# Secure Infrastructures in the Realm of Decentralization

**at the Faculty of Computer Science**
**of the Technische Universität Darmstadt**

Submitted in fulfilment of the requirements for the
Degree of Doctor rerum naturalium
(Dr. rer. nat.)

**Doctoral Thesis**

by **Poulami Das, M.Sc.**
born in Kolkata, India

First Assessor: Prof. Sebastian Faust, PhD.
Second Assessor: Prof. Aggelos Kiayias, PhD.

Darmstadt 2022

# Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Poulami Das

(P. Das)

## Curriculum Vitae

*June 2008 till June 2012:* Bachelor of Computer Science and Engineering at Jadavpur University, India

*February 2014 till January 2017:* Master of Science in Computer Science and Engineering at Indian Institute of Kharagpur, India

*March 2018 till September 2022:* Doctoral studies at Technical University of Darmstadt, Germany at the Chair of Applied Cryptography under Prof. Sebastian Faust, PhD.

# Acknowledgments

My PhD years at TU Darmstadt will always remain specially etched in my memories – not only because it has been one of my best learning experiences, but also because it helped me to grow as a person. To start with a personal anecdote, I came from a hardware security background and stumbled upon cryptography research papers during an internship at MPI-SWS, Saarbruecken. Although my internship project was distantly related to cryptography, I got instantly intrigued by the magic of public key cryptography.

Thank you to my PhD supervisor Sebastian Faust, I could consequently pursue my interest in cryptography through my PhD research. I am immensely thankful to Sebastian for guiding me through the process of finding interesting research problems to work on; for being a constant source of encouragement and positivity. I would like to sincerely thank Aggelos Kiayias for agreeing to be my external examiner; Carsten Binnig for chairing my PhD defense; Jan Gugenheimer and Matthias Hollick for being in my PhD committee.

I want to express my sincere gratitude towards Julian, who co-mentored me in my first project, explaining many foundational concepts and having countless discussions that helped me to make progress. I am thankful to Julia who always made me remember, how important it is to strengthen the overall background knowledge, besides working on own projects; Designing exercises for the Introduction to Cryptography course had been an enjoyable as well as an amazing learning experience with Max; I was fortunate to work with Andreas in follow-up projects; a regular exchange of ideas and discussions kept me motivated to continue.

I want to express my heartfelt thank you towards: Clara, Kristina and Lisa, for always offering help and creating a lively environment in the office; Max and Andreas for making the coffee breaks worth looking forward to; Elena for being a sister and the best office-mate.

I would like to thank Dorothee and Jacqueline for always supporting me with the official procedures; Heike Schmitt-Spall for patiently answering all my questions regarding the formal process of thesis submission. This thesis is incomplete without my sincere gratitude towards the collaborative research center of CROSSING for funding my research work, for providing educational workshops and a platform to collaborate and connect with other researchers.

I would like to thank my Darmstadt flatmates (Can, Natalie, Olha) for the nice times spent together that helped me to cope with the stressful times; friends from outside academia, especially Ankita, Neha, Kamelia, Sohini for always being there; And final gratitude towards my family for being my core strength and encouraging me to follow my passion, always.

Poulami Das

Saarbrücken, 24.08.2023

# List of Own Publications

## Part of this Thesis

[6] N. A. Alkadri, P. Das, A. Erwig, S. Faust, J. Krämer, S. Riahi, and P. Struck. "Deterministic Wallets in a Quantum World". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020.* 2020, pp. 1017–1031. DOI: 10.1145/3372297.3423361. URL: https://doi.org/10.1145/3372297.3423361.

[41] P. Das, L. Eckey, S. Faust, J. Loss, and M. Maitra. *Round Efficient Byzantine Agreement from VDFs.* Cryptology ePrint Archive, Paper 2022/823. https://eprint.iacr.org/2022/823. 2022.

[43] P. Das, A. Erwig, S. Faust, J. Loss, and S. Riahi. "The Exact Security of BIP32 Wallets". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communication Security, Virtual Event, Republic of Korea, November 15 - 19, 2021.* 2021, pp. 1020–1042. DOI: 10.1145/3460120.3484807. URL: https://doi.org/10.1145/3460120.3484807.

[45] P. Das, S. Faust, and J. Loss. "A Formal Treatment of Deterministic Wallets". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019.* 2019, pp. 651–668. DOI: 10.1145/3319535.3354236. URL: https://doi.org/10.1145/3319535.3354236.

[48] P. Das, J. Hesse, and A. Lehmann. "DPaSE: Distributed Password-Authenticated Symmetric-Key Encryption, or How to Get Many Keys from One Password". In: *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022.* 2022, pp. 682–696. DOI: 10.1145/3488932.3517389. URL: https://doi.org/10.1145/3488932.3517389.

## Other Publications

[42] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A. Sadeghi. "FastKitten: Practical Smart Contracts on Bitcoin". In: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019.* 2019, pp. 801–818.

[49] P. Das, D. B. Roy, and D. Mukhopadhyay. “Automatic Generation of HCCA Resistant Scalar Multiplication Algorithm by Proper Sequencing of Field Multiplier Operands”. In: *PROOFS 2017, 6th International Workshop on Security Proofs for Embedded Systems, Taipei, Taiwan, September 29th, 2017.* 2017, pp. 33–49.

[50] P. Das, D. B. Roy, and D. Mukhopadhyay. “Automatic generation of HCCA-resistant scalar multiplication algorithm by proper sequencing of field multiplier operands”. In: *J. Cryptogr. Eng.* 3 (2019), pp. 263–275.

[51] P. Das, D. B. Roy, and D. Mukhopadhyay. “Improved Atomicity to Prevent HCCA on NIST Curves”. In: *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, AsiaPKC@AsiaCCS, Xi’an, China, May 30 - June 03, 2016.* 2016, pp. 21–30.

[130] D. B. Roy, P. Das, and D. Mukhopadhyay. “ECC on Your Fingertips: A Single Instruction Approach for Lightweight ECC Design in GF(p)”. In: *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers.* 2015, pp. 161–177.

# My Contribution

All projects included in this thesis are results of fruitful collaboration between my supervisor Sebastian Faust and my co-authors. The following text is an attempt to highlight my individual contributions in each of the projects, included as a part of this thesis.

Chapter 2 is based on joint works with Nabil Alkeilani Alkadri, Andreas Erwig, Sebastian Faust, Juliane Krämer, Julian Loss, Siavash Riahi and Patrick Struck. Together with Julian and Sebastian we put forth the first formal treatment of deterministic wallets in [45]. Through several iteration of discussions, we formalized the security model of deterministic wallets, which serves as a foundational ingredient to analyze the security of a wide class of wallet schemes. I worked with Julian in writing down the security definitions of wallet security properties and building our generic wallet construction. Besides, I was solely responsible for writing down the security proofs of wallet security properties and unforgeability of a multiplicatively rerandomized variant of an ECDSA scheme – which was the main technical contribution of this work. This being my first PhD project, several feedback rounds from Julian and Sebastian immensely helped to get the formalization into a good shape. Our follow-up work [43] was a natural extension of [45], where we considered the hierarchical feature of BIP32 wallets along with additive key derivation based ECDSA. This was a joint work with Andreas, Julian, Sebastian and Siavash. The security model for the hierarchical setting came into shape after several intense discussions between me, Andreas and Siavash. Andreas, Siavash and I worked together on writing down the security model. As for the security analysis, I mainly focused on proving the additively rerandomized ECDSA and the salt-free multiplicatively rerandomized ECDSA schemes. While Andreas mostly worked on the unforgeability proof of the hierarchical wallet construction as well as the impossibility proof of a tighter bound for wallet unforgeability. Siavash helped both Andreas and me in writing down the security proofs. I also worked on putting together the concrete security parameters of BIP32, based on multiplicatively as well as additively rerandomized ECDSA. Our third work [6] on deterministic wallets was another natural extension to [45]. This was a CROSSING collaboration between me, Andreas, Siavash, Sebastian with Juliane, Nabil and Patrick on post-quantum security. The main idea behind this work was to achieve post-quantum security of deterministic wallets. Andreas, Siavash and I had several detailed discussions with Nabil and Patrick on figuring out the main adaptations of our security model to accomodate a quantum adversary, with access to the quantum random oracle. Andreas and me worked on laying down our post-quantum variant of formal definitions, generic construction and security model. Patrick focused on proving the security properties of the generic wallet construction, while Nabil's main focus was to construct a rerandomizable signature scheme from Lattice-based Fiat-Shamir signatures. Both Andreas and I were involved in writing down the security proofs with Patrick and Nabil, to get the formalization into a good shape. While, Siavash and Andreas worked together on the practical evaluation section. In Chapter 2, I summarize the main results from the above three works, primarily in my own words. Some of the background and formal definitions have been taken verbatim from the first work [45].

Chapter 3 is based on joint work [41] with Lisa Eckey, Julian Loss, Monosij Maitra and Sebastian Faust. Originally, this project was initiated between Julian, Lisa and Sebastian to design round efficient byzantine agreement in the resource-constrained model. I joined this project at a later point and took over as the lead author. Through many earlier discussions between Julian, Lisa and me, we started to lay down the main ideas of our protocols. Later, it took many months of discussions and iterations between Julian, Sebastian and me to finalize our computational model based on verifiable delay functions. Julian focused on the formalization of our model, while I focused on designing all necessary protocols and their associated security analysis. Monosij joined our project towards the end and helped us in proving the sequentiality property of VDF construction from Wesolowski and Pietrzak. Chapter 4 is based on a joint work [48] with Julia Hesse and Anja Lehmann on a distributed password-based cloud storage solution for end-users. The main idea of this project was put forth by Anja. Anja, Julia and I had several rounds of back and forth discussions to construct efficient protocols for our use-case. I focused on the protocol design and description with help from Julia and Anja. Besides, I was also responsible for the practical implementation of our protocols. While, Julia took care of the security analysis. Chapter 3 and 4 contains text, taken verbatim from the articles [41, 48] of the above two projects, where I was involved as one of the main authors.

# Abstract

Most of today's online services such as e-commerce and online banking are based on centralized service providers, hence easily prone to single points of failure, cyber-attacks and censorship. An alternative approach to mitigate this issue, is to design decentralized systems, where the control is distributed among several entities, instead of a single centralized authority. A decentralized system often consists of a complex distribution of trust among different parties, or even organizations. Henceforth, it is often challenging to build a decentralized system. It is not surprising that security guarantees of such complex decentralized systems depend on several factors: not only conventional properties of data integrity, confidentiality and authentication, but also other relevant factors such as availability, accountability and authorization. In this thesis, we aim to build *provably secure infrastructures* that serves a decentralized system in one way or another. Our contribution is in three different settings of blockchain, byzantine agreement and cloud.

Firstly, we consider the decentralized payment platform offered by blockchains. Although the core blockchain protocol has been thoroughly analyzed, the underling infrastructure of blockchain wallets is rather ad-hoc. Cryptocurrency wallets constitute an indispensable key management mechanism for every user that wants to send or receive blockchain payments. However, it lacks formal security analysis. We close this gap by designing provably secure wallets in presence of a classical as well as a quantum adversary [6, 43, 45]. Through our security analysis, we provide concrete bit security achieved by BIP32 wallets, which is a wallet standardization deployed in many practical wallets. Interestingly, we observe that slightly modified yet equally efficient version of BIP32 achieves higher level of bit security than the original version.

Secondly, we consider the problem of byzantine agreement (BA) – a fundamental problem in distributed computing as well as an important building block in the design of decentralized systems. Classically, byzantine agreement is known to be impossible without a public key infrastructure (PKI), in presence of a corruption threshold of $< \frac{n}{2}$ parties. Interestingly, a class of BA protocols has emerged that overcomes this well-known impossibility by taking inspiration from the decentralized model of blockchains. This setting allows a group of pseudonymous parties to achieve consensus, via some proof of computation, for instance, proof of work (PoW). Taking inspiration from the above mentioned computational model, prior works were able to achieve BA protocols with time complexity of $O(n\kappa^2)$, $O(\kappa)$ or $O(n)$. We show for the first time [41], a BA protocol, in the computational model of verifiable delay function (VDF) that runs in expected constant time. On the negative side, we are able to show a lower bound on the communication complexity of such protocols. Precisely, we prove that no protocol can achieve BA in the VDF model without any PKI assumption, in less than $O(\sqrt{n})$ send-to-all steps.

Thirdly, we consider the setting of cloud storage for end-users, where we aim to design an usable yet secure encryption service that overcomes the dependence on centralized service providers. To this end, we build our primitive: Distributed Password-authenticated Symmetric-key Encryption [48] (DPaSE) that enables users to generate strong encryption keys from a single password with the assistance of a set of $n$ servers. We use a new variant of an oblivious PRF (OPRF) as the main building block to build DPaSE.

# Contents

# 1. Introduction

Since the upsurge of the Internet, there has been a tremendous shift in digitization through a steady growth in e-commerce, online banking, cloud services, and the list goes on. Each of these services demands a secure digital infrastructure comprising of small to large scale devices, exchanging information over insecure channels. Through the key principles of confidentiality, integrity and authenticity of data, cryptography serves as an essential tool to make such infrastructure secure against cyberattacks. To take a concrete example, let us consider a user Alice, who wants to store her files with a cloud storage provider such as Dropbox. First, Alice must provide her credentials (usually a username and password) in a login portal. Only if the credentials are correct then she is allowed to access her Dropbox account, containing her confidential data. The above two actions are securely attained through the authenticity and confidentiality properties, respectively. Authenticity can be achieved from cryptographic primitives such as digital signatures while, encryptions guarantee confidentiality. Without authenticity, an attacker can try to impersonate Alice and get access to her account. Alice's data will be furthermore stored in encrypted form in Dropbox, since an attacker could try to break in the storage server, bypassing the login portal. Even then, due to the confidentiality of the encrypted data, it will not be able to access Alice's files in plain text.

The above example of Dropbox is however a centralized service provider, where all operations are solely controlled by Dropbox. If Dropbox somehow turns malicious, then it can simply access Alice's files in plaintext, thus violating the confidentiality guarantees. While this is just a mere example, most of today's widely used online services such as online banking and e-commerce rely on a centralized service provider and consequently they are prone to single points of failure [13], cyber-attacks and censorship. The above problem can be mitigated through decentralization, where the control is shared among several independent entities instead of a centralized authority. Similar to a distributed system, a decentralized system requires numerous parties to jointly solve a common task through continuous exchange of messages. A distributed system, however, can still be controlled by a single authority. Current tech-giants of Google, Facebook, Microsoft are all instances of distributed yet centralized systems. A decentralized system [137], on the other hand, is built from a complex distribution of trust among several parties or even organizations, managing different aspects of the system. Few examples of currently deployed decentralized systems are mixnets [34], Tor [52], BitTorrent [20] and blockchains [29, 120]. Achieving security in a decentralized system depends on several factors; not only conventional security guarantees of confidentiality, integrity and authentication, but also other relevant factors such as availability, accountability, authorization and non-repudiation. Henceforth, ensuring security of a decentralized system often involves numerous cryptographic tools, such as advanced signature schemes [91], threshold cryptography [77, 101], zero-knowledge proofs [27]; as well as techniques from distributed computing such as consensus protocols.

A fundamental tool in modern cryptography to formally analyze the security of the underyling cryptographic protocols and schemes is *provable security* [94]. In this technique, first, a scheme is identified with a set of well-defined security properties (examples from above: data confidentiality, authentication) that it must satisfy. Next task is to define a security model, capturing adversarial capabilities. Finally, the scheme is shown to satisfy the security properties, in presence of an attacker, in the specified model. Such a proof is usually conducted via a *reduction*, where, (on a high level) it is shown that, if it is hard to break a hardness assumption A, then it is also hard to break the scheme. In fact, tools from provable security lets us improve the security analysis of a scheme, in several ways; considering an alternative security model that captures a more realistic attacker or substituting with a better hardness assumption, and so on.

Our goal in this thesis is to build provably secure infrastructures that serve the purpose of decentralization in one way or another. Our contribution is in three different settings of blockchain, byzantine agreement and cloud computing as listed below:

- Firstly, we consider the setting of a decentralized payment system provided by blockchains. An important blockchain infrastructure is the cryptocurrency wallets. Currently, such wallets are built in a rather ad-hoc fashion, lacking any formal security analysis. We fill this gap by providing a comprehensive security analysis of blockchain wallets [6, 43, 45].

- Secondly, we consider the problem of Byzantine agreement, also referred to as consensus: a crucial component in the design of decentralized systems. Inspired by the setting of blockchains, we design a round-efficient byzantine agreement protocol from Verifiable Delay Functions (VDF), without requiring a Public Key Infrastructure (PKI) as a setup assumption, tolerating the optimal threshold of $\frac{n}{2}$ corruptions [41]. As an immediate application, our byzantine agreement provides a decentralized solution to the generation of an unpredictable beacon – inherently required during setup of most blockchains.

- Finally, we provide a cloud storage solution, as an alternative to centralized providers (for example, Google Drive, OneDrive, Dropbox). To this end, we build a distributed password-authenticated encryption scheme, that enables secure cloud storage for end-users [48].

Before summarizing the contribution of this thesis, we first discuss each of the settings in the following Sections 1.1 to 1.3.

## 1.1. Blockchain Infrastructure

Cryptocurrencies and its underlying blockchain technology offer a solution to overcome the centralization of traditional banking systems. It provides a decentralized platform without the requirement of a trusted third party, that thwarts single points of failures, thus resisting cyberattacks and censorship. Although more than a decade has passed since the first cryptocurrency Bitcoin was proposed [120], until today, the cryptocurrency ecosystem suffer from numerous issues, in terms of security. In fact, the infrastructure surrounding blockchain is designed in a rather ad-hoc way. In particular, each cryptocurrency user needs to maintain a digital wallet to store its coins (keys) and interact with the blockchain for making payments. Such wallets are often mishandled, and hence suffer from countless attacks, resulting in users losing their money [38]. Below, we give a brief overview of the key blockchain principles followed by highlighting the major challenges in designing secure blockchain infrastructure.

**Blockchain Principles.** At the core of blockchain is a peer-to-peer network of anonymous parties, often referred to as *miners*, that jointly run a consensus protocol to agree on a globally consistent append-only record of financial data. This record is nothing but a set of transactions that are collected into consecutive *blocks* to form a *blockchain*. Each transaction corresponds to a monetary transfer between two users. Miners are responsible for collecting sets of incoming transactions, check their validity according to a well-specified set of rules before including them into a block. Next, to propose this newly formed block, they must solve a puzzle (computational puzzle in case of Proof of Work (PoW)). Once a solution is obtained, the block is send around to other miners. If only other miners are convinced, then this block is accepted as the *next block*. This process is iterated to form a *chain* of blocks. Here, proof of work acts as a voting mechanism, representing one vote per computation. This prevents an adversary from flooding the network with fake votes, hence obtaining a dishonest majority of votes. Such an attack is known as the *Sybil attack* [56]. The security guarantees [70, 96] underlying a blockchain consensus relies on an honest majority assumption, which means that the majority of the resources in the system (computational power in case of PoW) belongs to honest parties. Additionally, it requires to assume the existence of an unpredictable beacon during setup of the protocol, typically referred to as the *genesis block*. Indeed, Bitcoin's genesis block, contains text from The Times 03/Jan/2009 issue [144], which is not truly unpredictable but at most a heuristic. While the above mechanism of genesis block creation is sometimes problematic, there exist a more decentralized solution to genesis block creation as we will discuss in Section 1.2.

Cryptocurrency users are able to send or receive payments by interacting with the blockchain. In particular, they need a device to store the relevant information for sending/receiving coins. This device is usually referred to as a blockchain wallet. Below we discuss the core payment mechanisms in a blockchain through the use of wallets.

**Blockchain Wallets.** Let us consider a user Alice who wants to make a payment to Bob over the blockchain. In order to proceed with this payment, Alice first needs to know Bob's address which is represented by Bob's public key $\mathsf{pk}_\mathsf{B}$. Next, Alice prepares a transaction $\mathtt{tx}_\mathsf{AB}$ that roughly says: "Alice pays $v$ coins to Bob". $\mathtt{tx}_\mathsf{AB}$ must be authenticated by Alice, so that it is accepted as a valid transaction. For authentication, Alice digitally signs the transaction with her secret key $\mathsf{sk}_\mathsf{A}$, via a digital signature scheme. As is evident from this example, both Alice and Bob requires a device to store a pair of keys for spending (secret key) and receiving (public key) coins. The first choice is to store the key pair in an online wallet app, running on a smartphone. Since such a device is always connected to the internet, an attacker could get hold of her keys and steal all the money attached to Alice's secret key. Henceforth, a wallet containing secret key is an attractive target for hackers [38]. For safe-guarding the secret key, it is usually recommended to follow a hot wallet/cold wallet approach. Here, a hot wallet is a device, connected to the Internet; for instance, it can be the software application mentioned above, storing the public key. On the other hand, the cold wallet is mostly an offline device, such as a USB stick or even a piece of paper that stores the secret key. Keeping the secret key in an offline storage while coming online only during a payment largely reduces the risks of attacks. However, since the receiver's addresses are public on the blockchain, it is easily traceable for an attacker how much money is sent to a particular address. This makes the wallets with larger amounts of money an attractive target for hackers, who can attempt to steal the money during the online period of the cold wallet. This problem is addressed by using freshly generated keys for every payment, via the process of deterministic key derivation as proposed in the Bitcoin Improvement Proposal 32 (BIP32) [116]. Although this standard is currently deployed in many wallets such Trezor and Ledger [106, 136], surprisingly there has been no formal security analysis that specifies the security guarantees that such a standard comes with. In this thesis, we make progress in filling this gap. Our contribution is summarized in Section 1.4 and discussed in details in Chapter 2.

## 1.2. Byzantine Agreement

Byzantine Agreement (BA) is a fundamental problem in distributed computing, where a set of $n$ parties run a distributed protocol to agree on a value. Traditionally, most existing protocols for BA assume a setting in which the parties' identities are fixed and known at the beginning of the protocol. In the fixed identity setting, two types of protocols are studied: the first type requires setup, e.g., a Public Key Infrastructure (PKI) or some form of correlated randomness. These protocols typically tolerate the (optimal) corruption threshold of $t < \frac{n}{2}$. The second type does not require such assumptions but can tolerate only $t < \frac{n}{3}$ corruptions.

More recently, a third type of BA protocol has emerged that takes inspiration from the decentralized setting of blockchain consensus [10, 95]. Precisely, it does not require parties' identities to be known at the beginning of the protocol but still achieves the optimal corruption threshold of $\frac{n}{2}$. Moreover, these protocols do not assume the conventional setup assumption of a PKI. As a first hurdle, if identities are not fixed then without further measures, every party could pose as many parties and easily obtain a dishonest majority; this is commonly referred to as a *sybil attack* [56]. To avoid such attacks, parties must instead invest some expensive resources, such as computation or money, to participate in this type of protocol. A prominent example is the Proof-of-Work model (PoW) initially introduced by Bitcoin, where parties have limited access to a computational resource which they are forced to continuously expend in order to participate in the protocol. In the PoW model, it is possible to build a weaker variant of a PKI, namely a *Graded PKI*, by assigning 'ranks' to keys, such that honest parties do not disagree by much. Such a graded PKI establishes a pseudonymous identity infrastructure and has several applications such as multi-party computation in [10], byzantine agreement in [10, 72], leader election and genesis block creation in [72]. The round complexity of existing graded PKI is however $O(n\kappa^2)$ or $O(\kappa)$ (respectively in [10, 72]) leading to the same complexity for BA. In this thesis, we fill the gap of round-efficient BA by achieving (expected) constant round complexity. As an immediate application, setup-free BA enables a decentralized solution to create the genesis block of a blockchain [144]. Our contribution on round-efficient BA is summarized in Section 1.4 and detailed in Chapter 3.

## 1.3. Cloud Infrastructure

Cloud data storage for end-users is as crucial as for enterprises. Today, users can upload their data with centralized Cloud storage providers such as Google Drive, Dropbox, OneDrive etc. However, data privacy can be guaranteed only if the service provider is not compromised. In the following, we discuss the possibilities for the users' cloud data storage without entrusting a centralized provider. As a first attempt, users could encrypt their data on their own and then store the encrypted data with a service provider. The challenge here is to find a usable solution such that users are able to generate strong keys without much trouble. One of the most well-explored solution is to derive strong keys on the fly from a human-memorizable password. We look into password-based solutions in a bit more details below.

**Password-based Approach.** Let us first present a naive solution, such that a key is derived from a small password and thereafter used to encrypt files. An attacker could access the encrypted files at the cloud storage and simultaneously attempt to guess the password. Since the password is usually a short string with low entropy, the attacker could mount an attack offline, referred to as an *offline dictionary* attack. With the correct password guess, the attacker can decrypt all of the files, rendering the scheme insecure. As can be observed, storing the ciphertexts in a single location enables an attacker to successfully mount an attack. There has been a long line of work in the client-server setting, that addresses this problem by distributing the server storage/computation among a set of servers. This helps in mitigating single points of failures at the server end. One of the worth-noting schemes is Password-protected Secret Sharing, also termed as Password-Authenticated Secret Sharing (PASS) [14, 86, 87, 88].

On a high level, PASS is a protocol run between a client and $n$ servers. In PASS, a password from the client triggers generation of a secret key, secret shared among the $n$ servers. The password, on the other hand, is only known to the client and kept secret from the servers. Such a PASS scheme can then be augmented with a protocol for encrypting files via threshold encryption. As a result client's encrypted files will be stored in a distributed manner among the servers, such that

$t+1$ servers are required to recover the decrypted files. At this point, the previous offline attack would not work since the adversary needs to corrupt $t+1$ out of $n$ servers to recover the data, after a successful dictionary attack. However, PASS schemes suffer from several drawbacks when used for data encryption. Firstly, a PASS scheme does not provide password authentication. Precisely, a client can mistype a password and encrypt her file with password $pw^*$. Later, she tries to decrypt the same file with password $pw$, because she is not aware of her mistake in typing from the previous time. This leads to file locking that the client can not recover anymore. Secondly, if the same PASS key is used to encrypt all files, then a compromise of the encryption key leads to decryption of all files. Generating different PASS keys based on different passwords is a cumbersome solution, since now the user needs to remember many passwords. As a result, we clearly need a usable but secure solution for user data storage. Our contribution is summarized in Section 1.4 and detailed in Chapter 4.

## 1.4. Thesis Outline and Summary of Contributions

Below, we summarize the overall contribution of this thesis. Chapter 2 covers the security analysis on the infrastructure surrounding blockchains aka blockchain wallets. In Chapter 3, we discuss our design of round-efficient Byzantine agreement in the resource-contrained model. In Chapter 4, we present a distributed cloud storage service for end-users, as an alternative to centralized providers. Finally, we conclude in Chapter 5.

**Chapter 2.** In this chapter, we summarize our contributions towards provably secure blockchain infrastructure. As a first contribution, we analyze the BIP32 standardization [143] of hierarchical deterministic wallets as used in practice and ask the following question:

*What security guarantees are obtained from BIP32 wallets as is?*

To this end, we formally define the security model for deterministic wallets in the hot wallet/cold wallet setting. Typically, a deterministic wallet has a pair of master keys – a *master public key* stored in the hot wallet, while the *master secret key* is kept in the cold wallet. Both wallets share a common state, referred to as the *chaincode.* The key idea is to derive session keys deterministically from the respective master keys without any interaction between the two wallets. Our model captures adversarial capabilities through the security games of *wallet unlinkability* and *wallet unforgeability.* In short, a deterministic wallet is unlinkable, if a session public key generated from the hot wallet is indistinguishable from a random public key, given that the hot/cold wallets are secure. Recall that, in a standard game of Existential Unforgeability Under Chosen Message Attack (UFCMA), an adversary sees a challenge public key and can ask signatures signed under the corresponding secret key. On the other hand, in the wallet unforgeability game, an adversary is given access to a master public key, and any (polynomial) number of signatures signed under session keys. Although the unforgeability game looks similar to the standard game of UFCMA, proving unforgeability in this setting is far from straightforward. This is because here the game needs to simulate signatures under any (polynomial) number of *related* session secret key, without knowing any of these secret keys.

We provide two provably secure variants of deterministic wallets in our model. To this end, we follow a modular approach and provide a generic construction of deterministic wallets from signature scheme with rerandomizable keys. Through our modular analysis, we are able to show that wallet unforgeability holds as long as the underlying rerandomizable signature scheme is unforgeable. To instantiate our wallet, we consider the signature schemes of Schnorr [131], BLS [23] and ECDSA [91]. It was already known that Schnorr satisfies the rerandomizability property of signature schemes [68]. We show it for the first time for ECDSA and BLS. Particularly, the security proof for ECDSA tackles many subtle challenges in the security analysis. We consider both additive and multiplicative rerandomization of keys for ECDSA and observe that ECDSA with multiplicative rerandomization achieves higher level of security than the additive variant. To capture the formalization of BIP32 as is, we extend our model of deterministic wallets to the hierarchical setting of key derivation, where we are able to support secret key leakage of *hardened wallets*. We show that unforgeability of the hierarchical deterministic wallet incurs security loss in terms of the number of secret key leaked and such loss is inherent. Since BIP32 is built from ECDSA [143], our results on additive vs multiplicatively rerandomized ECDSA also have direct implications for BIP32. Concretely, we show that BIP32 with multiplicatively rerandomized ECDSA achieves higher level of security than the additive variant, considering 128 bit security for ECDSA. The detailed summary of our results can be found in Section 2.2.

As a second contribution, we explore post-quantum security of deterministic wallets and ask the following question:

*Can we build post-quantum secure deterministic wallets?*

On a high level, in the post-quantum setting, an adversary has access to a quantum computer while the wallet scheme is implemented in a classical machine. Due to the results of Shor [132], it is well-known that the wallet schemes discussed above are rendered insecure in the post-quantum world. In particular, for a classical signature scheme like ECDSA, a quantum attacker could extract the master secret key from the master public key, derive any session secret keys and forge signatures. This motivates us to design deterministic wallets that are post-quantum secure. To this end, we extend our model to the post-quantum setting, where the adversary has access to a quantum random oracle. We follow our modular approach for the generic wallet construction from (post-quantum) signature schemes with rerandomizable public keys. We instantiate the later with lattice-based Fiat-Shamir Signatures [97]. In particular, we show the exact bit security of our wallet once instantiated with qTESLA, which is one of the NIST second round finalists. Our contribution is detailed in Section 2.3.

**Chapter 3.** Motivated by the decentralized setting of blockchain consensus, we consider the problem of byzantine agreement without a PKI, with the optimal corruption tolerance of $\frac{n}{2}$ parties. We ask the following question in the above setting:

*Can we a build a round-efficient BA from scratch?*

We answer the above question affirmatively by designing an (expected) constant round BA protocol. To this end, we refine the Proof of Work (PoW) model and consider sequential computational power of parties via evaluation of a Verifiable Delay Function (VDF). In other words, to evaluate a VDF proof, certain number of sequential steps must be computed. The adversarial capabilities in the VDF model subtly differ from the PoW model. In PoW, it is possible to speedup computation of a single proof through increased parallelization. While in the VDF model, a single proof cannot be sped up, no matter how much parallelizable computational power an adversary may possess. Parallelization, however, allows to compute multiple proofs parallely. Additionally, the adversary might have faster computational speed than an honest party, such that it computes each sequential step faster. Although note that, increased speed does not affect the number of sequential steps to be computed to finish a proof. In our model, we abstract one VDF evaluation as one call to a VDF *oracle*. The capabilities of an adversary in the VDF model, are captured through the *sequentiality* property of the VDF oracle. In short, the sequentiality property ensures that the adversary, controlling $q$ parties should not be able to produce more than a fixed number of VDF outputs within a time duration of $\delta$. Such oracle abstraction of the VDF model allows us to modularly build and analyse our protocols. We explain our VDF oracle in details in Section 3.1.1.

We follow the approach from [10] and build a constant round Graded PKI (GPKI). To achieve constant round, we firstly replace a PoW with a VDF computation. Secondly, we replace $n$-graded ranking in [10] with simply 2 grades. Once the GPKI is setup, we design a graded consensus protocol and a VDF-based leader election protocol following approaches from [119] and [1, 2] respectively. Finally, we adapt the (expected) constant round BA protocol from [93] to run with a GPKI, instead of a full PKI. We use our graded consensus and leader election protocols as building blocks in the BA protocol.

Additionally, we show a lower bound on the communication complexity of BA without a PKI, in the same setting, where parties have access to a VDF oracle. For our analysis, we consider the multicast model of communication, where parties send the same message to everyone in the network. Precisely, a multicast complexity of $\theta$ corresponds to a classical communication (with bilateral channels) complexity of $n\theta$. We show that a minimum of $O(\sqrt{n})$ multicast complexity is required to achieve BA.

**Chapter 4.** This chapter focuses on the design of a secure cloud infrastructure for end-users. Currently, users trust centralized service providers such as Google Drive, OneDrive, Dropbox with the privacy of their confidential documents. If a service provider turns malicious, then the user's data privacy is violated. In this regard, we ask the following question:

*Can we design a usable but secure cloud storage service for end-users without entrusting centralized providers?*

To this end, we introduce our primitive Distributed Password-authenticated Symmetric-key Encryption (DPaSE). DPaSE is a key management system that enables creation of strong encryption keys from a single password as well as serves as an encryption service. As a result, DPaSE comes with a set of security properties relating to both passwords and encryption: namely, protection against offline/online attacks on passwords, partial obliviousness, no reuse of (encryption) keys and finally correct and authenticated encryption.

Our DPaSE construction is run as a protocol between a client and a set of servers. The high level idea of our DPaSE protocol follows the usual paradigm of password-based protocols [16, 90], where a strong cryptographic key material is generated from a password from a special PRF evaluation, namely an *Oblivious*

*PRF (OPRF).* An OPRF allows to *hide* some of the inputs (in our case, the password), from the parties evaluating the PRF. First, to create an account, a user provides a username $uid$ and a password $pw$. A pair of keys is generated as $(upk, usk) \leftarrow \mathsf{OPRF}(uid, pw, K)$, where the OPRF key $K$ is split into $n$ shares among the set of $n$ servers. The key pair is sent to the client as output, whereas the public key is sent back to the servers and is recorded as $(uid, upk)$. The password remains hidden from the servers during the OPRF evaluation. To encrypt or decrypt a file, the user first provides her combination of username and password $(uid, pw')$. The previous OPRF is re-evaluated exactly as before $(upk', usk') \leftarrow \mathsf{OPRF}(uid, pw')$ and password correctness is verified against $upk == upk'$ at the server end. With a successful password authentication, the client can proceed with an encryption or decryption of her files with a follow-up OPRF evaluation. The second OPRF thereby "reuses" the previously entered $uid, pw'$ to ensure that the actual encryption keys are also bound to the user' identity and correct password. This prevents users from accidentally encrypting data under a wrong password. To ensure obliviousness, the object for which the key is derived is hidden in the evaluation.

As mentioned above, we require a OPRF for our DPaSE construction. Our OPRF must satisfy the properties of partial obliviousness, distribution and extendability to enable DPaSE security. To this end, we introduce a new oblivious PRF variant, namely Extendable Distributed Partially-Oblivious PRF (edpOPRF), which is of independent interest. Protection against offline/online attacks in DPaSE are achieved due to the properties of distribution and partially obliviousness in epdOPRF. Lastly, the extendability property allows to run two consecutive edpOPRF evaluations on correlated oblivious inputs. More concretely, when the user provides the correct password, then correct encryption is guaranteed by running the two consecutive edpOPRF evaluations with the same correct password as input.

We use the universal composability framework to prove security of edpOPRF and in turn DPaSE. To this end, we provide UC functionalities for both of these primitives. Finally, we provide a proof-of-concept implementation of our DPaSE construction.

# 2. Deterministic Wallets from Rerandomizable Signatures

Money mechanics in a Blockchain are typically carried out via transactions. Let us recall the same through a concrete example. Say Alice wants to pay Bob $v$ coins; this is usually done by transferring $v$ coins from Alice's address, represented by her public key $\mathsf{pk_A}$ to Bob's address $\mathsf{pk_B}$. To this end, Alice creates a transaction $\mathtt{tx_{AB}}$ that roughly says "Transfer $v$ coins from $\mathsf{pk_A}$ to $\mathsf{pk_B}$". To ensure the authenticity of such a transfer, $\mathtt{tx_{AB}}$ is accompanied by a valid signature of $\mathsf{H}(\mathtt{tx_{AB}})$. Since only the owner of the corresponding $\mathsf{sk_A}$ – here Alice – can produce a valid signature, control over $\mathsf{sk_A}$ implies full control over the funds assigned to $\mathsf{pk_A}$. This makes secret keys a highly attractive target for attackers. Unsurprisingly, there are countless examples of spectacular hacks where the attacker was able to steal millions of dollars by breaking into a system and extracting the secret key [133]. According to the research firm Comparitech [38], the amount of cryptocurrencies stolen till date is more than 33 Billions worth of Dollars.

**Hot/Cold setting.** One reason for many of these attacks is that the secret key, controlling large amount of funds is often stored in an online wallet, referred to as the *hot wallet*. A hot wallet can be thought of as a software application, running on the user's smart phone or computer. On the positive side, the hot wallet makes it very convenient for the user to send or receive coins. However, due to the constant connection to the Internet, it is prone to attackers, that can exploit software vulnerabilities via malware or phishing. It is thus recommended to only keep a small amount of fund in the hot wallet, while, large amount of funds must be moved to an offline storage, referred to as the *cold wallet*. A straight-forward solution to construct a hot/cold wallet is to generate a key pair – a secret key and a public key; store the public key in the hot wallet, and the secret key in the cold wallet. Such a cold wallet may be realized in practice, via an USB device or even a paper wallet. With this storage separation, when Alice wants to receive coins from another user, Alice simply needs to publish her address, represented by her public key. Hence, only the hot wallet of Alice is involved during this process. Only when Alice wants to pay someone or wants to transfer coins to a different address, the cold wallet needs to come online, for a short period of time to sign the transaction.

Offline storage of the secret key in a cold wallet, majorly reduces the risks of online attacks. However, this solution suffers from an immediate drawback. Since transactions are published on the blockchain as public information, an attacker could easily trace how much coins in total, were sent to a specific address. Thus, addresses holding large amount of funds can be an attractive target for attackers. To mitigate this problem, it is usually recommended to use each secret key only once for signing a new transaction. This could be addressed by maintaining a set of $l$ randomly selected key pairs $\{(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2), \dots, (\mathsf{pk}_l, \mathsf{sk}_l)\}$ in respective wallets, i.e., store the set of public keys in the hot wallet, the set of secret keys in the cold wallet and use each secret key only once for signing a new transaction. Although this solution circumvents the linkability problem of public keys, it significantly blows up the key storage in each wallet, which grows linearly in terms of the number of keys. Looking at this problem a bit more closely, we notice that, we do not need truly random key pairs. It is sufficient to generate key pairs in such a way, that they look random on the blockchain. This is possible through a deterministic key derivation procedure which we explain next.

**Deterministic Wallets.** The solution to the key storage problem as mentioned above is so-called *Deterministic Wallets*. Here, the idea is to derive many *deterministic* (one-time) session keys from a single pair of master keys, so that the respective wallets need to store only the master keys. More precisely, such a wallet consists of a *master secret key msk* together with a matching *master public key mpk* and a deterministic *key derivation procedure*. At setup, the master public key is given to the hot wallet, whereas the master secret key is kept on the cold wallet. After setup, the hot and cold wallet can independently generate matching session keys using their respective key derivation procedures and master keys. In more details, the hot wallet runs the public key derivation procedure on an identifier *ID* and master public key *mpk* as inputs to produce session public key $\mathsf{pk}_{ID}$. While, the cold wallet correspondingly runs the secret key derivation algorithm on master secret key *msk* and *ID* as inputs to output the corresponding session secret key $\mathsf{sk}_{ID}$.

Informally, a deterministic wallet should offer two main security guarantees. First, an *unforgeability property*, which ensures that as long as the cold wallet is not compromised, signatures to authenticate new transactions can not be forged, and thus funds are safe. Second, an *unlinkability* property, which guarantees that public keys generated from the same master public key *mpk* are computationally indistinguishable from freshly generated public keys. Despite the widespread use of deterministic wallets (e.g., they are used in most hardware wallets such as *ledger* or TREZOR, and by common software wallets such as Jaxx), only limited formal security analysis of these schemes has been provided. In this thesis, we make progress to close this gap.

**Standardized Wallets in Practice: BIP32.** The BIP32 specification [143] standardizes the construction of a hierarchical deterministic wallet, as used by several cryptocurrencies deployed in the market. Let us look into a simpler variant of BIP32 (without the hierarchical component) to understand its core features. Since BIP32 is originally built on ECDSA, we take an ECDSA-based construction of deterministic wallets. Let $G$ denote the base point of an ECDSA elliptic curve. The deterministic ECDSA wallet uses an ECDSA key tuple as its master secret/public key pair, denoted by $(msk = x, mpk = x \cdot G)$. The master secret key $msk$ is stored on the cold wallet, while the corresponding master public key $mpk$ is kept on the hot wallet. In addition, the hot wallet and the cold wallet both keep a common secret string $ch$ which is called the "chaincode". To derive a new session public key with identifier $ID$, the hot wallet runs the deterministic public key derivation algorithm on inputs $mpk$, $ID$ and chaincode $ch$. Precisely, it computes $w \leftarrow \mathsf{H}(ch, ID), \mathsf{pk}_{ID} \leftarrow mpk + w \cdot G$. While, the cold wallet runs the deterministic secret key derivation algorithm on inputs $msk$, $ID$ and $ch$, computing the corresponding session secret key as $w \leftarrow \mathsf{H}(ch, ID), \mathsf{sk}_{ID} \leftarrow msk + w$. As argued, e.g., in [115], this construction satisfies both unlinkability and unforgeability as long as the chaincode and all derived secret keys remain hidden from the adversary.

Unfortunately, hot wallet breaches happen frequently, and hence the assumption that the chaincode stays secret is rather unrealistic. When $ch$ is revealed, however, the unlinkability property is trivially broken since the adversary can derive from $mpk$ and $ch$ the corresponding session public key $\mathsf{pk}_{ID}$ for any $ID$ of its choice. Even worse, a hot wallet security breach may in certain cases break the unforgeability property of the wallet scheme, through session secret key leakage [115] or a related key attack. As we will discuss in Section 2.2.1, our model formally captures the above-mentioned scenarios of an attacker in the hot/cold wallet setting.

The simpler construction of deterministic wallet as described above is extended to the hierarchical variant to capture the exact BIP32 specification. A hierarchical deterministic wallet can be thought of as a tree, where every node has a pair of public/secret keys. The key pair of each node has two roles: firstly they can sign/verify messages; secondly, they can derive key pairs for child nodes. In such a hierarchical setting, hot wallet breach of one or more of the wallets may break the unlinkability property of some wallets but must not break unforgeabilty of any of the wallets. It is also possible to consider secret key leakage of a wallet in this model through a slightly different key derivation termed as *hardened key derivation* in [143]. We summarize our formal security analysis of BIP32 as is in the hot/cold wallet setting in Section 2.2.

**Turning to Post-Quantum Security.** Although deterministic wallets such as BIP32 offer an elegant solution to increase the security of users' funds, they are particularly susceptible to attacks by quantum adversaries. More concretely, in a deterministic wallet, all session keys are related, and in particular efficiently computable from $(msk, mpk)$. Hence, if the adversary manages to learn $mpk$ then he can recover the corresponding master secret key $msk$ and thereby recover *all* session secret keys. Hence, all the money that was ever transferred to the cold wallet is at stake. This motivates us to make deterministic wallets post-quantum secure. Precisely, we design *post-quantum secure deterministic wallets* from (post-quantum secure) *signature schemes with rerandomizable public keys.* For the former, we need to carefully modify our model in the hot/cold setting to consider quantum attackers. While for the later we need to slightly modify the original definition of rerandomizable schemes and consequently show constructions from lattice-based signature schemes. We summarize our results on post-quantum secure wallets in section 2.3.

## 2.1. Preliminaries

**Notation.** We denote as $s \xleftarrow{\$} \mathcal{H}$ the uniform sampling of the variable $s$ from the set $\mathcal{H}$. If $\ell$ is an integer, then $[\ell]$ is the set $\{1, \ldots, \ell\}$. We use uppercase letters $\mathcal{A}, \mathcal{B}$ to denote algorithms. Unless otherwise stated, all our algorithms are probabilistic and we write $y \xleftarrow{\$} \mathcal{A}(x)$ to denote that $\mathcal{A}$ returns output $y$ when run on input $x$. We write $y \leftarrow \mathcal{A}(x, \rho)$ to denote that $\mathcal{A}$ returns output $y$ when run on input $x$ and randomness $\rho$. Note that in this way, $\mathcal{A}$ becomes a deterministic algorithm. We use the notation $\mathcal{A}(x)$ to denote the set of all possible outputs of (probabilistic) algorithm $\mathcal{A}$ on input $x$.

We write $\mathcal{A}^{\mathcal{B}}$ to denote that $\mathcal{A}$ has oracle access to $\mathcal{B}$ during its execution. For ease of notation, we generally assume that boolean variables are initialized to $\mathsf{false}$, integers are set initially to 0, lists are initialized to $\emptyset$, and undefined entries of lists are initialized to $\perp$. To further simplify our definitions and notation, we assume that public parameters $par$ have been securely generated and define the scheme or algebraic structure in context. We denote $\kappa$ as the security parameter. For bit strings $a, b \in \{0,1\}^*$ if we write "$a = (b, \cdot)$" we check if the prefix of $a$ is equal to $b$; likewise with "$a \neq (b, \cdot)$" we check if the prefix of $a$ is different from $b$.

**Background on Signature schemes.** A *signature scheme* $\mathsf{Sig}$ is given by a triple of algorithms $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$. $\mathsf{Sig.Gen}$ is a randomized key generation algorithm that takes security parameter $par$ as input and outputs a pair of keys - a secret key $\mathsf{sk}$ and a public key $\mathsf{pk}$. $\mathsf{Sig.Sign}$ is a randomized signing algorithm that takes the secret key $\mathsf{sk}$, a message $m$ as inputs and outputs a signature $\sigma$. Finally, the deterministic verification algorithm $\mathsf{Sig.Verify}$ takes message $m$, signature $\sigma$ and public key $\mathsf{pk}$ as inputs and outputs 1 for a valid signature, and 0 otherwise.

We require *correctness*: For all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{Sig.Gen}(par)$, and all $m \in \{0,1\}^*$, we have that

$$\Pr_{\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}, m)} [\mathsf{Sig.Verify}(pk, \sigma, m) = 1] = 1.$$

For our security analysis, we adopt the notion of signature schemes with rerandomizable keys from Fleischhacker et al. [68]. A *signature scheme with rerandomizable keys* $\mathsf{RSig}$ extends a normal signature scheme with two additional algorithms of key derivation. Precisely, it contains the following algorithms ($\mathsf{RSig.Gen}$, $\mathsf{RSig.RandSK}, \mathsf{RSig.RandPK}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}$), where ($\mathsf{RSig.Gen}, \mathsf{RSig.Sign}$, $\mathsf{RSig.Verify}$) works similar to the standard algorithms of ($\mathsf{Sig.Gen}$, $\mathsf{Sig.Sign}, \mathsf{Sig.Verify}$) in a signature scheme. Moreover, we assume that the public parameters $par$ define a randomness space $\chi := \chi(par)$. The randomized secret key derivation algorithm $\mathsf{RandSK}$ takes a secret key $\mathsf{sk}$ and a randomness $\rho \in \chi$ as inputs and outputs a secret key $\mathsf{sk}'$. The randomized public key derivation algorithm takes a public key $\mathsf{pk}$ and $\rho$ as inputs and outputs a corresponding public key $\mathsf{pk}'$. We make the convention that for the empty string $\epsilon$, we have that $\mathsf{RSig.RandPK}(pk, \epsilon) = pk$ and $\mathsf{RSig.RandSK}(sk, \epsilon) = sk$. We further require:

1. *(Perfect) rerandomizability of keys:* For all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{RSig.Gen}(par)$ and $\rho \xleftarrow{\$} \chi$, the distributions of $(\mathsf{sk}', \mathsf{pk}')$ and $(\mathsf{sk}'', \mathsf{pk}'')$ are identical, where:

$$(\mathsf{sk}', \mathsf{pk}') \leftarrow (\mathsf{RSig.RandPK}(\mathsf{pk}, \rho), \mathsf{RSig.RandSK}(\mathsf{sk}, \rho)),$$
$$(\mathsf{sk}'', \mathsf{pk}'') \xleftarrow{\$} \mathsf{RSig.Gen}(par).$$

2. *Correctness under rerandomized keys:* For all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{RSig.Gen}(par)$, for all $\rho \in \chi$, and for all $m \in \{0,1\}^*$, the rerandomized keys $\mathsf{sk}' \leftarrow \mathsf{RSig.RandSK}(\mathsf{sk}, \rho)$ and $\mathsf{pk}' \leftarrow \mathsf{RSig.RandSK}(\mathsf{pk}, \rho)$ satisfy:

$$\Pr_{\sigma \xleftarrow{\$} \mathsf{RSig.Sign}(\mathsf{sk}', m)} [\mathsf{RSig.Verify}(\mathsf{pk}', \sigma, m) = 1] = 1.$$

```
main $\mathbf{uf\text{-}cma}_{\mathsf{Sig}}$
00 $(sk, pk) \xleftarrow{\$} \mathsf{Sig.Gen}\,(par)$
01 $(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}}\,(pk)$
02 If $m^* \in Sigs$: $\mathsf{bad} \leftarrow \mathsf{true}$
03 $b' \leftarrow \mathsf{Sig.Verify}\,(m^*, pk^*, \sigma^*)$
04 Return $b' \wedge \neg\mathsf{bad}$

Oracle $\mathsf{Sign}\,(m)$
05 $\sigma \xleftarrow{\$} \mathsf{Sig.Sign}\,(sk, m)$
06 $Sigs \leftarrow Sigs \cup \{m\}$
07 Return $\sigma$
```

Figure 2.1.: Security game $\mathbf{uf\text{-}cma}_{\mathsf{Sig}}$ with adversary $\mathcal{A}$.

**Security of Signature Schemes.** We will use the standard security notion of *UFCMA*. We formalize this notion for a signature scheme $\mathsf{Sig}$ via the game $\mathbf{uf\text{-}cma}_{\mathsf{Sig}}$ (cf. Figure 2.1). In this game, the challenger begins by sampling $(\mathsf{sk}, \mathsf{pk})$ as $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Gen}(par)$. The adversary is then given the public key $\mathsf{pk}$ and can adaptively sign messages of its choice under the corresponding secret key via an oracle $\mathsf{Sign}$. Its goal is to forge a signature on a *fresh* message $m^*$, i.e., one that was not previously queried to the oracle $\mathsf{Sign}$.

For a signature scheme with rerandomizable keys $\mathsf{RSig}$, we also introduce a new security notion called *Unforgeability under Honestly Rerandomizable Keys (UFCMA-HRK)* that is formalized via game $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ (cf. Figure 2.2). This notion constitutes a weaker form of the notion of *Unforgeability Under Rerandomizable Keys (UFCMA-RK)* proposed in [68]. In the latter notion of UFCMA-RK, the adversary is able to query the signing oracle not only for signatures corresponding to the public key $\mathsf{pk}$ that it obtains in the unforgeability experiment, but also for signatures that correspond to *arbitrary rerandomizations of* $\mathsf{pk}$. Similarly, the winning condition is also relaxed in this notion by allowing the adversary to return a forgery under an (arbitrarily) rerandomized key (but still on a fresh message $m^*$). The main difference between the security notion of UFCMA-RK [68] and our new notion of UFCMA-HRK is that the adversary is restricted to *honest* rerandomizations of $\mathsf{pk}$, i.e., randomizations where the randomness is chosen by the challenger uniformly at random from $\chi$. We model this via an additional oracle $\mathsf{Rand}$ in the security game.

## 2.2. Our Contributions on BIP32 Wallets

Our first contribution is to formally analyze the security of hierarchical deterministic wallets. We took a two-step modular approach that helped us to keep the model clean and to capture every details of the BIP32 standard. To this end, we

```
main $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$
$\mathsf{RList} \leftarrow \{\epsilon\}$
$(sk, pk) \xleftarrow{\$} \mathsf{RSig.Gen}\,(par)$
$(m^*, \sigma^*, \rho^*) \xleftarrow{\$} \mathcal{A}^{\mathtt{Rand},\mathsf{RSign}}\,(pk)$
If $m^* \in Sigs$: $\mathsf{bad} \leftarrow \mathsf{true}$
If $\rho^* \notin \mathsf{RList}$: $\mathsf{bad} \leftarrow \mathsf{true}$
$pk^* \leftarrow \mathsf{RSig.RandPK}(pk, \rho^*)$
$b \leftarrow \mathsf{RSig.Verify}\,(pk^*, \sigma^*, m^*)$
Return $b \wedge \neg\mathsf{bad}$

Oracle $\mathsf{RSign}\,(m, \rho)$
If $\rho \notin \mathsf{RList}$: Return $\perp$
$sk' \leftarrow \mathsf{RSig.RandSK}(sk, \rho)$
$\sigma \xleftarrow{\$} \mathsf{RSig.Sign}\,(m, sk')$
$Sigs \leftarrow Sigs \cup \{m\}$
Return $\sigma$

Oracle $\mathtt{Rand}$
$\rho \xleftarrow{\$} \chi$
$\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$
Return $\rho$
```

Figure 2.2.: Security game $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ with adversary $\mathcal{A}$.

first focused on the simpler notion of deterministic wallets, without the hierarchical key derivation. Once we determined the core model and security properties, we extended our analysis to accommodate the hierarchical setting. Our comprehensive analysis lets us present concrete bit security of the BIP32 construction as used in practice. Although the BIP32 standard uses additive key derivation, we have considered both multiplicative and additive key derivation in our security analysis. Surprisingly, we found out that BIP32 with multiplicative key derivation has higher level of security than the one with additive key derivation. The results of our security analysis has been disseminated in the following two publications:

[45] P. Das, S. Faust, and J. Loss. "A Formal Treatment of Deterministic Wallets". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019.* 2019, pp. 651–668. DOI: 10.1145/3319535.3354236. URL: https://doi.org/10.1145/3319535.3354236.

[43] P. Das, A. Erwig, S. Faust, J. Loss, and S. Riahi. "The Exact Security of BIP32 Wallets". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communication Security, Virtual Event, Republic of Korea, November 15 - 19, 2021.* 2021, pp. 1020–1042. DOI: 10.1145/3460120.3484807. URL: https://doi.org/10.1145/3460120.3484807.

In the following Sections 2.2.1 to 2.2.4, we outline our main contributions. Section 2.2.1 summarizes the security model of deterministic wallets in the hot/cold wallet setting, outlining the security properties of wallet unlinkability and wallet unforgeability. The main results of our security analysis on ECDSA-based rerandomizable signatures – main building block of our wallet construction, is captured in Section 2.2.2. Next, we discuss how our model is extended to accommodate the hierarchical feature of deterministic wallets in Section 2.2.3. Finally, we provide concrete level of bit security achieved by the BIP32 wallet construction for additive as well as multiplicative key derivation in Section 2.2.4.

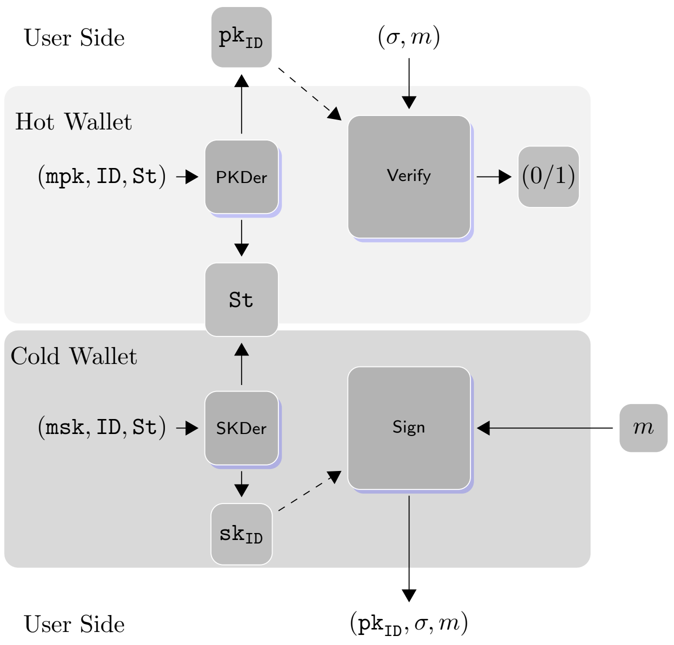
User Side
$\mathtt{pk}_{\mathtt{ID}}$
$(\sigma, m)$
Hot Wallet
$(\mathtt{mpk}, \mathtt{ID}, \mathtt{St})$
PKDer
Verify
(0/1)
St
Cold Wallet
$(\mathtt{msk}, \mathtt{ID}, \mathtt{St})$
SKDer
Sign
$m$
$\mathtt{sk}_{\mathtt{ID}}$
User Side
$(\mathtt{pk}_{\mathtt{ID}}, \sigma, m)$

Figure 2.3.: (1) The cold wallet signs a message $m$ with its session secret key $sk_{ID}$ as $\sigma \leftarrow \mathsf{SWal.Sign}(sk_{ID}, m)$. (2) Anyone can later verify the validity of a signature $\sigma$ on message $m$ as $(0/1) \leftarrow \mathsf{SWal.Verify}(\mathsf{pk}_{ID}, \sigma, m)$.

## 2.2.1. Security Model for Deterministic Wallets

Our security model for deterministic wallets, captures the security properties that a hot/cold wallet must satisfy. In particular, we incorporate into our model: hot wallet security breaches, access to derived public keys and corresponding signatures that may appear on the blockchain. Let us first define a wallet scheme SWal as the following tuple of algorithms: (SWal.MGen, SWal.SKDer, SWal.PKDer, SWal.Sign, SWal.Verify), where SWal.MGen denotes the master key generation algorithm. Next, the pair of algorithms (SWal.SKDer, SWal.PKDer) are used for deriving session secret/public keys respectively. Lastly, (SWal.Sign, SWal.Verify) represent the signing and verification algorithms of the underlying signature scheme. As already discussed previously, the master key generation algorithm $\mathsf{SWal.MGen}$ is run during setup, resulting in a pair of master keys and an initial state, referred to as the chaincode $ch$. The master secret key $msk$ is given to the cold wallet, the master public key $mpk$ to the hot wallet, while the chaincode $ch$ is shared between both wallets. The hot wallet can now run the public key derivation algorithm $\mathsf{SWal.PKDer}$ on inputs $mpk$, $ch$ and identifier $ID$ to generate session public key $\mathsf{pk}_{ID}$. While the cold wallet can generate the corresponding session secret key $\mathsf{sk}_{ID}$ by running the secret key derivation algorithm $\mathsf{SWal.SKDer}$ on inputs $mpk$, $ch$ and $ID$. When the user wants to sign a message $m$, the cold wallet runs the signing algorithm $\mathsf{SWal.Sign}$ on input $m$ and session secret key $\mathsf{sk}_{ID}$ to generate a signature $\sigma$. Later, $\sigma$ can be verified w.r.t $\mathsf{pk}_{ID}$ once both are publicly available on the blockchain.

The security of $\mathsf{SWal}$ is defined via two game-based security notions that we call *wallet unlinkability* and *wallet unforgeability*. Our notion of unlinkability can informally be described as a form of forward security – similar in spirit to key exchange models for analyzing TLS. It guarantees that all money that was sent to session public keys $\mathsf{pk}_{ID} \leftarrow \mathsf{SWal.PKDer}\,(mpk, ch, ID)$ derived *prior* to the hot wallet breach, can not be linked to $mpk$. Notably, our unlinkability property even holds against an adversary that sees a polynomial number of session public keys generated from $mpk$ and corresponding signatures for adversarially chosen messages. On the other hand, our unforgeability notion considers a natural threat model where funds on the cold wallet remain secure, even if the hot wallet is fully compromised. While at first sight it may seem that achieving unforgeability in such a setting is straightforward, it turns out that in particular for ECDSA-based wallets, we have to deal with several technical challenges. The main reason for this is that once the hot wallet is breached, the session public keys are not fresh anymore (i.e., all session public keys are now related to the master public key $mpk$). This hinders a straightforward reduction to the security of the underlying signature scheme used by the cryptocurrency. Even worse, we argue that for certain naive instantiations of wallet schemes, wallet unforgeability can be broken *without* ever breaking into the cold wallet. This is possible through a so-called *Related Key Attack (RKA).* In an RKA, after observing some signature $\sigma$ under public key $\mathsf{pk}$, the adversary can create a forgery $\sigma'$ under a public key $\mathsf{pk}'$, where $\mathsf{pk}$ and $\mathsf{pk}'$ are related via some randomness $\rho$. However, this RKA can be quite easily prevented within the construction of the signature scheme, by using public key prefixing, i.e. always signing a public key prefixed message $(\mathsf{pk}, m)$. Although, such an RKA can be mitigated without extra cost and does not pose a practical threat, as we will see in later sections, it turns out to be immensely useful in our reductions, while proving the unforgeability property.

**Forward Wallet Unlinkability and Stateful Deterministic Wallets.** In order to achieve our security definition of forward unlinkability, we consider the natural notion of *stateful deterministic wallets*. In contrast to the fixed chaincode as mentioned above, in a stateful wallet, the hot and cold wallet share a common secret state $St$ that is (deterministically) updated for every new session key pair. More concretely, the master key generation algorithm $\mathsf{SWal.MGen}$ outputs (together with the master key pair $(mpk, msk)$) an initial state $St_0$ that will be stored on both the hot and the cold wallet. Then, to derive new session keys, the secret/public key derivation algorithms $\mathsf{SWal.SKDer}$ and $\mathsf{SWal.PKDer}$ take as input additionally the current state $St_{i-1}$ and output the new state $St_i$, while the old state $St_{i-1}$ is erased from the hot/cold wallet. The update mechanism for deriving the new state has to guarantee that $St_i$ looks random even if future states $St_j$ (for $j > i$) are revealed. Together with a mechanism for deriving new session key pairs, our scheme achieves the strong aforementioned notion of forward unlinkability. We note that while state updates (together with secure erasures) are needed to achieve our new notion of forward unlinkability, our notion of unforgeability can also be achievable by some of the currently used (stateless) wallet schemes.

Next, we present a generic wallet construction from signature schemes with rerandomizable keys. As we will show afterwards, such a modular treatment helps us in proving both of the security properties of (forward) wallet unlinkability and wallet unforgeability in a clean manner.

**Generic Wallet Construction and proving Wallet Security.** Here we show how to generically instantiate our wallet scheme from a signature scheme with rerandomizable keys $\mathsf{RSig} := (\mathsf{RSig.Gen}, \mathsf{RSig.SKDer}, \mathsf{RSig.PKDer}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify})$ as follows (cf. Figure 2.4). Let $St$ be the current state of the hot/cold wallet. The public key derivation algorithm $\mathsf{SWal.PKDer}(mpk, St, ID)$ first computes $(\omega_{ID}, St') = \mathsf{H}(St, ID)$. Then, it derives the new session public key $\mathsf{pk}_{ID}$ by running the public key derivation algorithm $\mathsf{RSig.RandPK}$ via $\mathsf{pk}_{ID} \leftarrow \mathsf{RandPK}(mpk, \omega_{ID})$, and erases the old state $St$. Analogously, the cold wallet can compute $\mathsf{sk}_{ID}$ by computing $\omega_{ID}$ as above and calling $\mathsf{sk}_{ID} \leftarrow \mathsf{RSig.RandSK}(msk, \omega_{ID})$. For signing a message, the cold wallet runs the $\mathsf{RSig.Sign}$ algorithm on input secret key $\mathsf{sk}_{ID}$ and public key prefixed message $(\mathsf{pk}_{ID}, m)$ to obtain a signature $\sigma$. Verification of $\sigma$ can be carried out by running the respective verification algorithm $\mathsf{RSig.Verify}$ on input public key $\mathsf{pk}_{ID}$ and message $(\mathsf{pk}_{ID}, m)$.

Let us now briefly summarize our results on the security properties of (forward) wallet unlinkability and wallet unforgeability for a generic wallet construction as defined above. We define wallet unlinkability as a game between an adversary and a challenger, where the adversary gets access to a session public key oracle, a signing oracle and a state oracle. The adversary is allowed to query the public key oracle on any $ID$ to retrieve a session public key $\mathsf{pk}_{ID}$. Additionally, it can query the signing oracle on the retrieved session public key to receive a corresponding signature. Finally, the state oracle fetches the current wallet state $St_i$. The adversary wins the unlinkability game, if it is able to distinguish a randomly generated public key from a session public key (not seen by the adversary from a previous query), before calling the state oracle. We formally show that, if $\mathsf{H}$ is modeled as a random oracle that maps to the randomness space for rerandomizable keys, then the forward unlinkability property of the wallet construction is achieved from the property of rerandomizability of keys of the underlying rerandomizable signature scheme. In other words, the adversary is able to win the wallet unlinkability game only with a negligible probability, if the underlying signature scheme satisfies the rerandomizability property of keys. We formally define wallet unlinkability game in Section 3.1, Appendix A and prove the same in Theorem 4.2, Appendix A.

In the wallet unforgeability game, on the other hand, the adversary has corrupted the hot wallet. Hence, it gets access to the master public key $mpk$ and the current wallet state $St_i$. Additionally, it gets access to the session public key oracle and signing oracle as before. To win the unforgeability game, the adversary has to come up with a forgery on a fresh message, signed under any session key $\mathsf{pk}_{ID}$. We formally show that, if the underlying rerandomizable signature scheme satisfies the notion of UFCMA-HRK, then we achieve wallet unforgeability. We prove the same, by designing a reduction that tries to win the UFCMA-HRK game, by simulating the wallet unforgeability game to an adversary. In particular, the signing oracle can be simulated by querying the underlying $\mathsf{RSign}$ oracle in the UFCMA-HRK game (cf. Figure 2.2). While simulating the public key oracle, the randomness $\omega \leftarrow \mathsf{H}(mpk, St, ID)$ is derived by programming the hash function H as a random oracle. $\mathsf{H}$ internally calls the underlying $\mathsf{Rand}$ oracle in the UFCMA-HRK game, so that the public keys are always obtained from honestly derived randomness, as per UFCMA-HRK. Finally, when the adversary returns a valid forgery in the wallet unforgeability game, the same forgery can be forwarded to the UFCMA-HRK game. We formally define wallet unforgeability game and prove the same respectively in Section 3.2 and Theroem 4.2, Appendix A.

```
Algorithm SWal.MGen($par$)
00 $St \xleftarrow{\$} \{0,1\}^\kappa$
01 $(mpk, msk) \xleftarrow{\$} \mathsf{RSig.Gen}(par)$
02 Return $(St, msk, mpk)$

Algorithm SWal.Sign($m, \mathsf{sk}, \mathsf{pk}$)
03 $\hat{m} \leftarrow (\mathsf{pk}, m)$
04 $\sigma \xleftarrow{\$} \mathsf{RSig.Sign}(\mathsf{sk}, \hat{m})$
05 Return $\sigma$

Algorithm SWal.Verify($\mathsf{pk}, \sigma, m$)
06 $\hat{m} \leftarrow (\mathsf{pk}, m)$
07 Return $\mathsf{RSig.Verify}(\mathsf{pk}, \sigma, \hat{m})$

Algorithm SWal.SKDer($msk, ID, St$)
00 $(\omega_{ID}, St) \leftarrow \mathsf{H}(St, ID)$
01 $\mathsf{sk}_{ID} \xleftarrow{\$} \mathsf{RSig.RandSK}(msk, \omega_{ID})$
02 Return $(\mathsf{sk}_{ID}, St)$

Algorithm SWal.PKDer($mpk, ID, St$)
03 $(\omega_{ID}, St) \leftarrow \mathsf{H}(St, ID)$
04 $\mathsf{pk}_{ID} \leftarrow \mathsf{RSig.RandPK}(mpk, \omega_{ID})$
05 Return $(\mathsf{pk}_{ID}, St)$
```

Figure 2.4.: Construction of generic wallet scheme SWal from a signature scheme with rerandomizable keys RSig and a hash function H.

## 2.2.2. Rerandomizable Signature Schemes

Next, to instantiate our generic wallet scheme, we are left with the task of building signatures with rerandomizable keys from standard (practical) signature schemes ideally used by cryptocurrencies. As shown in [68] the Schnorr signature scheme [131] satisfies these properties. In addition, we show that ECDSA (cf. Figure 2.5) as well as BLS signatures [23] can be used to construct signatures with rerandomizable keys. Thus, these schemes are natural candidates for our wallet construction. Moreover, our main technical results are on the ECDSA-based scheme which we will describe next.

**Rerandomizable Signature Schemes from ECDSA.** While many cryptocurrencies (Algorand, Internet Computer [5, 84]) are currently considering BLS signatures, till date, legacy cryptocurrencies such as Bitcoin and Ethereum [29, 120] still rely on the ECDSA signature scheme [91].[1] Consequently, it is important to have a provably secure wallet construction from ECDSA. To this end, we need to build a rerandomizable signature scheme from ECDSA and plug it into our generic wallet construction. Surprisingly, due to the rather contrived nature of ECDSA, proving unforgeability for rerandomizable ECDSA is much more involved than

[1]Recently Bitcoin rolled an update [125, 126] to support the Schnorr signature scheme as well, but users can still use ECDSA signatures as before.

```
Algorithm
EC.Gen $(par)$
00 $x \xleftarrow{\$} \mathbb{Z}_p$
01 $X \leftarrow x \cdot G$
02 $sk \leftarrow x$
03 $pk \leftarrow X$
04 Return $(pk, sk)$
```

```
Algorithm
EC.Sign $(\mathsf{sk} = x, m)$
05 $z \leftarrow \mathsf{H}(m)$
06 $t \xleftarrow{\$} \mathbb{Z}_p$
07 $(e_x, e_y) \leftarrow t \cdot G$
08 $r \leftarrow e_x \mod p$
09 If $r = 0 \mod p$
10     Goto Step 2
11 $s \leftarrow t^{-1}(z + rx) \mod p$
12 If $s = 0 \mod p$
13     Goto Step 2
14 Return $\sigma := (r, s)$
```

```
Algorithm
EC.Verify $(\mathsf{pk} = X, \sigma, m)$
15 Parse $(r, s) \leftarrow \sigma$
16 If $(r, s) \notin \mathbb{Z}_p$
17     Return 0
18 $w \leftarrow s^{-1} \mod p$
19 $z \leftarrow \mathsf{H}(m)$
20 $u_1 \leftarrow zw \mod p$
21 $u_2 \leftarrow rw \mod p$
22 $(e_x, e_y) \leftarrow u_1 \cdot G + u_2 \cdot X$
23 If $(e_x, e_y) = (0, 0)$
24     Return 0
25 Return $r = e_x \mod p$
```

Figure 2.5.: $\mathsf{EC} = (\mathsf{EC.Gen}, \mathsf{EC.Sign}, \mathsf{EC.Verify})$: ECDSA Signature scheme relative to elliptic curve $\mathbb{E}$ and hash function $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p$.

that of Schnorr or BLS. Besides, both Schnorr and BLS satisfies the notion of UFCMA-RK security of rerandomizable signature schemes. While for ECDSA, we are only able to achieve the weaker notion of UFCMA-HRK security.

In a rerandomizable ECDSA scheme, keys can be rerandomized either additively as $\mathsf{sk}' \leftarrow \mathsf{sk} + \rho$; $\mathsf{pk}' \leftarrow \mathsf{pk} + \rho \cdot G$ or multiplicatively as $\mathsf{sk}' \leftarrow \mathsf{sk} \cdot \rho$; $\mathsf{pk}' \leftarrow \mathsf{pk} \cdot \rho$. We analyze both cases and detail our key findings below.

**ECDSA with Multiplicative Rerandomization.** Let us first consider the multiplicatively rerandomized ECDSA scheme $\mathsf{REC} = (\mathsf{REC.Gen}, \mathsf{REC.SKDer}, \mathsf{REC.PKDer}, \mathsf{REC.Sign}, \mathsf{REC.Verify})$, where REC.Gen generates an ECDSA key pair $(\mathsf{sk} = x, \mathsf{pk} = x \cdot G)$. REC.SKDer generates a rerandomized secret key as $\mathsf{sk}' = \mathsf{sk} \cdot \rho$, while REC.PKDer outputs a rerandomized public key as $\mathsf{pk}' = \mathsf{pk} \cdot \rho$. REC.Sign and REC.Verify works similar to the standard signing and verification algorithms of ECDSA (cf. Figure 2.5) with the following difference. In case of the REC scheme, messages are always public key prefixed before sign/verify. We now turn to the unforgeability proof of REC, highlighting, how we overcome the main hurdles in formulating our security proof. To prove unforgeability of REC, we want to show that UFCMA-HRK security of REC reduces to the standard notion of UFCMA security of the underlying ECDSA scheme EC. The proof consists of a reduction that tries to come up with a valid forgery in the UFCMA game (corresponding to EC), by simulating the UFCMA-HRK game (corresponding to REC) to an adver-

```
$\mathsf{Trf}[\mathsf{H},\mathsf{G}]_{\mathsf{EC}}(m_0, m_1, \sigma_1, \omega, X_0, X_1)$
00 $z_0 \leftarrow \mathsf{H}(m_0)$
01 $z_1 \leftarrow \mathsf{G}(m_1)$
02 If $(\mathsf{EC.Verify}(\sigma_1, m_1, X_1) = 0) \vee \left(\omega \neq \frac{z_1}{z_0} \vee X_1 \neq X_0 \cdot \omega\right)$ :
03   Return $\perp$
04 $(r, s_1) \leftarrow \sigma_1$
05 $s_0 \leftarrow \frac{s_1}{\omega} \mod p$
06 $\sigma_0 \leftarrow (r, s_0)$
07 Return $\sigma_0$
```

Figure 2.6.: Figure shows the $\mathsf{Trf}_{\mathsf{ECDSA}}$ algorithm for hash functions $\mathsf{H}, \mathsf{G}\colon \{0,1\}^* \to \mathbb{Z}_p$. It takes a signature $\sigma_1$ under public key $X_1$ on message $m_1$ and outputs a signature $\sigma_0$ under public key $X_0$ on message $m_0$, where $\omega = \frac{\mathsf{G}(m_1)}{\mathsf{H}(m_0)}$ and $X_1 = X_0 \cdot \omega$.

sary $\mathcal{A}$. The main difficulty in this reduction is that $\mathcal{A}$ can query for signatures under related (i.e., rerandomized) keys, where the relation between the keys may be known to $\mathcal{A}$. The signing oracle from the UFCMA game does not help in simulating signatures in a straight-forward manner. This is because, the signing oracle returns signatures under a particular challenge public key, while the reduction needs to answer signature queries under many related keys. We solve this problem by applying technique from an existing related key attack (RKA), present in the multiplicatively rerandomized ECDSA. Such an RKA lets us construct an efficient transfer algorithm (cf. Figure 2.6) that transforms a signature $\sigma' = (r, s)$ on message $m'$ relative to a key $\mathsf{pk}'$ into a signature $\sigma = (r, \frac{s}{\rho})$ on message $m$ that is valid under the related key $\mathsf{pk}$, such that $\mathsf{pk}' = \mathsf{pk} \cdot \rho$, where $\rho$ satisfies $\rho = \frac{\mathsf{G}(m')}{\mathsf{H}(m)}$ ($\mathsf{H}$ and $\mathsf{G}$ are the respective hash functions in $\mathsf{EC}$ and $\mathsf{REC}$). Given such an efficient transformation, we proceed with the reduction as follows. First of all, the challenge public key from the UFCMA game is embedded as the challenge key in the UFCMA-HRK game. Now, signature queries on related key $\mathsf{pk}'$ from $\mathcal{A}$ can be answered, by first querying the underlying sign oracle from UFCMA to retrieve a signature under $\mathsf{pk}$. Then, we run the transform algorithm to derive a valid signature under $\mathsf{pk}'$. Note here that, for $\rho$ to satisfy the condition $\rho = \frac{\mathsf{G}(m)}{\mathsf{H}(m')}$, we need to program the random oracle $\mathsf{H}$ in the UFCMA-HRK game as $\mathsf{H}(m') = \rho \cdot \mathsf{G}(m)$, where $\mathsf{G}(m)$ is obtained by querying the random oracle in the UFCMA game. Finally, when adversary returns a forgery in the UFCMA-HRK game, we need to derive a signature under the challenge key $\mathsf{pk}$ by running the transformation algorithm.

We formally prove UFCMA-HRK security of two variants of multiplicatively rerandomized ECDSA. The first one is a salted and public key prefixed construction (cf. Figure 10, App. A), where messages are always prepended with the public key and a salt. It is proven in Appendix A, Section 5. The second one is similar to the $\mathsf{REC}$ construction we explained above, basically a salt-free construction with only public key prefixing. The advantage of the salt-free variant is that it is directly compatible with Bitcoin, without any modification in the signature verification procedure. The proof of the later variant can be found in full version [44], Appendix C.

**ECDSA with Additive Rerandomization.** The additively rerandomized ECDSA scheme, denoted as $\mathsf{REC}' = (\mathsf{REC}'.\mathsf{Gen}, \mathsf{REC}'.\mathsf{SKDer}, \mathsf{REC}'.\mathsf{PKDer}, \mathsf{REC}'.\mathsf{Sign}, \mathsf{REC}'.\mathsf{Verify})$ differs from the multiplicative variant in the key derivation algorithms. Precisely, in $\mathsf{REC}'$ $\mathsf{REC}'.\mathsf{PKDer}$ derives a rerandomized public key as $\mathsf{pk}' \leftarrow \mathsf{pk}+\rho\cdot G$, while $\mathsf{REC}'.\mathsf{SKDer}$ derives the corresponding secret key as $\mathsf{sk}' \leftarrow \mathsf{sk} + \rho$. In contrast to the multiplicative variant, for additively rerandomized ECDSA, we are able to show UFCMA-HRK security, as long as *each message is signed only once per key*. For our unforgeability proof, we use a technique similar in spirit to the multiplicative case. However, note that the previous RKA is tailored to the multiplicative analysis and hence does not help us directly. To solve this issue, we design an RKA, specific for the additive case that works as follows: given a signature $(r, s)$ on $m$ relative to $\mathsf{pk}$, $(r, s)$ is also a valid signature relative to the public key $\mathsf{pk}' = \mathsf{pk} + \rho \cdot G$ on message $m'$, given that $\rho = \frac{(\mathsf{G}(m)-\mathsf{H}(m'))}{r}$ ($\mathsf{H}$ and $\mathsf{G}$ are the respective hash functions in $\mathsf{EC}$ and $\mathsf{REC}'$). To the best of our knowledge, we are the first to observe this RKA technique present in additive ECDSA. Having a closer look at the structure of $\rho = \frac{(\mathsf{G}(m)-\mathsf{H}(m'))}{r}$, notice that, it is dependent on $r$, which is part of the signature $(r, s)$. This is in contrast to the multiplicative case and poses additional problems in our reduction. First of all, the random oracle in the UFCMA-HRK game now needs to be programmed as $\mathsf{H}(m') = \mathsf{G}(m) - \rho \cdot r$. This in turn, fixes the signature $(r, s)$ for a message $m'$ and key $\mathsf{pk}'$. As a result, every message can be signed only once per key. This allows us to reduce to the notion of *One-Per Message Existential Unforgeability Under Chosen Message Attack (UFCMA1).* Secondly, since the signature under the related keys $\mathsf{pk}$ and $\mathsf{pk}'$ are surprisingly the same, when we already called the sign oracle from UFCMA game to fetch a signature $\sigma$ to simulate signatures in our reduction, the reduction cannot return the same signature to the UFCMA game. To solve this, we guess the position of adversary's forgery in the UFCMA-HRK game, among all queried public keys and embed the challenge key from UFCMA game at this position. To win the UFCMA-HRK game, adversary will return a valid forgery at this position. Hence, it must not have queried the sign oracle at this position. This in turn means that the reduction never needs to call the UFCMA sign oracle for this position. Hence, the forgery from UFCMA-HRK can also be returned as a valid forgery in the UFCMA game. On the downside, the reduction incurs a tightness loss in terms of total number of public keys due to the guess of the forgery's position. To conclude, the additive variant achieves only UFCMA1 with a tightness loss of total number of public keys, in contrast to the multiplicative variant which achieves standard UFCMA without any tightness loss. The security proof for the additively rerandomized ECDSA can be found in Appendix A, Section 3.

### 2.2.3. Security Model of Hierarchical Wallets

To complete the analysis of BIP32, we extend our flat model of deterministic wallets to accommodate the hierarchical key derivation feature. A hierarchical deterministic wallet can be thought of as a tree, where each node can act simultaneously as signing keys as well as master keys to derive new child keys. Such a hierarchical feature can be useful in a company setting, for instance, where higher officials would like to delegate signing keys to their subordinate employees. Unfortunately, it cannot be guaranteed that all official entities in a company have secure key storage. As we have seen before, if the secret key is not securely stored in a cold wallet, it can get leaked to an adversary. It is important to ensure in the hierarchical setting that, when a secret key of one of the wallet gets leaked then it must not reveal any secret information from other wallets. However, as observed in [80], the current key derivation mechanism does not guarantee this property. If the adversary learns secret key of a wallet as $\mathsf{sk}_{ID} = msk + \rho$, where $\rho = \mathsf{H}(mpk, ch, ID)$. Since adversary knows $mpk$ and $ch$, it can derive $msk = \mathsf{sk}_{ID} - \rho$ which is the secret key of the parent wallet. This process can be repeated, revealing secret keys of all nodes along the derivation path from the target node till the root node. To mitigate this issue, BIP32 considers a second type of key derivation mechanism, termed as *hardened key derivation*. In a hardened key derivation, $\rho$ is derived as $\rho = \mathsf{H}(msk, ch, ID)$. Then public/secret keys respectively are derived as: $\mathsf{pk}_{ID} = mpk + \rho \cdot G$, $\mathsf{sk}_{ID} = msk + \rho$. Let us now consider a wallet with hardened keys. If an adversary corrupts such a wallet, getting hold of $\mathsf{sk}_{ID}$, it still cannot derive $msk$ from $\mathsf{sk}_{ID}$, since $\rho$ is dependent on $msk$ itself. It is clearly evident from the above example that, hardened key derivation offers stronger security than the usual non-hardened key derivation we saw before. However, it comes with a cost. Since now $msk$ is involved in the derivation of both secret/public child keys, hardened key derivation cannot be achieved in the hot/cold wallet setting without interaction. As the main goal of storing keys in the hot/cold wallet is to derive child keys without any interaction between the two wallets, hardened key derivation is not suitable in this setting. However, both hardened and non-hardened key derivations can be meaningfully used in the hierarchical setting, based on the trust level of a node. Let us explain this in a bit more details. For nodes in the hierarchy which belongs to higher level employees of a company, can be trusted with secure key storage. So we assume that their key pair is derived via non-hardened key derivation and they store their keys in hot/cold wallet fashion. Note that, as long as a (non-hardened) secret key of a node is securely stored in a cold wallet, the above secret key leakage attack is prevented. Nodes that are related to lower level employees of a company, hence less trusted, shall be implemented in a hardened fashion. We further make no assumption on how a hardened key pair is stored. As a result, we consider in our model that secret key leakage from hardened wallets is possible.

Similar to the flat setting of deterministic wallets, we are concerned with the security properties of wallet unlinkability and wallet unforgeability in the hierarchical setting. Let us briefly summarize these properties here. The unlinkability property guarantees that a public key of any uncorrupted node in the hierarchy is indistinguishable from a randomly generated public key. When a hot wallet of a non-hardened node is corrupted, the adversary gets hold of the public key of the corrupted wallet. This further leads to revealing public keys of all non-hardened nodes within the sub-tree. Unlinkability for the rest of the tree will still hold, since the public key of any of its node was not derived from the corrupted node. On the other hand, the wallet unforgeability property gurantees the following. Even when the hot wallets of all nodes are corrupted as well as secret keys of some of the hardened nodes are leaked, the adversary should not be able to forge signatures for any of the uncorrupted nodes (all non-hardened nodes and honest hardened nodes) in the tree.

In our security proof for wallet unforgeability, we incur a multiplicative tightness loss in terms of total number of secret keys leaked. The security proof of our generic hierarchical wallet construction can be found in Appendix A, Section 5. Interestingly, we are also able to show that when we reduce wallet unforgeability to UFCMA-RK for the generic construction of hierarchical wallet, then it *must* lose this factor. To prove this lower bound, we adapt the reduction/metareduction techniques introduced by Coron in his seminal work [39]. [39] considered tightness of unique signatures (which is very different from our setting), this requires to adapt his technique to our model. Our impossibility proof of a tighter bound is presented in the full version [44], Appendix B.

### 2.2.4. Concrete Security Parameters

Finally, we compare the bit security level of BIP32 with additively rerandomized ECDSA (as in the specification [143]) versus BIP32 with multiplicatively rerandomized ECDSA. We find that original BIP32 gives roughly 94 bits of security according to our theorems and conservative choices of parameters. While, the multiplicative variant gives 114 bits of security with a similarly efficient scheme. Given these insights, we strongly recommend that the Bitcoin community switches rerandomizations in BIP32 from additive to multiplicative, in particular since these changes essentially come for free.

## 2.3. Our Contributions on Post-Quantum Deterministic Wallets

Our second contribution is to design deterministic wallets that are secure in the post-quantum setting. In the post-quantum security model, the adversary is considered to have quantum computational power, whereas the wallet is implemented in a classical computer. We show how to build post-quantum secure wallets from a certain class of post-quantum secure signature schemes. We relax the notion of signature schemes with rerandomizable keys to *rerandomizable public keys*, and show that the latter is sufficient to design post-quantum secure wallets. Next, we show generic instantiations of such signature scheme from Lattice-based Fiat-Shamir signatures. For a concrete instantiation, we consider the NIST second round finalist qTESLA, and use its rerandomizable variant to instantiate our wallet. Finally, we show transaction throughput of a blockchain, once integrated with rerandomizable qTESLA signatures. Our work has resulted in the following publication.

[6] N. A. Alkadri, P. Das, A. Erwig, S. Faust, J. Krämer, S. Riahi, and P. Struck. "Deterministic Wallets in a Quantum World". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020.* 2020, pp. 1017–1031. DOI: `10.1145/3372297.3423361`. URL: `https://doi.org/10.1145/3372297.3423361`.

Below we present our main findings of our work on post-quantum wallets. In Section 2.3.1, we present the security model for deterministic wallets needs in the post-quantum setting, followed by our analysis of wallet unlinkability and unforgeability properties. In Section 2.3.2, we summarize our construction of signature schemes with rerandomzable public keys, which is the main ingrediant of our wallet construction.

## 2.3.1. Security Model of Post-Quantum Wallets

For our post-quantum security analysis, we adapt the security model of deterministic wallets in the classical setting and carefully adjust it to the post-quantum setting. We use similar modularization for our generic wallet construction from signature schemes with rerandomizable public keys. We consider the same setting of hot/cold wallet separation of key storage, where master public key resides in the hot wallet, while the cold wallet contains the corresponding master secret key. Together with the master public key and a deterministic public key derivation algorithm, the hot wallet can derive session public key $\mathsf{pk}_{ID}$ for a specific identifier $ID$, independent from the cold wallet. The cold wallet can derive a corresponding session secret key, ideally without any interaction with the hot wallet, by running the deterministic secret key derivation algorithm with master secret key as input. Similar as before, we want to guarantee the properties of wallet unlinkability and wallet unforgeability. Our security proofs for both of these security properties now needs to consider a quantum adversary, with an access to a quantum random oracle. We briefly detail below the main hurdles in proving the security properties in the post-quantum setting.

We want to achieve wallet unlinkability, which roughly states the following. A session public key, derived from the master public key looks indistinguishable from a randomly generated public key to an adversary, who has observed a polynomial number of session public keys and signatures, generated from the wallet, as long as the hot wallet is uncorrupted. Let us first recall our security proof in the classical setting. There we proceed as follows. First, the adversary tries to guess any of the previous states of the wallet. A correct guess leads to breaking the unlinkability property. Next, we show that the this event is possible with negligible probability. The classical reduction keeps a list of all previous states to check whether adversary's guess hits one of the states in the list. In the post-quantum setting, we cannot maintain a list, since the adversary can query the random oracle on several inputs in superposition. To solve this issue, we use an additional game hop, where the quantum adversary has to distinguish between two random oracles. We bound the advantage of the adversary in distinguishing two such random oracles, and this further can be bound by the One-way to Hiding (O2H) Lemma [9].

In case of wallet unforgeability, we want to ensure that, even when the hot wallet is corrupted, after observing polynomial number of signatures on related session public keys, the adversary should not be able to produce a valid forgery on a fresh message. Here we highlight the main difference from the classical scenario. As before, our aim is to reduce wallet unforgeability to UFCMA-HRK security of the underlying rerandomizabe signature scheme. The reduction needs to program the random oracle to obtain the randomness to derive session keys. For the reduction to go through, the random oracle internally calls the $\mathsf{Rand}$ oracle from the underlying UFCMA-HRK game. In the post-quantum setting, the adversary however has access to a quantum random oracle. In particular, the adversary can query the quantum random oracle on equal superposition of all (exponentially many) inputs which would require exponential many queries to $\mathsf{Rand}$, since $\mathsf{Rand}$ is run by the classical challenger. This naive approach renders the reduction inefficient. As an alternative, we replace the random oracle with an oracle from small range distribution [146]. As a result, now the reduction would work with only polynomial number of queries to the $\mathsf{Rand}$ oracle.

### 2.3.2. Post-Quantum Secure Rerandomizable Signature Schemes

Next, we need to design post-quantum signature schemes to instantiate our wallet construction. Taking inspiration from the Schnorr-based rerandomizable signatures in the classical random oracle model, we consider lattice-based Schnorr like signature schemes as a natural candidate for designing our post-quantum variant of rerandomizable signatures in the quantum random oracle model (QROM). The key pair $(\mathsf{sk}, \mathsf{pk})$ of such Schnorr-based lattice schemes consists of an instance of a hard lattice problem, where the secret key $\mathsf{sk}$ typically follows either the discrete Gaussian distribution or the uniform distribution over a small set. Our straight forward technique of constructing a rerandomizable signature scheme however does not work here. In more details, recall that in classical signature schemes such as Schnorr, BLS, ECDSA, given $(\mathsf{sk}, \mathsf{pk})$ and randomness $\rho$, $\mathsf{sk}$ is rerandomized additively by computing $\mathsf{sk}' = \mathsf{sk} + \rho$. In the lattice setting however, we must ensure that the output $\mathsf{sk}'$ follows the correct distribution, e.g., the Gaussian or uniform distribution. If this is not the case, then the randomness $\rho$ needs to be re-sampled until $\mathsf{sk}'$ follows the correct distribution. Although, this technique renders signature schemes with rerandomizable keys, it becomes unsuitable in the hot/cold wallet setting of wallets, since it requires to syncronize the correctly chosen $\rho$ between both wallets. To mitigate this issue, we follow a different approach for designing rerandomizable signatures.

The main observation that we exploit is that the sum of two Gaussians is also Gaussian distributed (cf. Lemma 3, Appendix C). Based on this observation, our approach works as follows. Let $\mathsf{sk}$ be Gaussian distributed. Given randomness $\rho$, $\mathsf{sk}$ is rerandomized additively by adding to $\mathsf{sk}$ a freshly Gaussian distributed secret key $\mathsf{sk}^*$. The key $\mathsf{sk}^*$ is deterministically sampled using the randomness $\rho$, i.e., we use $\rho$ as the randomness required in the Gaussian sampler algorithm. We obtain a rerandomized secret key that is Gaussian distributed, but with a slightly larger standard deviation than the one of the original secret key. Consequently, we can construct a signature scheme with rerandomizable keys, in which the distribution of rerandomized public keys is computationally indistinguishable from the distribution of the original public key. However, rerandomized secret keys follow a different distribution than a freshly sampled secret key. We formally define such relaxed notion (cf. Section 2, App C) and call it a *signature scheme with rerandomizable public keys*. We then show in Section 3, App C that this notion is sufficient for post-quantum secure wallets. This is true because as long as the cold wallet remains secure, the secret key is never revealed to the adversary. We present a lattice-based construction of such a scheme in Section 4, App C with a security proof in the QROM. Finally, we show in Section 5, App C that our construction can be instantiated with state-of-the-art lattice-based signature schemes such as *qTESLA* [8]. Hence, it can use their proposed parameters and enjoy their performance and efficiency.

## 2.4. Related Work

**Signature schemes.** One of the techniques that we use in our works is that certain signature schemes support the following efficient transformation: given a signature under some public key pk, one can produce a signature with respect to a related key pk′. While for certain signature schemes such as Schnorr [131] this is a well-known trick that has been used in various works [67, 98, 147], we are not aware of any prior use of such an algorithm for the ECDSA signature scheme. In addition, as discussed above we make use of the abstraction of signature schemes with rerandomizable keys that was originally introduced by Fleischhacker et al. [68] in the context of sanitizable signatures. As for ECDSA, Fersch et. al. [66] proved for the first time, UFCMA security of ECDSA in an idealized model. While, [65] showed in the random oracle model that ECDSA satisfies the notion of one-per-message unforgeability. In our BIP32 analysis, we use additively rerandomized ECDSA as one of the core building block which we also show to be one-per-message unforgeable in the random oracle model. In a recent work by Groth et al. [79] additively rerandomized ECDSA is shown to satisfy standard UFCMA, however the proof is in the Generic Group Model.

There is a large body of work on threshold signature schemes: [32, 33, 53, 75, 76, 109, 110] on threshold ECDSA, [74, 101] on threshold Schnorr that can be applied to strengthen the security of wallets. In a threshold signature scheme,

$(t+1)$ out of $n$ parties jointly sign a signature. Even when the adversary corrupts at most $t$ parties and learn their secret key material, it will not be able to forge a signature. The secret key stored in a cold wallet can be split into multiple devices, followed by signing through a threshold signature scheme. This allows to satisfy unforgeability in presence of a stronger adversary, that can corrupt upto $t$ devices. A simpler notion is that of multisignatures [22, 121], where a group of $n$ parties jointly sign a signature.

**Research on Wallet Systems.** Hot/cold wallets are widely used in cryptocurrencies and various implementations on standard computing and dedicated hardware devices are available. Most related to our work is the result of Gutoski and Stebila [81] who discuss a flaw in BIP32 and propose a (provably secure) countermeasure against it. Concretely, they study the well known attack against deterministic wallets [28] that allows to recover the master secret key once a single session key has leaked from the cold wallet. They then propose a fix for this flaw which allows up to $d$ session keys to leak, and show by a counting argument that under a one-more discrete-log assumption the master secret key can not be recovered. We emphasize that their model is rather restricted and does not consider an adversary learning public keys or signatures for keys which have not been compromised. More importantly, [81] prove only a very weak security guarantee. Namely, instead of aiming at the standard security notion of unforgeability where the adversary's goal is to forge a signature (as considered in our work), [81] consider the much weaker guarantee where the adversary's goal is to extract the entire master secret key. Hence, the security analysis in [81] does not consider adversaries that forge a signature with respect to some session public key, while in practice this clearly violates security. Besides [81], various other works explore the security of hot/cold wallets. Similar to [81], Fan et al. [62] study the security against secret session key leakage (they call it "privilege escalation attacks"). Unfortunately, their proposed countermeasure is ad-hoc and no formal model nor security proof is provided. A work by Luzio et al. [112] designs a novel hierarchical wallet scheme from (deterministic) hierarchical key assignment schemes [12]. In fact, they prove that their hierarchical wallet construction is resistant against the above-mentioned "privilage escalation attacks", where secret key leakage from colluding child nodes do not propagate to the parent node. However, their solution is not compatible with cryptocurrencies such as Bitcoin since it requires a more sophisticated (signature) verification algorithm, where a certificate associated with the user needs to be verified along with the signature.

Another direction is taken by Turuani et al. [138] who provide an automated verification of the Bitcoin Electrum wallet in the Dolev Yao model. Since the

Dolev-Yao model assumes that ciphertexts, signatures etc. are all perfect, their analysis exclude potential vulnerabilities such as related key attacks, which turn out to be very relevant in the hot/cold wallet setting. Another line of recent work focuses on the security analysis of hardware wallets [11, 114]. Both works target different goals. The work of Marcedone et al. [114] aims at integrating two-factor authentication into wallet schemes, while Arapinis et al. [11] consider hardware attacks against hardware wallets and provide a formal modeling of such attacks in the UC framework. Similar to the latter, Curtoius et al. [40] investigate how implementation flaws such as bad and correlated randomness may affect security. Other works that study the implications of weak randomness in wallets are [24, 25]. Chaum et al. [35] proposes a solution to secret key leakage through the use of a back-up key. When a secret key is leaked, then the back-up key can be utilized to produce a proof of ownership of the leaked key. This in turn, can benefit in recovery of lost funds. They show constructions compatible with ECDSA, adding a W-OTS+ [83] signing key as a back-up key.

Karakostas et al. [92] for the first time provide formal definitions of a proof of stake wallet, built from the modelling of [11]. They focus on security properties such as “address malleability”, relevant in the proof of stake setting.

Kondi et al. [102] designs a threshold wallet scheme in an advanced security model of proactive security. In the proactive setting, originally by [82], time is divided into epochs. An adversary is able to corrupt more than $t$ devices across several epochs, rendering a threshold scheme insecure. It is thus recommended to refresh the secret key shares at the start of every epoch, such that a combination of $t + 1$ corruptions across several epochs, does not lead to secret key recovery. In the wallet setting, [102] formalizes the notion of refreshing of offline devices, such that any $t + 1$ out of $n$ parties run a refresh protocol, while other $n - t - 1$ parties can update their shares once they come online. They show how to upgrade a 2 out of $n$ threshold signature scheme to proactive security with offline refresh for ECDSA, EdDSA and Schnorr. However, they also provide an impossibility result for the $t > 2$ out of $n$ threshold case, with a dishonest majority of online parties.

**Post-quantum Security.** Many prior works have investigated lattice-based Fiat-Shamir signatures, e.g., [8, 15, 58, 59, 113], and in particular, their security was analyzed in the QROM, e.g., by [55, 97, 111, 139]. Various works build blockchains with security features against quantum adversaries. Most recently, Esgin et al. [60] have proposed a new ring signature scheme based on lattice assumptions for the blochchain setting which focus on similar anonymity guarantees to Monero [122]. Blockchain initiatives such as the “Bitcoin Post-Quantum” [19] and QRL [128] replace ECDSA with hash-based signature schemes which are post-quantum secure.

Despite the hash-based schemes being quite efficient, the underlying hash function does not permit to construct a signature scheme with rerandomizable keys which plays a key role in our wallet scheme.

## 2.5. Discussion and Future Work

In this chapter, we focused on the formal treatment of deterministic wallets, which serves as a core infrastructure in the decentralized setting of blockchains. In this regard, our contribution is two-fold. As our first contribution, we fill the gap of a comprehensive security analysis for the BIP32 standard of hierarchical deterministic wallets [143]. To this end, we take a modular approach and build wallets from signature schemes with rerandomizable keys. Our model captures the security properties of wallet unlinkability and wallet unforgeability in the setting of hot wallet/cold wallet. Our provable secure wallet constructions are built from the rerandomizable schemes of ECDSA, Schnorr and BLS. In particular, we provide the first provable secure construction of wallets from rerandomizable ECDSA. One of the key observations of our security analysis is the following: *multiplicatively rerandomized ECDSA achieves higher level of bit security than the additive variant*, considering the underlying ECDSA achieves 128 bit security level for standard unforgeability. Our results have direct implications for BIP32 security. Precisely, the original BIP32 specification considers additively rerandomized ECDSA; hence achieves lower level of security than the multiplicative variant. As per our observation, a slight modification to the BIP32 specification (i.e., considering multiplicatively rerandomized ECDSA) will render better security, but achieving the sa,e efficiency. Our security analysis left several interesting open questions: Firstly, we were only able to prove (in the random oracle model) *one-per-message unforgeability* for the additively rerandomized ECDSA, where every message can be signed at most once. Such notion is sufficient in the blockchain setting where, every transaction (message) is signed only once. However theoretically, it would be nice to lift the one-per-message restriction and achieve standard UFCMA security. Groth et al. [79] made progress in this direction, showing standard UFCMA security of additively rerandomized ECDSA, however the analysis is in the Generic Group model. Secondly, the one-per-message unforgeability of ECDSA was proven by [65] in the random oracle model, with a large security loss (in terms of the number of signing oracle queries). Naturally, there is scope to get a tighter security proof.

As a second contribution, we extend our security analysis of deterministic wallets in the post-quantum setting. In this setting, the adversary has quantum computational power, while the scheme is implemented in a classical computer. A quantum attacker can render a deterministic wallet scheme built from classical signatures insecure, by simply extracting the master secret key from the master public key [132]. As a result, we need to fill the gap of post-quantum secure deterministic wallets. To this end, we extend our security model to the post-quantum setting. Next, we built deterministic wallets from (post-quantum secure) lattice-based Fiat-Shamir signature schemes with rerandomizable public keys. In particular, we show instantiation of such a rerandomizable signature scheme from qTESLA [8], one of the second round NIST finalists. Finally, we evaluate blockchain transaction throughput, considering rerandomizable qTESLA signatures.

We have several immediate open questions that we discuss below. First, our instantiation from qTESLA is an example of rerandomizable lattice-based scheme based on Guassian distribution. When we tried to instantiate it with a rerandomizable scheme based on uniform distribution (eg. NIST PQ standard Dilithium [59]), the key size blows up, rendering the scheme impractical. Naturally, we need an efficient scheme based on uniform distribution. Next, we note that the blockchain transaction throughput based on qTESLA is roughly 72 times lower than that based on a classical signature scheme (ECDSA). Hence, there is a lot of room to improve the efficiency of signature/key size in rerandomizable post-quantum signature schemes. Lastly, we have only considered post-quantum wallets in the flat setting with one level of key derivation. The next step would be to analyze the full hierarchical deterministic wallet in the post-quantum setting. In particular, it is interesting to see what challenges are posed in the simulation of the (hierarchical) wallet unforgeability game, in the presence of a quantum random oracle.

In our security model of hierarchical deterministic wallets, we support secret key leakage of only hardened nodes. Our model considers non-hardened nodes to be implemented in a hot wallet/cold wallet fashion, where the secret key is stored offline in the cold storage. Henceforth, we assume in our model that the secret keys from a cold wallet are not leaked to the adversary. Since the cold wallet comes online time to time for signing payments, an attacker could still target a cold wallet during the online phase. If a secret key is leaked from the cold wallet of a non-hardened node, it can have devastating effects. Precisely, secret key leakage of a child node leads to revealing secret keys of the parent nodes, in the hierarchical derivation path, all the way up to the root secret key. One way to strengthen security of wallets in presence of secret key leakage of non-hardened nodes is to thresholdize the cold wallet, where the secret key is split among several devices. In the threshold $t + 1$ out of $n$ setting, adversary can corrupt upto $t$ devices, still not being able to forge signatures. We have already mentioned threshold signature schemes [32, 33, 53, 74, 75, 76, 101, 109, 110] in Section 2.4 that can be applied here to thresholdize cold wallets. However, none of the above schemes consider deterministic key derivation that is crucial in deterministic wallets. In other words, there is a need of a threshold *rerandomizable* signature scheme to build threshold wallets. Recently Groth et al. [78] presents a thershold ECDSA scheme with (non-hardened) key derivation of BIP32. However, they restrict to $\frac{n}{3}$ corruptions, specific to serve the setting of Internet computer [84]. They also do not consider the hardened key derivation of BIP32.

Another interesting direction is to consider password-based wallets. The simplest solution, close to that of [11] is to have a password authenticated wallet scheme in the client-server setting. Such a scheme combines ideas from password-based two-factor authentication and a threshold signature scheme. On a high level, password-based authentication ensures that, only when the client has provided the correct password, the protocol proceeds with the signature generation. Additionaly, the signature itself is generated via a 2-out-of-2 threshold signature scheme run between a client and the server. The problem with the scheme presented in [11] is the following: Firstly, whenever the server is corrupted, it can mount a offline dictionary attack on the client's password, get hold of secret key share of the client and thereby forge signatures. One solution to this issue is to distribute or thresholdize the server instead of a single server. This would require a $n$-out-of-$n$ or a $t$-out-of-$n$ access structure for the signature scheme. Secondly, such a scheme still do not consider deterministic key derivation feature of wallets. In other words, the password-based wallet must be able to derive fresh session keys for signing every new message. It would be interesting to design password-based wallets that solve either or both of the above issues, simultaneously not compromising on efficiency of the scheme.

# 3. Round Efficient Byzantine Agreement from VDFs

In the Byzantine agreement (BA) problem, a set of $n$ parties jointly run a distributed protocol to agree on a common output in the presence of some minority of $t$ malicious parties. BA is a well-studied and fundamental problem in distributed computing and has been used as a main ingredient in designing decentralized blockchain infrastructure [18, 29, 37, 96]. Traditionally, most existing protocols for BA assume a setting in which the parties' identities are fixed and known at the beginning of the protocol. In the fixed identity setting, two types of protocols are studied: the first type requires setup, e.g., a public key infrastructure (PKI) or some form of correlated randomness. These protocols typically tolerate the (optimal) corruption threshold of $t < n/2$. The second type does not require such assumptions but can tolerate only $t < n/3$ corruptions.

More recently, a third type of protocol has emerged [10, 95] that gives up on the fundamental assumption that parties know each other's identities at the beginning of the protocol. Moreover, these protocols do not require setup in the classical sense, yet still achieve the optimal corruption tolerance of $t < n/2$. Note that if identities are not fixed then without further measures, every party could pose as many parties and easily obtain a dishonest majority; this is commonly referred to as a *sybil attack* [56]. To avoid such attacks, parties must instead invest some expensive resources, such as computation or money, to participate in this type of protocol. A prominent example is the Proof-of-Work model (PoW) initially introduced by Bitcoin, where parties have limited access to a computational resource which they are forced to continuously expend in order to participate in the protocol. Known results [10, 72, 95] achieve byzantine agreement in the PoW model in $O(n\kappa^2)$, $O(\kappa)$ and $O(n)$ rounds. Our goal is to improve the round-efficiency of BA in this resource-constrained model.

## 3.1. Our Contributions

To achieve our goal of round efficiency, we refine the PoW model by considering the effort it takes to evaluate *verifiable delay functions (VDFs)* [21, 37] as the main computational resource. VDFs can be seen as a special type of proof-of-work whose computation cannot be sped up by much. This is in stark contrast to the typical lottery-type proofs-of-work, whose computation can be sped up almost arbitrarily, given sufficient parallel computation resources. We explore, for the first time, the implications of bounding the number of VDF evaluations that an (adaptive) adversary can compute in parallel: 1) We show an expected constant-round BA protocol that tolerates $t < n/2$ corrupted parties and does not rely on a PKI or known identities; 2) we give the first non-trivial communication lower bound by showing that any BA protocol in this setting requires at least $O(\sqrt{n})$ send-to-all steps.

In contrast to the work of Wan et. al [140], which allows the adversary to compute any (polynomial) number of puzzles in parallel, we aim at quantifying the number of VDFs computed by the adversary in parallel within a certain period of time. The above assumption is essential for us to obtain our results in the PKI-less setting as opposed to [140] which relies on a PKI. Our work has been disseminated in the following article.

[41] P. Das, L. Eckey, S. Faust, J. Loss, and M. Maitra. *Round Efficient Byzantine Agreement from VDFs.* Cryptology ePrint Archive, Paper 2022/823. `https://eprint.iacr.org/2022/823`. 2022.

Below we discuss our key results in more details[1].

### 3.1.1. The VDF Model

Our work introduces the VDF model as a refinement of the common PoW model to replace trusted setups and protect against Sybil attacks in permissionless consensus. Similar to the PoW model, we assume that the adversary only controls less than 1/2 of a computational resource to invest in producing *proofs of computation.* In contrast to the PoW model, however, we require a lower bound on the time it takes to create such proofs. This differs from the PoW model, where proofs can be computed almost arbitrarily fast, given sufficient parallel resources. We believe that the VDF model is a realistic alternative for the PoW model. Indeed, there exists various different constructions of VDFs [64, 127, 142] that leverage

[1]Major parts of this chapter have been taken verbatim from Section 1, Appendix D.

inherently sequential computation, and are used (or envisioned to be used) by blockchain projects for their consensus protocol (albeit not as an anti Sybil countermeasure as in our work). Examples include Chia, which relies on VDFs for leader election [37], and ETH 2.0, which plans to leverage VDFs for constructing a random beacon [29]. To make our model more realistic, we follow Wan et al. [140], and allow the adversary a small speedup in evaluating the VDF compared to the honest parties.

Let us describe our model with a concrete example. Suppose that the total amount computational power (over all protocol participants) over a fixed time period of length $t$ is 1000 VDF evaluations. Then, we demand that the total number of proofs produced by the adversary be at most 500 in the same time span. This is similar to the case of PoW model, where it is assumed that the majority of computational power belongs to honest parties. As mentioned above, we additionally give the adversary a small *speedup*, meaning that it can compute proofs a little bit faster than the honest parties.

**The $\mathsf{VDF}_\delta$ Oracle.** We now explain our formalization of the $\mathsf{VDF}$ model in some more detail. At the center of our model, we introduce the oracle $\mathsf{VDF}_\delta$, which parties can query on an input $s$ to obtain one evaluation $\phi$ of the VDF after $\delta$ time. Thus, $\delta$ denotes the difficulty parameter, specifying the number of sequential steps to be computed for one VDF evaluation. To make the model more realistic, we allow corrupted parties a $\varkappa$*-speedup* (where $\varkappa \geq 1$), meaning that they can obtain an output from $\mathsf{VDF}_\delta$ after $\delta/\varkappa$ time. An adversary $\mathcal{A}$ in our model controls some $q$ number of parties, where each party has $\varkappa$-speed-up. For some $i > 1$, let us discuss how many proofs an adversary is able to compute within time $t$, where $(i-1)\cdot\delta < t < i\cdot\delta$. We can express $t$ more concretely as $t = (i-1)\cdot\delta + r\cdot\delta$, where $r \in (0,1)$. With a corruption budget of $q$ parties, $\mathcal{A}$ can invoke the $\mathsf{VDF}_\delta$ oracle $q$ times concurrently (once per party). Since each proof is obtained after time $\frac{\delta}{\varkappa}$, at time $(i-1)\cdot\delta$, each party computes $(i-1)\cdot\varkappa$ proofs. In the remaining $r\cdot\delta$ time, each party can compute ($at$ most) $\lfloor\frac{r.\delta}{\delta/\varkappa}\rfloor = \lfloor r\cdot\varkappa\rfloor$ proofs. Thus, in total $\mathcal{A}$ obtains at most $((i-1)\cdot\varkappa + \lfloor r\cdot\varkappa\rfloor)\cdot q$ proofs at time $t$. Figure 3.1 illustrates our model with a small example. We refer to this property of the $\mathsf{VDF}_\delta$ oracle as its *sequentiality* and give a formal definition in Appendix D, Section 2.

Such oracle abstraction of the VDF computation allows us to give a cleaner and more modular analysis of our main protocols. In support of our modelling approach, we conjecture that the $\mathsf{VDF}_\delta$ oracle can be instantiated in the standard model (full version [41], Appendix A, Lemma 20) and we prove that it can be instantiated in the strong algebraic group model for constructions of Wesolowski [142] and Pietrzak [127] (full version [41], Appendix A, Lemma 23).

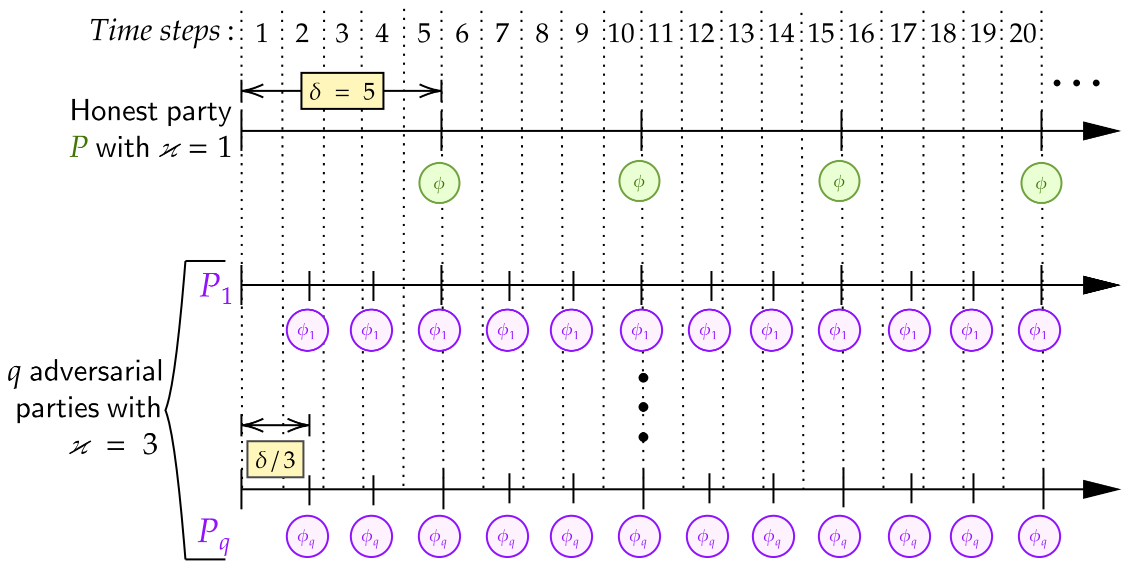

Figure 3.1.: Consider $i = 3, \delta = 5$. We have $i \cdot \delta = 15, (i-1) \cdot \delta = 10$. Say an adversary $\mathcal{A}$ controls $q$ parties $\{P_1\}_{i \in [q]}$ with a speed-up $\varkappa = 3$ compared to an honest party with $\varkappa = 1$. Consider two time steps: $t = 11$ and $t = 14$. In both cases, each $P_i$ can compute $(i-1)\varkappa = 6$ proofs in time $10 < i \cdot \delta$. For the remaining time $r \cdot \delta = 1$ (for $t = 11$) and $r \cdot \delta = 4$ (for $t = 14$), no extra proofs can be computed in the first case, whereas $\lfloor \frac{4}{5/3} \rfloor = 2$ extra proofs can be computed in the second case. Thus, $\mathcal{A}$ can compute in total $6q$ and $8q$ proofs for $t = 11$ and $t = 14$ respectively.

**Adversary Model.** The adversary in our protocol is modelled as a $(q, t_p, \varkappa)$-algorithm $\mathcal{A}$ as defined above. $\mathcal{A}$ can control at most $q$ parties each with a maximum speedup of $\varkappa$, such that $q < \frac{n}{\lfloor \varkappa \rfloor + 1}$ holds. In particular, the number of adversarial parties can be at most $< \frac{n}{2}$ (this is the case for $\varkappa = 1$). We consider an adaptive adversary which can corrupt a party at any point during the protocol execution. Once a party has been corrupted, it can arbitrarily deviate from the protocol execution. Furthermore, it can deliver a message over the multicast channel only to a subset of honest parties. In this way, it can send different messages to different subsets of honest parties over the multicast channel. However, the adversary can not drop the messages of honest parties from the channel or delay them for longer than $\Delta$. Our adversary is *rushing*, which means it can observe all the messages that the honest parties send in any round of the protocol, and then choose its own messages for that round adaptively. We notice that we consider the standard notion of an adaptive, rushing adversary, as opposed to the stronger notion of a *strongly rushing* (or strongly adaptive) adversary (see for e.g., [**cryptoeprint:2021:775**, 1, 2]) who can adaptively corrupt parties and then delete messages that they sent in the same round (prior to corruption).

## 3.1.2. Byzantine Agreement in the VDF Model

As our main technical contribution, we show how to obtain an expected constant-round Byzantine agreement protocol without any additional trusted setup in the VDF+random oracle model. This is of particular significance, given that we can also instantiate the VDF model without any trusted setup, using Wesolowski's construction. Given an upper bound $n$ on the number of parties, our protocol tolerates $q$ adaptively corrupted parties with $\varkappa$-speed-up, where $q(\lfloor \varkappa \rfloor + 1) < n$. In particular, our protocols tolerate up to $\frac{n}{2}$ corruptions when $\varkappa = 1$. In our protocols, we consider the *multicast* model of communication, where each party sends a message to all other parties via a multicast channel. Below, we present our necessary sub-protocols which finally leads us to achieve constant round BA.

- *Step 1: Establishing a Graded Public key Infratsructure (GPKI).* We adopt the idea of Andrychowicz and Dziembowski [10] and start by setting up a precursor to a full PKI called *graded PKI* (GPKI) among the parties. Roughly speaking, a GPKI differs from a full PKI in that the keys of the parties are additionally associated with grades. These grades can differ between parties, but not by too much. As a first step, to reduce the round complexity of their protocol to $O(1)$ from $O(n\kappa^2)$ (where $\kappa$ is a security parameter), we make two modifications: 1) We set up a $m$uch weaker GPKI with only two possible grades, whereas [10] sets up $n$ possible grades. 2) We borrow a technique from Katz et al. [95][2] and rely on VDFs to make the round complexity of our protocol independent of $\kappa$. As a second step, the difficulty parameter $\delta$ of the $\mathsf{VDF}_\delta$ must be adjusted to tolerate adversarial speedup in the $\mathsf{VDF}_\delta$ model.

  Here we give a short overview of our graded PKI protocol $\Pi_{\mathsf{KeyGrade}}$ from Section 3, Appendix D. Protocol $\Pi_{\mathsf{KeyGrade}}$ consists of three phases: *challenge phase*, *proof of computation phase* and a *key ranking phase.* The challenge phase consists of two rounds, where each party $P$ chooses a fresh challenge value as $c \xleftarrow{\$} \{0,1\}^{\mathsf{len}}$ (where, $\mathsf{len}$ is some well-defined bit length) and multicasts it to other parties. In the second round of challenge phase, $P$ gathers all first round challenges, including its own, into a set $\mathbf{c} = (c_1, c_2, \ldots)$; hashes this set to form a second round challenge $d = \mathsf{H}(\mathbf{c})$ and then multicasts $d$ to all other parties. After

[2]The construction in [95] uses only a proof of sequential as opposed to a VDF.

above two rounds of challenge phase, begins the proof of computation phase that works as follows. First, $P$ collects all second round challenges, into a set $\mathbf{d} = (d_1, d_2, \ldots)$, and hashes it into a final challenge value $\chi = \mathsf{H}(\mathbf{d})$. Here, $\chi$ serves as a membership proof of each second round challenge $d_i$ received by $P$: first by checking if $d_i \in \mathbf{d}$, then whether $\mathsf{H}(\mathbf{d}) = \chi$. Each of the second round challenges, in turn serves as a membership proof of first round challenges, received from first round multicasts. Next, $P$ samples a key pair $(\mathsf{sk}, \mathsf{pk})$ and computes a (sequential) proof of computation $\phi$, by querying the $\mathsf{VDF}_\delta$ oracle on input $\chi$ as $\phi = \mathsf{VDF}_\delta(\chi)$. Lastly, the key ranking phase starts with every $P$'s multicast of $\mathsf{pk}$, the evaluated proof of computation $\phi$, along with two additional values required for membership proof: the input $\chi$ on which the proof $\phi$ is computed and its preimage $\mathbf{d}$. If another party $P_j$ is convinced that $\phi$ is a valid proof on $\chi$ and the set $\mathbf{d}$ includes $P_j$'s second round challenge $d_j$, then $P_j$ ranks $\mathsf{pk}$ with the *highest grade* 2. $P_j$ further multicasts the tuple received from $P$, containing $\phi$, $\chi$, $\mathbf{d}$ and $P_j$'s first round challenge set $\mathbf{c_j}$ to all other parties – let us call such a party $P_k$. If $P_k$ did not grade $\mathsf{pk}$ before, then it assigns grade 1 to $\mathsf{pk}$, after verifying the information received from $P_j$, i.e. whether $\phi$ is a valid proof on $\chi$ and whether her own first round challenge $c_k$ is included in $\mathbf{c_j}$, where $d_j = \mathsf{H}(\mathbf{c_j})$ was included in $\mathbf{d}$.

We show that our protocol $\Pi_{\mathsf{KeyGrade}}$ satisfies the underlying properties of a graded PKI. One of the properties is *bounded number of identities*, which states that the number of adversarial identities in a GPKI protocol must be at most half of the total number of identities. In our analysis, we needed to be very careful such that this property is satisfied. This is because, our adversary, controlling $q$ parties and having a computational speedup factor of $\varkappa$, can compute multiple proofs in parallel, faster than honest parties. However, we were able to show that, this property is satisfied, even when we set the duration of the proof of computation phase to a constant time (independent of the security parameter or the number of parties). In particular, the time complexity of the proof of computation phase is dependent on the duration of the rest of the protocol, which is in turn, a constant number of rounds. As a result, we obtain a constant round GPKI protocol for the first time. We present our formal protocol and its security in Section 3, Appendix D.

- *Step 2: Graded Consenus from GPKI.* One of the ingredients needed in our BA protocol is a graded consensus protocol. Graded consensus is similar to a BA, with the difference that at the end of the protocol, every party outputs a value with some grade, where the grades of different parties must satisfy some consistency properties. We build upon the graded consensus protocol of Micali and Vaikuntanathan [119] where we modify their protocol such that it requires only a GPKI instead of a full PKI.

- *Step 3: Leader election protocol from VDFs.* It is well known that expected constant round protocols are inherently randomized, e.g., by electing a random leader in every protocol iteration. However, electing a random leader efficiently is challenging without prior setup. We overcome this issue by presenting a novel leader election protocol that leverages the oracle $\mathsf{VDF}_\delta$ to efficiently elect a random leader that all honest parties agree on with high probability. Our protocol is inspired by leader elections based on verifiable random functions [1, 2] and implements a leader election lottery with unique tickets. This makes the tickets hard to bias from the perspective of the adversary.

- *Step 4: BA protocol.* Finally, we combine all of the above components to adopt the expected constant round protocol of Katz and Koo [93] to our setting.

  We informally state our overall result in the following theorem.

  **Theorem 1** (Informal)**.** *Let $n$ denote an upper bound on the number of parties. Then, there exists an expected-constant round BA protocol in the VDF model that is secure against any adversary $\mathcal{A}$ that controls at most $q$ parties with $\varkappa$-speed-up, where $q \cdot (\lfloor \varkappa \rfloor + 1) < n$.*

  We present formal descriptions of our graded consensus, leader election and BA protocols and their security analysis in Sections 4 and 5, Appendix D.

### 3.1.3. A Lower Bound on Communication Complexity for BA

As a third contribution, we give the *first* lower bound for communication complexity of BA, assuming parties have bounded computational resources. Concretely, we show that in the VDF model without additional trusted setup, no protocol can realize BA with overwhelming probability by multicasting fewer than $O(\sqrt{n})$ messages in the presence of adaptive corruptions. In the multicast model of communication, honest parties are restricted to sending messages to *all parties at once*, whereas the adversary can send to only a subset of the parties. This models a setting in which parties communicate via a *gossip network* [10, 69]. The cost of running the same protocol from the multicast model in the bilateral channel model [54] would be $O(n^{3/2})$. We remark, however, that the multicast restriction is crucially used in the lower bound, and thus, a better communication complexity might be possible in the bilateral channels model. Our lower bound builds on ideas of Abraham et al. [1] who show a bound for Byzantine broadcast in the multicast model without setup.

Our bound has to overcome several technical challenges that arise when parties have limited computational resources. The adversary in our attack has to carry out a simulation of the protocol in its head (this is a standard technique used in lower bounds), which may require to query the VDF oracle. At the same time, the adversary must also participate in an actual execution of the protocol it is attacking, which, of course, also results in queries to the oracle. Thus, the key difficulty in our lower bound is to carefully balance the adversary's limited budget of queries to VDF over the two executions of the protocol (real and simulated).

Although our lower bound is relatively weak compared to most existing lower bounds in this area (which are quadratic, or of the form $O(n)$ in the multicast model, respectively), we argue that it is still meaningful. Namely, protocols that require significantly above $O(\kappa)$ multicasts are deemed impractical for large-scale settings with millions or even billions of users. This means that our bound essentially rules out efficient solutions in the VDF model unless further setup is assumed among the parties. Second, we point out that our lower bound actually holds in the relatively weak VDF-*model* and can likely be carried over to a less restrictive model (e.g., to the PoW model used by Bitcoin). It also leaves room for a tighter bound in such more general models. Details on our results can be found in Section 6, Appendix D.

## 3.2. Implications of Our Results and Related Work

Our model can be instantiated using Wesolowski's VDF, which does not require trusted setup. Thus, our results show, for the first time, how to perform expected-constant round BA in a permissionless model with a simple honest majority and no trusted setup (beyond a random oracle). This has many important implications. For example, one could use our protocol to efficiently agree on a random string in a permissionless setting. This string could be used as a genesis block or as a uniform common reference string to perform an MPC protocol. Our results also significantly improves over the result of Andrychowicz and Dziembowski, who presented a protocol that achieves essentially the same, but requires $O(n\kappa^2)$ rounds to do so [10]. We also improve over a similar (slightly more efficient) version of this idea shown by Katz et al. [95]. Another closely related work is that of Garay et al. [72] who show how to bootstrap the classical Nakamoto consensus protocol [71, 120, 124] in the PoW model without trusted setup. However, it requires $O(\kappa)$ rounds, and therefore also does not constitute an expected constant round protocol.

**Further Related Work.** There is a large body of research on BA and related problems (sometimes colloquially referred to as "consensus"), and we focus here on the most closely related works. We have already mentioned the works of [10, 72, 95] who achieve BA in the PoW model without setup and run in $O(n\kappa^2)$, $O(\kappa)$, and $O(n)$ rounds, respectively. It should also be noted that we require stronger assumptions (namely a VDF and the random oracle model (ROM)) than the protocols in [10, 72, 73], who require only the ROM that can be queried at a bounded rate by any party, but (possibly) weaker assumptions than required in the work of Katz et al. [95], who also require some form of an unpredictable beacon in their protocol. The more recent work of Aggarwal et al. [3] presents a setup-free solution in the PoW model that also runs in expected $O(1)$ rounds, but assumes a static adversary (while we consider the much stronger adversarial model of adaptive corruptions). Another related work that focuses on the PoW model is by Garay et al. [73]. They show how to achieve UC-secure BA and multi-party computation (MPC) protocols in the PoW model. Similar to [10], their BA takes $O(n\kappa^2)$ rounds.

Although the above-mentioned prior works achieve BA without a PKI [10, 72, 95], their techniques are not what we need to achieve $O(1)$-round BA. In fact, we notice that achieving $O(1)$-round BA protocols requires very particular techniques that have been studied [1, 2, 63, 93, 118, 119, 129] in the classical setting for many years and this round reduction is very challenging to achieve.

## 3.3. Discussion and Future Work

In this work, we are able to achieve (expected) constant round byzantine agreement in the resource-constrained VDF model of computation, without any PKI assumption. We consider an adaptive adversary, controlling $q$ parties, with a computational speedup of $\varkappa$, such that $q \cdot (\lfloor \varkappa \rfloor + 1) < n$ holds. One of the crucial assumptions of our VDF model is to *bound* the number of VDF evaluations an adversary is able to compute in parallel, as opposed to the model of Wan et. al [140], where the adversary can make any polynomial number of parallel evaluations. Such a restriction models an adversary, whose total adversarial computational power is less than half of the total computational power – this is same as honest majority assumption in the PoW model. One of the crucial steps for achieving (expected) constant round BA is to setup a graded PKI in constant number of rounds. To achieve the property of bounded number of identities of our graded PKI, we need to rely on the honest majority assumption of computational power. Wan et al. however assumes a PKI to obtain their results, hence they do not come across a similar problem. Naturally, an open question would be to achieve our results in presence of a stronger adversary, that can compute *arbitrary* (polynomial) number of VDF evaluations in parallel. Our results are in the synchronous communication model, where parties are assumed to have synchronized clocks. It would be interesting to lift our results to a more realistic setting of partial synchrony or even asynchronous mode of communication. Finally, we have only considered sequential runs of all our protocols. It will also be interesting to investigate concurrent run of our protocols, for instance in the universal composabality framework.

In this work, we also provide the first lower bound for communication complexity of BA, in the resource-constrained model of computation. In more details, we show that in the VDF model, without any trusted setup assumptions, no protocol can achieve BA with less than $O(\sqrt{n})$ multicast communication complexity, in presence of an adaptive adversary. An immediate open question would be to extend our bound to the more generic PoW model. It also leaves room for a tighter bound in such more generic models.

# 4. Distributed Password-Authenticated Symmetric-key Encryption

Outsourcing storage to cloud providers is not only a common approach in enterprise settings, but is also widely appreciated by end users relying on services such as Dropbox, Google Drive, iCloud or Microsoft OneDrive to manage their personal data. With data breaches happening on a daily basis, it is essential that personal data kept in such cloud storage must be protected accordingly. The prevalent approach is to trust the cloud with properly encrypting the data, where the service provider controls access to the respective encryption keys via standard user authentication, mostly relying on username-password authentication. Clearly, such a solution crucially relies on the honesty of the service provider who can otherwise gain plaintext access to the users' data.

A different approach is to let the user already encrypt the data before storing it in the cloud, which is offered e.g., by Tresorit [135] or Mega [117]. Therein a user client is locally encrypting the data and only uploads ciphertexts to the cloud. The cryptographic keys are either generated and stored directly by the user client, or (re)-derived from a human-memorizable password that the user enters into the local client. The former provides strong security guarantees, but is cumbersome to use as it relies on users' being able to manage and securely store cryptographic keys. The latter provides (roughly) the same convenience and usability as standard cloud provides as it does not require secure storage on the user side, but is inherently vulnerable to so-called offline attacks: Since encryption keys are derived from a low-entropy password, a corrupt service provider or an attacker gaining access to the ciphertexts, can attempt to decrypt the files by guessing the user's password.

While recently some service providers have moved away from password and deploy solutions where users are required to store key material (e.g., [84]), password-based systems remain the only truly device-independent solution at our disposal. In this work, we investigate how users can password-encrypt their cloud data *without* storing any key material, and *without* making their encrypted data prone to offline password-guessing attacks.

**Known approaches to password-based encryption.** One way to avoid the two aforementioned issues is to use a *distributed password-based key management* system: a user retrieves her encryption key from a set of servers, using only a password as input. This does not require the user to store any cryptograhpic material, since the servers take over this role, and the distribution of keys among servers thwarts of offline attacks on the password. There exist various cryptographic primitives suitable to implement such password-protected key retrieval, for example Password-Authenticated Secret Sharing (PASS/PPSS) [14] and Oblivious Key Management [89] (discussed in details in Section 4.2).

All aforementioned schemes allow users to turn a password into an encryption key. In practice, this means that users either encrypt all their data with *the same key*, or they must memorize *as many passwords as keys* that they want to use. For optimal usability *and* security, in a password-based key management scheme, we want to ask the user to remember only *few but strong* passwords, and "behind the scenes" still use different encryption keys for every piece of data she wants to encrypt. Varying encryption keys is desirable to mitigate the effect of security breaches of the user's device, or of irresponsible handling of secret keys on the user side. We note that there exist other ways to mitigate the effect of such attacks, for example allowing for efficient updates of the encryption key, which however provide no protection in case the attacker is already in posession of ciphertexts. In this work, we prefer one-time usage of encryption keys over updatability, since then revelance of one key upon compromise does not impact the confidentiality of more than one encrypted piece of data.

## 4.1. Our Contributions

From the discussion above, the requirement for a usable yet strongly secure password-based encryption scheme is clearly evident. To close this gap, we build our primitive DPaSE : Distributed Password-Authenticated Symmetric-key Encryption scheme. DPaSE allows users to securely and conveniently encrypt and decrypt their data with different encryption keys while relying only on a *single password* and the assistance of $n$ servers. We provide an efficient realization based on a new type of Oblivious Pseudorandom Function (OPRF) that supports correlated evaluations of blind inputs, which we believe to be of independent interest. Our results have been disseminated in the following publication, which can be found in Appendix E.

[48] P. Das, J. Hesse, and A. Lehmann. "DPaSE: Distributed Password-Authenticated Symmetric-Key Encryption, or How to Get Many Keys from One Password".

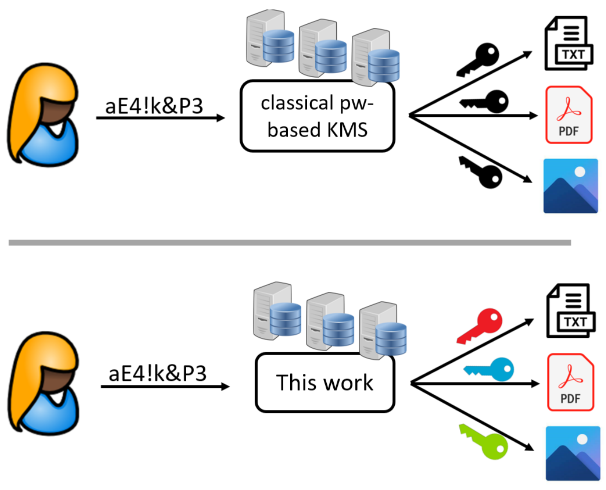

Figure 4.1.: Classical password-based server-assisted KMS yields one key per password to encrypt all the different user data. Our solution DPaSE is a server-assisted encryption scheme that allows to derive different encryption keys from only one password.

In: *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*. 2022, pp. 682–696. DOI: `10.1145/3488932.3517389`. URL: `https://doi.org/10.1145/3488932.3517389`.

We summarize our main ideas from [48] below[1].

### 4.1.1. Our Primitive DPaSE and its Properties

DPaSE is not a mere key management system, but has built-in encryption of data with the retrieved keys already. Encryption and decryption is carried out locally by the user using the retrieved key. This integrated modeling allows us to demand the following strong security and functionality from a DPaSE system, covering guarantees with respect to both passwords *and* encryption of data.

- *Correct Encryption.* If a user types an incorrect password upon encryption, her data is not encrypted and the user instead obtains an error message. This property is important to avoid that a user accidentally encrypts her data with unrecoverable secret keys.

[1]Major parts of this chapter has been taken verbatim from Section 1, Appendix E.

- *No Reuse of Keys.* Every ciphertext is created with an individual key. Hence, in case a user loses one of her encryption keys, all but one of her encrypted files remain confidential.
- *Security against Offline Attacks.* As long as at least one server is honest, the encrypted data (or rather the underlying password) cannot be offline attacked. And even if eventually *all* servers are corrupted, they cannot decrypt the data immediately but must still perform an offline attack on the password – thus when users have chosen strong passwords, their data remains secure.
- *Security against Online Attacks.* To detect and prevent online guessing attacks, the servers learn which user is trying to encrypt or decrypt, and whether her entered password was correct. In particular, we require that every file access/decryption requires explicit approval of all servers. When an honest server has recognized suspicious behaviour or was alerted by the user herself, it can enforce user-specific rate limiting or even fully block a certain account.
- *Obliviousness.* Servers do not learn anything about the files (plain- or ciphertext) the user wants to access[2]. It was demonstrated [85, 103] that such leakage would have devastating effects on the user's privacy.
- *Authenticated Encryption.* An adversary cannot plant wrong information into the outsourced storage. Thus, unless the adversary knows the user's password (and is assisted by all servers) it must be infeasible to create valid ciphertexts.

**Security Model in the UC Framework.** We formally define these properties by means of an ideal functionality $\mathcal{F}_{\mathsf{DPaSE}}$ using the Universal Composability (Universally Composable (UC)) framework [31], which is known to allow for the most realistic modeling for how users (mis)handle passwords. In game-based security models, users choose their passwords at random from known distributions and are assumed to behave perfectly, i.e., never make a typo when using a password. This clearly does not reflect reality, where users share or re-use passwords, and make mistakes when typing them. The UC framework models that much more naturally as therein the environment provides the passwords. Thus, a UC security notion guarantees the desired security properties without making any assumptions

[2]We do not want to go further and hide the identity of the user in his requests, since otherwise we would not be able to protect against online guessing attacks.

regarding the passwords' distributions or usages. Our modeling also ensures that any DPaSE protocol is secure when executed concurrently with other systems, thanks to the strong composability guarantees of the UC framework.

However, these desirable features come at a cost. In order to end up with a manageable and understandable security definition (i.e., UC functionality), we need to make compromises and protect against some attacks that might not be of high relevance to DPaSE in practise[3]. For example, we need to prevent servers from intentionally deriving encryption keys from wrong passwords, which makes our protocol a bit more costly and restricted to security against semi-honest servers. There exist many ways of protecting against such attacks, each with different trade-offs. For example, we could use client-side caching of password-dependent inputs, leaving it up to the client to use the correct password. Such a solution would however not suffice for our purpose of achieving a concise UC definition (a malicious client could simply mess up the caching then, introducing valid encryptions under wrong passwords to the system). Hence, in this paper, we opt for a stronger and cleaner definition, at the cost of slightly worse efficiency and slightly weaker corruption model.

## 4.1.2. Our DPASE Protocol

We present an efficient protocol that provably realizes our functionality $\mathcal{F}_{\mathsf{DPaSE}}$. The high-level idea of the protocol is very simple and follows the known paradigm of password-based protocols to turn the password into cryptographic key material using an OPRF [16, 90]. More detailed, to create an account, the user derives a signing key $(upk, usk) \leftarrow \mathsf{OPRF}(K, uid, pw)$ from her username and password, where the OPRF key $K$ is split among the $n$ servers and the evaluation reveals the username to the servers to later allow for user-specific rate limiting. The servers store $(uid, upk)$ upon registration.

To encrypt a file, the user again enters $uid, pw'$ and starts by re-running the steps from account creation to recover her signing key pair $(upk, usk)$. She then signs a fresh nonce with $usk$ and sends it to the servers who verify it against the stored $upk$, thereby verifying that $pw = pw'$. If the password is correct, the user and server engage in a follow-up OPRF evaluation where an object-specific encryption key is

[3]A UC functionality needs to "list" *all* potential attacks that can be mounted against a protocol. While some attacks might be benign in practise and we might be okay with the threat they are imposing on us, every such attack still shows in the functionality. It is one of the main challenges in using the UC framework to find a mid-way between a not overly strong notion that still allows for efficient instantiations, and one that is not overly cluttered with such benign attacks.

derived. The OPRF evaluation thereby "reuses" the previously entered *uid*, $pw'$ to ensure that the actual encryption keys are also bound to the user' identity and correct password. This prevents users from accidentally encrypting data under a wrong password. To ensure obliviousness, the object for which the key is derived is hidden in the evaluation.

Decryption works almost analogously to encryption, verifying the password and – if correct – recovering the object-specific encryption key via the distributed OPRF. The generated ciphertexts and decryption procedure also include checks to guarantee the desired ciphertext integrity.

**Extendable Distributed Partially-Oblivious PRF.** The core of our DPaSE protocol is a new type of OPRF that we believe to be of independent interest for many password-based applications. So far, OPRFs have been designed as *single*-evaluation primitives[4] that can either be fully or partially-blind. Thus, the user sends a (partially) blind query, and receives a single output related to that input. What we need for DPaSE though is an OPRF that "remembers" the blindly provided password from a previous query and re-uses it in a follow-up evaluation: we need to perform a dedicated password check and also want to ensure that encryption is done with the same password that was verified. We model that as an extension query, where a second OPRF query re-uses the blinded input from a previous request. This extension feature is required on top of *partial*-blindness (as the *uid*'s must be a known input to all parties) and the *distributed* setting. We formalize the desired properties of such an extendable OPRF in the UC framework and propose a secure instantiation. We believe that this is a contribution of independent interest. Namely, using an OPRF with th extendability property could generally add secure password verification to protocols that deploy an OPRF to bootstrap cryptographic material from passwords.

Our OPRF construction is based on the classical double-hash DH scheme, basically combining all tricks that have been used in this context into a single scheme. The challenge thereby is that our second OPRF call which blindly carries over the input from the first call now has *three* inputs: the non-blind part ($x_{\mathsf{pub}} = uid$), and two blinded values, namely the blinded ($x_{\mathsf{priv1}} = pw$) from the previous evaluation and the new input ($x_{\mathsf{priv2}} = oid$). Previous partially-blind OPRFs deal with two inputs only $x_{\mathsf{pub}}$ and $x_{\mathsf{priv}}$ which are mostly combined through a pairing [16, 61], with the final PRF being of the form $\mathsf{H}_T(e(\mathsf{H}_1(x_{\mathsf{priv}}), x_{\mathsf{pub}})^K, x_{\mathsf{priv}})$. In our construction, we will already need both "slots" of the pairing to combine the two blinded

[4]With the exception of OPRF with batch evaluations under several keys [100, 107]. This is orthogonal to our problem since we have a single OPRF key.

inputs, and therefore must find a different place to include the public input. We take inspiration from [89] and replace the direct use of the server's secret key $K$ by $K' \leftarrow \mathsf{F}(K, \mathit{uid})$ where $\mathsf{F}$ is a standard PRF. Thus, overall our new OPRF computes the output for an extended query as $\mathsf{H}_T(e(\mathsf{H}_1(x_{\mathsf{priv1}}), \mathsf{H}_2(x_{\mathsf{priv2}}))^{\mathsf{F}(K, x_{\mathsf{pub}})}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}})$. The first (non-extended) query, just consisting of $x_{\mathsf{pub}}$ and $x_{\mathsf{priv1}}$ has the same form and simply sets $x_{\mathsf{priv2}} = 1$.

This construction allows us to combine three values into a single evaluation, but this extendability feature comes for a price. First, relying on exponents that are derived from a standard PRF $K' \leftarrow \mathsf{F}(K, \mathit{uid})$ only allows for a distributed, but not threshold protocol. The distributed version simply considers the additive combination of all $K'$ as the implicit overall secret key (per $x_{\mathsf{pub}}$). Second, there are currently no efficient proofs that allow to check whether the servers have computed the second evaluation correctly – which again stems from the use of the standard PRF to derive the OPRF secret key share. As we will require correct computation of OPRF outputs in our DPaSE protocol, we must assume that the servers in the OPRF are at most honest-but-curious. We stress that considering honest-but-curious servers already captures the main threat to passwords: an adversary stealing the password database (or other offline-attackable information). To our knowledge, DPaSE is currently the only protocol being secure in the presence of such attacks.

Lastly, we note that extendability is a property that could as well be ensured on the application level by, e.g., caching the user's password on the client machine. While this would enable using DPaSE with a standard, i.e., single-evaluation OPRF and make our protocol simpler and more efficient, it allows for a „benign“ attack which prevents a security proof. Namely, an adversary knowing the password of an honest user could produce encryption keys under bogus passwords. If the honest user later tries to decrypt such a maliciously crafted ciphertext, decryption would fail – yet the adversary can decrypt using the bogus password again. While this attack is rather harmless in practice, to prove the password-caching version secure one would have to include this imperfection into the security definition, with a different set of "shadow passwords" for each (!) ciphertext that the adversary could use (even for honest accounts). With the extendability property, we enforce password consistency *on the protocol level* and hence avoid cluttering the security definition of DPaSE with attacks resulting from inconsistent usage of passwords.

Formalization of our OPRF primitive, instantiation of our OPRF protocol and its security analysis can be found in Section 3, Appendix E, while DPaSE functionality, protocol instantiation and corresponding security analysis is provided in Section 4, Appendix E.

### 4.1.3. Evaluation and Comparison

| Scheme | #(Exponentiations + Pairings) per Encryption | |
|---|---|---|
| | client/rate limiter | server |
| PHE [104] | 7 exps (in $\mathbb{G}$) | 10 exps (in $\mathbb{G}$) |
| $\Pi_{\mathsf{DPaSE}}$ (Our Work) | 10 exps (= $2\mathbb{G}_1 + 2\mathbb{G}_2 + 4\mathbb{G}_T + 2\mathbb{G}_{\mathsf{p\text{-}256}}$) | 4 exps (= $2\mathbb{G}_T + 2\mathbb{G}_{\mathsf{p\text{-}256}}$) +2 pairings |

Table 4.1.: Comparison of our DPaSE protocol with closest password-based encryption scheme PHE, where the exponentiations are counted per group $\mathbb{G}$, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, the pairing is mapped as $\mathbb{G}_1 \times \mathbb{G}_2 \colon \to \mathbb{G}_T$ and $\mathbb{G}_{\mathsf{p\text{-}256}}$ represents the prime group in the ECDSA signature scheme secp256r1.

In this section, we consider an instantiation of our DPaSE protocol, where the OPRF functionality is instantiated with our OPRF protocol, and the signature scheme SIG is instantiated with ECDSA. We report on the efficiency of our scheme, by counting the number of exponentiations per group and pairings, being the most expensive operations of such protocols. We compare our DPaSE protocol with what we believe to be the closest related password-based encryption scheme, namely Password Hardened Encryption (PHE) [104] (see also Table 4.3 for the overlap of properties of both schemes). Considering each exchange of messages between the client and servers as one round of communication, our DPaSE protocol requires 2 rounds for Account Creation and 3 rounds for Authentication followed by a Encryption (Decryption) request.

| Protocol | No. of Servers | Execution Time (in ms) | | Requests per server (per second) |
|---|---|---|---|---|
| | | user | server | |
| Account Creation | 2 | 18 | 13 | 76 |
| | 6 | 19 | 13 | 76 |
| | 8 | 19 | 13 | 76 |
| | 10 | 19 | 13 | 76 |
| Authenticate + Encrypt | 2 | 32 | 27 | 37 |
| | 6 | 37 | 27 | 37 |
| | 8 | 40 | 27 | 37 |
| | 10 | 43 | 27 | 37 |

Table 4.2.: Timing measurements of our DPaSE protocol run between one user and $k = \{2, 6, 8, 10\}$ number of servers.

We carried out a proof-of-concept implementation [57] of our DPaSE protocol and report preliminary benchmarks on the same. We implement in Java, and use the MIRACL - AMCL library for the pairing computation and exponentiation operations. We use the Boneh-Lynn-Shacham pairing with 461 bit curves for the pairing $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ in our OPRF protocol, ECDSA with sec256r1, SHA-512 as the underlying hash function $\mathsf{H}$, AES-256 to construct the standard PRF function $\mathsf{F}$ and $\mathsf{H\text{-}PRG}$ and the Java's inbuilt KeyPairGenerator class for user key pair generation $\mathsf{SIG.KGen}$. The elements in groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are implemented using single exponentiation operations with the respective group generators. The underlying hash functions are implemented by first applying SHA-512 followed by an exponentiation in the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ respectively.

We measured our implementation on a machine running a Intel Core i7-7500U series CPU with 4 virtual CPUs, 16 GiB of RAM. We focused on measuring the local computation times both on the client and the server sides, and did not consider delays due to network latency. The details of our timing measurements corresponding to our DPaSE protocol run between one user and a number of servers can be found in the Table 4.2. The time taken by each server for processing an account creation is 13 milliseconds, while that required for processing a user authentication followed by an encryption request is 27 milliseconds. Consequently, each server is able to process 76 account creation requests and 37 encryption requests per second. Since the computation underlying an encryption or a decryption is almost the same, we have only detailed the encryption timings. We stress here that the timing benchmarks can be further improved by exploiting the parallelizability of the underlying algorithms as well as utilizing the capabilities of multiple cores of a computer. Since this enhancement was not the focus of our work, in our implementation, we have relied on standard cryptographic libraries as mentioned above, which create the bottleneck in our timing measures. Close to our work, Pythia achieves a throughput of  130 requests ps [ECS+15] (also pointed in [LER+18]). In theory, efficiency of our DPaSE protocol is lower-bounded by half the throughput of Pythia, which is  65 enc/dec requests ps. This is because each enc/dec request in our DPaSE protocol requires 2 OPRF evaluations, and each has the same computational cost as one Pythia evaluation. We note that password verification and encryption both add only little overhead.

| Properties\Schemes | **Key Management Schemes (KMS)** | | **Encryption Schemes** | | | |
|---|---|---|---|---|---|---|
| | PASS scheme [88] | PASS scheme Memento [30] | OKMS [89] | DiSE [4] | (Threshold) PHE [104], [26] | DPaSE this work |
| Password correctness ensured | - | - | | | ✓ | ✓ |
| Can derive multiple keys per password | - | - | | | - | ✓ |
| Security against online attacks | - | ✓ | | | ✓ | ✓ |
| Security against offline attacks | ✓ | ✓ | | | ✓ | ✓ |
| Password remains private | ✓ | ✓ | | | - | ✓ |
| Access pattern remains private | | | - | - | ✓ | ✓ |
| Authenticated encryption | | | ✓ | ✓ | ✓ | ✓ |
| Who encrypts? (U=User, S=Server) | | | U | S | S | U |
| Mitigation of compromised encryption keys | | | no reuse & key rotation | - | key rotation | no reuse |
| Secure in concurrent settings | ✓ | ✓ | - | - | - | ✓ |

Table 4.3.: Properties of server-assisted encryption and encryption key retrieval (KMS) schemes. Gray cells are not applicable. More precisely, password properties do not apply to OKMS and DiSE schemes, as they rely on strong user authentication. Likewise, encryption properties do not apply to the KMS schemes, since their purpose is to recover an encryption key from a password.

## 4.2. Related Work

Password-authenticated secret sharing (PASS/PPSS) allows a user to recover a strong secret that is shared among $n$ servers when she can enter the correct password [14, 86, 87, 88]. In contrast to end users, servers can easily maintain strong cryptographic keys which is leveraged by PASS to thwart offline attacks against the password (and consequently on the shared secret key) if at least one, or a certain threshold, of the servers is not compromised. While this concept is shared between PASS and DPaSE, PASS can only be used to derive one encryption key per password, while DPaSE is required to encrypt each piece of data under different keys, yet enabling the user to encrypt all her data under the same password.

Password-hardened encryption (PHE) [104] targets a related setting, where a user outsources key management, encryption and decryption to a so-called rate limiter. The user can send encryption/decryption requests through a server, but needs to provide a correct password. The rate limiter can be implemented in a threshold version [26] to further enhance PHE's security. The scheme allows a mechanism of key rotation, to mitigate against compromises or simply as a routine process. Key rotation involves the server and the rate limiter updating their respective keys as well as the ciphertexts accordingly. In PHE the frontend server is fully trusted, as it learns the user's password and keys. PHE schemes are very efficient (no OPRF is required!) and a good option in settings where the client fully trusts the server, since both password and access pattern on user's data are shared with the server. In our work, we do not want to assume such trust and hence opt for client-side encryption of data to hide access patterns.

Updatable Oblivious Key Management [89] also relies on a OPRF to derive file-specific encryption keys with the help of a (single) external server for increased security. Their work focuses on an enterprise setting for storage systems though, i.e., it relies on strong authentication between the client (that wants to encrypt or decrypt) and the server that holds the OPRF key. This is a first difference to DPaSE: our scheme achieves oblivious key management without strong client-authentication. Second, their system uses key rotation – similar to PHE and the general concept of *updatable encryption* with post-compromise security [99, 108] – to update encryption keys and the corresponding ciphertexts as a measure to mitigate the effect of security breaches. The approach of our DPaSE protocol is orthogonal: we mitigate the risk of data breaches by using a distributed setting instead of key rotation: the information to recover the encryption keys is split across $n$ servers, and the file-specific encryption keys are secure as long as one server remains uncompromised.

The DiSE protocol [4] and its improvements [36, 141] for distributed symmetric encryption consider strong authentication only. In these protocols, a group of $n$ parties jointly controls encryption keys under which ciphertexts for the group get encrypted. The secret key material is split among the group and *any* member of the group can request decryption of ciphertexts which is again done jointly by all member. DiSE implicitly – yet crucially – relies on strong authentication to ensure that only valid members of the group can make such requests, whereas we want only a single user to encrypt or decrypt her files from a password. Nevertheless, the authenticity checks in the encryption/decryption process of our protocol are build upon the ideas of the DiSE protocol.

Finally, the PESTO protocol [16] for distributed single sign-on (SSO) relies on a similar idea of first deriving a strong key pair from a distributed OPRF in order to let a user authenticate to a number of servers. The overall application is different though, SSO vs. encryption, and consequently also the desired functionality and security are different. PESTO is in one aspect stronger than DPaSE since it features *proactive security*, meaning that a once corrupted server can be sanitized to be honest again. This strong aspect comes at a cost that is much more critical for the targeted encryption use case than in SSO: PESTO guarantees no security whatsoever when all servers are corrupt. In DPaSE, even in case of a full corruption of all servers, the user's data still remains confidential unless all servers jointly mount a successful offline attack on her password. Thus, a dedicated offline attack for each user (on top of corrupting all servers) would be required, and the encrypted files of users with reasonably strong passwords can remain secure.

We give a detailed comparison of properties of the schemes that are closely related to DPaSE in Table 4.3.

## 4.3. Discussion and Future Work

In this work, we formalize our interpretation of strong security and privacy by introducing the notion of *Distributed Password-Authenticated Symmetric Encryption* (DPaSE). DPaSE uses ciphertext-specific encryption keys, prevents encryption under mistyped passwords, hides users' access pattern, protects against on- and offline attacks on the user password, and maintains all these guarantees even in a concurrent setting with arbitrary other protocols.

We answer the question above in the affirmative, by providing a DPaSE protocol based on a new type of oblivious pseudo-random functions (OPRF). The OPRF is evaluated twice: first, to let the user turn her password into high-entropy authentication data, and second, to let the user compute a password- and ciphertext-dependent symmetric key. We give a construction for such an OPRF, which we believe is of independent interest as a new building block for password-based cryptographic protocols. We provide proof-of-concept implementations for our DPaSE construction (including our OPRF construction) and compare efficiency to related protocols in the literature. Our protocol provides only little overhead over existing solutions for password-based key retrieval/encryption, scales well in the number of users and servers, and features provable security under standard bilinear discrete-log based assumptions in the random oracle model.

An interesting future direction is the construction of a *threshold* version of DPaSE, where only an arbitrary subset of all servers is required to participate in each user request. This would improve usability of the protocol, since users would not have to wait for answers of busy servers. Our user-specific OPRF keys ($osk_i = \mathsf{F}(k, uid)$) hinders us to choose $osk_i$ as shares of standard threshold scheme. However constructing $\mathsf{F}$ as threshold PRF might give an interesting solution towards thresholdization. Finally, security in the presence of malicious servers would be enabled by constructing a maliciously secure extendable distributed partially-oblivious PRF. Alternatively, for ensuring correct encryption it seems to be sufficient to have servers use the same keys in both OPRF evaluations. This flavor of verifiability in our OPRF seems to be achievable with standard techniques. Although key switching between different requests of a specific user would not significantly weaken but clutter the description of our DPaSE functionality, we decide to present the more secure and cleaner version here, and leave the slightly weaker but maliciously secure version as future work.

# 5. Conclusion

In this thesis, we considered decentralized systems in three different settings of blockchain, byzantine agreement and cloud storage. As our contribution, we put forth provable secure infrastructures in each of the above-mentioned settings. In Chapter 2, we investigated blockchain wallets that constitutes a core infrastructural element in blockchains. Although wallets are an indispensable key management scheme for every cryptocurrency user to send/receive payments via blockchain, there has been a significant lack of formal security analysis of wallets. In this thesis, we made progress to close this gap, by designing provably secure wallets from rerandomizable signature schemes. Our contribution is two-fold. First, we analyzed the BIP32 standardization [143] of hierarchical deterministic wallets and were able to provide concrete level of bit security for such wallets. To this end, we put forth the first formalization of deterministic wallets in the hot/cold wallet setting and were able to analyze the exact specification of BIP32 in our security model. In particular, our analysis revealed interesting implications for the BIP32 standard as follows. The original BIP32 construction built from additively rerandomized ECDSA achieves lower bit security than a (slightly) modified version, obtained from multiplicatively rerandomized ECDSA. Second, we extended our security analysis to the post-quantum setting and were able to provide first constructions of post-quantum secure deterministic wallets from a post-quantum variant of rerandomizable signatures.

Our work on deterministic wallets left several interesting open questions. Let us go through some of the most important ones here. In our security analysis of hierarchical deterministic wallets, we considered protection of secret keys through cold wallet storage. However, as was observed for (non-hardened) derivation of keys, secret key leakage from a wallet leads to secret key leakage of all parent wallets. Thus, it is rather important to consider a stronger security model that tolerates secret key leakage from (non-hardened) wallets. Threshold signature schemes [32, 33, 53, 74, 75, 76, 101, 109, 110] provides a solution to achieve such a stronger model of security. Through a threshold signature scheme, it is possible to split the secret key among several (cold wallet) devices and then subsequently use a threshold $t$ of these devices for signing/verifying signatures. For designing deterministic wallets, however we require *rerandomizable* variant of threshold signature schemes. It is not clear whether existing schemes are immediately rerandomizable. It will be even more interesting to explore, whether it is possible to obtain a generic transformation of threshold signature schemes to its rerandomizable variant.

Our analysis on BIP32 wallets mainly covers legacy cryptocurrencies such as Bitcoin and Ethereum. However, wallets are equally relevant in other important cryptocurrency settings such as proof of stake blockchains [5, 96] or privacy-preserving currencies [17, 123]. There has been some prior work that addresses provably secure design of proof of stake wallets [92]. However, their analysis considers signature schemes as idealized cryptographic objects, hence lacks any concrete security parameters for practical wallets. Worth-mentioning is the works on Ring-Confidential Transactions (Ring-CT) [105, 134, 145] that enables privacy-preserving payment mechanism in Monero. However, there still remains a gap in the formalization of wallets in such settings.

Through our analysis of post-quantum secure wallets, we were able to provide concrete transaction throughput based on rerandomizable qTESLA signatures. Our throughput is significantly ($\approx 70$ times) lower than classical ECDSA-based transactsions. Hence there is a lot of room to improve efficiency of signature/key size in case of rerandomizable post-quantum signatures.

In Chapter 3, we considered the problem of byzantine agreement (BA), which is a fundamental problem in distributed computing as well one of the key elements in the design of decentralized systems. Inspired by the setting of blockchains, we explore the question of achieving BA in a resource-constrained model of computation without any PKI assumption. The core idea in this class of BA protocols is to use a computational resource such as proof of work (PoW) to protect against Sybil attacks. Prior works [10, 72, 95] were able to achieve BA in the resource-constrained model of PoW, without setup and run in $O(n\kappa^2)$, $O(\kappa)$, and $O(n)$ rounds, respectively. The natural question was whether it is possible to achieve BA in constant rounds. We answered this question affirmatively, by designing for the first time, an (expected) constant round BA protocol in this setting. At the core of our results, we replace the PoW model with the sequential computational power of a verifiable delay function (VDF) or the VDF model. We considered an adaptive adversary that controls $q$ number of parties and that can compute VDF outputs at a slightly faster speed than honest parties. Similar to the honest majority assumption in case of PoW, we bound the number of VDF evaluations an adversary can compute in parallel, such that over a fixed interval of time $t$, the total number of adversarial evaluations is less than half of the total VDF evaluations. A natural open question would be: whether it is possible to obtain our results of constant round BA in presence of an adversary, that can compute any (polynomial) number of VDFs in parallel. Secondly, all of our results are in the synchronous communication model.

It would be interesting to extend our results to the more realistic model of partial synchrony or asynchronous mode of communication. We were also able to show a lower bound on the communication complexity of our BA protocol. Concretely, we showed that no protocol can obtain BA without a PKI assumption in the VDF model of computation, in less than $O(\sqrt{n})$ number of multicasts. An interesting open question is to investigate whether our results can be extended to the more generic PoW model of computation. However, we can not immediately see the solution, since we would need a different simulation strategy in case of PoW. There is also a lot of room to tighten our existing bound, which is comparitively weaker than bounds in this area (which are quadratic, or of the form $O(n)$ in the multicast model, respectively).

In Chapter 3, we consider the problem of secure cloud storage for end-users. Current solutions are provided by centralized service providers such as OneDrive, Google Drive, Dropbox and so on. To overcome the dependence on centralized services, we aimed to design a usable yet secure encryption mechanism suitable for end-users. To this end, we build our new primitive: Distributed Password-authenticated Symmetric-key Encryption or DPaSE. DPaSE serves as a key management scheme as well as an encryption service, that guarantees the following: ciphertext-specific encryption keys, prevents encryption under mistyped passwords, hides users' access pattern, protects against on- and offline attacks on the user password, and maintains all these guarantees even in a concurrent setting with arbitrary other protocols. As most password-based protocols, DPaSE also relies on an oblivious PRF (OPRF) as the main building block. To instantiate our DPaSE protocol, we need two consecutive OPRF evaluations on correlated blind inputs. To this end, we built a new variant of OPRF, that can be of independent interest. An interesting open question, is to get a *threshold* variant of DPaSE, where only a threshold number of servers need to be available. Because of the inherent structure of our OPRF keys as $\mathsf{F}(K, uid)$, we could not directly use them as standard shares of a threshold scheme. An alternative approach would be to replace $\mathsf{F}(\cdot, \cdot)$ with a threshold PRF variant, without compromising too much on efficiency.

# 6. Bibliography

[1] I. Abraham, T.-H. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi. "Communication Complexity of Byzantine Agreement, Revisited". In: 2019, pp. 317–326. DOI: 10.1145/3293611.3331629.

[2] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren. "Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience". In: 2019, pp. 320–334. DOI: 10.1007/978-3-030-32101-7_20.

[3] A. Aggarwal, M. Movahedi, J. Saia, and M. Zamani. "Bootstrapping Public Blockchains Without a Trusted Setup". In: 2019, pp. 366–368. DOI: 10.1145/3293611.3331570.

[4] S. Agrawal, P. Mohassel, P. Mukherjee, and P. Rindal. "DiSE: Distributed Symmetric-key Encryption". In: 2018, pp. 1993–2010. DOI: 10.1145/3243734.3243774.

[5] *Algorand.* https://www.algorand.com/. 2019.

[6] N. A. Alkadri, P. Das, A. Erwig, S. Faust, J. Krämer, S. Riahi, and P. Struck. "Deterministic Wallets in a Quantum World". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020.* 2020, pp. 1017–1031. DOI: 10.1145/3372297.3423361. URL: https://doi.org/10.1145/3372297.3423361.

[7] N. A. Alkadri, P. Das, A. Erwig, S. Faust, J. Krämer, S. Riahi, and P. Struck. *Deterministic Wallets in a Quantum World.* Cryptology ePrint Archive, Paper 2020/1149. https://eprint.iacr.org/2020/1149. 2020. URL: https://eprint.iacr.org/2020/1149.

[8] E. Alkim, P. S. L. M. Barreto, N. Bindel, J. Krämer, P. Longa, and J. E. Ricardini. "The Lattice-Based Digital Signature Scheme qTESLA". In: *Applied Cryptography and Network Security - 18th International Conference, ACNS 2020.* 2020.

[9] A. Ambainis, M. Hamburg, and D. Unruh. "Quantum Security Proofs Using Semi-classical Oracles". In: *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, 2019.* 2019, pp. 269–295.

[10] M. Andrychowicz and S. Dziembowski. "PoW-Based Distributed Cryptography with No Trusted Setup". In: 2015, pp. 379–399. DOI: 10.1007/978-3-662-48000-7_19.

[11] M. Arapinis, A. Gkaniatsou, D. Karakostas, and A. Kiayias. *A Formal Treatment of Hardware Wallets*. Cryptology ePrint Archive, Report 2019/034. https://eprint.iacr.org/2019/034. 2019.

[12] M. J. Atallah, M. Blanton, N. Fazio, and K. B. Frikken. "Dynamic and Efficient Key Management for Access Hierarchies". In: *ACM Trans. Inf. Syst. Secur.* 3 (2009). DOI: 10.1145/1455526.1455531. URL: https://doi.org/10.1145/1455526.1455531.

[13] *AWS Outage that Broke the Internet Caused by Mistyped Command*. https://www.datacenterknowledge.com/archives/2017/03/02/aws-outage-that-broke-the-internet-caused-by-mistyped-command. 2017.

[14] A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. "Password-protected secret sharing". In: 2011, pp. 433–444. DOI: 10.1145/2046707.2046758.

[15] S. Bai and S. Galbraith. "An improved compression technique for signatures based on learning with errors". In: *Cryptographers' Track at the RSA Conference*. Springer. 2014, pp. 28–47.

[16] C. Baum, T. K. Frederiksen, J. Hesse, A. Lehmann, and A. Yanai. *PESTO: Proactively Secure Distributed Single Sign-On, or How to Trust a Hacked Server*. IEEE European Symposium on Security and Privacy. 2020.

[17] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. 2014, pp. 459–474.

[18] "Bitcoin: A peer-to-peer electronic cash system". In: (2008). URL: http://bitcoin.org/bitcoin.pdf.

[19] *Bitcoin Post-Quantum*. https://bitcoinpq.org/.

[20] *BitTorrent*. http://www.bittorrent.org/. 2015.

[21] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. "Verifiable Delay Functions". In: 2018, pp. 757–788. DOI: 10.1007/978-3-319-96884-1_25.

[22] D. Boneh, M. Drijvers, and G. Neven. *Compact Multi-Signatures for Smaller Blockchains*. Cryptology ePrint Archive, Report 2018/483. https://eprint.iacr.org/2018/483. 2018.

[23] D. Boneh, B. Lynn, and H. Shacham. "Short Signatures from the Weil Pairing". In: *J. Cryptology* 4 (2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9. URL: https://doi.org/10.1007/s00145-004-0314-9.

[24] J. Breitner and N. Heninger. “Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies”. In: *IACR Cryptology ePrint Archive* (2019), p. 23. URL: `https://eprint.iacr.org/2019/023`.

[25] M. Brengel and C. Rossow. “Identifying Key Leakage of Bitcoin Users”. In: *Research in Attacks, Intrusions, and Defenses - 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings.* 2018, pp. 623–643.

[26] J. Brost, C. Egger, R. W. F. Lai, F. Schmid, D. Schröder, and M. Zoppelt. “Threshold Password-Hardened Encryption Services”. In: *CCS ’20: 2020 ACM SIGSAC Conference on Computer and Communications Security.* 2020. URL: `https://doi.org/10.1145/3372297.3417266`.

[27] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. 2018, pp. 315–334.

[28] V. Buterin. *Deterministic Wallets, Their Advantages and their Understated Flaws.* `https://bitcoinmagazine.com/articles/deterministic-wallets-advantages-flaw-1385450276/`. 2013.

[29] V. Buterin. “Ethereum white paper”. In: (2013). URL: `https://ethereum.org/en/whitepaper/`.

[30] J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. “Memento: How to Reconstruct Your Secrets from a Single Password in a Hostile Environment”. In: 2014, pp. 256–275. DOI: `10.1007/978-3-662-44381-1_15`.

[31] R. Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: 2001, pp. 136–145. DOI: `10.1109/SFCS.2001.959888`.

[32] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. “UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts”. In: *CCS ’20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020.* 2020, pp. 1769–1787.

[33] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. “Two-Party ECDSA from Hash Proof Systems and Efficient Instantiations”. In: *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III.* 2019, pp. 191–221.

[34] D. Chaum. “Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms”. In: *Commun. ACM* 2 (1981), pp. 84–88.

[35] D. Chaum, M. Larangeira, M. Yaksetig, and W. Carter. "W-OTS$^+$ Up My Sleeve! A Hidden Secure Fallback for Cryptocurrency Wallets". In: *Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part I.* 2021, pp. 195–219.

[36] M. Christodorescu, S. Gaddam, P. Mukherjee, and R. Sinha. "Amortized Threshold Symmetric-key Encryption". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security.* 2021. URL: `https://doi.org/10.1145/3460120.3485256`.

[37] B. Cohen and K. Pietrzak. *The Chia Network Blockchain.* Tech. rep. Chia Network, 2019.

[38] Comparitech. *Worldwide cryptocurrency heists tracker (updated daily).* `https://www.comparitech.com/crypto/biggest-cryptocurrency-heists/`. Updated daily.

[39] J.-S. Coron. "Optimal Security Proofs for PSS and Other Signature Schemes". In: 2002, pp. 272–287. DOI: `10.1007/3-540-46035-7_18`.

[40] N. T. Courtois, P. Emirdag, and F. Valsorda. "Private Key Recovery Combination Attacks: On Extreme Fragility of Popular Bitcoin Key Management, Wallet and Cold Storage Solutions in Presence of Poor RNG Events". In: *IACR Cryptology ePrint Archive* (2014), p. 848.

[41] P. Das, L. Eckey, S. Faust, J. Loss, and M. Maitra. *Round Efficient Byzantine Agreement from VDFs.* Cryptology ePrint Archive, Paper 2022/823. `https://eprint.iacr.org/2022/823`. 2022.

[42] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A. Sadeghi. "FastKitten: Practical Smart Contracts on Bitcoin". In: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019.* 2019, pp. 801–818.

[43] P. Das, A. Erwig, S. Faust, J. Loss, and S. Riahi. "The Exact Security of BIP32 Wallets". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communication Security, Virtual Event, Republic of Korea, November 15 - 19, 2021.* 2021, pp. 1020–1042. DOI: `10.1145/3460120.3484807`. URL: `https://doi.org/10.1145/3460120.3484807`.

[44] P. Das, A. Erwig, S. Faust, J. Loss, and S. Riahi. *The Exact Security of BIP32 Wallets.* Cryptology ePrint Archive, Paper 2021/1287. `https://eprint.iacr.org/2021/1287`. 2021. URL: `https://eprint.iacr.org/2021/1287`.

[45] P. Das, S. Faust, and J. Loss. "A Formal Treatment of Deterministic Wallets". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. 2019, pp. 651–668. DOI: 10.1145/3319535.3354236. URL: https://doi.org/10.1145/3319535.3354236.

[46] P. Das, S. Faust, and J. Loss. *A Formal Treatment of Deterministic Wallets*. Cryptology ePrint Archive, Paper 2019/698. https://eprint.iacr.org/2019/698. 2019. URL: https://eprint.iacr.org/2019/698.

[47] P. Das, J. Hesse, and A. Lehmann. *DPaSE: Distributed Password-Authenticated Symmetric Encryption*. Cryptology ePrint Archive, Paper 2020/1443. https://eprint.iacr.org/2020/1443. 2020. URL: https://eprint.iacr.org/2020/1443.

[48] P. Das, J. Hesse, and A. Lehmann. "DPaSE: Distributed Password-Authenticated Symmetric-Key Encryption, or How to Get Many Keys from One Password". In: *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*. 2022, pp. 682–696. DOI: 10.1145/3488932.3517389. URL: https://doi.org/10.1145/3488932.3517389.

[49] P. Das, D. B. Roy, and D. Mukhopadhyay. "Automatic Generation of HCCA Resistant Scalar Multiplication Algorithm by Proper Sequencing of Field Multiplier Operands". In: *PROOFS 2017, 6th International Workshop on Security Proofs for Embedded Systems, Taipei, Taiwan, September 29th, 2017*. 2017, pp. 33–49.

[50] P. Das, D. B. Roy, and D. Mukhopadhyay. "Automatic generation of HCCA-resistant scalar multiplication algorithm by proper sequencing of field multiplier operands". In: *J. Cryptogr. Eng.* 3 (2019), pp. 263–275.

[51] P. Das, D. B. Roy, and D. Mukhopadhyay. "Improved Atomicity to Prevent HCCA on NIST Curves". In: *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, AsiaPKC@AsiaCCS, Xi'an, China, May 30 - June 03, 2016*. 2016, pp. 21–30.

[52] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium (USENIX Security 04)*. San Diego, CA, 2004. URL: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router.

[53] J. Doerner, Y. Kondi, E. Lee, and A. Shelat. "Secure Two-party Threshold ECDSA from ECDSA Assumptions". In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. 2018, pp. 980–997.

[54] D. Dolev and H. R. Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM Journal on Computing* 4 (1983), pp. 656–666.

[55] J. Don, S. Fehr, C. Majenz, and C. Schaffner. "Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model". In: 2019, pp. 356–383. DOI: `10.1007/978-3-030-26951-7_13`.

[56] J. R. Douceur. "The sybil attack". In: *International Workshop on Peer-to-Peer Systems.* Springer. 2002, pp. 251–260.

[57] *DPaSE PoC Implementation.* `https://gitlab.com/DPaSEcode/dpase-submission-code`.

[58] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. "Lattice signatures and bimodal Gaussians". In: *Advances in Cryptology–CRYPTO 2013.* 2013, pp. 40–56.

[59] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme". In: *Transactions on Cryptographic Hardware and Embedded Systems - TCHES 2018* 1 (2018), pp. 238–268.

[60] M. F. Esgin, R. K. Zhao, R. Steinfeld, J. K. Liu, and D. Liu. "MatRiCT: Efficient, Scalable and Post-Quantum Blockchain Confidential Transactions Protocol". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* 2019, pp. 567–584.

[61] A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart. "The Pythia PRF Service". In: 2015, pp. 547–562.

[62] C. Fan, Y. Tseng, H. Su, R. Hsu, and H. Kikuchi. "Secure Hierarchical Bitcoin Wallet Scheme Against Privilege Escalation Attacks". In: *IEEE Conference on Dependable and Secure Computing, DSC 2018.* 2018, pp. 1–8.

[63] P. Feldman and S. Micali. "Optimal Algorithms for Byzantine Agreement". In: 1988, pp. 148–161. DOI: `10.1145/62212.62225`.

[64] L. D. Feo, S. Masson, C. Petit, and A. Sanso. "Verifiable Delay Functions from Supersingular Isogenies and Pairings". In: *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I.* 2019, pp. 248–277.

[65] M. Fersch, E. Kiltz, and B. Poettering. "On the One-Per-Message Unforgeability of (EC)DSA and Its Variants". In: *Theory of Cryptography.* Cham, 2017, pp. 519–534.

[66] M. Fersch, E. Kiltz, and B. Poettering. "On the Provable Security of (EC)DSA Signatures". In: 2016, pp. 1651–1662. DOI: `10.1145/2976749.2978413`.

[67] M. Fischlin and N. Fleischhacker. "Limitations of the Meta-reduction Technique: The Case of Schnorr Signatures". In: *Advances in Cryptology - EUROCRYPT 2013.* 2013, pp. 444–460.

[68] N. Fleischhacker, J. Krupp, G. Malavolta, J. Schneider, D. Schröder, and M. Simkin. "Efficient Unlinkable Sanitizable Signatures from Signatures with Re-randomizable Keys". In: 2016, pp. 301–330. DOI: 10.1007/978-3-662-49384-7_12.

[69] C. Ganesh, Y. Kondi, A. Patra, and P. Sarkar. "Efficient Adaptively Secure Zero-Knowledge from Garbled Circuits". In: 2018, pp. 499–529. DOI: 10.1007/978-3-319-76581-5_17.

[70] J. Garay, A. Kiayias, and N. Leonardos. *The Bitcoin Backbone Protocol: Analysis and Applications.* Cryptology ePrint Archive, Paper 2014/765. https://eprint.iacr.org/2014/765. 2014. URL: https://eprint.iacr.org/2014/765.

[71] J. A. Garay, A. Kiayias, and N. Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10.

[72] J. A. Garay, A. Kiayias, N. Leonardos, and G. Panagiotakos. "Bootstrapping the Blockchain, with Applications to Consensus and Fast PKI Setup". In: 2018, pp. 465–495. DOI: 10.1007/978-3-319-76581-5_16.

[73] J. A. Garay, A. Kiayias, R. M. Ostrovsky, G. Panagiotakos, and V. Zikas. "Resource-Restricted Cryptography: Revisiting MPC Bounds in the Proof-of-Work Era". In: *Advances in Cryptology - EUROCRYPT 2020, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II.* 2020, pp. 129–158.

[74] F. Garillot, Y. Kondi, P. Mohassel, and V. Nikolaenko. "Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions". In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I.* 2021, pp. 127–156.

[75] R. Gennaro and S. Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018.* 2018, pp. 1179–1194.

[76] R. Gennaro, S. Goldfeder, and A. Narayanan. "Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security". In: *Applied Cryptography and Network Security - ACNS 2016.* 2016, pp. 156–174.

[77] J. Groth. "Non-interactive distributed key generation and key resharing". In: *IACR Cryptol. ePrint Arch.* (2021), p. 339. URL: https://eprint.iacr.org/2021/339.

[78] J. Groth and V. Shoup. *Design and analysis of a distributed ECDSA signing service.* Cryptology ePrint Archive, Paper 2022/506. https://eprint.iacr.org/2022/506. 2022. URL: https://eprint.iacr.org/2022/506.

[79] J. Groth and V. Shoup. "On the Security of ECDSA with Additive Key Derivation and Presignatures". In: *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I.* 2022, pp. 365–396.

[80] G. Gutoski and D. Stebila. "Hierarchical Deterministic Bitcoin Wallets that Tolerate Key Leakage". In: 2015, pp. 497–504. DOI: `10.1007/978-3-662-47854-7_31`.

[81] G. Gutoski and D. Stebila. "Hierarchical Deterministic Bitcoin Wallets that Tolerate Key Leakage". In: *Financial Cryptography and Data Security - 19th International Conference, FC 2015.* 2015, pp. 497–504.

[82] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. "Proactive Secret Sharing Or: How to Cope With Perpetual Leakage". In: *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings.* 1995, pp. 339–352.

[83] A. Hülsing. "WOTS+ - Shorter Signatures for Hash-Based Signature Schemes". In: *IACR Cryptol. ePrint Arch.* (2017), p. 965.

[84] *Internet Identity: The End of Usernames and Passwords.* `https://tinyurl.com/6rrhvzr2`.

[85] M. S. Islam, M. Kuzu, and M. Kantarcioglu. "Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation". In: 2012.

[86] S. Jarecki, A. Kiayias, and H. Krawczyk. "Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model". In: 2014, pp. 233–253. DOI: `10.1007/978-3-662-45608-8_13`.

[87] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. "Highly-Efficient and Composable Password-Protected Secret Sharing (Or: How to Protect Your Bitcoin Wallet Online)". In: *IEEE European Symposium on Security and Privacy, EuroS&P.* 2016.

[88] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. "TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF". In: 2017, pp. 39–58. DOI: `10.1007/978-3-319-61204-1_3`.

[89] S. Jarecki, H. Krawczyk, and J. K. Resch. "Updatable Oblivious Key Management for Storage Systems". In: 2019, pp. 379–393. DOI: `10.1145/3319535.3363196`.

[90] S. Jarecki, H. Krawczyk, and J. Xu. "OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-computation Attacks". In: 2018, pp. 456–486. DOI: `10.1007/978-3-319-78372-7_15`.

[91] D. Johnson and A. Menezes. *The Elliptic Curve Digital Signature Algorithm (ECDSA).* Tech. rep. 1999.

[92] D. Karakostas, A. Kiayias, and M. Larangeira. “Account Management in Proof of Stake Ledgers”. In: *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*. 2020, pp. 3–23.

[93] J. Katz and C.-Y. Koo. “On Expected Constant-Round Protocols for Byzantine Agreement”. In: 2006, pp. 445–462. DOI: `10.1007/11818175_27`.

[94] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. 2014. URL: `https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269`.

[95] J. Katz, A. Miller, and E. Shi. *Pseudonymous Broadcast and Secure Computation from Cryptographic Puzzles*. Cryptology ePrint Archive, Report 2014/857. `https://eprint.iacr.org/2014/857`. 2014.

[96] T. Kerber, A. Kiayias, M. Kohlweiss, and V. Zikas. “Ouroboros Crypsinous: Privacy-Preserving Proof-of-Stake”. In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. 2019, pp. 157–174.

[97] E. Kiltz, V. Lyubashevsky, and C. Schaffner. “A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model”. In: *Advances in Cryptology–EUROCRYPT 2018*. 2018, pp. 552–586.

[98] E. Kiltz, D. Masny, and J. Pan. “Optimal Security Proofs for Signatures from Identification Schemes”. In: *Advances in Cryptology - CRYPTO 2016, Part II*. 2016, pp. 33–61.

[99] M. Klooß, A. Lehmann, and A. Rupp. “(R)CCA Secure Updatable Encryption with Integrity Protection”. In: 2019, pp. 68–99. DOI: `10.1007/978-3-030-17653-2_3`.

[100] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. “Efficient Batched Oblivious PRF with Applications to Private Set Intersection”. In: 2016, pp. 818–829. DOI: `10.1145/2976749.2978381`.

[101] C. Komlo and I. Goldberg. “FROST: Flexible Round-Optimized Schnorr Threshold Signatures”. In: *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*. 2020, pp. 34–65.

[102] Y. Kondi, B. Magri, C. Orlandi, and O. Shlomovits. “Refresh When You Wake Up: Proactive Threshold Wallets with Offline Devices”. In: *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. 2021, pp. 608–625.

[103] M.-S. Lacharité, B. Minaud, and K. G. Paterson. “Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage”. In: 2018, pp. 297–314. DOI: `10.1109/SP.2018.00002`.

[104] R. W. F. Lai, C. Egger, M. Reinert, S. S. M. Chow, M. Maffei, and D. Schröder. "Simple Password-Hardened Encryption Services". In: *27th USENIX Security Symposium, USENIX Security.* 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/lai.

[105] R. W. F. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang. "Omniring: Scaling Private Payments Without Trusted Setup". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019.* 2019, pp. 31–48.

[106] *Ledger Support,Ledger Nano OS.* https://support.ledger.com/hc/en-us/articles/115005297709-Export-your-accounts. 2014.

[107] A. Lehmann. "ScrambleDB: Oblivious (Chameleon) Pseudonymization-as-a-Service". In: *Proc. Priv. Enhancing Technol.* (2019). URL: https://doi.org/10.2478/popets-2019-0048.

[108] A. Lehmann and B. Tackmann. "Updatable Encryption with Post-Compromise Security". In: 2018, pp. 685–716. DOI: 10.1007/978-3-319-78372-7_22.

[109] Y. Lindell. "Fast Secure Two-Party ECDSA Signing". In: 2017, pp. 613–644. DOI: 10.1007/978-3-319-63715-0_21.

[110] Y. Lindell and A. Nof. "Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018.* 2018, pp. 1837–1854.

[111] Q. Liu and M. Zhandry. "Revisiting Post-quantum Fiat-Shamir". In: 2019, pp. 326–355. DOI: 10.1007/978-3-030-26951-7_12.

[112] A. D. Luzio, D. Francati, and G. Ateniese. "Arcula: A Secure Hierarchical Deterministic Wallet for Multi-asset Blockchains". In: 2020, pp. 323–343. DOI: 10.1007/978-3-030-65411-5_16.

[113] V. Lyubashevsky. "Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures". In: *Advances in Cryptology–ASIACRYPT 2009.* 2009, pp. 598–616.

[114] A. Marcedone, R. Pass, and abhi shelat. *Minimizing Trust in Hardware Wallets with Two Factor Signatures.* Cryptology ePrint Archive, Report 2019/006. https://eprint.iacr.org/2019/006. 2019.

[115] G. Maxwell and I. Bentov. *Deterministic Wallets.* https://www.cs.cornell.edu/~iddo/detwal.pdf. 2018.

[116] Mediawiki. *BIP32 Specification.* https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki. 2018.

[117] *MEGA: Secure Cloud Storage and Communication Privacy by Design.* `https://mega.nz/`.

[118] S. Micali. "Very Simple and Efficient Byzantine Agreement". In: 2017, 6:1–6:1. DOI: `10.4230/LIPIcs.ITCS.2017.6`.

[119] S. Micali and V. Vaikuntanathan. "Optimal and player-replaceable consensus with an honest majority". In: (2017).

[120] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* `https://bitcoin.org/bitcoin.pdf`. 2008.

[121] J. Nick, T. Ruffing, and Y. Seurin. "MuSig2: Simple Two-Round Schnorr Multi-signatures". In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I.* 2021, pp. 189–221.

[122] S. Noether. *Ring Signature Confidential Transactions for Monero.* Cryptology ePrint Archive, Report 2015/1098. `https://eprint.iacr.org/2015/1098`. 2015.

[123] S. Noether and A. Mackenzie. "Ring Confidential Transactions". In: *Ledger* (2016), pp. 1–18.

[124] R. Pass, L. Seeman, and a. shelat. "Analysis of the Blockchain Protocol in Asynchronous Networks". In: 2017, pp. 643–673. DOI: `10.1007/978-3-319-56614-6_22`.

[125] J. N. Pieter Wuille and T. Ruffings. *Schnorr Signatures for secp256k1.* `https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki`. 2020-01-19.

[126] J. N. Pieter Wuille and A. Towns. *Taproot: SegWit version 1 spending rules.* `https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki`. 2020-01-19.

[127] K. Pietrzak. "Proofs of Catalytic Space". In: 2019, 59:1–59:25. DOI: `10.4230/LIPIcs.ITCS.2019.59`.

[128] *Quantum Resistant Ledger (QRL).* `https://github.com/theQRL/Whitepaper/blob/master/QRL_whitepaper.pdf`.

[129] M. O. Rabin. "Randomized Byzantine Generals". In: 1983, pp. 403–409. DOI: `10.1109/SFCS.1983.48`.

[130] D. B. Roy, P. Das, and D. Mukhopadhyay. "ECC on Your Fingertips: A Single Instruction Approach for Lightweight ECC Design in GF(p)". In: *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers.* 2015, pp. 161–177.

[131] C. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings.* 1989, pp. 239–252.

[132] P. W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: 1994, pp. 124–134. DOI: `10.1109/SFCS.1994.365700`.

[133] R. Skellern. *Cryptocurrency Hacks: More Than $2b USD lost between 2011-2018.* `https://medium.com/ecomi/cryptocurrency-hacks-more-than-2b-usd-lost-between-2011-2018_-67054b342219`. 2018.

[134] S. Sun, M. H. Au, J. K. Liu, and T. H. Yuen. "RingCT 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero". In: *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II.* 2017, pp. 456–474.

[135] *Tresorit: Cloud Storage + End-to-end Encryption.* `https://tresorit.com/security/encryption`.

[136] *Trezor Wiki,Cryptocurrency standards,Hierachical deterministic wallets.* `https://wiki.trezor.io/Cryptocurrency_standards`. 2014.

[137] C. Troncoso, G. Danezis, M. Isaakidis, and H. Halpin. "Systematizing Decentralization and Privacy: Lessons from 15 years of research and deployments". In: *CoRR* (2017). URL: `http://arxiv.org/abs/1704.08065`.

[138] M. Turuani, T. Voegtlin, and M. Rusinowitch. "Automated Verification of Electrum Wallet". In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC.* 2016, pp. 27–42.

[139] D. Unruh. "Post-quantum Security of Fiat-Shamir". In: 2017, pp. 65–95. DOI: `10.1007/978-3-319-70694-8_3`.

[140] J. Wan, H. Xiao, S. Devadas, and E. Shi. "Round-Efficient Byzantine Broadcast Under Strongly Adaptive and Majority Corruptions". In: *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I.* 2020, pp. 412–456.

[141] X. Wang and B. Huson. "Robust distributed symmetric-key encryption". In: *IACR ePrint* (2020).

[142] B. Wesolowski. "Efficient Verifiable Delay Functions". In: 2019, pp. 379–407. DOI: `10.1007/978-3-030-17659-4_13`.

[143] B. Wiki. *BIP32 proposal.* `https://en.bitcoin.it/wiki/BIP_0032`. 2018.

[144] B. Wiki. *Genesis Block.* `https://en.bitcoin.it/wiki/Genesis_block`.

[145] T. H. Yuen, S. Sun, J. K. Liu, M. H. Au, M. F. Esgin, Q. Zhang, and D. Gu. “RingCT 3.0 for Blockchain Confidential Transaction: Shorter Size and Stronger Security”. In: *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers.* 2020, pp. 464–483.

[146] M. Zhandry. “How to Construct Quantum Random Functions”. In: 2012, pp. 679–687. DOI: 10.1109/FOCS.2012.37.

[147] Z. Zhang, Y. Chen, S. S. M. Chow, G. Hanaoka, Z. Cao, and Y. Zhao. “Black-Box Separations of Hash-and-Sign Signatures in the Non-Programmable Random Oracle Model”. In: *Provable Security - 9th International Conference, ProvSec 2015.* 2015, pp. 435–454.

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **RKA** | Related Key Attack |
| **BIP32** | Bitcoin Improvement Proposal 32 |
| **UFCMA** | Existential Unforgeability Under Chosen Message Attack |
| **UFCMA1** | One-Per Message Existential Unforgeability Under Chosen Message Attack |
| **UFCMA-RK** | Unforgeability Under Rerandomizable Keys |
| **UFCMA-HRK** | Unforgeability under Honestly Rerandomizable Keys |
| **RKA** | Related Key Attack |
| **PKI** | Public Key Infrastructure |
| **VDF** | Verifiable Delay Function |
| **BA** | Byzantine Agreement |
| **GPKI** | Graded PKI |
| **UC** | Universally Composable |
| **DPaSE** | Distributed Password-authenticated Symmetric-key Encryption |
| **OPRF** | Oblivious PRF |
| **edpOPRF** | Extendable Distributed Partially-Oblivious PRF |

# A. A Formal Treatment of Deterministic Wallets

This chapter corresponds to our published article in CCS 2019 [45], with minor edits. Our full version can be found in [46].

[45] P. Das, S. Faust, and J. Loss. “A Formal Treatment of Deterministic Wallets”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. 2019, pp. 651–668. DOI: 10.1145/3319535.3354236. URL: https://doi.org/10.1145/3319535.3354236.

# A Formal Treatment of Deterministic Wallets

Poulami Das*[1] Sebastian Faust†[1] Julian Loss‡ [2]

[1] TU Darmstadt, Germany

[2] University of Maryland, USA

**Abstract**

In cryptocurrencies such as Bitcoin or Ethereum, users control funds via secret keys. To transfer funds from one user to another, the owner of the money signs a new transaction that transfers the funds to the new recipient. This makes secret keys a highly attractive target for attacks, and has lead to prominent examples where millions of dollars worth in cryptocurrency have been stolen. To protect against these attacks, a widely used approach are so-called hot/cold wallets. In a hot/cold wallet system, the hot wallet is permanently connected to the network, while the cold wallet stores the secret key and is kept without network connection. In this work, we propose the first comprehensive security model for hot/cold wallets and develop wallet schemes that are provable secure within these models. At the technical level our main contribution is to provide a new provably secure ECDSA-based hot/cold wallet scheme that can be integrated into legacy cryptocurrencies such as Bitcoin. Our construction and security analysis uses a modular approach, where we show how to generically build secure hot/cold wallets from signature schemes that exhibit a rerandomizing property of the keys.

**Keywords:** Wallets; cryptocurrencies; foundations

## 1 Introduction

In decentralized cryptocurrencies such as Bitcoin or Ethereum, the money mechanics (e.g., who owns what and how money is transferred) are controlled by a network of miners. To this end, the miners agree via a consensus protocol about the current balance that each party has in the system. Changes to these balances are validated by the miners according to well-specified rules. In most cryptocurrencies, balance updates are executed

*Email: poulami.das@crisp-da.de

†Email: sebastian.faust@cs.tu-darmstadt.de

‡Email: julian.loss@ruhr-uni-bochum.de. Work done while author was at Ruhr-University Bochum, Germany.

via *transactions*. A transaction transfers money between *addresses*, which is the digital identity of a party and technically is represented by a public key of a digital signature scheme.[1] For better illustration, consider the example where Alice wants to send some of her coins – say 1 $\mathtt{BTC}$ – from her address $pk_{\mathtt{A}}$ to Bob's address $pk_{\mathtt{B}}$. To this end, she creates a transaction $\mathtt{tx}_{\mathtt{AB}}$ that informally says: "Transfer 1 $\mathtt{BTC}$ from $pk_{\mathtt{A}}$ to $pk_{\mathtt{B}}$". To ensure that only Alice can send her coins to Bob, we require that $\mathtt{tx}_{\mathtt{AB}}$ is accompanied by a valid signature of $\mathsf{H}(\mathtt{tx}_{\mathtt{AB}})$. Since only the owner of the corresponding $sk_{\mathtt{A}}$ – here Alice – can produce a valid signature, control over $sk_{\mathtt{A}}$ implies full control over the funds assigned to $pk_{\mathtt{A}}$. This makes secret keys a highly attractive target for attacks. Unsurprisingly, there are countless examples of spectacular hacks where the attacker was able to steal millions of dollars by breaking into a system and extracting the secret key [Ske18, Blo18]. According to the cryptocurrency research firm CipherTrace, in 2018 alone, attackers managed to steal more than USD 1 billion worth in cryptocurrency [Bit18].

One reason for many of these attacks is that large amounts of funds are often controlled by so-called *hot wallets*. A hot wallet is a piece of software that runs on a computer or a smart phone and has a direct connection to the Internet. This make hot wallets very convenient to use since they can move funds around easily. On the downside, however, their permanent Internet connection often makes them an easy target for attackers, e.g., by exploiting software vulnerabilities via malware or phishing. Thus, it is generally recommended to store only a small amount of cryptocurrency on a hot wallet, while larger amounts of money should be transferred to a *cold wallet*. A cold wallet stays disconnected from the network most of the time and may in practice be realized by a dedicated hardware device [Wik18b], or by a paper wallet where the secret key is printed on paper and stored in a secure place.

A simple way to construct a hot/cold wallet is to generate a key pair $(pk_{\mathsf{cold}}, sk_{\mathsf{cold}})$ and store the secret key $sk_{\mathsf{cold}}$ on the cold wallet, while the corresponding public key $pk_{\mathsf{cold}}$ is kept on the hot wallet (or published over the Internet). A user can then directly transfer money to the cold wallet by publishing a transaction on the blockchain that sends money to $pk_{\mathsf{cold}}$. As long as the owner of the cold wallet does not want to spend its funds, the cold wallet never needs to come online. This naive approach has one important drawback. Since all transactions targeting the cold wallet send money to the *same* public key $pk_{\mathsf{cold}}$, the cold wallet may accumulate, over time, a large amount of money. Moreover, all transactions are publicly recorded on the blockchain, and thus $pk_{\mathsf{cold}}$ becomes an attractive target for an attack the next time the wallet goes online (which will happen at the latest when the owner of the wallet wants to spend its coins).

To mitigate this attack, it is common practice in the cryptocurrency community to use each key pair only for a single transaction. Hence, we may generate a "large number" of fresh key pairs $(sk_1, pk_1), \dots, (sk_\ell, pk_\ell)$. Then, the $\ell$ public keys are sent to the hot wallet, while the corresponding $\ell$ secret keys $sk_i$ are kept on the cold wallet. While this approach keeps individual transactions unlinkable, it only works for an a-priori fixed number of transactions, and requires storage on the hot/cold wallet that grows linearly with $\ell$.

Fortunately, in popular cryptocurrencies such as Bitcoin, these two shortcomings can be solved by exploiting the algebraic structure of the underlying signature scheme (e.g., the ECDSA signature scheme in Bitcoin). In the cryptocurrency literature, this approach

[1]To be more precise, in Bitcoin funds are assigned to the hash of a public key, and not to the public key itself.

is often called *deterministic wallets* [But13] and is standardized in the BIP32 improvement proposal [Wik18a].[2] At a high level, a deterministic wallet consists of a *master secret key msk* together with a matching *master public key mpk* and a deterministic *key derivation procedure*. At setup, the master public key is given to the hot wallet, whereas the master secret key is kept on the cold wallet. After setup, the hot and cold wallet can independently generate matching session keys using the key derivation procedure and their respective master keys. Using this approach, we only need to store a single (master) key on the hot/cold wallet in order to generate an arbitrary number of (one-time) session keys.

Informally, a deterministic wallet should offer two main security guarantees. First, an *unforgeability property*, which ensures that as long as the cold wallet is not compromised, signatures to authenticate new transactions can not be forged, and thus funds are safe. Second, an *unlinkability* property, which guarantees that public keys generated from the same master public key *mpk* are computationally indistinguishable from freshly generated public keys. Despite the widespread use of deterministic wallets (e.g., they are used in most hardware wallets such as *ledger* or TREZOR, and by common software wallets such as Jaxx), only limited formal security analysis of these schemes has been provided (we will discuss the related work in Section 1.3). The main contribution of our work is to close this gap.

## 1.1 Deterministic hot/cold wallets

Before we outline our contribution, we recall (a slightly simplified version of) the BIP32 wallet construction as used by popular cryptocurrencies. We emphasize that for ease of presentation, we abstract from some of the technical details of the BIP32 scheme. In particular, we focus in this work on the (conceptually cleaner) deterministic wallets ignoring the "hierarchical" component of BIP32 (see [Med18] for a full specification). We leave it as an important open problem to also develop a formal model for hierarchical wallets (see Section 7 for a more detailed discussion). In the following description we focus on ECDSA-based wallets as ECDSA is the underlying signature scheme used by most popular cryptocurrencies.

Let $G$ denote the base point of an ECDSA elliptic curve. The deterministic ECDSA wallet uses an ECDSA key tuple as its master secret/public key pair, denoted by $(msk = x, mpk = x \cdot G)$. The master secret key $msk$ is stored on the cold wallet, while the corresponding master public key $mpk$ is kept on the corresponding hot wallet. In addition, the hot wallet and the cold wallet both keep a common secret string $ch$ which is called the "chaincode". To derive a new session public key with identifier $ID$, the hot wallet computes $w \leftarrow \mathsf{H}(ch, ID), pk_{ID} \leftarrow mpk + w \cdot G$ and the cold wallet computes the corresponding session secret key as $w \leftarrow \mathsf{H}(ch, ID), sk_{ID} \leftarrow msk + w$. As argued, e.g., in [MB18], this construction satisfies both unlinkability and unforgeability as long as the chaincode and all derived secret keys remain hidden from the adversary.

Unfortunately, hot wallet breaches happen frequently, and hence the assumption that the chaincode stays secret is rather unrealistic. When $ch$ is revealed, however, the unlinkability property is trivially broken since the adversary can derive from $mpk$ and $ch$ the corresponding session public key $pk_{ID}$ for any $ID$ of its choice. Even worse, as we

[2]BIP32 stands for Bitcoin improvement proposal. The same approach is also used for other cryptocurrencies such as Ethereum or Dash.

discuss in Section 4.1.2 (and as already suggested in [MB18]), a hot wallet security breach may in certain cases even break the unforgeability property of the wallet scheme.

## 1.2 Our contributions

At the conceptual level, our main contribution is to introduce a formal comprehensive security model to analyze hot/cold wallets. On the other hand, at the technical level, we design a new ECDSA-based wallet scheme and prove its security within our model. The latter is achieved using a modular approach, which shows that signature schemes exhibiting certain rerandomizability properties for the key suffice to securely instantiate wallets in our model. Further details are provided below.

Security model for wallets. As our first contribution we provide a formal security model that precisely captures the security properties that a hot/cold wallet should satisfy. In particular, we incorporate into our model hot wallet security breaches, access to derived public keys and corresponding signatures that may appear on the blockchain. More concretely, let $\mathsf{SWal} = (\mathsf{SWal.MGen}, \mathsf{SWal.SKDer}, \mathsf{SWal.PKDer}, \mathsf{SWal.Sign}, \mathsf{SWal.Verify})$ be a wallet scheme, where $\mathsf{SWal.MGen}$ denotes the master key generation algorithm, ($\mathsf{SWal.SKDer}$, $\mathsf{SWal.PKDer}$) are used for deriving session keys and $(\mathsf{SWal.Sign}, \mathsf{SWal.Verify})$ represent the signing and verification algorithms of the underlying signature scheme. The security of $\mathsf{SWal}$ is defined via two game-based security notions that we call *wallet unlinkability* and *wallet unforgeability.*

Our notion of unlinkability can informally be described as a form of forward security – similar in spirit to key exchange models for analyzing TLS. It guarantees that all money that was sent to session public keys $pk_{ID} \leftarrow \mathsf{SWal.PKDer}(mpk, ch, ID)$ derived *prior* to the hot wallet breach, can not be linked to $mpk$. Notably, our unlinkability property even holds against an adversary that sees a polynomial number of session public keys generated from $mpk$ and signatures for adversarially chosen messages. On the other hand, our unforgeability notion considers a natural threat model where funds on the cold wallet remain secure even if the hot wallet is fully compromised. While at first sight it may seem that achieving unforgeability in such a setting is straightforward, it turns out that in particular for ECDSA-based wallets, we have to deal with several technical challenges. The main reason for this is that once the hot wallet is breached, the session public keys are not fresh anymore (i.e., all session public keys are now related to the master public key $mpk$). This hinders a straightforward reduction to the security of the underlying signature scheme used by the cryptocurrency. Even worse, we argue that for certain naive instantiations of wallet schemes, wallet unforgeability can be broken and an adversary may steal money from the cold wallet *without* ever breaking into it.

Stateful deterministic wallets. In order to achieve our security definition of forward unlinkability, we consider the natural notion of *stateful deterministic wallets.* In a stateful wallet, the hot and cold wallet share a common secret state $St$ that is (deterministically) updated for every new session key pair. More concretely, the master key generation algorithm $\mathsf{SWal.MGen}$ outputs (together with the master key pair $(mpk, msk)$) an initial state $St_0$ that will be stored on both the hot and the cold wallet. Then, to derive new session keys, the secret/public key derivation algorithms $\mathsf{SWal.SKDer}$ and $\mathsf{SWal.PKDer}$ take as input additionally the current state $St_{i-1}$ and output the new state $St_i$, while the old state $St_{i-1}$ is erased from the hot/cold wallet. The update mechanism for deriving the new state has to guarantee that $St_i$ looks random even if future states $St_j$ (for $j > i$) are revealed. Together with a mechanism for deriving new session key pairs, our scheme achieves the strong aforementioned notion of forward unlinkability. We note that while state updates (together with secure erasures) are needed to achieve our new notion of forward unlinkability, our notion of unforgeability might also be achievable by some of the currently used (stateless) wallet schemes.

Modular approach for provably secure wallets. To securely instantiate our stateful deterministic wallets, we provide a modular approach that uses digital signature schemes with rerandomizable keys. This notion – originally due to Fleischhacker et al. [FKM+16] – extends standard digital signature schemes with two additional algorithms: $\mathsf{RandSK}$ and $\mathsf{RandPK}$. These algorithms take as input a secret key $sk$, respectively public key $pk$, and some randomness $\rho$ and output fresh keys $sk'$, respectively $pk'$. Besides the standard unforgeability property, signatures with rerandomizable keys guarantee that the key pair $(sk', pk')$ is fresh and independent of the original keys $(sk, pk)$ from which they were generated.

Given a secure signature scheme with rerandomizable keys, we show how to generically instantiate our wallet scheme as follows. Let $St$ be the current state of the hot/cold wallet. The public key derivation algorithm $\mathsf{SWal.PKDer}(mpk, St, ID)$ first computes $(\omega_{ID}, St') = \mathsf{H}(St, ID)$. Then, it derives the new session public key $pk_{ID}$ by running the public key rerandomizing algorithm $\mathsf{RandPK}$ via $pk_{ID} \leftarrow \mathsf{RandPK}(mpk, \omega_{ID})$, and erases the old state $St$. Analogously, the cold wallet can compute $sk_{ID}$ by computing $\omega_{ID}$ as above and calling $sk_{ID} \leftarrow \mathsf{RandSK}(msk, \omega_{ID})$. If $\mathsf{H}$ is modeled as a random oracle that maps to the randomness space for rerandomizing keys, then the rerandomizability property mentioned above satisfies that our wallet construction achieves forward unlinkability. On the other hand, wallet unforgeability follows from the unforgeability of the underlying signature scheme. For the latter to go through, we rely on the special `RSign` oracle that is provided in the unforgebaility game of signatures with rerandomizable keys (see below). Besides its strong security guarantees, our generic wallet construction preserves the storage efficiency of the BIP32 standard and only requires one hash computation more per hot/cold wallet for every derived session key pair.

Of course, before we can use our wallet scheme in practice, we need to build signatures with rerandomizable keys from standard (practical) signature schemes ideally used by cryptocurrencies. As shown in [FKM+16] the Schnorr signature scheme [Sch89] satisfies these properties. In addition, we show that also BLS signatures [BLS04] can be used to construct signatures with rerandomizable keys. Thus, these schemes are natural candidates for our wallet construction.

Provably secure ECDSA-based wallets. While many cryptocurrencies plan to use Schnorr and BLS signatures in the future, to date almost all legacy cryptocurrencies (e.g., Bitcoin or Ethereum) rely on the ECDSA signature scheme. The main technical contribution of our work is thus to propose the first provably secure construction of stateful deterministic wallets that work together with ECDSA-based cryptocurrencies such as Bitcoin. To achieve this, we make several subtle changes to the current way hot/cold wallets are built in BIP32 for Bitcoin. An important goal of our construction is that all these changes come with minimal overheads to guarantee efficiency and are compatible with Bitcoin and other state-of-the-art cryptocurrencies. The latter ensures that our wallet scheme can be readily deployed as a more secure alternative for existing hot/cold wallet systems. At the technical level, the main challenge of our work lies in proving that the ECDSA signatures can be used to construct a signature scheme with rerandomizable keys. Due to the rather "contrived" nature of ECDSA signatures our analysis is, however, more involved than for Schnorr and BLS signatures, and also requires us to slightly weaken the original notion of *unforgeability under rerandomized keys* due Fleischhacker et al. [FKM+16]. We call this notion *unforgeability under honestly rerandomized keys (***uf-cma-hrk***).*

Formally, we prove **uf-cma-hrk** of a "salted version" of the ECDSA signature scheme assuming that the standard ECDSA signature scheme is existentially unforgeable under chosen message attacks (**uf-cma**). The main challenge for this reduction is that in the **uf-cma-hrk** game, the adversary may see signatures under related (i.e., rerandomized) keys, where the relation between these keys may be known to the adversary. This significantly complicates the reduction. More precisely, in the reduction we need to embed the target public key $pk^*$ of the **uf-cma** game for the ECDSA signature scheme into the simulation of the adversary in the **uf-cma-hrk** game. Once $pk^*$ has been embedded, the reduction may have to answer signing queries for *any* of the rerandomized keys that the adversary can ask via the oracle `RSign`. Unfortunately, for this simulation we neither know the corresponding secret keys nor can the reduction answer these queries by using the underlying ECDSA signing oracle from the **uf-cma** game.

To overcome this challenge, we develop an efficient method that transfers ECDSA signatures wrt. $pk^*$ to signatures wrt. a related public key, and show how to apply it for proving the **uf-cma-hrk** security. The later is the main technical contribution of our work.

Practical considerations. As a final contribution, we explore the practical implications of our work. First, we argue that a careless implementation of hot/cold wallets using as underlying signature scheme, e.g., Schnorr or BLS, may result into a severe security vulnerability if the hot wallet is compromised. This may seem a bit surprising as the hot wallet does not contain any secret key material. At a high level, the vulnerability exploits a "related key attack" in these signature schemes, where an adversary that knows the "relation" between two related public keys $pk_{ID}$ and $pk_{ID'}$ can transform a signature $\sigma_{ID}$ scheme under $pk_{ID}$ to a signature $\sigma_{ID'}$ under $pk_{ID'}$. This may have severe consequences because once an adversary sees a signature $\sigma_{ID}$ that transfers funds assigned to $pk_{ID}$, it can also transfer the funds held by $pk_{ID'}$.

As a second practical contribution, we describe how our ECDSA-based wallet scheme can be integrated into Bitcoin. One difficulty is that for the proof to go through, we need that signatures produced by the cold wallet are salted with fresh randomness and prefixed by the pulic key (or the hash of it). Fortunately, Bitcoin supports a simple scripting language such that these changes can be integrated at very low additional costs.

## 1.3 Related work

Research on wallet systems. Hot/cold wallets are widely used in cryptocurrencies and various implementations on standard computing and dedicated hardware devices are available. Most related to our work is the result of Gutoski and Stebila [GS15] who discuss a flaw in BIP32 and propose a (provably secure) countermeasure against it. Concretely, they study the well known attack against deterministic wallets [But13] that allows to recover the master secret key once a single session key has leaked from the cold wallet. They then propose a fix for this flaw which allows up to $d$ session keys to leak, and show by a counting argument that under a one-more discrete-log assumption the master secret key can not be recovered. We emphasize that their model is rather restricted and does not consider an adversary learning public keys or signatures for keys which have not been compromised. More importantly, [GS15] prove only a very weak security guarantee. Namely, instead of aiming at the standard security notion of unforgeability where the adversary's goal is to forge a signature (as considered in our work), [GS15] consider the much weaker guarantee where the adversary's goal is to extract the entire master secret key. Hence, the security analysis in [GS15] does not consider adversaries that forge a signature with respect to some session public key, while in practice this clearly violates security.

Besides [GS15], various other works explore the security of hot/cold wallets. Similar to [GS15], Fan et al. [FTS+18] study the security against secret session key leakage (they call it "privilege escalation attacks"). Unfortunately, their proposed countermeasure is ad-hoc and no formal model nor security proof is provided. Another direction is taken by Turuani et al. [TVR16] who provide an automated verification of the Bitcoin Electrum wallet in the Dolev Yao model. Since the Dolev-Yao model assumes that ciphertexts, signatures etc. are all perfect, their analysis exclude potential vulnerabilities such as related key attacks, which turn out to be very relevant in the hot/cold wallet setting.

Another line of recent work focuses on the security analysis of hardware wallets [MPas19, AGKK19]. Both works target different goals. The work of Marcedone et al. [MPas19] aims at integrating two-factor authentication into wallet schemes, while Arapinis et al. [AGKK19] consider hardware attacks against hardware wallets and provide a formal modeling of such attacks in the UC framework. Similar to the latter, Curtoius et al. [CEV14] investigate how implementation flaws such as bad and correlated randomness may affect security. Other works that study the implications of weak randomness in wallets are [BR18, BH19].

Orthogonal to our work is a large body of work on threshold ECDSA [GGN16, LN18, DKLS18] and multisignatures [BDN18] to construct more secure wallets. Both approaches aim at distributing trust by requiring that multiple key holders authenticate transactions. These techniques can be combined with our hot/cold wallet to mitigate attacks against the cold wallet.

Other related work. One of the techniques that we use in this work is that certain signature schemes support the following efficient transformation: given a signature under some public key $pk$, one can produce a signature with respect to a related key $pk'$. While for certain signature schemes such as Schnorr [Sch89] this is a well-known trick that has been used in various works [FF13, KMP16, ZCC+15], we are not aware of any prior use of such an algorithm for the ECDSA signature scheme. In addition, as discussed above we make use of the abstraction of signature schemes with rerandomizable keys that was originally introduced by Fleischhacker et al. [FKM+16] in the context of sanitizable signatures.

## 2 Preliminaries

Notation. We denote as $s \xleftarrow{\$} \mathcal{H}$ the uniform sampling of the variable $s$ from the set $\mathcal{H}$. If $\ell$ is an integer, then $[\ell]$ is the set $\{1, \ldots, \ell\}$. We use uppercase letters $\mathsf{A}, \mathsf{B}$ to denote algorithms. Unless otherwise stated, all our algorithms are probabilistic and we write $y \xleftarrow{\$} \mathsf{A}(x)$ to denote that $\mathsf{A}$ returns output $y$ when run on input $x$. We write $y \leftarrow \mathsf{A}(x, \rho)$ to denote that $\mathsf{A}$ returns output $y$ when run on input $x$ and randomness $\rho$. Note that in this way, $\mathsf{A}$ becomes a deterministic algorithm. We use the notation $\mathsf{A}(x)$ to denote the set of all possible outputs of (probabilistic) algorithm $\mathsf{A}$ on input $x$.

We write $\mathsf{A}^{\mathsf{B}}$ to denote that $\mathsf{A}$ has oracle access to $\mathsf{B}$ during its execution. For ease of notation, we generally assume that boolean variables are initialized to $\mathsf{false}$, integers are set initially to 0, lists are initialized to $\emptyset$, and undefined entries of lists are initialized to $\perp$. To further simplify our definitions and notation, we assume that public parameters *par* have been securely generated and define the scheme or algebraic structure in context. We denote throughout the paper $\kappa$ as the security parameter. For bit strings $a, b \in \{0,1\}^*$ if we write "$a = (b, \cdot)$" we check if the prefix of $a$ is equal to $b$; likewise with "$a \neq (b, \cdot)$" we check if the prefix of $a$ is different from $b$.

Security Games. We use standard code-based security games [BR04]. A *game* $\mathbf{G}$ is an interactive probability experiment between an *adversary* $\mathsf{A}$ and an (implicit) *challenger* which provides answers to oracle queries posed by $\mathsf{A}$. $\mathbf{G}$ has one *main procedure* and can have any number of additional *oracle procedures* that describe how oracle queries are answered. We distinguish such oracle procedures from algorithmic ones by using a distinct font `Oracle`. The output of $\mathbf{G}$ when interacting with adversary $\mathsf{A}$ is denoted as $\mathbf{G}^{\mathsf{A}}$. Finally, the randomness in any probability term of the form $\Pr[\mathbf{G}^{\mathsf{A}} = 1]$ is assumed to be over all the random coins in game $\mathbf{G}$.

Random Oracle Model. We model hash functions as random oracles [BR93]. The code of hash function $\mathsf{H}$ is defined as follows. On input $x$ from the domain of the hash function, $\mathsf{H}$ checks whether $\mathsf{H}(x)$ has been previously defined. If so, it returns $\mathsf{H}(x)$. Else, it sets $\mathsf{H}(x)$ to a uniformly random element from the range of $\mathsf{H}$ and then returns $\mathsf{H}(x)$.

Elliptic Curve Cryptography. We denote an elliptic curve group as $\mathbb{E} = \mathbb{E}(par)$ with order $p$. The base point of the group $\mathbb{E}$ is denoted as $G := (x_b, y_b)$. Any point $S := (x_s, y_s)$ in the group $\mathbb{E}$ can be written as $S = aG$, where $a\mathbb{Z}_p$ and we use additive notation.

### 2.1 Signature Schemes

In this section, we introduce the syntax and relevant security notions for signature schemes.

**Definition 2.1** (Signature Scheme). A *signature scheme* $\mathsf{Sig}$ is a triple of algorithms $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$. The randomized *key generation algorithm* $\mathsf{Sig.Gen}$ takes as input public parameters *par* and returns a pair $(sk, pk)$, of secret and public keys. The randomized *signing algorithm* $\mathsf{Sig.Sign}$ takes as input a secret key $sk$ and a message $m$ and returns a signature $\sigma$. The deterministic *verification algorithm* $\mathsf{Sig.Verify}$ takes as input a public key $pk$, a signature $\sigma$, and a message $m$. It returns 1 (accept) or 0 (reject).

We require *correctness*: For all $(sk, pk) \in \mathsf{Sig.Gen}\,(par)$, and all $m \in \{0,1\}^*$, we have that

$$\Pr_{\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(sk,m)} [\mathsf{Sig.Verify}\,(pk, \sigma, m) = 1] = 1.$$

We also adopt the notion of signature schemes with rerandomizable keys from Fleischhacker et al. [FKM+16].

**Definition 2.2** (Signature Scheme with Perfectly Rerandomizable Keys). A *signature scheme with perfectly rerandomizable keys* is a tuple of algorithms $\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK})$ where $(\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify})$ are the standard algorithms of a signature scheme as defined above. Moreover, we assume that the public parameters $par$ define a randomness space $\chi := \chi(par)$. The probabilistic *secret key rerandomization algorithm* $\mathsf{RSig.RandSK}$ takes as input a secret key $sk$ and randomness $\rho \in \chi$ and outputs a rerandomized secret key $sk'$. The probabilistic *public key rerandomization algorithm* $\mathsf{RSig.RandPK}$ takes as input a public key $pk$ and randomness $\rho \in \chi$ and outputs a rerandomized public key $pk'$. We make the convention that for the empty string $\epsilon$, we have that $\mathsf{RSig.RandPK}(pk, \epsilon) = pk$ and $\mathsf{RSig.RandSK}(sk, \epsilon) = sk$. We further require:

1. *(Perfect) rerandomizability of keys:* For all $(sk, pk) \in \mathsf{RSig.Gen}\,(par)$ and $\rho \xleftarrow{\$} \chi$, the distributions of $(sk', pk')$ and $(sk'', pk'')$ are identical, where:

$$\begin{aligned}&(sk', pk') \leftarrow (\mathsf{RSig.RandPK}(pk, \rho), \mathsf{RSig.RandSK}(sk, \rho))\,,\\&(sk'', pk'') \xleftarrow{\$} \mathsf{RSig.Gen}\,(par)\,.\end{aligned}$$

2. *Correctness under rerandomized keys:* For all $(sk, pk) \in \mathsf{RSig.Gen}\,(par)$, for all $\rho \in \chi$, and for all $m \in \{0,1\}^*$, the rerandomized keys $sk' \leftarrow \mathsf{RSig.RandSK}(sk, \rho)$ and $pk' \leftarrow \mathsf{RSig.RandSK}(pk, \rho)$ satisfy:

$$\Pr_{\sigma \xleftarrow{\$} \mathsf{RSig.Sign}(sk',m)} [\mathsf{RSig.Verify}\,(pk', \sigma, m) = 1] = 1.$$

Security of Signature Schemes. In this work we will use the standard security notion of *existential unfogeability under chosen message attacks (UFCMA)*. We formalize this notion for a signature scheme $\mathsf{Sig}$ via the game $\textbf{uf-cma}_{\mathsf{Sig}}$ (Figure 1). In this game, the challenger begins by sampling $(sk, pk)$ as $(sk, pk) \xleftarrow{\$} \mathsf{Gen}\,(par)$. The adversary is then given the public key $pk$ and can adaptively sign messages of its choice under the corresponding secret key via an oracle `SignO`. Its goal is to forge a signature on a *fresh* message $m^*$, i.e., one that was not previously queried to `SignO`. For an algorithm $\mathsf{C}$, we define $\mathsf{C}$'s advantage in game $\textbf{uf-cma}_{\mathsf{Sig}}$ as $\mathsf{Adv}^{\mathsf{C}}_{\textbf{uf-cma},\mathsf{Sig}} = \Pr\left[\textbf{uf-cma}^{\mathsf{C}}_{\mathsf{Sig}} = 1\right]$.

For a signature scheme with rerandomizable keys $\mathsf{RSig}$, we also introduce a new security notion called *unforgeability under honestly rerandomized keys* that is formalized via game $\textbf{uf-cma-hrk}_{\mathsf{RSig}}$ (Figure 2). This notion constitutes a weaker form of the notion of *existential unforgeability under rerandomized keys* proposed in [FKM+16]. In the latter notion, the adversary is able to query the signing oracle not only for signatures corresponding to the public key $pk$ that it obtains in the unforgeability experiment, but also for signatures that correspond to *arbitrary rerandomizations of pk*. Similarly, the

```
main $\mathbf{uf\text{-}cma}_{\mathsf{Sig}}$
00 $(sk, pk) \xleftarrow{\$} \mathsf{Sig.Gen}\,(par)$
01 $(m^*, \sigma^*) \xleftarrow{\$} \mathsf{C}^{\mathtt{SignO}}\,(pk)$
02 If $m^* \in Sigs$: $\mathsf{bad} \leftarrow \mathsf{true}$
03 $b' \leftarrow \mathsf{Sig.Verify}\,(m^*, pk^*, \sigma^*)$
04 Return $b' \wedge \neg\mathsf{bad}$

Oracle $\mathtt{SignO}\,(m)$
05 $\sigma \xleftarrow{\$} \mathsf{Sig.Sign}\,(sk, m)$
06 $Sigs \leftarrow Sigs \cup \{m\}$
07 Return $\sigma$
```

Figure 1: Security game $\mathbf{uf\text{-}cma}_{\mathsf{Sig}}$ with adversary $\mathsf{C}$.

```
main $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$
00 $\mathsf{RList} \leftarrow \{\epsilon\}$
01 $(sk, pk) \xleftarrow{\$} \mathsf{RSig.Gen}\,(par)$
02 $(m^*, \sigma^*, \rho^*) \xleftarrow{\$} \mathsf{C}^{\mathtt{Rand},\mathtt{RSign}}\,(pk)$
03 If $m^* \in Sigs$: $\mathsf{bad} \leftarrow \mathsf{true}$
04 If $\rho^* \notin \mathsf{RList}$: $\mathsf{bad} \leftarrow \mathsf{true}$
05 $pk^* \leftarrow \mathsf{RSig.RandPK}(pk, \rho^*)$
06 $b \leftarrow \mathsf{RSig.Verify}\,(pk^*, \sigma^*, m^*)$
07 Return $b \wedge \neg\mathsf{bad}$

Oracle $\mathtt{RSign}\,(m, \rho)$
08 If $\rho \notin \mathsf{RList}$: Return $\perp$
09 $sk' \leftarrow \mathsf{RSig.RandSK}(sk, \rho)$
10 $\sigma \xleftarrow{\$} \mathsf{RSig.Sign}\,(m, sk')$
11 $Sigs \leftarrow Sigs \cup \{m\}$
12 Return $\sigma$

Oracle $\mathtt{Rand}$
13 $\rho \xleftarrow{\$} \chi$
14 $\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$
15 Return $\rho$
```

Figure 2: Security game $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ with adversary $\mathsf{C}$.

winning condition is also relaxed in this notion by allowing the adversary to return a forgery under an (arbitrarily) rerandomized key (but still on a fresh message $m^*$). The main difference between the security notion from [FKM$^+$16] and our new one is that the adversary is restricted to *honest* rerandomizations of $pk$, i.e., randomizations where the randomness is chosen by the challenger uniformly at random from $\chi$. We model this via an additional oracle in the security game. For an algorithm $\mathsf{C}$, we define $\mathsf{C}$'s advantage in game $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ as $\mathsf{Adv}^{\mathsf{C}}_{\mathbf{uf\text{-}cma\text{-}hrk},\mathsf{RSig}} = \Pr\left[\mathbf{uf\text{-}cma\text{-}hrk}^{\mathsf{C}}_{\mathsf{RSig}} = 1\right]$.

## 3 The Stateful Model for Wallets

In this section, we introduce our formal security model for stateful deterministic wallets. At a high level, a stateful deterministic wallet scheme allows two parties $A$ (the cold wallet) and $B$ (the hot wallet) to derive matching session key pairs (for signing/verification) from a pair of master keys. As presented in Figure 3, $A$ keeps her master secret key $msk$ and gives the master public key $mpk$ to $B$. $A$ and $B$ can now use the key derivation procedures $\mathsf{SKDer}$ and $\mathsf{PKDer}$, respectively, to derive an arbitrary number of session key pairs, locally, i.e., without further interaction. Intuitively, this is possible since every part of the key derivation procedure is deterministic and therefore, both $A$ and $B$ "automatically" carry out the same sequence of derivations.

In contrast to standard hot/cold wallets, we will make one conceptual change and add

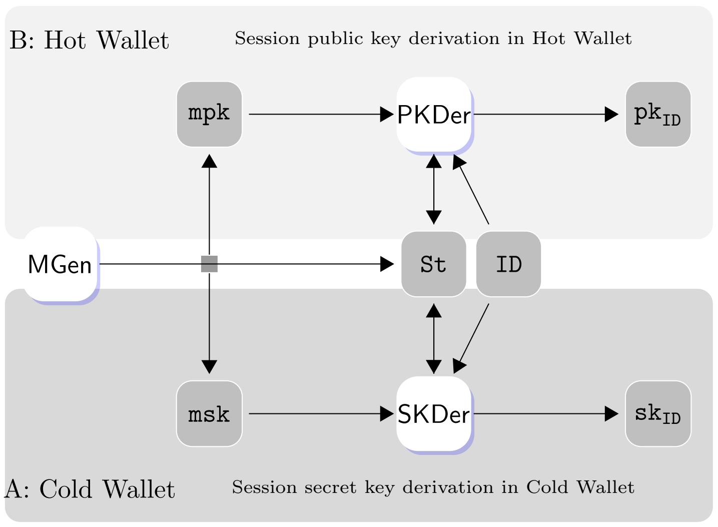
B: Hot Wallet
Session public key derivation in Hot Wallet
mpk
PKDer
pk_ID
MGen
St
ID
msk
SKDer
sk_ID
A: Cold Wallet
Session secret key derivation in Cold Wallet

Figure 3: Both Hot/ Cold wallet internally stores the common state $St$. The master keys are stored in the respective wallets. When a session secret key is generated within the cold wallet as $(sk_{ID}, St) \leftarrow \mathsf{SWal.SKDer}(msk, ID, St)$, the state $St$ gets refreshed. The session public key $pk_{ID}$ is generated within the hot wallet as $(pk_{ID}, St) \leftarrow \mathsf{SWal.PKDer}(mpk, ID, St)$, and the corresponding state $St$ is refreshed in the same manner.

to our schemes a *state*, denoted $St$ below. The state $St$ is updated (deterministically) during each call to one of the key derivation procedures. As we will explain shortly, this allows to obtain a strong form of forward privacy, which we will refer to as *unlinkability*. For $A$ to easily identify keys on the blockchain for which she can derive a corresponding secret key and to keep track of the order they where derived in by $B$, session keys also have an identifier $ID \in \{0,1\}^*$ which is used as an argument for the key derivation procedures. We now proceed to give the syntax of a stateful wallet scheme.

**Definition 3.1** (Stateful Wallet). A *stateful wallet* is a tuple of algorithms

$$\mathsf{SWal} = (\mathsf{SWal.MGen}, \mathsf{SWal.SKDer}, \mathsf{SWal.PKDer}, \mathsf{SWal.Sign}, \mathsf{SWal.Verify}),$$

which are defined as follows. The randomized *master key generation algorithm* $\mathsf{SWal.MGen}(par)$ takes public parameters $par$ as input and outputs a tuple $(St_0, mpk, msk)$ consisting of an *initial state* $St_0$, a *master secret key* $msk$ and a *master public key* $mpk$. The deterministic *secret key derivation algorithm* $\mathsf{SWal.SKDer}$ takes as input a master secret key $msk$, an identity $ID$, and a state $St$. It outputs a session secret key $sk_{ID}$ and an updated state $St'$. The deterministic *public key derivation algorithm* $\mathsf{SWal.PKDer}$ takes as input a master public key $mpk$, an identity $ID$, and a state $St$. It outputs a session public key $pk_{ID}$ and an updated state $St'$. The randomized signing algorithm $\mathsf{SWal.Sign}$ takes as input a (session) secret key $sk$ and a message $m$ and returns a signature $\sigma$. The deterministic verification algorithm $\mathsf{SWal.Verify}$ takes as input a (session) public key $pk$, a signature $\sigma$, and a message $m$. It returns 1 (accept) or 0 (reject).

Let $\mathsf{SWal} = (\mathsf{SWal.MGen}, \mathsf{SWal.SKDer}, \mathsf{SWal.PKDer}, \mathsf{SWal.Sign}, \mathsf{SWal.Verify})$ denote a stateful wallet according to Definition 3.1, for the remainder of this section. We now define correctness of stateful deterministic wallets. Roughly speaking, correctness should ensure that if the cold wallet $A$ and the hot wallet $B$ derive session key pairs on the same set of identities $ID_0, ..., ID_{n-1} \in \{0,1\}^*$ and in the same order, any signature created under one of the resulting signing keys of $A$ should correctly verify under the corresponding verification key of $B$. In other words, all the derived session keys should "match".

**Definition 3.2** (Correctness of Stateful Wallets). For all $(St_0, msk, mpk) \in \mathsf{SWal.MGen}(par)$, all $n \in \mathbb{N}$, all $\vec{ID} := (ID_0, ..., ID_{n-1}) \in \{0,1\}^*$, we set $St_0[\vec{ID}, St_0, msk] = St_0[\vec{ID}, St_0, mpk] := St_0$ and define the sequence $\left\{\left(sk_i[\vec{ID}, St_0, msk], St_i[\vec{ID}, St_0, msk]\right)\right\}_{1\le i\le n}$ recursively as

$$\left(sk_i[\vec{ID}, St_0, msk], St_i[\vec{ID}, St_0, msk]\right) := \mathsf{SWal.SKDer}(msk, ID_{i-1}, St_{i-1}[\vec{ID}, St_0, msk]).$$

Analogously, we define $\left\{\left(pk_i[\vec{ID}, St_0, mpk], St_i[\vec{ID}, St_0, mpk]\right)\right\}_{1\le i\le n}$ recursively as

$$\left(pk_i[\vec{ID}, St_0, mpk], St_i[\vec{ID}, St_0, mpk]\right) := \mathsf{SWal.PKDer}(mpk, ID_{i-1}, St_{i-1}[\vec{ID}, St_0, mpk]).$$

We say that $\mathsf{SWal}$ is *correct* if for all $n \in \mathbb{N}$, all $(ID_0, ..., ID_{n-1}) \in \{0,1\}^*$, all $(St_0, msk, mpk) \in \mathsf{SWal.MGen}(par)$, and all $m \in \{0,1\}^*$, we have for $pk := pk_n[\vec{ID}, St_0, mpk]$ and $sk := sk_n[\vec{ID}, St_0, msk]$:

$$\Pr_{\sigma \xleftarrow{\$} \mathsf{SWal.Sign}(sk,m)}[\mathsf{SWal.Verify}(pk, \sigma, m) = 1] = 1.$$

In the next subsection, we introduce the two basic security notions for stateful wallets, namely a) *unlinkability* of generated public session keys, and b) *unforgeability* of corresponding signatures.

## 3.1 Wallet Unlinkability

We begin by introducing the notion of *wallet unlinkability.* Intuitively, unlinkabililty guarantees that transactions sending money to different public session keys that were derived from the *same master key* should be unlinkable. Formally, we require that, given the master public key, the distribution of session public keys is computationally indistinguishable from session public keys that are generated from a fresh (i.e., independently and randomly chosen) master public key and state. Unfortunately, there is little hope to achieve this guarantee for keys to which the adversary knows the state $St$ used to derive them. Therefore, our notion of unlinkability satisfies a weaker form of privacy called *forward unlinkability.* This means that keys generated prior to a hot wallet breach (i.e., when the adversary learns the state) cannot be linked to $mpk$.

The wallet unlinkability game $\mathbf{unl}_{\mathsf{SWal}}$ is presented in Figure 5. Initially, $\mathsf{A}$ receives as input a master public key $mpk$ generated via $\mathsf{MGen}\,(par)$ and subsequently interacts with oracles $\mathsf{PK}$, `WalSign` and $\mathsf{Chall}$ that reflect $\mathsf{A}$'s capabilities. The game internally maintains a state $St$, which is updated when $\mathsf{A}$ calls the oracle $\mathsf{PK}$ to derive new keys. In addition, at any point in time $\mathsf{A}$ can read out the current state $St$ by calling the oracle `getSt`. This models $\mathsf{A}$'s capability to break into the hot wallet on which the state is stored. Finally, the oracle $\mathsf{Chall}$ allows $\mathsf{A}$ to obtain a challenge public key $pk_{ID}$ for a user identity $ID$ of its choice. This challenge public key is either "real" or "random", i.e., it depends on $mpk$

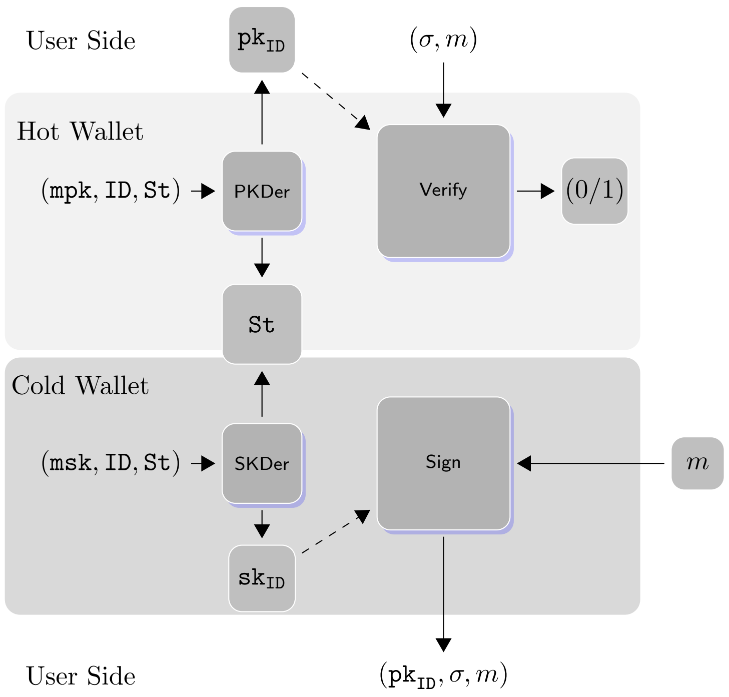

Figure 4: (1) The cold wallet signs a message $m$ with its session secret key $sk_{ID}$ as $\sigma \leftarrow \mathsf{SWal.Sign}(sk_{ID}, m)$. (2) Anyone can later verify the validity of a signature $\sigma$ on message $m$ as $(0/1) \leftarrow \mathsf{SWal.Verify}(pk_{ID}, \sigma, m)$.

or was sampled freshly and independently of $mpk$ (see below for details). $\mathsf{A}$'s goal is to distinguish these two scenarios. However, $\mathsf{A}$ is only considered successful if it obtains $St$ (via oracle `getSt`) *after* being given the challenge public key $pk_{ID}$.[3] We now proceed in explaining the oracles to which $\mathsf{A}$ has access in more detail.

**$\mathsf{PK}(ID)$:** The oracle $\mathsf{PK}$ takes as input an $ID$ and returns a corresponding session public key $pk_{ID}$. It models $\mathsf{A}$'s capability to observe transactions stored on the blockchain that transfer money to $pk_{ID}$. A typical setting where this may occur is when funds are sent via the blockchain to the cold wallet. For simplicity of bookkeeping, we make the convention that identifiers are unique and thus $\mathsf{A}$ can call $\mathsf{PK}$ only once per $ID$.

**`WalSign`$(m, ID)$:** The oracle `WalSign` takes as input an identity $ID$ and a message $m$ and returns the corresponding signature if $pk_{ID}$ has been previously returned as a result to a $\mathsf{PK}(ID)$ query. As such, it allows $\mathsf{A}$ to sign, in an adaptive fashion, messages of its choice under public keys that it previously obtained via the oracle $\mathsf{PK}$. `WalSign` models that an adversary $\mathsf{A}$ may obtain signatures that are produced by the cold wallet with $sk_{ID}$, when funds are spent from the cold wallet (e.g., when the owner of the cold wallet buys something with the collected coins).

[3] Recall that otherwise the adversary can trivially distinguish between “real” or “random”.

```
main $\mathbf{unl}_{\mathsf{SWal}}$
$(St, msk, mpk) \xleftarrow{\$} \mathsf{SWal.MGen}(par)$
$b \xleftarrow{\$} \{0,1\}$
$\mathtt{Orc} \leftarrow \{\mathsf{PK}, \mathtt{WalSign}, \mathsf{Chall}, \mathtt{getSt}\}$
$b' \xleftarrow{\$} \mathsf{A}^{\mathtt{Orc}}(mpk)$
Return $b' = b$

Oracle $\mathtt{WalSign}(m, ID)$
If $SSNKeys[ID] = \perp$: Return $\perp$
$(pk_{ID}, sk_{ID}) \leftarrow SSNKeys[ID]$
$\sigma \xleftarrow{\$} \mathsf{SWal.Sign}(sk_{ID}, m)$
Return $\sigma$

Oracle $\mathtt{PK}\,(ID)$ // Once per ID
$tmp_1 \leftarrow (msk, ID, St)$
$tmp_2 \leftarrow (mpk, ID, St)$
$(sk_{ID}, St) \leftarrow \mathsf{SWal.SKDer}(tmp_1)$
$(pk_{ID}, St) \leftarrow \mathsf{SWal.PKDer}(tmp_2)$
$SSNKeys[ID] \leftarrow (pk_{ID}, sk_{ID})$
Return $pk_{ID}$

Oracle $\mathtt{getSt}$
$StateQuery \leftarrow \mathsf{true}$
Return $St$

Oracle $\mathsf{Chall}\,(ID)$ //One time
If $StateQuery$: Return $\perp$
If $SSNKeys[ID] \neq \perp$: Return $\perp$
// Generate real key
$tmp_1 \leftarrow (msk, ID, St)$
$tmp_2 \leftarrow (mpk, ID, St)$
$(sk^0_{ID}, St) \leftarrow \mathsf{SWal.SKDer}(tmp_1)$
$(pk^0_{ID}, St) \leftarrow \mathsf{SWal.PKDer}(tmp_2)$
// Generate random key
$(\hat{St}, \hat{msk}, \hat{mpk}) \xleftarrow{\$} \mathsf{SWal.MGen}(par)$
$tmp_1 \leftarrow (\hat{msk}, ID, \hat{St})$
$tmp_2 \leftarrow (\hat{mpk}, ID, \hat{St})$
$(sk^1_{ID}, \cdot) \leftarrow \mathsf{SWal.SKDer}(tmp_1)$
$(pk^1_{ID}, \cdot) \leftarrow \mathsf{SWal.PKDer}(tmp_2)$
$SSNKeys[ID] \leftarrow (pk^b_{ID}, sk^b_{ID})$
Return $pk^b_{ID}$
```

Figure 5: Adversary $\mathsf{A}$ playing in Game $\mathbf{unl}_{\mathsf{SWal}}$.

**$\mathtt{getSt}$:** The oracle $\mathtt{getSt}$ returns the current state $St$ and records this event by setting $StateQuery$ to $\mathsf{true}$. As mentioned above, this models hot wallet corruption.

**$\mathsf{Chall}\,(ID)$:** The oracle $\mathsf{Chall}$ takes as input an $ID$ and returns a public key $pk^b_{ID}$ that depends on the uniformly random bit $b$ sampled internally by the game $\mathbf{unl}_{\mathsf{SWal}}$. $\mathsf{Chall}$ can be called only a single time. If $b = 0$, $pk^0_{ID}$ is derived from the current state $St$ and $mpk$ as $\left(pk^0_{ID}, \cdot\right) \leftarrow \mathsf{SWal.PKDer}(mpk, ID, St)$. If $b = 1$, $pk^1_{ID}$ is derived from a freshly generated master public key and state for the same identity $ID$, i.e., via the sequence of steps:

- $(\hat{St}, \cdot, \hat{mpk}) \xleftarrow{\$} \mathsf{SWal.MGen}(par)$
- $\left(pk^1_{ID}, \cdot\right) \leftarrow \mathsf{SWal.PKDer}(\hat{mpk}, ID, \hat{St})$

If $\mathsf{A}$ sets $StateQuery$ prior to calling $\mathsf{Chall}$, or queries $\mathsf{Chall}$ on an identity $ID$ that it previously queried $\mathsf{PK}$ on, $\mathsf{Chall}$ always returns $\perp$ in order to prevent a trivial attack on unlinkability. We define $\mathsf{A}$'s advantage in $\mathbf{unl}_{\mathsf{SWal}}$ as

$$\mathsf{Adv}^{\mathsf{A}}_{\mathbf{unl},\mathsf{SWal}} = \left|\Pr\left[\mathbf{unl}^{\mathsf{A}}_{\mathsf{SWal}} = 1\right] - \frac{1}{2}\right|. \tag{1}$$

```
main $\mathbf{wunf}_{\mathsf{SWal}}$
00 $(St, msk, mpk) \xleftarrow{\$} \mathsf{SWal.MGen}(par)$
01 $(m^*, \sigma^*, ID^*) \xleftarrow{\$} \mathsf{A}^{\mathsf{PK},\mathtt{WalSign}}(mpk, St)$
02 If $SSNKeys[ID^*] = \perp$
03     Return 0
04 $(pk_{ID^*}, sk_{ID^*}) \leftarrow SSNKeys[ID^*]$
05 If $m^* \in Sigs[ID^*]$
06     Return 0
07 If $\mathsf{SWal.Verify}(pk_{ID^*}, \sigma^*, m^*) = 0$
08     Return 0
09 Return 1

Oracle $\mathtt{WalSign}(m, ID)$
10 If $SSNKeys[ID] = \perp$: Return $\perp$
11 $(pk_{ID}, sk_{ID}) \leftarrow SSNKeys[ID]$
12 $\sigma \xleftarrow{\$} \mathsf{SWal.Sign}(sk_{ID}, m)$
13 $Sigs[ID] \leftarrow Sigs[ID] \cup \{m\}$
14 Return $\sigma$

Oracle $\mathtt{PK}(ID)$ // Once per ID
15 $tmp_1 \leftarrow (msk, ID, St)$
16 $tmp_2 \leftarrow (mpk, ID, St)$
17 $(sk_{ID}, St) \leftarrow \mathsf{SWal.SKDer}(tmp_1)$
18 $(pk_{ID}, St) \leftarrow \mathsf{SWal.PKDer}(tmp_2)$
19 $SSNKeys[ID] \leftarrow (pk_{ID}, sk_{ID})$
20 $Sigs[ID] \leftarrow \emptyset$
21 Return $pk_{ID}$
```

Figure 6: Adversary $\mathsf{A}$ playing in Game $\mathbf{wunf}$.

## 3.2 Wallet Unforgeability

In this subsection we describe the *wallet unforgeability* notion. In Game $\mathbf{wunf}^{\mathsf{A}}_{\mathsf{SWal}}$ (depicted in Figure 6) we consider again an adversary $\mathsf{A}$ that receives as input a master public key $mpk$ and has subsequently access to oracles $\mathsf{PK}$ and $\mathtt{WalSign}$, which work as their corresponding oracles in the unlinkability game. In addition, $\mathsf{A}$ gets as input the *initial state St*. $\mathsf{A}$ wins if it can produce a triple $(m^*, \sigma^*, ID^*)$ such that $\sigma^*$ is a valid forgery on message $m^*$ under a public key $pk_{ID^*}$ previously obtained from a call to $\mathsf{PK}$. Here, valid means that no signature on message $m^*$ under $pk_{ID^*}$ was previously obtained from a call to $\mathtt{WalSign}$. We denote $\mathsf{A}$'s advantage in $\mathbf{wunf}_{\mathsf{SWal}}$ as

$$\mathsf{Adv}^{\mathsf{A}}_{\mathbf{wunf},\mathsf{SWal}} = \Pr\Big[\mathbf{wunf}^{\mathsf{A}}_{\mathsf{SWal}} = 1\Big]. \tag{2}$$

Unforgeability for Keys with Compromised State. At a high-level, the $\mathbf{wunf}_{\mathsf{SWal}}$ game models that once funds are transferred to the cold wallet they remain secure even if (a) the hot wallet is compromised, and (b) the adversary can see transfers of coins sent from the cold wallet. We now explain the game in more detail. In contrast to the $\mathbf{unl}_{\mathsf{SWal}}$ game from the previous section, in the $\mathbf{wunf}_{\mathsf{SWal}}$ game the adversary is given the *state St* as part of its initial input. This models the "worst-case" adversary that breaks into the hot wallet right after the hot/cold wallet has been initialized. In addition, to giving $\mathsf{A}$ the initial state $St$ and the master public key $mpk$, we also give it access to the $\mathsf{PK}$ and $\mathtt{WalSign}$ oracle. The first can be queried by the adversary on identity $ID$ to derive a new key pair $(pk_{ID}, sk_{ID})$ from the master keys and the current state, and is used mainly for bookkeeping purposes.[4] The second oracle $\mathtt{WalSign}$ is as in the $\mathbf{unl}_{\mathsf{SWal}}$ game except that we also keep track of the messages that were already signed via the map $Sigs[ID]$.

[4]Notice that in $\mathbf{wunf}_{\mathsf{SWal}}$ the adversary obtains $mpk$ and the initial state, and hence can compute the output $pk$ of $\mathsf{PK}$ himself.

As already mentioned above, since the adversary receives $mpk$ and the initial state $St$ in the $\mathbf{wunf}_{\mathsf{SWal}}$ game, it can derive all possible $pk_{ID}$ (even without calling $\mathtt{PK}(ID)$). This subtle difference significantly complicates the security proof in the subsequent sections and is a crucial aspect of our unforgeability notion. More concretely, since $\mathsf{A}$ knows the state throughout the entire game $\mathbf{wunf}_{\mathsf{SWal}}$, it may be able to mount a related key attack (RKA) against the underlying signature scheme used in our wallet construction. At a high-level the RKA allows the adversary to "transfer" a signature $\sigma_{ID}$ with respect to $pk_{ID}$ to a valid signature $\sigma_{ID^*}$ for $pk_{ID^*}$. Signature schemes that are susceptible to such an RKA are for instance the Schnorr or BLS signature scheme, and the discussion on the attack of a hot/cold wallet instantiated in a naive way with these schemes can be found in Appendix, full version. Let us briefly explain how an adversary in the $\mathbf{wunf}_{\mathsf{SWal}}$ game can exploit such an RKA to break the underlying wallet scheme.

To this end, consider an adversary $\mathsf{A}$ that breaks into the hot wallet and obtains $mpk$ and $St$. This break-in is modeled in $\mathbf{wunf}_{\mathsf{SWal}}$ by giving the adversary $mpk, St$ at the beginning of the game. Next, the adversary waits until some funds are transferred to the cold wallet, which we model by calls to the $\mathsf{PK}$ oracle. Finally, $\mathsf{A}$ queries the `WalSign` oracle to transfer some fraction of funds – say the funds stored under $pk_{ID}$ – from the cold wallet to some new address. In practice, this may happen for example when some of the funds kept on the cold wallet are spent for a purchase. Once the adversary has received a single signature $\sigma_{ID}$ produced by the cold wallet, it can apply the RKA to steal *all funds* that have ever been transferred to the cold wallet. More precisely, given the signature $\sigma_{ID}$, the master public key $mpk$, and the state $St$ it can produce valid signatures $\sigma_{ID^*}$ for $pk_{ID^*}$ where $pk_{ID^*}$ resulted from a previous call to $\mathsf{PK}$ on input $ID^*$.

This attack results into a severe security breach as the owner of the cold wallet can loose its entire funds stored on the cold wallet. Since the attack does not require to break into the cold wallet, it strongly violates the original purpose of the hot/cold wallet concept in cryptocurrencies. Indeed, a user that transfers its funds to the cold wallet would assume that once the funds are transferred to the cold wallet, they are safe except for a break-in to the cold wallet.

As demonstrated in the subsequent section, an easy way to thwart this attack is to use *public key prefixing*, i.e., to compute a signature on $m$ as $\mathsf{Sign}(sk,(pk, m))$. Interestingly, this technique was also used in [MSM+15], with the purpose of preventing an RKA. This further highlights the close relation between resistance to RKAs and unforgeability in our model.

Of course, exploiting an RKA is only one possibility of stealing funds from the cold wallet, and there may be other types of attacks allowing the adversary to forge signatures with respect to keys stored on the cold wallet, given that it knows the state. Nevertheless, it also clearly underlines the importance of a formal security analysis of hot/cold wallet schemes within a strong security model. In the next section, we show how to reduce the security of a wallet scheme in the above unforgeability game to the security of the signature scheme that underlies the wallet construction.

```
Algorithm SWal[H].MGen($par$)
00 $St \xleftarrow{\$} \{0,1\}^\kappa$
01 $(mpk, msk) \xleftarrow{\$}$ RSig.Gen($par$)
02 Return $(St, msk, mpk)$

Algorithm SWal[H].SKDer($msk, ID, St$)
00 $(\omega_{ID}, St) \leftarrow$ H($St, ID$)
01 $sk_{ID} \xleftarrow{\$}$ RSig.RandSK($msk, \omega_{ID}$)
02 Return $(sk_{ID}, St)$

Algorithm SWal[H].PKDer($mpk, ID, St$)
03 $(\omega_{ID}, St) \leftarrow$ H($St, ID$)
04 $pk_{ID} \leftarrow$ RSig.RandPK($mpk, \omega_{ID}$)
05 Return $(pk_{ID}, St)$

Algorithm
SWal[H].Sign($m, sk, pk$)
03 $\hat{m} \leftarrow (pk, m)$
04 $\sigma \xleftarrow{\$}$ RSig.Sign($sk, \hat{m}$)
05 Return $\sigma$

Algorithm
SWal[H].Verify($pk, \sigma, m$)
06 $\hat{m} \leftarrow (pk, m)$
07 Return RSig.Verify($pk, \sigma, \hat{m}$)
```

Figure 7: Construction of swal[H] from RSig and H.

## 4 Generic Constructions

In this section, we show how to realize a stateful wallet from any signature scheme with uniquely rerandomizable keys. Such a signature scheme is defined as follows.

**Definition 4.1** (Signature scheme with uniquely rerandomizable keys). A rerandomizable signature scheme RSig, is said to have *uniquely rerandomizable public keys* if for all $(\rho, \rho') \in \chi$, we have that $\mathsf{RandPK}(pk, \rho) = \mathsf{RandPK}(pk, \rho')$ implies $\rho = \rho'$.

We begin by explaining our generic construction. We then prove its security with respect to the security notions introduced in Section 3. We assume in the following a signature scheme with uniquely rerandomizable keys

$$\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK})$$

. Our construction swal[H] of a stateful wallet which internally uses the hash function $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p \times \{0,1\}^\kappa$ (for state updates) is depicted in Figure 7.

### 4.1 Security Analysis

We proceed to analyze the properties of *unlinkability* and *unforgeability* of our stateful wallet construction (c.f. Figure 7) below.

#### 4.1.1 Unlinkability

We begin by proving unlinkability of our generic construction. The proof is rather simple and follows from collision resistance of H and that H is modeled as a random oracle. It also relies on the rerandomizability property of the underlying signature scheme.

**Theorem 4.2** *Let* $\mathsf{swal}[\mathsf{H}]$ *be the construction defined in Figure 7. Then for any adversary* $\mathsf{A}$ *playing in game* $\mathbf{unl}_{\mathsf{swal}[\mathsf{H}]}$, *we have*

$$\mathsf{Adv}^{\mathsf{A}}_{\mathbf{unl},\mathsf{swal}[\mathsf{H}]} \leq \frac{q_H(q_P+2)}{2^\kappa},$$

*where* $q_H$ *and* $q_P$ *are the number of random oracle queries and queries to oracle* $\mathsf{PK}$, *respectively, that* $\mathsf{A}$ *makes.*

*Proof.* Consider an adversary $\mathsf{A}$ playing in game $\mathbf{unl}_{\mathsf{swal}[\mathsf{H}]}$. $\mathsf{A}$ interacts with oracles $\mathsf{PK}, \mathtt{WalSign}, \mathtt{getSt}, \mathsf{Chall}$, and the random oracle $\mathsf{H}$. We can assume without loss of generality that $\mathsf{A}$ always calls $\mathsf{Chall}\,(ID)$ *before* calling $\mathtt{getSt}$ and exclusively on an identity $ID$ that was never previously queried to $\mathsf{PK}$; otherwise, $\mathsf{Adv}^{\mathsf{A}}_{\mathbf{unl},\mathsf{swal}[\mathsf{H}]} = 0$ and the theorem holds trivially. In the following, let $\mathcal{S}$ denote the set of values taken by the variables $St, \hat{St}$ *before* $\mathsf{A}$ calls $\mathsf{Chall}\,(ID)$. Furthermore, let $pk^0_{ID}, pk^1_{ID}$ denote the keys internally sampled in $\mathbf{unl}_{\mathsf{swal}[\mathsf{H}]}$ upon $\mathsf{A}$'s call $\mathsf{Chall}\,(ID)$. Note that by definition of $\mathsf{swal}[\mathsf{H}].\mathsf{PKDer}$, unless $\mathsf{A}$ manages to make a query of the form $\mathsf{H}\left(St', ID\right)$ where $St' \in \mathcal{S}$, $pk^0_{ID}$ and $pk^1_{ID}$ are identically distributed from its point of view. The reason is that as long as such a query hasn't been made, the values of $St, \hat{St}$ used to derive $pk^0_{ID}$ and $pk^1_{ID}$, respectively, are uniformly distributed from $\mathsf{A}$'s point of view. Now, the rerandomizability property of $\mathsf{RSig}$ ensures that both $pk^0_{ID}$ and $pk^1_{ID}$ are identically distributed to a freshly generated public key $pk \xleftarrow{\$} \mathsf{RSig}(par)$ (and therefore identically distributed to each other). In this case we again have that $\mathsf{Adv}^{\mathsf{A}}_{\mathbf{unl},\mathsf{swal}[\mathsf{H}]} = 0$. It therefore remains to argue that $\mathsf{A}$ makes such a call to $\mathsf{H}$ with probability at most $\left(q_H(q_P+2)\right)/2^\kappa$. This can be seen as follows. Since $\mathsf{A}$ makes at most $q_P$ queries to $\mathsf{PK}$ throughout $\mathbf{unl}_{\mathsf{swal}[\mathsf{H}]}$, in particular $|\mathcal{S}| \leq q_P + 2$. Since we have assumed that $\mathsf{A}$ always calls $\mathtt{getSt}$ *after* calling $\mathsf{Chall}$ (which internally updates $St$), *all* values in $\mathcal{S}$ are uniformly distributed from $\mathsf{A}$'s point of view, until it learns any particular value $St' \in \mathcal{S}$ (note that after such $St'$ becomes known to $\mathsf{A}$, it is able to infer all values that were added to $\mathcal{S}$ after $St'$). Therefore, the probability that for any particular query of the form $\mathsf{H}\left(St', ID\right)$, $St' \in \mathcal{S}$, is at most $(q_P+2)/2^\kappa$. Since $\mathsf{A}$ makes at most $q_H$ such queries of the form $\mathsf{H}\left(St', ID\right)$, the probability that for any of them, $St' \in \mathcal{S}$, is at most $\left(q_H(q_P+2)\right)/2^\kappa$, which proves the lemma. ■

### 4.1.2 Unforgeability

We now turn towards the unforgeability of our construction. Before giving the proof, we provide some intuition about our proof technique. At a high level, the idea is to reduce the security of the stateful wallet scheme $\mathsf{swal}[\mathsf{H}]$ (relative to $\mathbf{wunf}_{\mathsf{swal}[\mathsf{H}]}$) to the security of $\mathsf{RSig}$ (relative to $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$). As such, the proof consists mainly of the description of a reduction $\mathsf{C}$ trying to come up with a valid forgery in order to win the game $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ by simulating $\mathbf{wunf}_{\mathsf{swal}[\mathsf{H}]}$ to an adversary $\mathsf{A}$. Recall that $\mathsf{C}$ obtains a public key $pk_{\mathsf{C}}$ from its challenger in $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ and has access to oracles $\mathtt{Rand}$, $\mathtt{RSign}$. It can call the oracle $\mathtt{Rand}$ to obtain a random value $\rho$. Later, $\mathsf{C}$ can use the signing oracle $\mathtt{RSign}$ on input $(m, \rho)$, which provides signatures on a message $m$ of $\mathsf{C}$'s choice under the rerandomized key $pk' := \mathsf{swal}[\mathsf{H}].\mathsf{RandPK}(pk_{\mathsf{C}}, \rho)$. Note that $\mathsf{C}$ can query $\mathtt{RSign}$ also to get signatures under $pk_{\mathsf{C}}$ by setting $\rho = \epsilon$. $\mathsf{C}$'s goal is to simulate the oracles in the $\mathbf{wunf}_{\mathsf{swal}[\mathsf{H}]}$ experiment and to suitably embed $pk_{\mathsf{C}}$ into the key $pk_{ID^*}$ under which $\mathsf{A}$ eventually returns a forgery $(\sigma^*, m^*)$. The hope is that it can use $(\sigma^*, m^*)$ to win $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$.

A promising approach is therefore to embed $pk_{\mathsf{C}}$ into the master public key $mpk$ within the simulation. This way, every answer to a query $\mathsf{PK}(ID)$ can easily be computed as $(\omega_{ID}, \cdot) \leftarrow \mathsf{H}(St, ID)$, $pk_{ID} \leftarrow \mathsf{swal[H].RandPK}(mpk, \omega_{ID})$. To simulate any signature under $pk_{ID}$ to $\mathsf{A}$, $\mathsf{C}$ can make a query of the form $\mathtt{RSign}(\hat{m}, \omega_{ID})$, where $\hat{m} = (pk_{ID}, m)$. When $\mathsf{A}$ returns the forgery $(m^*, \sigma^*, ID^*)$, it is valid under the following conditions: (i) $pk_{ID^*}$ is a valid session public key that was returned to $\mathsf{A}$ as the answer to a query $\mathsf{PK}(ID^*)$, (ii) $\mathsf{A}$ has not yet queried $\mathtt{WalSign}$ for a signature on $m^*$ under $pk_{ID^*}$, (iii) the signature $\sigma^*$ is valid, i.e., $\mathsf{swal[H].Verify}(pk_{ID^*}, \sigma^*, m^*) = 1$. As part of the proof, we show that $\mathsf{C}$ can win $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ by returning the forgery $(m^*, \sigma^*, \rho^*)$, where $\rho^* = \omega^*_{ID}$.

**Theorem 4.3** *Let $\mathsf{A}$ be an algorithm that plays in the unforgeability game $\mathbf{wunf}_{\mathsf{swal[H]}}$, where $\mathsf{swal[H]}$ denotes the construction defined in Figure 7. Then if $\mathsf{RSig}$ is a signature scheme with uniquely rerandomizable keys, then there exists an algorithm $\mathsf{C}$ running in roughly the same time as $\mathsf{A}$, such that*

$$\mathsf{Adv}^{\mathsf{A}}_{\mathbf{wunf},\mathsf{swal[H]}} \leq \mathsf{Adv}^{\mathsf{C}}_{\mathbf{uf\text{-}cma\text{-}hrk},\mathsf{RSig}} + \frac{q^2}{p}$$

*where $q$ is the number of random oracle queries that $\mathsf{A}$ makes.*

*Proof.* Consider an adversary $\mathsf{A}$ playing $\mathbf{wunf}_{\mathsf{swal[H]}}$. As such, $\mathsf{A}$ is given the initial master public key $mpk$ and the initial state $St$, and is granted access to the oracles $\mathsf{PK}, \mathtt{WalSign}$ and the random oracle $\mathsf{H}$. We prove the Theorem via a sequence of games.

Game $\mathbf{G}_0$: This game behaves exactly as $\mathbf{wunf}_{\mathsf{swal[H]}}$, i.e., $\mathbf{G}_0 := \mathbf{wunf}_{\mathsf{swal[H]}}$. Internally however, $\mathbf{G}_0$ additionally sets $\mathtt{flag} \leftarrow \mathsf{true}$, whenever there is a call of the form $\mathsf{PK}(ID)$, such that the tuple $(sk_{ID}, pk_{ID})$ of session keys corresponding to this query, collides with a pair of session keys that was previously derived for another identity $ID' \neq ID$, i.e., $(pk_{ID}, sk_{ID}) = (pk_{ID'}, sk_{ID'}) = SSNKeys[ID']$.

Game $\mathbf{G}_1$: $\mathbf{G}_1$ behaves as $\mathbf{G}_0$, but aborts whenever $\mathtt{flag}$ is set to true. We let $E_{0,1}$ denote the event that $\mathtt{flag} = \mathsf{true}$ during the execution of $\mathbf{G}_1$.

**Claim 4.4** $\Pr[E_{0,1}] \leq \frac{q^2}{p}$.

*Proof.* A collision of the form $(pk_{ID}, sk_{ID}) = (pk_{ID'}, sk_{ID'})$, where $ID \neq ID'$ implies that

$$\mathsf{RSig.RandPK}(mpk, \omega_{ID}) = \mathsf{RSig.RandPK}(mpk, \omega_{ID'}).$$

From the property of signature scheme with uniquely rerandomizable keys of $\mathsf{RSig}$, this would mean $\omega_{ID} = \omega_{ID'}$, where $(\omega_{ID}, \cdot) = \mathsf{H}(\cdot, ID)$, $(\omega_{ID'}, \cdot) = \mathsf{H}(\cdot, ID')$. Since there are $q$ queries to $\mathsf{H}$ the probability of event $E_{0,1}$ is bounded by $\frac{q^2}{p}$. ■

Thus, $\mathsf{Adv}^{\mathsf{A}}_{\mathbf{wunf},\mathsf{swal[H]}} \leq \mathsf{Adv}^{\mathsf{A}}_{\mathbf{G}_1} + \frac{q^2}{p}$.

Next, we show how winning game $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ reduces to winning game $\mathbf{G}_1$. To this end, we describe an algorithm $\mathsf{C}^{\mathtt{Rand},\mathtt{RSign}}$ (depicted in Figure 8) that plays in game $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$. $\mathsf{C}$ obtains as input a public key $pk_{\mathsf{C}}$ and is given access to the oracles $\mathtt{Rand}$ and $\mathtt{RSign}$. $\mathsf{C}$ simulates $\mathbf{G}_1$ to $\mathsf{A}$ as described in the following.

```
Algorithm $\mathsf{C}^{\mathtt{RSign},\mathtt{Rand}}(pk_{\mathsf{C}})$
00 $St \xleftarrow{\$} \{0,1\}^{\kappa}$
01 $(m^*, \sigma^*, ID^*) \xleftarrow{\$} \mathsf{A}^{\mathsf{PK},\mathsf{WSign},\mathsf{H}}(mpk, St)$
02 If $SSNKeys[ID^*] = \bot$: Abort
03 If $m^* \in Sigs[ID^*]$: Abort
04 $(pk_{ID^*}, \omega_{ID^*}) \leftarrow SSNKeys[ID^*]$
05 If $\mathsf{SWal}[\mathsf{H}].\mathsf{Verify}(pk_{ID^*}, \sigma^*, m^*) = 0$ :
06    Abort
07 $\hat{m}^* \leftarrow (pk_{ID^*}, m^*)$
08 Return $(\hat{m}^*, \sigma^*, \omega_{ID^*})$

Procedure $\mathtt{PK}\,(ID)$ //Once per ID
09 $(\omega_{ID}, St) \leftarrow \mathsf{H}(St, ID)$
10 $pk_{ID} \leftarrow \mathsf{SWal}[\mathsf{H}].\mathsf{RandPK}(mpk, \omega_{ID})$
11 If $(pk_{ID}, \omega_{ID}) \in SSNKeys$: Abort
12 $SSNKeys[ID] \leftarrow (pk_{ID}, \omega_{ID})$
13 Return $pk_{ID}$

Procedure $\mathtt{WalSign}(m, ID)$
14 If $SSNKeys[ID] = \bot$: Return $\bot$
15 $(pk_{ID}, \omega_{ID}) \leftarrow SSNKeys[ID]$
16 $\hat{m} \leftarrow (pk_{ID}, m)$
17 $\sigma \leftarrow \mathtt{RSign}(\hat{m}, \omega_{ID})$
18 $Sigs[ID] \leftarrow Sigs[ID] \cup \{\hat{m}\}$
19 Return $\sigma$

Procedure $\mathsf{H}\,(s)$
20 If $H[s] \neq \bot$
21    Return $H[s]$
22 $\rho \xleftarrow{\$} \mathtt{Rand}$
23 $\varphi \xleftarrow{\$} \{0,1\}^{\kappa}$
24 $H[s] \leftarrow (\rho, \varphi)$
25 Return $H[s]$
```

Figure 8: $\mathsf{C}$'s simulation of $\mathbf{wunf}_{\mathsf{swal[H]}}$ to $\mathsf{A}$.

Setup. $\mathsf{C}$ first samples an initial state $St \xleftarrow{\$} \{0,1\}^{\kappa}$ and uses the public key $pk_{\mathsf{C}}$ from the $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{RSig}}$ game as the master public key $mpk$ in its simulation of $\mathbf{wunf}_{\mathsf{swal[H]}}$, i.e., it runs $\mathsf{A}$ on input $mpk = pk_{\mathsf{C}}, St$ in $\mathbf{wunf}_{\mathsf{swal[H]}}$. Throughout the game, $\mathsf{C}$ keeps updating $St$ each time it answers a query to $\mathsf{PK}$ from $\mathsf{A}$, as we describe below.

Simulation of Random Oracle Queries. $\mathsf{C}$ has to answer queries of the form $\mathsf{H}\,(s)$: Queries of this type are simulated in the programmable random oracle model as follows. When $\mathsf{A}$ makes a query of the form $\mathsf{H}(s)$, $\mathsf{C}$ returns $H[s]$ if it was already set. Otherwise, it proceeds as follows. Firstly, $\mathsf{C}$ fetches $\rho \xleftarrow{\$} \mathtt{Rand}$ by querying the oracle $\mathtt{Rand}$. Let us note that $\mathtt{Rand}$ internally updates $\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$. Secondly, $\mathsf{C}$ freshly samples $\varphi \xleftarrow{\$} \{0,1\}^{\kappa}$. Finally, $\mathsf{C}$ returns $H[s] = (\rho, \varphi)$.

Simulation of Public Key Queries. $\mathsf{C}$ answers a call of $\mathsf{A}$ to $\mathtt{PK}\,(ID)$ by computing $pk_{ID}$ as $pk_{ID} \leftarrow \mathsf{RSig}.\mathsf{RandPK}(pk_{\mathsf{C}}, \omega_{ID})$ where $(\omega_{ID}, St) \leftarrow \mathsf{H}\,(St, ID)$. If $\mathsf{C}$ detects a collision among $(pk_{ID}, \omega_{ID})$ and a value previously stored in $SSNKeys$, $\mathsf{C}$ aborts the simulation. Otherwise, it sets $SSNKeys[ID] \leftarrow (pk_{ID}, \omega_{ID})$ and returns $pk_{ID}$.

Simulation of Signing Queries. When $\mathsf{A}$ queries $\mathtt{WalSign}$ on input $(m, ID)$, $\mathsf{C}$ first recovers the pair $(pk_{ID}, \omega_{ID}) \leftarrow SSNKeys[ID]$ (it returns $\bot$ if $SSNKeys[ID] = \bot$). Next, $\mathsf{C}$ sets $\hat{m} = (pk_{ID}, m)$ and obtains $\sigma \xleftarrow{\$} \mathtt{RSign}(\hat{m}, \omega_{ID})$ by querying its own challenge signing oracle. Since $\mathsf{H}\,(\cdot,\cdot)$ is programmed as explained above by making a call to $\mathtt{Rand}$, we know that $\omega_{ID} \in \mathsf{RList}$. Hence, the query $\mathtt{RSign}(\hat{m}, \omega_{ID})$ is indeed valid, i.e, does not return $\bot$. From the definition of signature schemes with rerandomizable keys, $\mathsf{SWal}[\mathsf{H}].\mathsf{Verify}(pk_{ID}, \sigma, m) = \mathsf{RSig}.\mathsf{Verify}(\mathsf{RSig}.\mathsf{RandPK}(pk_{\mathsf{C}}, \omega_{ID}), \sigma, \hat{m})) = 1$, and so the simulated signatures are also correctly distributed.

Extracting the forgery. When $\mathsf{A}$ returns the tuple $(m^*, \sigma^*, ID^*)$, $\mathsf{C}$ aborts if it encounters any of the cases in which $\mathbf{G}_1$ would return 0 at this point (c.f. Figure 8). Otherwise it proceeds as follows. It first recovers the pair $(pk_{ID^*}, \omega_{ID^*}) \leftarrow SSNKeys[ID^*]$, and then returns $(\hat{m}^*, \sigma^*, \omega_{ID^*}) = ((pk_{ID^*}, m^*), \sigma^*, \omega_{ID^*})$. $(\hat{m}^*, \sigma^*, \omega_{ID^*})$ is a valid forgery in **uf-cma-hrk** game since:

1. From the simulation, we have that $pk_{ID^*} = pk_{\mathsf{C}} \cdot \omega_{ID^*}$ and $\omega_{ID^*} \in \mathsf{RList}$.
2. Since $\mathsf{swal[H].Verify}(pk_{ID^*}, \sigma^*, m^*) = 1$, it follows from the previous point that $\mathsf{RSig.Verify}(pk_{ID^*},$
$\sigma^*, \hat{m}^*) = 1$.
3. $m^* \notin Sigs[ID^*]$ implies that $\mathsf{C}$ never simulated a signature on message $m^*$ under public key $pk_{ID^*}$ to $\mathsf{A}$ before. Since every identifier has a unique key in $\mathbf{G}_1$, it follows that $\mathsf{C}$ never made a query of the form $\mathtt{RSign}(\hat{m}^*, \cdot)$ throughout its simulation. Consequently, $\hat{m}^* \notin Sigs$.

It is clear that $\mathsf{C}$ provides a perfect simulation of $\mathbf{G}_1$ to $\mathsf{A}$. Therefore, we obtain

$$\mathsf{Adv}^{\mathsf{A}}_{\mathbf{wunf},\mathsf{swal[H]}} \leq \mathsf{Adv}^{\mathsf{A}}_{\mathbf{G}_1,\mathsf{swal[H]}} + \frac{q^2}{p} = \mathsf{Adv}^{\mathsf{C}}_{\mathbf{uf\text{-}cma\text{-}hrk},\mathsf{RSig}} + \frac{q^2}{p},$$

which implies the theorem.

■

## 5 A Construction from ECDSA

In this section, we prove security of a construction based on the $\mathsf{EC}[\mathsf{H}]$ scheme (cf. Figure 11). For the following discussion, let $\mathbb{E}(par)$ denote an elliptic curve with base point $G$ and prime order $p$. Furthermore, assume hash functions $\mathsf{G}\colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_0\colon \{0,1\}^* \to \mathbb{Z}_p$(modeled as random oracles). We prove that a salted variant of the standard $\mathsf{EC}[\mathsf{H}]$ scheme, denoted as $\mathsf{REC[H]}$ and depicted in Figure 10, satisfies the notion of unforgeability under honestly rerandomized keys.

### 5.1 Security Analysis of Our Construction

We now proceed to the main technical contribution of this paper, where we analyze the notion of unforgeability under honestly rerandomized keys of the construction $\mathsf{REC}[\mathsf{H}_0]$ presented in Figure 10. We prove the following theorem.

**Theorem 5.1** *Let $\mathsf{G}, \mathsf{H}_0\colon \{0,1\}^* \to \mathbb{Z}_p$ be hash functions (modeled as random oracles). Let $\mathsf{A}$ be an algorithm that plays in game $\mathbf{uf\text{-}cma\text{-}hrk}_{\mathsf{REC}[\mathsf{H}_0]}$. Then there exists an algorithm $\mathsf{C}$ running in roughly the same time as $\mathsf{A}$, such that*

$$\mathsf{Adv}^{\mathsf{A}}_{\mathbf{uf\text{-}cma\text{-}hrk},\mathsf{REC}[\mathsf{H}_0]} \leq \mathsf{Adv}^{\mathsf{C}}_{\mathbf{uf\text{-}cma},\mathsf{EC}[\mathsf{G}]} + \frac{5q^2}{p},$$

*where $q$ is the number of random oracle queries that $\mathsf{A}$ makes.*

```
Algorithm
$\mathsf{EC}[\mathsf{H}].\mathsf{Gen}\,(par)$
00 $x \xleftarrow{\$} \mathbb{Z}_p$
01 $X \leftarrow x \cdot G$
02 $sk \leftarrow x$
03 $pk \leftarrow X$
04 Return $(pk, sk)$
```

```
Algorithm
$\mathsf{EC}[\mathsf{H}].\mathsf{Sign}\,(sk = x, m)$
05 $z \leftarrow \mathsf{H}\,(m)$
06 $t \xleftarrow{\$} \mathbb{Z}_p$
07 $(e_x, e_y) \leftarrow t \cdot G$
08 $r \leftarrow e_x \mod p$
09 If $r = 0 \mod p$
10     Goto Step 2
11 $s \leftarrow t^{-1}\,(z + rx) \mod p$
12 If $s = 0 \mod p$
13     Goto Step 2
14 Return $\sigma := (r, s)$
```

```
Algorithm
$\mathsf{EC}[\mathsf{H}].\mathsf{Verify}\,(pk = X, \sigma, m)$
15 Parse $(r, s) \leftarrow \sigma$
16 If $(r, s) \notin \mathbb{Z}_p$
17     Return 0
18 $w \leftarrow s^{-1} \mod p$
19 $z \leftarrow \mathsf{H}\,(m)$
20 $u_1 \leftarrow zw \mod p$
21 $u_2 \leftarrow rw \mod p$
22 $(e_x, e_y) \leftarrow u_1 \cdot G + u_2 \cdot X$
23 If $(e_x, e_y) = (0, 0)$
24     Return 0
25 Return $r = e_x \mod p$
```

Figure 9: $\mathsf{EC}\,[\mathsf{H}] = (\mathsf{EC}[\mathsf{H}].\mathsf{Gen}, \mathsf{EC}[\mathsf{H}].\mathsf{Sign}, \mathsf{EC}[\mathsf{H}].\mathsf{Verify})$: ECDSA Signature scheme relative to elliptic curve $\mathbb{E}$ and hash function $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p$.

Algorithm $\mathsf{Trf}[\mathsf{H}, \mathsf{G}]_{\mathsf{EC}}$ The algorithm $\mathsf{Trf}[\mathsf{H}, \mathsf{G}]_{\mathsf{EC}}$ which serves as an essential tool in our proof of Theorem 5.1 is presented in Figure 11. It takes as input two distinct messages $m_0, m_1$, two ECDSA public keys $X_0, X_1$ related via the offset $\omega$ and a signature $\sigma_1$ of $m_1$ wrt. public key $X_1$. The algorithm then carries out several consistency checks and if they pass outputs a valid signature $\sigma_0$ of $m_0$ under the related public key $X_0$. Notice that the two signatures $\sigma_0$ and $\sigma_1$ are valid with respect to different hash function, i.e., $\sigma_1$ is a signature with respect to $\mathsf{G}$, while $\sigma_0$ is a signature with respect to $\mathsf{H}$. This in particular implies that the transformation in $\mathsf{Trf}[\mathsf{H}, \mathsf{G}]_{\mathsf{EC}}$ does not result into a practical related key attack as both signatures $\sigma_0$ and $\sigma_1$ are valid with respect to different hash functions and the consistency checks in $\mathsf{Trf}[\mathsf{H}, \mathsf{G}]_{\mathsf{EC}}$ strongly restrict on what messages the related signature can be computed.[5] The following lemma formalizes the properties of $\mathsf{Trf}[\mathsf{H}, \mathsf{G}]_{\mathsf{EC}}$.

**Lemma 5.2** *Consider the algorithm* $\mathsf{Trf}[\mathsf{H}, \mathsf{G}]_{\mathsf{EC}}$ *in Figure 11. Suppose that:*

- $\omega = \mathsf{G}\,(m_1)\,/\mathsf{H}\,(m_0) \in \mathbb{Z}_p$,
- $X_0, X_1 \in \mathbb{E}$ *s.t.* $X_0 = x_0 \cdot G$ *and* $X_1 = \omega \cdot X_0$,
- $\mathsf{EC}[\mathsf{G}].\mathsf{Verify}(X_1, \sigma_1, m_1) = 1$,
- $\sigma_0 \leftarrow \mathsf{Trf}[\mathsf{H}, \mathsf{G}]_{\mathsf{EC}}(m_0, m_1, \sigma_1, \omega, X_0, X_1)$.

*Then* $\mathsf{EC}[\mathsf{H}].\mathsf{Verify}(X_0, \sigma_0, m_0) = 1$.

*Proof.* Let $\sigma_1 = (r, s_1)$ be a valid signature on $m_1$ relative to $\mathsf{G}$ and public key $X_1$, i.e., $\mathsf{EC}[\mathsf{G}].\mathsf{Verify}(X_1, \sigma_1, m_1) = 1$. We have to show that $\sigma_0 = (r, \frac{s_1}{\omega}) = \mathsf{Trf}[\mathsf{H}, \mathsf{G}]_{\mathsf{EC}}(m_0, m_1, \sigma_1, \omega, X_0, X_1)$

[5]The RKA against ECDSA can also be deployed when setting $\mathsf{H} = \mathsf{G}$. However, this attack is not particularly useful for our simulation argument. For the simulation argument we require to move signatures between different hash functions.

```
Algorithm $\mathsf{REC}[\mathsf{H}_0].\mathsf{Sign}\,(sk, m)$
00 $\psi \xleftarrow{\$} \{0,1\}^\kappa$
01 $\hat{m} \leftarrow (pk, \psi, m)$
02 $\sigma' \leftarrow \mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}\,(sk, \hat{m})$
03 Return $\sigma = (\psi, \sigma')$

Algorithm
$\mathsf{REC}[\mathsf{H}_0].\mathsf{Verify}\,(pk, \sigma, m)$
04 $(\psi, \sigma') \leftarrow \sigma$
05 $\hat{m} \leftarrow (pk, \psi, m)$
06 Return
$\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}\,(pk, \sigma', \hat{m})$

Algorithm
$\mathsf{REC}[\mathsf{H}_0].\mathsf{RandSK}\,(sk, \rho)$
00 $sk' \leftarrow sk \cdot \rho \bmod p$
01 Return $sk'$

Algorithm
$\mathsf{REC}[\mathsf{H}_0].\mathsf{RandPK}\,(pk, \rho)$
02 $pk' \leftarrow pk \cdot \rho$
03 Return $pk'$
```

Figure 10: Salted and key-prefixed version of the ECDSA signature scheme with perfectly rerandomizable keys $\mathsf{REC}[\mathsf{H}_0] := (\mathsf{REC}[\mathsf{H}_0].\mathsf{Gen} = \mathsf{EC}[\mathsf{H}_0].\mathsf{Gen}, \mathsf{REC}[\mathsf{H}_0].\mathsf{Sign}, \mathsf{REC}[\mathsf{H}_0].\mathsf{Verify}, \mathsf{REC}[\mathsf{H}_0].\mathsf{RandSK}, \mathsf{REC}[\mathsf{H}_0].\mathsf{RandPK})$ from the ECDSA signature scheme $\mathsf{EC}[\mathsf{H}_0]$. $\mathsf{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p$ denotes a hash function.

```
$\mathsf{Trf}[\mathsf{H},\mathsf{G}]_{\mathsf{EC}}(m_0, m_1, \sigma_1, \omega, X_0, X_1)$
00 $z_0 \leftarrow \mathsf{H}\,(m_0)$
01 $z_1 \leftarrow \mathsf{G}\,(m_1)$
02 If $\left(\mathsf{Verify}^{\mathsf{G}}(\sigma_1, m_1, X_1) = 0\right) \vee \left(\omega \neq \frac{z_1}{z_0} \vee X_1 \neq X_0 \cdot \omega\right)$ :
03     Return $\perp$
04 $(r, s_1) \leftarrow \sigma_1$
05 $s_0 \leftarrow \frac{s_1}{\omega} \bmod p$
06 $\sigma_0 \leftarrow (r, s_0)$
07 Return $\sigma_0$
```

Figure 11: Figure shows the $\mathsf{Trf}_{\mathsf{ECDSA}}$ algorithm for hash functions $\mathsf{H}, \mathsf{G} \colon \{0,1\}^* \to \mathbb{Z}_p$.

is a valid signature on $m_0$ relative to $\mathsf{H}$ and public key $X_0$, i.e., $\mathsf{EC}[\mathsf{H}].\mathsf{Verify}(X_0, \sigma_0, m_0) = 1$. To this end, let $z_1 = \mathsf{G}\,(m_1)$ and suppose that $s_1$ was computed as $s_1 = \frac{z_1 + r\omega x}{t}$ for some $t \in \mathbb{Z}_p$. We show that $\mathsf{EC}[\mathsf{H}].\mathsf{Verify}(X_0, \sigma_0, m_0) = 1$. The algorithm $\mathsf{EC}[\mathsf{H}].\mathsf{Verify}$ on input $(X_0, \sigma_0, m_0)$ first computes $w_0 = (s_0)^{-1} = \frac{\omega}{s_1} = \frac{\omega t}{z_1 + r\omega x} = \frac{\omega t}{\omega z_0 + r\omega x} = \frac{t}{z_0 + rx} = \frac{t}{\mathsf{H}(m_0) + rx}$, where the last equation follows, because $\mathsf{Trf}[\mathsf{H},\mathsf{G}]_{\mathsf{EC}}(m_0, m_1, \sigma_1, \omega, X_0, X_1)$ did not return $\perp$ (by the prerequisites of the lemma). Therefore, since $z_0 = z_1/\omega = \mathsf{G}\,(m_1)\,/\omega$, it must hold that $z_0 = \mathsf{H}\,(m_0)$.

$\mathsf{EC}[\mathsf{H}].\mathsf{Verify}$ next computes $u_{1,0} \equiv_p z_0 w_0 \equiv_p \mathsf{H}\,(m_0)\, w_0, u_{2,0} \equiv_p r w_0$ and

$$\begin{aligned} u_{1,0} \cdot G + u_{2,0} \cdot X_0 =& \mathsf{H}\left(m_0\right) w_0 \cdot G + r w_0 \cdot x \cdot G \\ =& \mathsf{H}\left(m_0\right) w_0 \cdot G + x r w_0 \cdot G \\ =& \left(w_0 \left(\mathsf{H}\left(m_0\right) + xr\right)\right) \cdot G \\ =& t \cdot G =: \left(e_x, e_y\right) \end{aligned} \tag{3}$$

To ensure that $\mathsf{EC}[\mathsf{H}].\mathsf{Verify}(X_0, \sigma_0, m_0) = 1$, it remains to show that $r \equiv_p e_x$, where $r$ is the first component of the signature. To this end, consider the computation performed via $\mathsf{EC}[\mathsf{G}].\mathsf{Verify}(X_1, \sigma_1, m_1)$. First, the algorithm computes

$$w_1 = (s_1)^{-1} = \frac{t}{z_1 + r\omega x} = \frac{t}{\mathsf{G}\left(m_1\right) + \omega r x}.$$

Next it computes $u_{1,1} \equiv_p z_1 w_1 \equiv_p \mathsf{G}\left(m_1\right) w_1, u_{2,1} \equiv_p r w_1$,

$$\begin{aligned} u_{1,1} \cdot G + u_{2,1} \cdot X_1 =& \mathsf{G}\left(m_1\right) w_1 \cdot G + r w_1 \cdot x\omega \cdot G \\ =& \mathsf{G}\left(m_1\right) w_1 \cdot G + x\omega r w_1 \cdot G \\ =& \left(w_1 (\mathsf{G}\left(m_1\right) + x\omega r)\right) \cdot G \\ =& t \cdot G = \left(e_x, e_y\right), \end{aligned} \tag{4}$$

Therefore, since $\mathsf{EC}[\mathsf{G}].\mathsf{Verify}(X_1, \sigma_1, m_1) = 1$, we have that $r \equiv_p e_x$. It follows now that also $\mathsf{EC}[\mathsf{H}].\mathsf{Verify}(X_0, \sigma_0, m_0) = 1$. ■

Before giving the formal proof of Theorem 5.1, we give some intuition about the main difficulties that we need to overcome. At a high level, the idea is to reduce the security of the salted ECDSA construction $\mathsf{REC}[\mathsf{H}_0]$(relative to **uf-cma-hrk**$_{\mathsf{REC}[\mathsf{H}_0]}$) to the security of $\mathsf{EC}[\mathsf{G}]$ (relative to **uf-cma**$_{\mathsf{EC}[\mathsf{G}]}$). As such, the proof consists mainly of the description of a reduction $\mathsf{C}$ trying to come up with a valid forgery in order to win the game **uf-cma**$_{\mathsf{EC}[\mathsf{G}]}$ by simulating **uf-cma-hrk**$_{\mathsf{REC}[\mathsf{H}_0]}$ to the adversary $\mathsf{A}$. $\mathsf{C}$ obtains a public key $pk_{\mathsf{C}}$ from its challenger and can query a signing oracle $\mathtt{SignO}(\cdot)$ which provides signatures on messages of $\mathsf{C}$'s choice under $pk_{\mathsf{C}}$. It also can query the random oracle $\mathsf{G}$. $\mathsf{C}$'s goal is to simulate the oracles in the **uf-cma-hrk**$_{\mathsf{REC}[\mathsf{H}_0]}$ experiment and to suitably embed $pk_{\mathsf{C}}$ into the key $pk^*$ under which $\mathsf{A}$ eventually returns a forgery $(\sigma^*, m^*, \rho^*)$. The hope is that it can use $(\sigma^*, m^*, \rho^*)$ to win **uf-cma**$_{\mathsf{EC}[\mathsf{G}]}$.

$\mathsf{C}$ embeds $pk_{\mathsf{C}}$ as $\mathsf{A}$'s input public key $pk$. This allows $\mathsf{C}$ to rerandomize $pk$ into $pk'$ which is a crucial requirement for answering oracle queries posed by $\mathsf{A}$. However, there are several issues with this approach. Firstly, $\mathsf{C}$ is not aware of any of the secret keys for the public keys generated as $pk' \leftarrow pk \cdot \rho = pk_{\mathsf{C}} \cdot \rho$. Secondly, the signatures obtained by making a query $\mathtt{SignO}(\cdot)$ to $\mathsf{C}$'s challenger are only valid under $pk_{\mathsf{C}}$, so cannot be directly used to simulate signing queries of the form $\mathtt{RSign}(m, \rho)$ to $\mathsf{A}$. To solve the latter problem, $\mathsf{C}$ can convert a signature of the form $\sigma \leftarrow \mathtt{SignO}(m')$ under $pk_{\mathsf{C}}$ into a signature $\hat{\sigma}$ under $pk'$, and on message $\hat{m}$ using algorithm $\mathsf{Trf}[\mathsf{H}_0, \mathsf{G}]_{\mathsf{EC}}$. Here, $pk_{\mathsf{C}}$ and $pk'$ are related as $pk_{\mathsf{C}} = pk' \cdot \rho^{-1}$, and $\rho^{-1} = \frac{G(m')}{\mathsf{H}_0(\hat{m})}$. Similarly, it can convert a forgery $(\sigma^*, m^*)$ under an arbitrary related key $pk^*$ into a forgery that is valid under $pk_{\mathsf{C}}$, using $\mathsf{Trf}[\mathsf{G}, \mathsf{H}_0]_{\mathsf{EC}}$ in the "reverse" direction (note the inverted order of $\mathsf{H}_0$ and $\mathsf{G}$). To satisfy the relationship between the (hash of) messages involved in the signatures, $\mathsf{C}$ needs to carefully program the random oracle $\mathsf{H}_0$ to make everything consistent with what $\mathsf{A}$ expects to see. This gets even more complicated because $\mathsf{A}$ can make direct queries to the programmed oracle $\mathsf{H}_0(\cdot)$ where each of the queries should look random from $\mathsf{A}$'s point of view.

We now turn to the formal proof of Theorem 5.1.

*Proof.* Consider an adversary $\mathsf{A}$ playing in Game $\textbf{uf-cma-hrk}_{\mathsf{REC}[\mathsf{H}_0]}$. As such $\mathsf{A}$ is granted access to the oracles $\mathtt{Rand}, \mathtt{RSign}$, and the random oracle $\mathsf{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p$. In the following, we use that $2^\kappa \leq p$. We prove the statement via a sequence of games. Each game $\mathbf{G}_{i(i>0)}$ is presented in Figure 13 via the description of the oracles that are modified with respect to the previous game $\mathbf{G}_{i-1}$. The exact differences of game $\mathbf{G}_i$ to game $\mathbf{G}_{i-1}$ are highlighted in the form of boxed pseudocode. Moreover, we denote by $E_{i-1,i}$ a difference event, where the indices of the event correspond to games $\mathbf{G}_{i-1}, \mathbf{G}_i$ that are affected by the event.

Game $\mathbf{G}_0$: The initial game $\mathbf{G}_0$ (Figure 12) corresponds to $\textbf{uf-cma-hrk}_{\mathsf{REC}[\mathsf{H}_0]}$, i.e., $\mathbf{G}_0 := \textbf{uf-cma-hrk}_{\mathsf{REC}[\mathsf{H}_0]}$. Since we are in the random oracle model, we explicitly list the random oracle $\mathsf{H}_0$ in $\mathbf{G}_0$.

Game $\mathbf{G}_1$: In $\mathbf{G}_1$, the way that random oracle queries to $\mathsf{H}_0$ from $\mathsf{A}$ are answered, is internally modified as follows. To answer queries to $\mathsf{H}_0$, $\mathbf{G}_1$ internally keeps two lists $H_0$ and $H_0'$ which it programs throughout its interaction with $\mathsf{A}$. Depending on whether a queried message $m$ contains as part of its prefix a public key $pk'$, it programs $H_0[m]$ and $H_0'[m]$ in two different possible ways. Note that $pk'$ is the result of rerandomizing $pk$ as $pk' = pk \cdot \rho$, where $\rho \leftarrow \mathtt{Rand}(\rho \in \mathsf{RList})$ is a previous answer to a oracle query $\mathtt{Rand}$. We now analyze the three types of queries to $\mathsf{H}_0$ that can occur.

- $\mathsf{H}_0[m] \neq \perp$: In this case, $\mathbf{G}_1$ returns $H_0[m]$.
- $\mathsf{H}_0[m] = \perp$ and $m$ is of the form $m = (\cdot, pk', \cdot)$, s.t. $pk' = pk \cdot \rho$ for some $\rho \in \mathsf{RList}$: In this case, $\mathbf{G}_1$ computes $h \leftarrow \mathsf{G}(ctr)$, where $ctr \xleftarrow{\$} \{0,1\}^\kappa$. Consequently, $\mathbf{G}_1$ sets $H_0[m] \leftarrow \rho \cdot h \mod p$ and $H_0'[m] \leftarrow ctr$. It returns $H_0[m]$.
- Otherwise, $\mathbf{G}_1$ samples $h \xleftarrow{\$} \mathbb{Z}_p$ and sets $H_0[m] \leftarrow h$, $H_0'[m] \leftarrow \epsilon$. It then returns $H_0[m]$.

It is easy to see that all answers for queries to $\mathsf{H}_0$ that $\mathbf{G}_1$ returns are uniformly distributed from $\mathsf{A}$'s perspective. This follows from the uniformity of output $h$ computed via random oracle $\mathsf{G}$. Therefore, $\mathbf{G}_1$ behaves exactly as $\mathbf{G}_0$.

Game $\mathbf{G}_2$: In $\mathbf{G}_2$, the way in which queries to $\mathtt{Rand}$ are answered, is internally modified as follows. When $\mathsf{A}$ asks a query of the form $\mathtt{Rand}$, the game aborts if there exists a message of the form $m = \left(\cdot, pk', \cdot\right)$ for which $H_0'[m]$ evaluates to $\epsilon$ and where $pk'$ is the (rerandomized) key that corresponds to the return value $\rho$ of $\mathtt{Rand}$, i.e., $pk' = pk \cdot \rho$. The following claim bounds the probability of such an abort scenario.

**Claim 5.3** Let $E_{1,2}$ denote the event that $\mathbf{G}_2$ aborts during a $\mathtt{Rand}$ query, for which $H_0'[m]$ evaluates to $\epsilon$, where $m = \left(\cdot, pk', \cdot\right)$. Then $\Pr[E_{1,2}] \leq \frac{q^2}{p}$.

```
Game $\mathbf{G}_0$
00 $\mathsf{RList} \leftarrow \{\epsilon\}$
01 $\mathsf{bad} \leftarrow \mathsf{false}$
02 $(sk, pk) \xleftarrow{\$} \mathsf{REC}[\mathsf{H}_0].\mathsf{Gen}\,(par)$
03 $(m^*, \sigma^*, \rho^*) \xleftarrow{\$} \mathsf{C}^{\mathsf{H}_0,\mathtt{Rand},\mathtt{RSign}}\,(pk)$
04 $pk^* \leftarrow pk \cdot \rho^*$
05 If $m^* \in Sigs$: $\mathsf{bad} \leftarrow \mathsf{true}$
06 If $\rho^* \notin \mathsf{RList}$: $\mathsf{bad} \leftarrow \mathsf{true}$
07 $b \leftarrow \mathsf{REC}[\mathsf{H}_0].\mathsf{Verify}\,(pk^*, \sigma^*, m^*)$
08 Return $b \wedge \neg\mathsf{bad}$

Oracle $\mathtt{Rand}$
09 $\rho \xleftarrow{\$} \chi$
10 $\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$
11 Return $\rho$

Oracle $\mathtt{RSign}\,(m, \rho)$
12 If $\rho \notin \mathsf{RList}$: Return $\perp$
13 $\psi \xleftarrow{\$} \{0,1\}^\kappa$
14 $pk' \leftarrow pk \cdot \rho \mod p$
15 $sk' \leftarrow sk \cdot \rho \mod p$
16 $\hat{m} \leftarrow (\psi, pk', m)$
17 $\sigma \leftarrow \mathsf{REC}[\mathsf{H}_0].\mathsf{Sign}\left(\hat{m}, sk'\right)$
18 $Sigs \leftarrow Sigs \cup \{m\}$
19 Return $(\psi, \sigma)$

Oracle $\mathsf{H}_0\,(m)$
20 If $H_0\,[m] \neq \perp$
21     Return $H_0\,[m]$
22 $H_0\,[m] \xleftarrow{\$} \mathbb{Z}_p$
23 Return $H_0\,[m]$
```

Figure 12: Game $\mathbf{G}_0 = \textbf{uf-cma-hrk}_{\mathsf{REC}[\mathsf{H}_0]}$ with adversary $\mathsf{C}$.

```
Oracle $\mathsf{H}_0\,(m)$ in $\mathbf{G}_1$
00 If $H_0\,[m] \neq \perp$
01     Return $H_0\,[m]$
02 Parse $m$ as $\left(\cdot, pk', \cdot\right)$
03 If $\exists \rho \in \mathsf{RList} : pk' = pk \cdot \rho$
04     $ctr \leftarrow \{0,1\}^\kappa$
05     $h \leftarrow \mathsf{G}\,(ctr)$
06     $H_0\,[m] \leftarrow \rho \cdot h \mod p$
07     $H_0'\,[m] \leftarrow ctr$
08 Else
09     $h \xleftarrow{\$} \mathbb{Z}_p$
10     $H_0\,[m] \leftarrow h$
11     $H_0'\,[m] \leftarrow \epsilon$
12 Return $H_0\,[m]$

Oracle $\mathtt{Rand}$ in $\mathbf{G}_2$
13 $\rho \xleftarrow{\$} \chi$
14 $\boxed{pk' \leftarrow pk \cdot \rho}$
15 $\boxed{\forall m = \left(\cdot, pk', \cdot\right) :}$
16     $\boxed{\text{If } H_0'\,[m] = \epsilon\text{: Abort}}$
17 $\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$
18 Return $\rho$

Oracle $\mathtt{RSign}\,(m, \rho)$ in $\mathbf{G}_3$
19 If $\rho \notin \mathsf{RList}$: Return $\perp$
20 $\psi \xleftarrow{\$} \{0,1\}^\kappa$
21 $pk' \leftarrow pk \cdot \rho \mod p$
22 $sk' \leftarrow sk \cdot \rho \mod p$
23 $\hat{m} \leftarrow (\psi, pk', m)$
24 $\boxed{\text{If } H_0'[\hat{m}] \neq \perp\text{: Abort}}$
25 $\hat{\sigma} \leftarrow \mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}(sk', \hat{m})$
26 $Sigs \leftarrow Sigs \cup \{m\}$
27 Return $(\psi, \hat{\sigma})$

Oracle $\mathtt{RSign}\,(m, \rho)$ in $\mathbf{G}_4$
28 If $\rho \notin \mathsf{RList}$: Return $\perp$
29 $\psi \xleftarrow{\$} \{0,1\}^\kappa$
30 $pk' \leftarrow pk \cdot \rho$
31 $\hat{m} \leftarrow (\psi, pk', m)$
32 If $H_0'[\hat{m}] \neq \perp$: Abort
33 $\boxed{\text{Query } \mathsf{H}_0(\hat{m})}$
34 $\boxed{m' \leftarrow H_0'[\hat{m}]}$
35 $\boxed{\sigma' \leftarrow \mathsf{EC}[\mathsf{G}].\mathsf{Sign}(sk, m')}$
36 $\boxed{\hat{\sigma} \leftarrow \mathsf{Trf}[\mathsf{H}_0, \mathsf{G}]_{\mathsf{EC}}(\hat{m}, m', \sigma', \rho^{-1}, pk', pk)}$
37 $Sigs \leftarrow Sigs \cup \{m\}$
38 Return $(\psi, \hat{\sigma})$

main in $\mathbf{G}_5$
39 $(pk, sk) \leftarrow \mathsf{EC}.\mathsf{Gen}(par)$
40 $(m^*, \sigma^*, \rho^*) \xleftarrow{\$} \mathsf{A}^{\mathsf{H}_0,\mathtt{Rand},\mathtt{RSign}}(pk)$
41 $pk^* \leftarrow pk \cdot \rho^*$
42 $\boxed{\hat{m}^* \leftarrow (\psi, pk^*, m^*)}$
43 $\boxed{\text{If } H_0'[\hat{m}^*] = \epsilon\text{: Abort}}$
44 If $m^* \in Sigs$: $\mathsf{bad} \leftarrow \mathsf{true}$
45 If $\rho^* \notin \mathsf{RList}$: $\mathsf{bad} \leftarrow \mathsf{true}$
46 $b \leftarrow \mathsf{REC}[\mathsf{H}_0].\mathsf{Verify}\,(pk^*, \sigma^*, m^*)$
47 Return $b \wedge \neg\mathsf{bad}$
```

Figure 13: Games $\mathbf{G}_1$-$\mathbf{G}_5$

*Proof.* During any particular call to the oracle `Rand`, this event can only occur if $\mathsf{A}$ has already made a query of the form $\mathsf{H}_0(m)$, where $m = (\cdot, pk', \cdot)$ (prior to the oracle `Rand` returning the value $\rho$ for this query). Since $\mathsf{RList}$ contains at most $q$ values at any point during the game, any of them coincide with the (uniformly chosen) value $\rho$ with probability at most $\frac{q}{p}$. Since keys are uniquely rerandomizable, a query of the form $\mathsf{H}_0(m)$ thus also has probability at most $\frac{q}{p}$ of having been made prior to this particular call to `Rand`. Since there at most $q$ queries to `Rand`, it follows that $\Pr[E_{1,2}] \leq \frac{q^2}{p}$. ∎

Since the games $\mathbf{G}_1$, $\mathbf{G}_2$ are equivalent unless the event $\Pr[E_{1,2}]$ occurs, $\mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_2},\mathsf{REC}[\mathsf{H_0}]} \leq \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_1},\mathsf{REC}[\mathsf{H_0}]} + \Pr[E_{1,2}] \leq \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_1},\mathsf{REC}[\mathsf{H_0}]} + \frac{q^2}{p}$.

Game $\mathbf{G}_3$: In $\mathbf{G}_3$, the way in which signing queries from $\mathsf{A}$ are answered, is internally modified as follows. When $\mathsf{A}$ makes a query of the form `RSign` $(m, \rho)$, $\mathbf{G}_3$ first checks whether $\rho \in \mathsf{RList}$ and if not, returns $\perp$. Otherwise, it samples $\psi \xleftarrow{\$} \{0,1\}^\kappa$, computes $pk' \leftarrow pk \cdot \rho, sk' := sk \cdot \rho \mod p$, and sets $\hat{m} \leftarrow \left(\psi, pk', m\right)$. If the list $H_0'$ already contains an element for $H_0'[\hat{m}]$, i.e. $H_0'[\hat{m}] \neq \perp$, then the game aborts at this point. Otherwise, a signature $\hat{\sigma}$ is computed as $\hat{\sigma} \xleftarrow{\$} \mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}\left(sk', \hat{m}\right)$. $\mathbf{G}_3$ subsequently returns $(\psi, \hat{\sigma})$. The only difference of game $\mathbf{G}_3$ to $\mathbf{G}_2$, is that game $\mathbf{G}_3$ potentially aborts at line 24 if $H_0'[\hat{m}] \neq \perp$. Hence, we obtain the following claim.

**Claim 5.4** Let $E_{2,3}$ denote the event that $\mathbf{G}_3$ aborts during a signing query, when $H_0[\hat{m}] \neq \perp$, where $\hat{m} = (\psi, pk', m)$. Then $\Pr[E_{2,3}] \leq \frac{q^2}{p}$.

*Proof.* This event can only happen when $\mathsf{A}$ makes a correct guess of the message $\hat{m}$ and makes a query of the form $\mathsf{H}_0(\hat{m})$ prior to a `RSign`$(m, \rho)$ query. $\hat{m}$ is constructed as $\hat{m} = (\psi, pk', m)$ where $\psi$ is uniformly sampled as $\psi \xleftarrow{\$} \{0,1\}^\kappa$. Since $\mathsf{A}$ makes atmost $q$ queries to $\mathsf{H}_0(\cdot)$, $\mathsf{A}$ can correctly guess a particular $\hat{m} = (\psi, pk', m)$ for a fixed $m$, with probability $\frac{q}{p}$. Since $\mathsf{A}$ makes at most $q$ signing queries to `RSign` $(m, \rho)$, $\mathsf{A}$ can correctly guess any $\hat{m}$ with a probability bounded by $\sum_{i=1}^{q} \frac{q}{p} \leq \frac{q^2}{p}$. ∎

Since the games $\mathbf{G}_2$, $\mathbf{G}_3$ are equivalent unless the event $\Pr[E_{2,3}]$ occurs, $\mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_2},\mathsf{REC}[\mathsf{H_0}]} \leq \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_3},\mathsf{REC}[\mathsf{H_0}]} + \Pr[E_{2,3}] \leq \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_3},\mathsf{REC}[\mathsf{H_0}]} + \frac{q^2}{p}$.

Game $\mathbf{G}_4$: In $\mathbf{G}_4$, the way that signing queries from $\mathsf{A}$ are answered, is again internally modified as follows. When $\mathsf{A}$ makes a query of the form `RSign`$(m, \rho)$, $\mathbf{G}_4$ first checks whether $\rho \in \mathsf{RList}$ and if not, returns $\perp$. Otherwise, it samples $\psi \xleftarrow{\$} \{0,1\}^\kappa$ computes $pk' \leftarrow pk \cdot \rho$, and sets $\hat{m} \leftarrow \left(\psi, pk', m\right)$. The game aborts at this point if $H_0[\hat{m}] \neq \perp$. If it does not abort, it internally queries $\mathsf{H}_0$ on input message $\hat{m}$. This means it queries $h \leftarrow \mathsf{G}(ctr)$, where $ctr \xleftarrow{\$} \{0,1\}^\kappa$. $\mathbf{G}_4$ internally sets $H_0[\hat{m}] \leftarrow \rho \cdot h \mod p$ and stores $H_0'[\hat{m}] \leftarrow ctr$. After making the query to $\mathsf{H}_0$, $\mathbf{G}_4$ fetches $m' \leftarrow H_0'[\hat{m}]$, where $m'$ was set to $ctr$ during $\mathsf{H}_0$ query. Since $sk$ is known to the game, it can now compute the signature $\sigma'$ as $\sigma' \xleftarrow{\$} \mathsf{EC}[\mathsf{G}].\mathsf{Sign}(sk, m')$. Finally, it computes and returns the signature $\hat{\sigma}$ as $\hat{\sigma} \leftarrow \mathsf{Trf}[\mathsf{H}_0, \mathsf{G}]_{\mathsf{EC}}(\hat{m}, m', \sigma', \rho^{-1}, pk', pk)$, where $pk = pk' \cdot \rho^{-1}$.

**Claim 5.5** $\mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_3},\mathsf{REC}[\mathsf{H_0}]} = \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_4},\mathsf{REC}[\mathsf{H_0}]}$

*Proof.* We argue that in both games, the answers to signing queries are identically distributed. To this end, we analyze how $\mathbf{G}_4$ replies to a query of the form `RSign` $(m, \rho)$.

First note that the explicit query to $\mathsf{H}_0$ at line 33 is implicitly also made in $\mathbf{G}_3$ at line 25 and therefore does not change the behaviour of $\mathbf{G}_4$ (compared to $\mathbf{G}_3$). Next, $\mathbf{G}_4$ derives signature $(\psi, \hat{\sigma})$ on input $(m, \rho)$ as $\hat{\sigma} \leftarrow \mathsf{Trf}[\mathsf{H}_0, \mathsf{G}]_{\mathsf{EC}}(\hat{m}, m', \sigma', \rho^{-1}, pk', pk)$, where $m' = H_0'[\hat{m}], pk = pk' \cdot \rho^{-1}, \mathsf{EC}[\mathsf{G}].\mathsf{Verify}(pk, \sigma', m') = 1$, and $\frac{\mathsf{G}(m')}{H_0[\hat{m}]} = \frac{h'}{H_0[\hat{m}]} = \frac{h'}{\rho \cdot h'} = \rho^{-1}$ mod $p$. It follows from Lemma 5.2 that $\hat{\sigma}$ constitutes a correct signature on message $\hat{m}$ and under public key $pk'$ relative to $\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}$. It follows immediately that the signature $(\psi, \hat{\sigma})$ constitutes a valid signature relative to $\mathsf{REC}[\mathsf{H}_0].\mathsf{Verify}$. Moreover, the value of $\psi$ is identically distributed in games $\mathbf{G}_3, \mathbf{G}_4$, which concludes the proof. ■

GAME $\mathbf{G}_5$: $\mathbf{G}_5$ behaves identically to $\mathbf{G}_4$ except for the following modification in the main procedure: Upon receiving a forgery of the form $(m^*, \sigma^* = (\psi, \hat{\sigma}), \rho^*)$ from $\mathsf{A}$, it sets $\hat{m}^* \leftarrow (\psi, pk^*, m^*)$ and aborts if $H_0'[\hat{m}^*] = \epsilon$.

**Claim 5.6** Let $E_{4,5}$ be the event that $\mathbf{G}_5$ aborts if $H_0'[\hat{m}^*] = \epsilon$, where $\hat{m}^* = (\psi, pk^*, m^*)$. Then $\Pr[E_{4,5}] \le \frac{q^2}{p}$.

*Proof.* The only way this event can happen, is if $\mathsf{A}$ manages to make a query of the form $\mathsf{H}_0(\hat{m}^*)$ before querying `Rand` to obtain the corresponding value of $\rho^*$. The proof of this claim follows in a similar way as the corresponding proof in claim 5.3. ■

Since the games $\mathbf{G}_4$, $\mathbf{G}_5$ are equivalent unless event $E_{4,5}$ occurs, $\mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_4},\mathsf{REC}[\mathsf{H}_0]} \le \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_5},\mathsf{REC}[\mathsf{H}_0]} + \frac{q^2}{p}$.

REDUCTION TO UF-CMA SECURITY. We describe an algorithm $\mathsf{C}^{\mathtt{SignO},\mathsf{G}}$ (depicted in Figure 14) that plays in the $\mathbf{uf\text{-}cma}_{\mathsf{EC}[\mathsf{G}]}$ game. $\mathsf{C}$ obtains as input a public key $pk_{\mathsf{C}}$ and is given access to the signing oracle `SignO` to obtain signatures under $pk_{\mathsf{C}}$ under messages of its choice. Furthermore, $\mathsf{C}$ has access to the random oracle $\mathsf{G}$. $\mathsf{C}$ runs $\mathsf{A}$ on input $pk_{\mathsf{C}}$ and simulates $\mathbf{G}_5$ to $\mathsf{A}$ as described in Figure 14.

SIMULATION OF RANDOMNESS QUERIES. Queries to `Rand` from $\mathsf{A}$ do not require knowledge of the secret key corresponding to $pk_{\mathsf{C}}$ and hence are straight forward to simulate.

SIMULATION OF RANDOM ORACLE QUERIES. $\mathsf{C}$'s simulation of random oracle queries coincides with the above programming strategy that is already internally present in $\mathbf{G}_5$.

SIMULATION OF SIGNING QUERIES. Recall that in $\mathbf{G}_5$, queries of the form `RSign` $(m, \rho)$ internally prompt the computation of signature $\sigma' = \mathsf{EC}[\mathsf{G}].\mathsf{Sign}(sk_{\mathsf{C}}, m')$, where $m' \leftarrow ctr$. Since $\mathsf{C}$ does not know $sk_{\mathsf{C}}$, it needs to compute $\sigma'$ via a call to its signing oracle, i.e., as $\sigma' \leftarrow \mathtt{SignO}(m')$. Other than that $\mathsf{C}$ simulates such a query exactly as internally done for $\mathbf{G}_5$.

EXTRACTING THE FORGERY. When the tuple $(m^*, \sigma^*, \rho^*)$ is returned as an answer from $\mathsf{A}$, $\mathsf{C}$ first parses it as $(m^*, \sigma^*, \rho^*) = (m^*, (\psi^*, \hat{\sigma}^*), \rho^*)$, checks whether it constitutes a valid forgery, and aborts otherwise (note that in this case, $\mathbf{G}_5$ would return 0, so $\mathsf{C}$ can safely abort). In case $\mathsf{C}$ does not abort, it computes $pk^* = pk_{\mathsf{C}} \cdot \rho^*$, where $pk^*$ is the public key under which $\mathsf{A}$'s forgery is valid. $\mathsf{C}$ computes $\hat{m}^* \leftarrow (\psi^*, pk^*, m^*)$ and if $H_0'[\hat{m}^*] = \epsilon$, it aborts. Otherwise, $\mathsf{C}$ fetches $m' \leftarrow H_0'[\hat{m}^*]$ and computes

$$\sigma' \leftarrow \mathsf{Trf}[\mathsf{G}, \mathsf{H}_0]_{\mathsf{ECDSA}}(m', \hat{m}^*, \hat{\sigma}^*, \rho^*, pk_{\mathsf{C}}, pk^*)\,.$$

```
main $\mathsf{C}^{\mathtt{SignO},\mathsf{G}}(pk_{\mathsf{C}})$
$(m^*, \sigma^*, \rho^*) \xleftarrow{\$} \mathsf{A}^{\mathsf{H}_0,\mathtt{Rand},\mathtt{RSign}}(pk_{\mathsf{C}})$
$(\psi, \hat{\sigma}) \leftarrow \sigma^*$
$pk^* \leftarrow pk \cdot \rho^*$
$\hat{m}^* \leftarrow (\psi, pk^*, m^*)$
If $H'_0[\hat{m}^*] = \epsilon$: Abort
If $m^* \in Sigs$: $\mathsf{bad} \leftarrow \mathsf{true}$
If $\rho^* \notin \mathsf{RList}$:
    $\mathsf{bad} \leftarrow \mathsf{true}$
$b \leftarrow \mathsf{REC}[\mathsf{H}_0].\mathsf{Verify}\,(pk^*, \sigma^*, m^*)$
If $\neg b \lor \mathsf{bad}$: Abort
$m' \leftarrow H'_0[\hat{m}^*]$
$\mathtt{tmp} \leftarrow (m', \hat{m}^*, \hat{\sigma}^*, \rho^*, pk_{\mathsf{C}}, pk^*)$
$\sigma' \leftarrow \mathsf{Trf}[\mathsf{G}, \mathsf{H}_0]_{\mathsf{EC}}(\mathtt{tmp})$
Return $(m', \sigma')$
```

```
Procedure $\mathtt{Rand}$
$\rho \xleftarrow{\$} \chi$
$pk' \leftarrow pk \cdot \rho$
$\forall m = \left(\cdot, pk', \cdot\right)$ :
    If $H'_0\,[m] = \epsilon$: Abort
$\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$
Return $\rho$
```

```
Procedure $\mathtt{RSign}\,(m, \rho)$
If $\rho \notin \mathsf{RList}$: Return $\perp$
$\psi \xleftarrow{\$} \{0,1\}^\kappa$
$pk' \leftarrow pk \cdot \rho$
$\hat{m} \leftarrow (\psi, pk', m)$
If $H'_0[\hat{m}] \neq \perp$: Abort
Query $\mathsf{H}_0(\hat{m})$
$m' \leftarrow H'_0[\hat{m}]$
$\sigma' \leftarrow \mathtt{SignO}(m')$
$\mathtt{tmp} \leftarrow (\hat{m}, m', \sigma', \rho^{-1}, pk', pk_{\mathsf{C}})$
$\hat{\sigma} \leftarrow \mathsf{Trf}[\mathsf{H}_0, \mathsf{G}]_{\mathsf{EC}}(\mathtt{tmp})$
$Sigs \leftarrow Sigs \cup \{m\}$
Return $(\psi, \hat{\sigma})$
```

```
Procedure $\mathsf{H}_0\,(m)$
If $H_0\,[m] \neq \perp$
    Return $H_0\,[m]$
Parse $m$ as $\left(\cdot, pk', \cdot\right)$
If $\exists \rho \in \mathsf{RList} : pk' = pk \cdot \rho$
    $ctr \leftarrow \{0,1\}^\kappa$
    $h \leftarrow \mathsf{G}\,(ctr)$
    $H_0\,[m] \leftarrow \rho \cdot h \mod p$
    $H'_0\,[m] \leftarrow ctr$
Else
    $h \xleftarrow{\$} \mathbb{Z}_p$
    $H_0\,[m] \leftarrow h$
    $H'_0\,[m] \leftarrow \epsilon$
Return $H_0\,[m]$
```

Figure 14: Reduction to UF-CMA game.

Since $H_0[\hat{m}^*] = \mathsf{G}(H_0'[\hat{m}^*]) \cdot \rho^* = \mathsf{G}(m') \cdot \rho^*$, we have that $\frac{H_0[\hat{m}^*]}{\mathsf{G}(m')} = \frac{\mathsf{G}(m') \cdot \rho^*}{\mathsf{G}(m')} = \rho^*$. Together with $pk^* = pk_{\mathsf{C}} \cdot \rho^*$ and $\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}(pk^*, \hat{\sigma}^*, \hat{m}^*) = 1$, Lemma 5.2 implies that

$$\mathsf{EC}[\mathsf{G}].\mathsf{Verify}(pk_{\mathsf{C}}, \sigma', m') = 1.$$

**Claim 5.7** $(m', \sigma')$ constitutes a valid forgery in $\mathbf{uf\text{-}cma}_{\mathsf{EC}[\mathsf{G}]}$ with probability $1 - q^2/p$.

*Proof.* We have to show that the query `SignO`$(m')$ was not made by $\mathsf{C}$ during its simulation and hence $(m', \sigma')$ is a valid forgery in $\mathbf{uf\text{-}cma}_{\mathsf{EC}[\mathsf{G}]}$. Note that $\mathsf{A}$ has not made a query of the form `RSign`$(m^*, \rho^*)$ throughout the simulation. Namely, if it had, $(m^*, \sigma^*, \rho^*)$ would not constitute a valid forgery in $\mathbf{G}_5$ and the simulation would have aborted at this point. This implies that $\mathsf{C}$ never had to simulate a query `RSign`$(m^*, \rho^*)$ to $\mathsf{A}$ which entailed a $\mathsf{H}_0$ query on message $\hat{m}^* \leftarrow (\psi^*, pk^*, m^*)$. Hence, $m'$ associated with query $\mathsf{H}_0(\hat{m}^*)$ was not queried by $\mathsf{C}$ to the oracle `SignO` in any query of the form `RSign`$(m, \rho)$ with $m \neq m^*$ unless there exist (any) two values $m_1, m_2$ s.t. $H_0'[m_1] = H_0'[m_2] \neq \perp$. It is easy to see that this happens with probability at most $q^2/p$ during $\mathsf{C}$'s simulation, since all values that $\mathsf{C}$ queries to the oracle `SignO` are sampled independently and uniformly at random from $\{0,1\}^\kappa$. ■

From claims 5.3-5.6, we have $\mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_0},\mathsf{REC}[\mathsf{H}_0]} \leq \mathbf{Adv}^{\mathsf{A}}_{\mathbf{G_5},\mathsf{REC}[\mathsf{H}_0]} + \frac{4q^2}{p}$. Since $\mathsf{C}$ provides a perfect simulation of $\mathbf{G}_5$ to $\mathsf{A}$ up to an error of $q^2/p$, as shown in the previous claim, we obtain

$$\mathsf{Adv}^{\mathsf{A}}_{\mathbf{uf\text{-}cma\text{-}hrk},\mathsf{REC}[\mathsf{H}_0]} \leq \mathsf{Adv}^{\mathsf{A}}_{\mathbf{G}_5} + \frac{4q^2}{p} \leq \mathsf{Adv}^{\mathsf{C}}_{\mathbf{uf\text{-}cma},\mathsf{EC}[\mathsf{G}]} + \frac{4q^2}{p},$$

which implies the theorem.

■

## 6 Practical Considerations

Synchronizing hot/cold wallet. To achieve correctness according to Definition 3.2, the cold wallet and hot wallet (party A and party B in Fig. 3) respectively, need to derive their keys in the same (ordered) sequence. Fortunately, this can be realized easily in practice. A simple solution is to use an increasing counter for every freshly derived pair of session keys in place of the *ID* argument. In this case, no additional synchronization between the hot and cold wallet is necessary. However, it is also possible to include a more complicated *ID* structure, where the *ID* is provided by the wallet user as an input parameter. Consider a scenario, where a wallet user Bob wants to receive some payment for some *ID*. To this end, the hot wallet generates a fresh session public key $pk_{ID}$ for *ID* via $\mathsf{SWal.PKDer}$. Then, *ID* is added to the transaction `tx` that is published on the blockchain. Later, when Bob wants to spend the transaction via the cold wallet, he can extract the *ID* from `tx` to generate the corresponding secret key $sk_{ID}$ on the cold wallet. Notice, of course, that the values for *ID* have to be chosen "somewhat randomly" as otherwise the unlinkability property of the wallet scheme is broken. One simple way to achieve this is to let the hot wallet encrypt the *ID* and add the ciphertext to the transaction that sends money to the address $pk_{ID}$.

Statefulness of our scheme. We point out that the state in our stateful wallet scheme $\mathsf{SWal}$ may make our scheme more complex to use in practice (as evidented from the previous discuss on synchronization). However, the state is *only* needed in order to achieve forward unlinkability after compromise of the hot wallet. The unforgeability property proven in our work also works for the simpler stateless wallets. Hence, if forward unlinkability is not needed, one can use a stateless version of our constructions and benefit from our security analysis (i.e., unlinkability *without* state compromise and unforgeability).

The winning condition of wallet unforgeability. In Figure 6 the adversary wins the game if she manages to output a valid forgery $(pk_{ID^*}, \sigma^*, m^*)$ such that $\mathsf{SWal.Verify}(pk_{ID^*}, \sigma^*, m^*) = 1$. We emphasize that in practice for breaking a wallet in, e.g., Bitcoin, it suffices that the adversary creates a transaction spending money from address $pk_{ID^*}$ *and* is accepted by the miners. The latter is quite important because there is no reason why in legacy cryptocurrencies, miners should execute the $\mathsf{SWal.Verify}$ algorithm of our $\mathsf{SWal}$ construction. Fortunately, however, in Bitcoin miners implicitly execute $\mathsf{REC[H].Verify}$ when verifying transactions, and hence our scheme and its security analysis is compatible with Bitcoin.[6]

Transaction Cost Analysis. To integrate our scheme into Bitcoin, we have to make sure that (a) transactions are salted, (b) they are pre-fixed[7] by the public key $pk$ from which the money is sent, and (c) such transactions are accepted by the miners. Fortunately, in Bitcoin this can be achieved using the simple scripting language, and we explain it in detail in App. B, full version. While the pre-fixing of the public key (b) is naturally happening in Bitcoin, the random salting (a) is non-standard and results into additional costs. We discuss them briefly below and compare them with the standard costs of creating transactions in Bitcoin (i.e., without salting). Consider a transaction $\mathtt{tx}_0$ that transfers money from the cold wallet to a new address, and hence in our scheme has to be randomized. Due to the mechanics of Bitcoin also the transaction $\mathtt{tx}_1$ that spends $\mathtt{tx}_0$ will include this random salt. Thus, our cost analysis includes these two transactions. We summarize the costs in `Satoshi` and USD, depending on whether the transaction gets included in the next block, or within the next 6 subsequent blocks. Note that confirmation of a transaction in an earlier block results into higher costs[8]

[6]At a more technical level, in Bitcoin if we want to spend money from an address $pk_{ID}$, then the spending transaction (that is signed with $sk_{ID}$) contains $pk_{ID}$. Hence, it has a form that is compatible with the verification done by $\mathsf{REC[H].Verify}$. In fact, our security proof can also be adjusted to match *exactly* with the verification that is carried out by the miners.

[7]Notice that in our generic wallet construction (c.f. Figure 7), messages are key pre-fixed to prevent from the related key attack. For the salted ECDSA construction to satisfy the property of signatures with uniquely rerandomizable keys (c.f. Figure 10), messages are again key-prefixed. The key prefixing in the latter case is necessary as an essential technique for the proof of Th 5.1. Although theoretically our ECDSA based wallet construction is key pre-fixed twice, in practice key pre-fixing the message once will be enough.

[8]We have used the currency value from [Cur19] timestamped on 14th May, 2019. Notice that the increase in costs are around 3% compared to standard Bitcoin transactions. However the cost increase also depends on the application, and we leave it as an interesting question for future work to provide an application-dependent optimization of costs.

Table 1: Standard vs Randomized Transactions Costs

| Transaction Type | Confirmation in next block<br>Fees (`Satoshi`/ USD) | Confirmation in next 6 blocks<br>Fees (`Satoshi`/ USD) |
|---|---|---|
| $\mathtt{tx}_0$ (Standard) | 7665/0.54 | 2190/0.17 |
| $\mathtt{tx}_1$ (Standard) | 8505/0.60 | 2430/0.19 |
| $\mathtt{tx}_0$ (*Randomized*) | 7875/0.56 | 2250/0.18 |
| $\mathtt{tx}_1$ (*Randomized*) | 8610/0.61 | 2460/0.19 |

## 7 Conclusion

In this work, we focused on analyzing the security of deterministic wallets. We developed two new security guarantees that we call wallet unlinkability and wallet unforgeability, and showed a modular approach for constructing such wallets from certain signature schemes. At the technical level, we proved that a simple extension of the ECDSA-based hot/cold wallet as used in Bitcoin can be proven secure in our model. A natural extension of our work will be to consider the case of hierarchical wallets. However, the hierarchical setting will require a significantly more complex model (additional oracles, more complex bookkeeping). The security analysis in this setting is also believed to be more involved. Hence, it is certainly an excellent direction for future research to extend our model to the hierarchical case.

## References

[AGKK19] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. Cryptology ePrint Archive, Report 2019/034, 2019. https://eprint.iacr.org/2019/034. (Cited on page 7.)

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. Cryptology ePrint Archive, Report 2018/483, 2018. https://eprint.iacr.org/2018/483. (Cited on page 7.)

[BH19] Joachim Breitner and Nadia Heninger. Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. *IACR Cryptology ePrint Archive*, 2019:23, 2019. (Cited on page 7.)

[Bit18] BitcoinExchangeGuide. CipherTrace Releases Report Exposing Close to $1 Billion Stolen in Crypto Hacks During 2018. https://bitcoinexchangeguide.com/ciphertrace-releases-report-exposing-close-to-1-billion-stolen-in_-crypto-hacks-during-2018/, 2018. (Cited on page 2.)

[Blo18] Bloomberg. How to Steal $500 Million in Cryptocurrency. http://fortune.com/2018/01/31/coincheck-hack-how/, 2018. (Cited on page 2.)

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17(4):297–319, 2004. (Cited on page 5.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. (Cited on page 8.)

[BR04] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. Cryptology ePrint Archive, Report 2004/331, 2004. http://eprint.iacr.org/2004/331. (Cited on page 8.)

[BR18] Michael Brengel and Christian Rossow. Identifying key leakage of bitcoin users. In *Research in Attacks, Intrusions, and Defenses - 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings*, pages 623–643, 2018. (Cited on page 7.)

[But13] Vitalik Buterin. Deterministic Wallets, Their Advantages and their Understated Flaws. https://bitcoinmagazine.com/articles/deterministic-wallets-advantages-flaw-1385450276/, 2013. (Cited on page 3, 7.)

[CEV14] Nicolas T. Courtois, Pinar Emirdag, and Filippo Valsorda. Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events. *IACR Cryptology ePrint Archive*, 2014:848, 2014. (Cited on page 7.)

[Cur19] Bitcoin Fees for Transactions. https://bitcoinfees.earn.com/, 2019. (Cited on page 31.)

[DFL19] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. Cryptology ePrint Archive, Paper 2019/698, 2019. https://eprint.iacr.org/2019/698. (Cited on page .)

[DKLS18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 980–997, 2018. (Cited on page 7.)

[FF13] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of schnorr signatures. In *Advances in Cryptology - EUROCRYPT 2013*, pages 444–460, 2013. (Cited on page 7.)

[FKM$^+$16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 301–330. Springer, Heidelberg, March 2016. (Cited on page 5, 6, 7, 9, 10.)

[FTS+18] Chun-I Fan, Yi-Fan Tseng, Hui-Po Su, Ruei-Hau Hsu, and Hiroaki Kikuchi. Secure hierarchical bitcoin wallet scheme against privilege escalation attacks. In *IEEE Conference on Dependable and Secure Computing, DSC 2018*, pages 1–8, 2018. (Cited on page 7.)

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *Applied Cryptography and Network Security - ACNS 2016*, pages 156–174, 2016. (Cited on page 7.)

[GS15] Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015*, pages 497–504, 2015. (Cited on page 7.)

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In *Advances in Cryptology - CRYPTO 2016, Part II*, pages 33–61, 2016. (Cited on page 7.)

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1837–1854, 2018. (Cited on page 7.)

[MB18] Gregory Maxwell and Iddo Bentov. Deterministic Wallets. `https://www.cs.cornell.edu/~iddo/detwal.pdf`, 2018. (Cited on page 3, 4.)

[Med18] Mediawiki. BIP32 Specification. `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`, 2018. (Cited on page 3.)

[MPas19] Antonio Marcedone, Rafael Pass, and abhi shelat. Minimizing trust in hardware wallets with two factor signatures. Cryptology ePrint Archive, Report 2019/006, 2019. `https://eprint.iacr.org/2019/006`. (Cited on page 7.)

[MSM+15] Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. On the security of the schnorr signature scheme and DSA against related-key attacks. In *ICISC 2015 - 18th International Conference, Seoul, South Korea, November 25-27, 2015, Revised Selected Papers*, pages 20–35, 2015. (Cited on page 16.)

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 239–252, 1989. (Cited on page 5, 7.)

[Ske18] Rhys Skellern. Cryptocurrency Hacks: More Than $2b USD lost between 2011-2018. `https://medium.com/ecomi/cryptocurrency-hacks-more-than-2b-usd-lost-between-2011-2018_-67054b342219`, 2018. (Cited on page 2.)

[TVR16] Mathieu Turuani, Thomas Voegtlin, and Michael Rusinowitch. Automated verification of electrum wallet. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC*, pages 27–42, 2016. (Cited on page 7.)

[Wik18a] Bitcoin Wiki. BIP32 proposal. `https://en.bitcoin.it/wiki/BIP_0032`, 2018. (Cited on page 3.)

[Wik18b] Wikipedia. Hardware Wallet. `https://en.bitcoin.it/wiki/Hardware_wallet`, 2018. (Cited on page 2.)

[ZCC$^+$15] Zongyang Zhang, Yu Chen, Sherman S. M. Chow, Goichiro Hanaoka, Zhenfu Cao, and Yunlei Zhao. Black-box separations of hash-and-sign signatures in the non-programmable random oracle model. In *Provable Security - 9th International Conference, ProvSec 2015*, pages 435–454, 2015. (Cited on page 7.)

# B. Exact Security of BIP32 Wallets

This chapter corresponds to our published article in CCS 2021 [43], with minor edits. Our full version can be found in [44].

[43] P. Das, A. Erwig, S. Faust, J. Loss, and S. Riahi. "The Exact Security of BIP32 Wallets". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communication Security, Virtual Event, Republic of Korea, November 15 - 19, 2021.* 2021, pp. 1020–1042. DOI: 10.1145/3460120.3484807. URL: https://doi.org/10.1145/3460120.3484807.

# The Exact Security of BIP32 Wallets

Poulami Das [1] Andreas Erwig [1] Sebastian Faust [1]
Julian Loss [2] Siavash Riahi [1]

[1] TU Darmstadt, Germany
firstname.lastname@tu-darmstadt.de
[2] University of Maryland, USA
lossjulian@gmail.com

**Abstract**

In many cryptocurrencies, the problem of key management has become one of the most fundamental security challenges. Typically, keys are kept in designated schemes called *wallets*, whose main purpose is to store these keys securely. One such system is the BIP32 wallet (Bitcoin Improvement Proposal 32), which since its introduction in 2012 has been adopted by countless Bitcoin users and is one of the most frequently used wallet system today. Surprisingly, very little is known about the concrete security properties offered by this system. In this work, we propose the first formal analysis of the BIP32 system in its entirety and without any modification. Building on the recent work of Das et al. (CCS '19), we put forth a formal model for hierarchical deterministic wallet systems (such as BIP32) and give a security reduction in this model from the existential unforgeability of the ECDSA signature algorithm that is used in BIP32. We conclude by giving concrete security parameter estimates achieved by the BIP32 standard, and show that by moving to an alternative key derivation method we can achieve a tighter reduction offering an additional 20 bits of security (111 vs. 91 bits of security) at no additional costs.

**Keywords:**Wallets, cryptocurrencies, foundations, BIP32

## 1 Introduction

Decentralized cryptocurrencies such as Bitcoin or Ethereum have introduced a new digital payment paradigm which does not rely on a central authority such as a bank or a credit card company. The main building block used in many popular cryptocurrencies to facilitate secure transfer and holding of assets are digital signatures. Loosely speaking, a user Alice in the system is identified by her public key $\mathsf{pk}_A$ which she uses as her address for receiving and sending payments. If Alice wants to send $c$ coins of the underlying currency to another user Bob with address $\mathsf{pk}_B$, she creates a transaction $\mathsf{tx}$ saying "Send $c$ coins from $\mathsf{pk}_A$ to $\mathsf{pk}_B$" and signs $\mathsf{tx}$ using her secret key $\mathsf{sk}_A$. She then uploads the transaction $\mathsf{tx}$ together with the signature $\sigma$ to the public ledger (often also called blockchain) of the cryptocurrency. Once the tuple $(\mathsf{tx}, \sigma)$ is visible on the public ledger, the payment is completed meaning that now Bob owns an additional $c$ coins of the underlying currency. Clearly, Alice's funds remain secure only as long as no one can forge a signature $\sigma$ on her behalf that verifies under $\mathsf{pk}_A$. On top of this, it is generally recommended to use a fresh signing key for every new transaction stored on the public ledger to avoid that all transactions are linkable to the same user Alice. In the cryptocurrency space, the management and storage of secret keys is typically carried out by so-called *wallets* – which are pivotal for the security of cryptocurrency funds. Indeed, cryptocurrency wallets are a highly attractive target for hackers as illustrated by spectacular attacks against common cryptocurrency projects. For example, in 2018 alone, hackers managed to steal more than one billion USD worth of cryptocurrency from wallets [Ske18, Blo18, Bit18].

While several recent works study the formal security properties of cryptocurrency wallets (see related work for a detailed discussion), one of the most widely used schemes – *the BIP32 wallet* [Wik18] – has not been formally analyzed so far. This is somewhat surprising as BIP32 became a standard for deterministic Bitcoin wallets in 2012, and has been widely adopted since then (e.g., it is used in the deployment of popular wallets [Ele13, Tre14, Led14]). In this work, we address this gap and provide the first comprehensive study of the security properties achieved by the BIP32 wallet standard.

## 1.1 Deterministic Wallets

As we have already pointed out, to improve privacy it is important to not re-use the same signing key for too many public transactions. To explain why privacy is also beneficial for security, let us consider a user Alice who holds a single secret/public key pair $(\mathsf{sk}_A, \mathsf{pk}_A)$, and that she receives multiple payments to her address $\mathsf{pk}_A$. As we have explained, such transactions contain her public key $\mathsf{pk}_A$ and are posted to the public ledger. Hence, an attacker can easily extract Alice's balance via the public transaction ledger. Over time, $\mathsf{pk}_A$'s balance might grow, and at some point, the attacker may identify $\mathsf{pk}_A$ as a high-priority target. The obvious approach to thwart an attacker's attempts of linking Alice's transactions would be for Alice to keep a set of $l$ one-time random key pairs $\{(\mathsf{sk}_1, \mathsf{pk}_1), \ldots, (\mathsf{sk}_l, \mathsf{pk}_l)\}$ within her wallet, where each key pair is used for a single transaction on the public ledger. However, this approach has the obvious downside of Alice having to store all of her keys on disk (as long as they still retain some amount of currency). This requires a lot of storage space and bears the risk of losing one of her keys, at which point the associated funds of that key are irrevocably lost. A simple approach to overcome these issues are *deterministic wallets*, proposed by Buterin [But13]. A deterministic wallet usually contains a pair of master keys $(\mathsf{msk}, \mathsf{mpk})$ and a seed $\mathsf{ch}$, which is also referred to as the *chaincode*. For every new transaction, the wallet deterministically derives a fresh session key pair $(\mathsf{sk}, \mathsf{pk})$ from the master keys with the help of deterministic key derivation algorithms. More precisely, the public key derivation algorithm takes as input the master public key $\mathsf{mpk}$, the chaincode $\mathsf{ch}$ and an identifier $\mathsf{ID}$ and deterministically computes a one-time public key $\mathsf{pk}_{\mathsf{ID}}$. An analogous secret key derivation algorithm takes in $\mathsf{msk}$, $\mathsf{ch}$, and $\mathsf{ID}$ and deterministically computes a one-time secret key $\mathsf{sk}_{\mathsf{ID}}$ that matches $\mathsf{pk}_{\mathsf{ID}}$ (given that the arguments $\mathsf{ch}$ and $\mathsf{ID}$ in both derivations are identical). Going back to the example of BIP32, $(\mathsf{msk}, \mathsf{mpk})$ are generated as ECDSA keys and public key derivation is done by computing the offset $\omega := \mathsf{H}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$, where $\mathsf{ID} \in [2^{32}]$ and then rerandomizing $\mathsf{mpk}$ to $\mathsf{pk}_{\mathsf{ID}}$ by computing $\mathsf{pk}_{\mathsf{ID}} := \mathsf{mpk} + G \cdot \omega$. Here, $G$ denotes the base point of an elliptic curve group of prime order $p$. A matching secret key can be derived via $\mathsf{sk}_{\mathsf{ID}} := \mathsf{msk} + \omega \pmod p$.

**Hot/Cold Wallets.** A typical way of using deterministic wallets in practice is via the hot/cold wallet paradigm. With this approach, Alice maintains two wallets. The first wallet is referred to as the *cold wallet*. It keeps the master secret key $\mathsf{msk}$ as well as the chaincode. The cold wallet is usually implemented via some simple storage device that should be almost permanently disconnected from the internet, so as to minimize the risk of attack. The second wallet is the so-called *hot wallet*, which is permanently online and keeps the master public key as well as the chaincode. Using the deterministic key derivation procedures, the two wallets can independently derive matching keys to use for one-time transaction on the public ledger. In a bit more detail, Alice uses her hot wallet as a low-security spending wallet which, at any point in time, keeps only a small amount of currency. Whenever the funds stored on the hot wallet exceed a certain amount, Alice can use the public key derivation algorithm to derive a new public key $\mathsf{pk}_{\mathsf{ID}}$ on her hot wallet and transfer the excess funds to $\mathsf{pk}_{\mathsf{ID}}$. Note that this requires no interaction with the cold wallet. At a later point in time, the cold wallet can come online for a brief moment and spend the funds from $\mathsf{pk}_{\mathsf{ID}}$, using a matching secret key $\mathsf{sk}_{\mathsf{ID}}$ derived via the secret key derivation algorithm. As far as security goes, we would like to ensure two properties. First, *unlinkability* ensures that keys derived from the same master key pair are indistinguishable from random keys, given that the hot wallet has not leaked the chaincode to the attacker. Second, *unforgeability* ensures that even if the hot wallet leaks the chaincode (e.g., because it has been corrupted), signatures from derived keys should still remain unforgeable. While unlinkability is easy to achieve, unforgeability is a much more subtle issue in this setting, as the derived keys are all correlated once the chaincode has been revealed to the attacker. Hence, the standard unforgeability property of the underlying signature scheme is no longer sufficient to ensure unforgeability of signatures under these derived keys.

### 1.2 Limitations of Existing Works

The work of Das et al. [DFL19] was the first to provide a formal model to reason about the aforementioned security properties. It also showed how to achieve secure constructions in their proposed model from various different signature schemes used in practice, e.g., Schnorr, BLS, and ECDSA. Notably, the latter construction is very practical and can be integrated directly with the (unmodified) Bitcoin system. In spite of these achievements, their work makes no progress towards formally proving security properties for the BIP32 wallet standard that is widely used in many real-world systems. Let us discuss the reasons for this in a little more detail.

First, the construction of Das et al. uses a *multiplicative rerandomization* to derive keys, in which keys for identity $\mathsf{ID}$ are computed from $\omega = \mathsf{H}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$ as $\mathsf{pk}_{\mathsf{ID}} := \mathsf{mpk} \cdot \omega$, and $\mathsf{sk}_{\mathsf{ID}} := \mathsf{msk} \cdot \omega \pmod p$. By comparison, as we saw above, BIP32 uses an *additive rerandomization*. Although this might look like a minor difference, we will see later that the proof technique and security guarantees achieved by the additive version differ significantly from the multiplicative one. Second, the work of Das et al. does not consider the hierarchical key derivation mechanism provided by BIP32. Hierarchical deterministic wallets allow for keys in the wallet to act simultaneously as signing keys *and* as parent (master) keys to derive new child keys in their own right. As a useful example, consider a company that wishes to delegate new signing key pairs to different entities within the company. Unfortunately, it cannot be guaranteed that all entities in the company store their keys securely and some of them might be leaked to the adversary over time. Such a strong adversary cannot be captured by the model and constructions of Das et. al. Since many wallets that are used in practice follow the BIP32 standard, it is crucial to provide a formal analysis of the scheme *as is*, meaning without any modifications to it.

## 1.3 Our Contributions

In this work we address the above shortcomings and provide, for the first time, a formal analysis of the full BIP32 specification in the hot/cold wallet setting. An important implication of our work is that we can establish the exact security that is achieved by the current standard, which also leads us to propose a minor modification that can significantly improve security without any additional costs.

**Rerandomizing ECDSA.** We begin by recalling the notion of *unforgeability under honestly rerandomized keys (UFCMA-HRK)* introduced by Das et al. [DFL19]. As this notion will serve as the basis of our wallet constructions, we review it in detail below. Compared to the standard notion of unforgeability under chosen message attacks (UFCMA), the adversary in the UFCMA-HRK game initially obtains a challenge public key $\mathsf{pk}$ and gets to query for rerandomizations of $\mathsf{pk}$. The game returns the rerandomized public key $\tilde{\mathsf{pk}}$ together with the (uniformly chosen) randomness $\rho$ that was used in the rerandomization process. The exact way that the rerandomization is actually done depends on the scheme; we are mostly interested in the case where ECDSA keys are additively rerandomized as $\mathsf{pk} + G \cdot \rho$. The game then allows the adversary to query for signatures relative to any of the rerandomized public keys that it has previously obtained from the game. It is considered successful if it can return a forgery relative to any of the requested keys $\tilde{pk}$ on a message for which it has not previously asked for a signature under $\tilde{pk}$. As observed by Das et al., this security notion is a weakened version of *unforgeability under rerandomized keys* [FKM+16] in which the adversary can choose the random coins $\rho$ itself and provide them to the game. In Section 3, we prove that ECDSA with additive rerandomization satisfies UFCMA-HRK as long as *each message is signed only once per key.* A first attempt is to naively follow the approach of Das et al. who showed that ECDSA with multiplicative rerandomization satisfies UFCMA-HRK (without any restrictions on the number of signatures per message). The main idea of Das et al.'s reduction from UFCMA-HRK to UFCMA (both with respect to the ECDSA scheme) is to rely on a *related key attack* (RKA) that is present in the multiplicatively rerandomized version of the ECDSA scheme. Concretely, the RKA allows to transform a signature $(r, s)$ on message $m_0$ relative to a key $\mathsf{pk}_0$ into a signature $(r, s/\rho)$ on message $m_1$ that is valid under the related key $\mathsf{pk}_1 = \mathsf{pk}_0 \cdot \rho$, where $\rho$ satisfies $\rho = \frac{\mathsf{H}(m_0)}{\mathsf{H}(m_1)}$. This attack can be leveraged by the reduction to answer all signing queries in the UFCMA-HRK game. More precisely, using the RKA, it is possible to transform signatures obtained from the signing oracle in the UFCMA game into signatures relative to any of the rerandomized keys in the UFCMA-HRK game (via programming of the random oracle). Hence, we are immediately faced with the following obstacle: this RKA does not work if keys are additively rerandomized.

**Extending to Additive Rerandomization.** To overcome this issue with the existing reduction, we present a new RKA which works for additively rerandomized ECDSA. The attack works as follows: given a signature $(r, s)$ on $m_0$ relative to $\mathsf{pk}_0$, $(r, s)$ is also a valid signature relative to the public key $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$ on message $m_1$, given that $\rho = (\mathsf{H}(m_0) - \mathsf{H}(m_1))/r$. Rather surprisingly, considering ECDSA's huge popularity, we are not aware of this attack having been noticed previously. Using our new RKA, we are now able to (almost) make the simulation of signatures in Das et al.'s approach work. However, there is a further issue that comes from the structure of the additive RKA. Suppose that the reduction is directed to program the random oracle $\mathsf{H}$ on a message $m$ so as to provide the attacker with a signature relative to a (rerandomized) public key $\tilde{\mathsf{pk}}$ in the UFCMA-HRK game. The above RKA forces the reduction to program $\mathsf{H}$ on a value that depends on a *particular signature* $(r, s)$ on $m$, which it obtains from the signing oracle in the underlying UFCMA game. Now, the only signature on $m$ that the reduction can hand to the adversary under $\tilde{\mathsf{pk}}$ is $(r, s)$. If the adversary requests another signature *on the same message* $m$, we are not able to reply with a fresh signature, as we can program $\mathsf{H}$ on $m$ only a single time. For this reason, we have to restrict ourselves to one-per-message unforgeability. We emphasize, however, that this notion of security (one signature per-message) is sufficient in our setting, as transactions are identified by unique nonces in most cryptocurrencies (including Bitcoin) and hence never signed twice. An additional benefit of our new reduction (compared to [DFL19]) is that it only requires the weaker assumption that the underlying ECDSA scheme is one signature per-message unforgeable in its own right. This is worth noting, as the work of Fersch et al. shows that ECDSA achieves this property in the random oracle model [FKP17] (albeit with a very large security loss). By comparison, the unrestricted security (i.e., UFCMA) of ECDSA remains only a conjecture in the plain random oracle model. Our reduction also removes the need for the random salt present in Das et al.'s construction. This is an important improvement, as it allows using BIP32 without Bitcoin's scripting language, which was required by the construction of Das et al. due to their use of the salt. Finally, we remark that our reduction (by comparison to Das et al.) is *non-tight* and loses a factor proportional to the total number of keys derived in the UFCMA-HRK game. We provide further discussion on this issue in the next section and in Section 3.

**Hierarchical Wallets.** To complete the analysis of BIP32, the second part of our work focuses on formal security properties when supporting hierarchies in deterministic wallet constructions (as is the case for BIP32). As already hinted, the core difficulty in this setting is that some of the wallet's keys may be given to untrustworthy users who may leak their cold wallet keys to the adversary. If this happens, it is important to ensure that the adversary does not gain information about secret keys further up in the hierarchy. It is easy to see that this property is not achieved if all keys are derived using the derivation algorithms described so far: if the adversary learns $\mathsf{sk}_{\mathsf{ID}} = \mathsf{msk} + \rho \pmod p$, where $\rho$ is computed as $\rho = \mathsf{H}(\mathsf{mpk}, \mathsf{ch}, \mathsf{ID})$, then it can recover $\mathsf{msk}$ as $\mathsf{msk} = \mathsf{sk}_{\mathsf{ID}} - \rho \pmod p$ and learn all cold wallet keys that were ever derived using $\mathsf{msk}$. Because of this, BIP32 offers a second mode of deriving keys called *hardened key derivation*. Hardened keys are derived by changing the computation of the offset $\rho$ above to $\rho = \mathsf{H}(\mathsf{msk}, \mathsf{ch}, \mathsf{ID})$. Now, even when learning $\mathsf{sk}_{\mathsf{ID}}$, it is not possible for the adversary to recover $\mathsf{msk}$. The downside of hardened key derivation is that the hot and cold wallet can no longer independently derive keys (as the hot wallet does not know msk). Thus, this mode of derivation is not intended for use in the hot/cold wallet paradigm, but simply to create keys with a higher degree of security. These keys can either be stored (efficiently) as part of the main wallet or handed to users in the system without any concern for other cold wallet keys. In Section 4, we state the syntactical definition and correctness properties of a hierarchical deterministic wallet. We then introduce a security model that supports both types of key derivations (hardened and non-hardened), as well as secret key leakage of hardened keys. We refer to this notion of security as WUFCMA. In Section 5, we provide a generic construction HDWal that transforms a signature scheme satisfying UFCMA-HRK into a hierarchical deterministic wallet with WUFCMA security.[1] In this way, we are able to complete the analysis of BIP32 by instantiating HDWal with ECDSA using additive rerandomization.

**On the Tightness of Our Construction.** A particular focus of our work is to analyze the tightness and concrete security achieved by our constructions, most notably BIP32. We have already mentioned that our reduction from UFCMA-HRK to UFCMA of the ECDSA scheme with additive rerandomization is non-tight. More precisely, it loses a factor proportional to the number of keys derived by the adversary in the UFCMA-HRK game. Thus, our goal is to at least achieve the best possible tightness of our generic transform HDWal. To this end, let us first consider the possible options for potential security losses. From worst to best (excluding a tight reduction), the options are:

- Loss in the number of random oracle queries.
- Loss in the number of keys derived in the wallet (hardened or non-hardened).
- Loss in the number of signing oracle queries (assuming keys are used only once).
- Loss in the number of hardened keys leaked to the adversary.

The first three possibilities are quite catastrophic as the number of random oracle queries, signing oracle queries, or keys derived in practice could be quite high. On the other hand, we expect the number of *leaked keys* to be only a small portion of all the keys in a given wallet (we use 1% as an estimate in our calculations). We are able to prove that HDWal indeed achieves a multiplicative security loss proportional to only the hardened keys leaked to the adversary over the course of the lifetime of the wallet. Furthermore, we show that any *generic transform* from UFCMA-RK (a stronger notion than what is used in our construction) to WUFCMA *must lose at least this factor*. Hence, our construction HDWal achieves the *best possible parameters*. To prove our results, we adapt the reduction/metareduction techniques introduced by Coron in his seminal work [Cor02]. Given that his results deal with the tightness of unique signatures (which is very different from our setting), this requires careful insight into his technique in order to adapt it to our model.

**Concrete Security Parameters.** We conclude by giving a discussion of the concrete security levels achieved by BIP32 and the multiplicative ECDSA scheme of Das et al.,

[1]In case the underlying signature scheme has the one signature per message restriction, then the resulting wallet scheme also does.

when plugged into HDWal. We find that BIP32 gives roughly 94 bits of security according to our theorems and conservative choices of parameters. We find that by comparison, the multiplicative version of Das et al. gives 114 bits of security with a similarly efficient scheme. (We remark that using the techniques introduced in our paper, we can also remove the salt in the multiplicatively rerandomizable ECDSA version of Das et al.). Given these insights, we strongly recommend that the Bitcoin community switch rerandomizations in BIP32 from additive to multiplicative, in particular since these changes essentially come for free.

## 1.4 Related Work

The most relevant previous work for us is by Das et al. [DFL19] as mentioned previously. However, there have been other works which try to formalize cryptographic wallets. The work of Gutoski and Stebila [GS15] proposes an alternative construction for hierarchical wallets where up to $d$ session keys can leak without the master secret key being compromised under the one-more discrete-log assumption. However, their security model is weaker than our model (or the security model of Das et al. on which we base our work). More precisely, in their model, the adversary cannot query the game for signatures under uncompromised wallet keys. Furthermore, instead of the traditional security model where the adversary wins if she can forge a signature, the adversary's goal in their security definition is to extract the master secret/public key pair. Another more recent work is by Luzio et al. [LFA20] where the authors design a new hierarchical wallet scheme by using (deterministic) hierarchical key assignment schemes [ABFF09]. Unfortunately, their solution is not compatible with cryptocurrencies such as Bitcoin since their solution requires a more sophisticated (signature) verification algorithm, where a certificate associated with the user needs to be verified along with the signature.

Turuani et al. [TVR16] analyzed the Bitcoin Electrum wallet using automated verification in the Dolev-Yao model. However, many automated verification models only consider "idealized" building blocks, i.e., cryptographic building blocks that are perfectly secure. Consequently, this type of analysis excludes weaknesses such as related key attacks, which are of fundamental relevance in the setting of deterministic wallets.

Another line of work has considered the security of hardware wallets [MPs19, AGKK19] and implementation bugs in wallets (such as weak randomness) [CEV14, BR18, BH19]. Additionally, there have been several works with focus on the use of threshold ECDSA signatures [KMOS19, GGN16, LN18, DKLs18] and multi-signatures [BDN18] in (and outside of) wallet systems.

In a recent work, Alkadri et al. [ADE+20] have shown how to realize deterministic wallets that are post-quantum secure. To this end, they suitably adapt the model and techniques of Das et al. by considering an adversary with quantum computing power.

The concept of rerandomizable signature schemes was first introduced by Fleischhacker et al. [FKM+16] and later used by [DFL19, ADE+20] for their wallet schemes. In addition, related key attacks have been studied for signature schemes such as Schnorr [Sch90] in many previous works [FF13, KMP16, ZCC+15]. For ECDSA, Das et al. leveraged related key attacks to achieve a multiplicatively rerandomizable ECDSA scheme which they prove secure w.r.t. the security notion of unforgeability under honestly rerandomizable keys. Finally, Fersch et. al. [FKP16] provided the first security analysis of ECDSA in an idealized model.

## 2 Preliminaries

**Notation.** We use the notation $s \xleftarrow{\$} H$ to denote the uniform sampling of a variable $s$ from the set $H$. For an integer $l$, $[l]$ denotes the set of integers $\{1, \cdots, l\}$. We use upper case letters to denote algorithms. For an algorithm $A$, we write $y \xleftarrow{\$} A(x)$ to denote the execution of a randomized algorithm $A$ on input $x$ that outputs $y$. We write $y \leftarrow B(x; \rho)$ to denote the execution of an algorithm $B$ that, on input $x$ and randomness $\rho$, outputs $y$. Note that in this notation, $B$ *is deterministic.* We use the notation $y \in A(x)$ to denote that $y$ is in the set of possible outputs of $A$ on input $x$.

In order to simplify our notation and definitions, we assume that public parameters $\mathsf{par}$ have been securely generated and can be used throughout the paper as input to algorithms. We generally assume that, initially, boolean variables are set to false, integers are set to 0, lists are set to $\emptyset$, and undefined entries of lists are set to $\perp$. For strings $a, b \in \{0,1\}^*$, we write $a = (b, \cdot)$ if $b$ is a prefix of $a$ and likewise, we write $a \neq (b, \cdot)$ if $a$ is not prefixed by $b$. We denote by $\kappa$ the security parameter throughout the paper.

We use standard code-based security games [Sho04]. A *game* $\mathbf{G}$ is an interactive probability experiment between an *adversary* $\mathcal{A}$ and an (implicit) *challenger* which provides answers to oracle queries posed by $\mathcal{A}$. The output of $\mathbf{G}$ when interacting with adversary $\mathcal{A}$ is denoted as $\mathbf{G}^{\mathcal{A}}$. Finally, the randomness in any probability term of the form $\Pr[\mathbf{G}^{\mathcal{A}} = 1]$ is assumed to be over all the random coins in game $\mathbf{G}$.

### 2.1 Signature Schemes

We now recall the definition of signature schemes and that of signature schemes with perfectly rerandomizable keys from [DFL19].

**Definition 2.1** (Signature Scheme). A *signature scheme* is a tuple of algorithms $\mathsf{Sig} = (\mathsf{Sig.Gen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ which are defined as follows:

- $\mathsf{Sig.Gen}(\mathsf{par})$: The randomized *key generation* algorithm $\mathsf{Sig.Gen}$ takes as input public parameters $\mathsf{par}$ and outputs a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$.

- $\mathsf{Sig.Sign}(\mathsf{sk}, m)$: The (possibly) randomized *signing* algorithm $\mathsf{Sig.Sign}$ takes as input a secret key $\mathsf{sk}$ and a message $m$ and outputs a signature $\sigma$.

- $\mathsf{Sig.Verify}(m, \mathsf{pk}, \sigma)$: The deterministic *verification* algorithm $\mathsf{Sig.Verify}$ takes as input a public key $\mathsf{pk}$, a signature $\sigma$, and a message $m$. It outputs either 1 (accept) or 0 (reject).

A signature scheme $\mathsf{Sig}$ is *correct* if the following holds: For all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Sig.Gen}(\mathsf{par})$ and all $m \in \{0,1\}^*$ we have that

$$\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}, m) \Pr[\mathsf{Sig.Verify}(\mathsf{pk}, \sigma, m) = 1] = 1.$$

**Definition 2.2** (Signature Scheme with Perfectly Rerandomizable Keys). A *signature scheme with perfectly rerandomizable keys* is a tuple of algorithms $\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK})$ where $(\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify})$ are the standard algorithms of a signature scheme. Moreover, we assume that the public parameters $\mathsf{par}$ define a randomness space $\mathcal{R} := \mathcal{R}(\mathsf{par})$. Then the algorithms $\mathsf{RSig.RandSK}$ and $\mathsf{RSig.RandPK}$ are defined as follows:

- $\mathsf{RSig.RandSK}(\mathsf{sk}; \rho)$: The deterministic *secret key rerandomization algorithm* $\mathsf{RSig.RandSK}$ takes as input a secret key $sk$ and randomness $\rho \in \mathcal{R}$ and outputs a rerandomized secret key $sk'$.

- $\mathsf{RSig.RandPK}(\mathsf{pk}; \rho)$: The deterministic *public key rerandomization algorithm* $\mathsf{RSig.RandPK}$ takes as input a public key $pk$ and randomness $\rho \in \mathcal{R}$ and outputs a rerandomized public key $pk'$.

We make the convention that for the empty string $\epsilon$, we have that $\mathsf{RSig.RandPK}(\mathsf{pk}; \epsilon) = \mathsf{pk}$ and $\mathsf{RSig.RandSK}(\mathsf{sk}; \epsilon) = \mathsf{sk}$.

We further require:

1. *(Perfect) rerandomizability of keys:* For all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{RSig.Gen}(\mathsf{par})$ and $\rho \xleftarrow{\$} \mathcal{R}$, the distributions of $(\mathsf{sk}', \mathsf{pk}')$ and $(\mathsf{sk}'', \mathsf{pk}'')$ are identical, where:

$$\begin{aligned} (\mathsf{sk}', \mathsf{pk}') &\leftarrow (\mathsf{RSig.RandSK}(\mathsf{sk}; \rho), \mathsf{RSig.RandPK}(\mathsf{pk}; \rho)), \\ (\mathsf{sk}'', \mathsf{pk}'') &\xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par}). \end{aligned}$$

2. *Correctness under rerandomized keys:* For all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{RSig.Gen}(\mathsf{par})$, for all $\rho \in \mathcal{R}$, and for all $m \in \{0,1\}^*$, the rerandomized keys $\mathsf{sk}' \leftarrow \mathsf{RSig.RandSK}(\mathsf{sk}; \rho)$ and $\mathsf{pk}' \leftarrow \mathsf{RSig.RandPK}(\mathsf{pk}; \rho)$ satisfy:

$$\Pr_{\sigma \xleftarrow{\$} \mathsf{RSig.Sign}(\mathsf{sk}', m)} [\mathsf{RSig.Verify}(\mathsf{pk}', \sigma, m) = 1] = 1.$$

**Security notion uf-cma1.** In this work, we use the security notion of *one-per message existential unforgeability under chosen message attacks* (**uf-cma1**) [FKP17] which is a slightly weaker variant of the standard notion of existential unforgeability under chosen message attacks (**uf-cma**) security. In contrast to standard **uf-cma**, in **uf-cma1**, the adversary is restricted to querying the signing oracle at most once for each message. We formalize the **uf-cma1** notion for a signature scheme $\mathsf{Sig}$ in the form of a game $\textbf{uf-cma1}_{\mathsf{Sig}}$ as follows.

Game $\textbf{uf-cma1}_{\mathsf{Sig}}$:

- **Setup Phase:** The challenger initiates a list as $\mathsf{SigList} \leftarrow \{\epsilon\}$ for storing messages and samples a pair of keys $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Sig.Gen}(\mathsf{par})$. Then, $\mathcal{A}$ is run on input $\mathsf{pk}$.

- **Online Phase:** $\mathcal{A}$ is given access to a signing oracle `Sign` which works as follows. On input a message $m$, if $m$ was queried in a previous `Sign` query, i.e., if $m \in \mathsf{SigList}$, then $\perp$ is returned. Otherwise, `Sign` computes a signature on message $m$ as $\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(\mathsf{sk}, m)$. The message $m$ is stored in the $\mathsf{SigList}$ and the signature $\sigma$ is returned as the answer.

- **Output Phase:** Finally, $\mathcal{A}$ wins the game if it can provide a forgery $\sigma^*$ on a message $m^*$, where (1) $m^*$ is fresh, i.e., $m^* \notin \mathsf{SigList}$ and (2) $\sigma^*$ is a valid forgery, i.e., $\mathsf{Sig.Verify}(\mathsf{pk}, \sigma^*, m^*) = 1$.

For an algorithm $\mathcal{A}$ we define $\mathcal{A}$'s advantage in the game $\mathbf{uf\text{-}cma1}_{\mathsf{Sig}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}cma1}_{\mathsf{Sig}}} = \Pr[\mathbf{uf\text{-}cma1}^{\mathcal{A}}_{\mathsf{Sig}} = 1]$.
**Security notion uf-cma-hrk1.** For signature schemes with perfectly rerandomizable keys, we introduce the notion of *one-per message existential unforgeability under honestly rerandomizable keys* (**uf-cma-hrk1**), which restricts the security notion of *existential unforgeability under honestly rerandomizable keys* (**uf-cma-hrk**) as introduced by Das et al. [DFL19]. In this security notion, the signing oracle cannot only return signatures under $\mathsf{sk}$, but it can also return signatures that were produced with keys that represent *honest* rerandomizations of $\mathsf{sk}$. The term *honest* indicates that the randomness for the rerandomization is chosen uniformly at random from $\mathcal{R}$ (by the game itself). Our security notion of **uf-cma-hrk1** restricts the notion of **uf-cma-hrk** in the sense that the signing oracle returns at most one signature for each randomness/message pair $(\rho, m)$. We formally model the notion of **uf-cma-hrk1** for a rerandomizable signature scheme $\mathsf{RSig}$ in the form of a game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ as follows.

Game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$:

- **Setup Phase:** The challenger initializes two lists as $\mathsf{SigList} \leftarrow \{\epsilon\}$ and $\mathsf{RList} \leftarrow \{\epsilon\}$ and samples a pair of keys $(\mathsf{pk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{RSig.Gen}(\mathsf{par})$. Then $\mathcal{A}$ is run on input $\mathsf{pk}$.

- **Online Phase:**

  – $\mathcal{A}$ is given access to an oracle `Rand`, which, upon a query, samples a fresh random value from $\mathcal{R}$ as $\rho \overset{\$}{\leftarrow} \mathcal{R}$, stores $\rho$ in the list $\mathsf{RList}$, and returns $\rho$.

  – $\mathcal{A}$ is given access to a signing oracle `RSign` which works as follows. On input a message $m$ and a randomness $\rho$, if $\rho$ was not obtained via a prior `Rand` query (i.e., $\rho \notin \mathsf{RList}$), then return $\perp$. Otherwise, derive a pair of keys rerandomized with the randomness $\rho$, as $\mathsf{sk}' \leftarrow \mathsf{RSig.SKDer}(\mathsf{sk}; \rho)$ and $\mathsf{pk}' \leftarrow \mathsf{RSig.PKDer}(\mathsf{pk}; \rho)$. If $(\mathsf{pk}', m) \in \mathsf{SigList}$ then return $\perp$. Otherwise, a signature is derived on message $m$ under the secret key $\mathsf{sk}'$ as $\sigma \leftarrow \mathsf{RSig.Sign}(\mathsf{sk}', m)$. The tuple $(\mathsf{pk}', m)$ is stored in the $\mathsf{SigList}$ and the signature $\sigma$ is returned as the answer.

- **Output Phase:** $\mathcal{A}$ wins if it returns a forgery $\sigma^*$ together with a message $m^*$ and a public key $\mathsf{pk}^* \leftarrow \mathsf{RSig.PKDer}(\mathsf{pk}; \rho^*)$,[2] s.t. following holds: (1) the randomness $\rho^*$ has been derived via a `Rand` query, i.e., $\rho^* \in \mathsf{RList}$, (2) $(m^*, \rho^*)$ is fresh, i.e., $(\mathsf{pk}^*, m^*) \notin \mathsf{SigList}$, and (3) $\sigma^*$ is a valid forgery, i.e., $\mathsf{RSig.Verify}(\mathsf{pk}^*, \sigma^*, m^*) = 1$.

For an algorithm $\mathcal{A}$ we define $\mathcal{A}$'s advantage in game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ as $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} = \Pr[\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{A}}_{\mathsf{RSig}} = 1]$.

Other than only allowing the adversary to ask for at most one signature per message, our definition deviates from the one presented in [DFL19] by storing the tuples $(pk', m)$

[2]For simplicity, we tacitly assume that $\mathsf{pk}^*$ identifies $\rho^*$. This can easily be achieved using appropriate bookkeeping.

in the list $\mathsf{SigList}$ instead of just storing $m$. This change allows an adversary in the **uf-cma-hrk1** game to query a signature for the same message but under different public keys.

## 3 Security Analysis of Additively Rerandomizable ECDSA

In the following discussion, let $\mathbb{E}(\mathsf{par})$ denote an elliptic curve with base point G and prime order p. Furthermore, assume hash functions $\mathsf{H}_0 \colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ (modeled as random oracles). In this section, we present a signature scheme with rerandomizable keys $\mathsf{REC}[\mathsf{H}_1]$ based on the standard ECDSA scheme which we denote by $\mathsf{EC}[\mathsf{H}_0]$ (cf. Figure 1). $\mathsf{REC}[\mathsf{H}_1]$, as illustrated in Figure 2, works in a similar way as $\mathsf{EC}[\mathsf{H}_0]$ with two main differences. (1) It is extended by two algorithms $\mathsf{RandSK}$ and $\mathsf{RandPK}$ for the key rerandomization and (2) it is designed for key-prefixed messages. First, the two algorithms $\mathsf{RandSK}$ and $\mathsf{RandPK}$ randomize a key pair by *adding* a random value to each key. This is in contrast to the signature scheme with *multiplicatively* rerandomizable keys based on ECDSA as presented by Das et al. [DFL19], where the rerandomization algorithms multiply a random value to each key. Second, $\mathsf{REC}[\mathsf{H}_1]$ is designed for key-prefixed messages, i.e., upon executing $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}, m)$ for a secret key $\mathsf{sk}$ and a message $m$, the message is first extended to a *key-prefixed message* $\mathsf{pm} \leftarrow (\mathsf{pk}, m)$ where $\mathsf{pk}$ represents the public key corresponding to $\mathsf{sk}$. Then the prefixed message $\mathsf{pm}$ is signed under $\mathsf{sk}$.

We prove that $\mathsf{REC}[\mathsf{H}_1]$ satisfies **uf-cma-hrk1** security by providing a reduction from the **uf-cma1** security of the standard ECDSA scheme $\mathsf{EC}[\mathsf{H}_0]$. An integral part of the reduction is the observation that there exists a so-called "related key attack" (RKA) in the scheme $\mathsf{EC}[\mathsf{H}_0]$. An RKA allows to transform a signature that is valid under a public key $\mathsf{pk}_0$ into a signature that is valid under another public key $\mathsf{pk}_1$ given there exists a specific relation between $\mathsf{pk}_1$ and $\mathsf{pk}_0$. The RKA in $\mathsf{EC}[\mathsf{H}_0]$ allows to use a signature $\sigma$ that is valid under a public key $\mathsf{pk}_0$ as a valid signature under a public key $\mathsf{pk}_1$ in case $\mathsf{pk}_1$ and $\mathsf{pk}_0$ are related as $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$, where $\rho$ must satisfy $\rho = \frac{H_0(m_0) - H_1(m_1)}{r}$. We formally describe this related key attack in the following Lemma.

**Lemma 3.1** *Let* $\mathsf{H}_0$, $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ *be hash functions (modeled as random oracles). Suppose that* $\sigma = (r, s)$ *is a valid signature on message* $m_0 \in \{0,1\}^*$ *w.r.t.* $\mathsf{EC}[\mathsf{H}_0]$ *and public key* $\mathsf{pk}_0$*, i.e.,* $\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}(\mathsf{pk}_0, \sigma, m_0) = 1$*. Furthermore, let* $\rho = \frac{H_0(m_0) - H_1(m_1)}{r}$ $(\mathrm{mod}\ p)$*. Then* $\sigma$ *is also a valid signature on message* $m_1 \in \{0,1\}^*$ *w.r.t.* $\mathsf{EC}[\mathsf{H}_1]$ *and public key* $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$*, i.e.,* $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m_1) = 1$.

*Proof of Lemma 3.1.* We have to show that $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m_1) = 1$ for $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$ and $\rho = \frac{H_0(m_0) - H_1(m_1)}{r}$ $(\mathrm{mod}\ p)$. Note that $\sigma = (r, s)$, where $s = t^{-1}(\mathsf{H}_0(m_0) + r\mathsf{sk}_0)$ $(\mathrm{mod}\ p)$ and $r$ represents the $x$-coordinate of the elliptic curve point $t \cdot G$ for $t \overset{\$}{\leftarrow} \mathbb{Z}_p$. As shown in Figure 1, $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m_1)$ computes the following:

```
Algorithm $\mathsf{EC}[\mathsf{H}_0].\mathsf{Gen}\,(\mathsf{par})$
00 $x \xleftarrow{\$} \mathbb{Z}_p$
01 $X \leftarrow x \cdot G$
02 $\mathsf{sk} \leftarrow x$
03 $\mathsf{pk} \leftarrow X$
04 Return $(\mathsf{pk}, \mathsf{sk})$

Algorithm $\mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}\,(\mathsf{sk} = x, m)$
05 $z \leftarrow \mathsf{H}_0(m)$
06 $t \xleftarrow{\$} \mathbb{Z}_p$
07 $(e_x, e_y) \leftarrow t \cdot G$
08 $r \leftarrow e_x \mod p$
09 If $r = 0 \mod p$
10    Goto Step 06
11 $s \leftarrow t^{-1}(z + rx) \mod p$
12 If $s = 0 \mod p$
13    Goto Step 06
14 Return $\sigma := (r, s)$

Algorithm $\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}\,(\mathsf{pk} = X, \sigma, m)$
15 Parse $(r, s) \leftarrow \sigma$
16 If $(r, s) \notin \mathbb{Z}_p$
17    Return 0
18 $w \leftarrow s^{-1} \mod p$
19 $z \leftarrow \mathsf{H}_0(m)$
20 $u_1 \leftarrow zw \mod p$
21 $u_2 \leftarrow rw \mod p$
22 $(e_x, e_y) \leftarrow u_1 \cdot G + u_2 \cdot X$
23 If $(e_x, e_y) = (0, 0)$
24    Return 0
25 Return $r = e_x \mod p$
```

Figure 1: $\mathsf{EC}\,[\mathsf{H}_0] = (\mathsf{EC}[\mathsf{H}_0].\mathsf{Gen}, \mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}, \mathsf{EC}[\mathsf{H}_0].\mathsf{Verify})$: ECDSA signature scheme over to elliptic curve $\mathbb{E}$ using hash function $\mathsf{H}_0\colon \{0,1\}^* \to \mathbb{Z}_p$.

$$\begin{aligned}
&u_1 \cdot G + u_2 \cdot \mathsf{pk}_1 \\
=&\mathsf{H}_1(m_1) \cdot s^{-1} \cdot G + r \cdot s^{-1} \cdot \left(\mathsf{pk}_0 + \frac{\mathsf{H}_0(m_0) - \mathsf{H}_1(m_1)}{r} \cdot G\right) \\
=&s^{-1} \cdot G\,(\mathsf{H}_1(m_1) + r \cdot \mathsf{sk}_0 + \mathsf{H}_0(m_0) - \mathsf{H}_1(m_1)) \\
=&s^{-1} \cdot G\,(r \cdot \mathsf{sk}_0 + \mathsf{H}_0(m_0)) \\
=&t \cdot (\mathsf{H}_0(m_0) + r\mathsf{sk}_0)^{-1} \cdot (\mathsf{H}_0(m_0) + r\mathsf{sk}_0) \cdot G = t \cdot G
\end{aligned}$$

Since the $x$-coordinate of $t \cdot G$ equals $r \pmod p$, it holds that $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m_1) = 1$.
■

The RKA from Lemma 3.1 can be extended to an RKA between the schemes $\mathsf{EC}[\mathsf{H}_0]$ and $\mathsf{REC}[\mathsf{H}_1]$ such that a valid signature under $\mathsf{pk}_0$ for a *prefixed* message $\mathsf{pm} \leftarrow (\mathsf{pk}_1, m)$ in $\mathsf{EC}[\mathsf{H}_0]$ is also valid in $\mathsf{REC}[\mathsf{H}_1]$ under $\mathsf{pk}_1$ for message $m$. This RKA allows to transfer a valid signature from $\mathsf{EC}[\mathsf{H}_0]$ to a valid signature in $\mathsf{REC}[\mathsf{H}_1]$ and vice versa in case $\mathsf{pk}_0$ and $\mathsf{pk}_1$ satisfy the relation from Lemma 3.1. We formally present this RKA in the following Lemma.

**Lemma 3.2** *Let* $\mathsf{H}_0$, $\mathsf{H}_1\colon \{0,1\}^* \to \mathbb{Z}_p$ *be hash functions (modeled as random oracles). Let* $m \in \{0,1\}^*$ *and suppose that* $\sigma = (r, s)$ *is a valid signature on message* $\mathsf{pm} \leftarrow (\mathsf{pk}_1, m)$ *w.r.t.* $\mathsf{EC}[\mathsf{H}_0]$ *and public key* $\mathsf{pk}_0$*, i.e.,* $\mathsf{EC}[\mathsf{H}_0].\mathsf{Verify}(\mathsf{pk}_0, \sigma, \mathsf{pm}) = 1$*. Furthermore, suppose that* $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$ *where* $\rho = \frac{H_0(\mathsf{pm}) - H_1(\mathsf{pm})}{r} \pmod p$*. Then* $\sigma$ *is also a valid signature on message* $m$ *w.r.t.* $\mathsf{REC}[\mathsf{H}_1]$ *and public key* $\mathsf{pk}_1$*, i.e.,* $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m) = 1$.

```
Algorithm $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}\,(\mathsf{sk}, m)$
00 $\mathsf{pm} \leftarrow (\mathsf{pk}, m)$
01 $\sigma \leftarrow \mathsf{EC}[\mathsf{H}_1].\mathsf{Sign}\,(\mathsf{sk}, \mathsf{pm})$
02 Return $\sigma$

Algorithm $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}\,(\mathsf{pk}, \sigma, m)$
03 $\mathsf{pm} \leftarrow (\mathsf{pk}, m)$
04 Return $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}\,(\mathsf{pk}, \sigma, \mathsf{pm})$

Algorithm $\mathsf{REC}[\mathsf{H}_1].\mathsf{RandSK}\,(\mathsf{sk}; \rho)$
00 $\mathsf{sk}' \leftarrow (\mathsf{sk} + \rho) \mod p$
01 Return $\mathsf{sk}'$

Algorithm $\mathsf{REC}[\mathsf{H}_1].\mathsf{RandPK}\,(\mathsf{pk}; \rho)$
02 $\mathsf{pk}' \leftarrow (\mathsf{pk} + \rho \cdot G)$
03 Return $\mathsf{pk}'$
```

Figure 2: Key-prefixed version of the ECDSA signature scheme with perfectly rerandomizable keys $\mathsf{REC}[\mathsf{H}_1] := (\mathsf{REC}[\mathsf{H}_1].\mathsf{Gen} = \mathsf{EC}[\mathsf{H}_1].\mathsf{Gen}, \mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}, \mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}, \mathsf{REC}[\mathsf{H}_1].\mathsf{RandSK}, \mathsf{REC}[\mathsf{H}_1].\mathsf{RandPK})$ based on the ECDSA signature scheme $\mathsf{EC}[\mathsf{H}_1]$. Above $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ denotes a hash function.

*Proof of Lemma 3.2.* We have to show that $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m) = 1$ for $\mathsf{pk}_1 = \mathsf{pk}_0 + \rho \cdot G$ and $\rho = \frac{H_0(\mathsf{pm}) - H_1(\mathsf{pm})}{r} \pmod p$, where $\mathsf{pm} \leftarrow (\mathsf{pk}_1, m)$. Note that $\sigma = (r, s)$, where $s = t^{-1}(\mathsf{H}_0(\mathsf{pm}) + r\mathsf{sk}_0) \pmod p$ and $r$ represents the $x$-coordinate of the elliptic curve point $t \cdot G$ for $t \overset{\$}{\leftarrow} \mathbb{Z}_p$. As shown in figure 2, $\mathsf{REC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, m)$ first computes the prefixed message $\mathsf{pm} \leftarrow (\mathsf{pk}_1, m)$ and then runs $\mathsf{EC}[\mathsf{H}_1].\mathsf{Verify}(\mathsf{pk}_1, \sigma, \mathsf{pm})$. The rest follows from the proof of Lemma 3.1 with $m_0 = m_1 = \mathsf{pm}$. ∎

## 3.1 Security analysis of REC

In this section, we analyze the one-per message unforgeability of the honestly rerandomizable signature scheme, or in short the **uf-cma-hrk1** security of the scheme $\mathsf{REC}[\mathsf{H}_1]$. We prove the following theorem.

**Theorem 3.3** *Let $\mathsf{H}_0, \mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p$ be hash functions (modeled as random oracles). Let $\mathcal{A}$ be an algorithm that plays in the game $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$. Then there exists an algorithm $\mathcal{C}$ running in roughly the same time as $\mathcal{A}$, such that*

$$\mathsf{Adv}^{\mathcal{C}}_{\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}} \geq \left( \mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} - \frac{q_{\mathsf{H}_1}^2}{p} \right) \cdot \frac{1}{q}$$

*where $q_{\mathsf{H}_1}$ and $q$ are the number of random oracle queries and* `Rand` *queries, respectively, that $\mathcal{A}$ makes.*

Before providing the full formal proof of Theorem 3.3, we give some intuition on how we overcome the main difficulties in our simulation. At a high level, the idea is to reduce the **uf-cma-hrk1** security of the additively rerandomizable ECDSA construction $\mathsf{REC}[\mathsf{H}_1]$ from the **uf-cma1** security of ECDSA construction $\mathsf{EC}[\mathsf{H}_0]$. Therefore, the proof essentially consists of building a reduction $\mathcal{C}$ trying to come up with a valid forgery to win the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game, by simulating the $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$ game to adversary $\mathcal{A}$ using the RKA from Lemma 3.2. In the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game, $\mathcal{C}$ obtains a public key $\mathsf{pk}_{\mathcal{C}}$ from its challenger. It can query an oracle `Sign` to get signatures w.r.t. $\mathsf{pk}_{\mathcal{C}}$. $\mathcal{C}$ also has access to a random oracle $\mathsf{H}_0$. $\mathcal{C}$'s goal is to somehow embed its public key $\mathsf{pk}_{\mathcal{C}}$ in one of the rerandomized public keys $\mathsf{pk}^*$ under which $\mathcal{A}$ eventually returns a forgery $(\mathsf{pk}^*, \sigma^*, m^*)$. The hope is that $\mathcal{C}$ can use $(\mathsf{pk}^*, \sigma^*, m^*)$ to win its own game $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$.

In more detail, $\mathcal{C}$'s strategy works as follows. Instead of directly using $\mathsf{pk}_{\mathcal{C}}$, $\mathcal{C}$ generates the challenge public key for $\mathcal{A}$ by additively shifting $\mathsf{pk}_{\mathcal{C}}$ with a freshly sampled $\tilde{\rho} \xleftarrow{\$} \mathcal{R}$, i.e., $\mathsf{pk} \leftarrow \mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G$. When $\mathcal{A}$ asks for a signature under a key $\mathsf{pk}' = \mathsf{pk} + \rho \cdot G$, $\mathcal{C}$ can simulate such signatures by querying its `Sign` oracle and employing the RKA from Lemma 3.2. This is because, to the adversary $\mathcal{A}$, $\mathsf{pk}'$ looks like a rerandomization of $\mathsf{pk}$, while in fact, it is derived from $\mathsf{pk}_{\mathcal{C}}$ as $\mathsf{pk}' = \mathsf{pk} + \rho \cdot G = (\mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G) + \rho \cdot G$. To make this simulation work, the random oracle $\mathsf{H}_1$ must be carefully programmed by $\mathcal{C}$ such that the relation between $\rho$, $\mathsf{H}_0$ and $\mathsf{H}_1$ satisfies $\mathsf{H}_1(m) = \mathsf{H}_0(m) - r \cdot \rho \pmod{p}$ (according to Lemma 3.2), where $(r, s) := \sigma$ is the signature[3].Note that, due to the programming of the random oracle, the first simulated signature for every message and randomness pair $(m, \rho)$ fully determines $\mathsf{H}_1(m)$. Hence, the simulated signing oracle in $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$ can be queried at most once on every input pair $(m, \rho)$. $\mathcal{C}$'s strategy to win $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ is to embed $\tilde{\rho}$ at random as an answer to one of the `Rand` queries in $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$. For signing queries w.r.t. $\tilde{\mathsf{pk}}$, $\mathcal{C}$ does not reprogram $\mathsf{H}_1$; instead, it uses $\mathsf{H}_0$ and signatures obtained from the signing oracle in $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ directly. If $\mathcal{A}$ returns a valid forgery $\sigma^*$ w.r.t. to $\mathsf{pk}^* = \tilde{\mathsf{pk}} = \mathsf{pk} + \tilde{\rho} \cdot G$, then $\mathcal{C}$ can simply use this forgery to win the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game. This is because $\mathsf{pk}^* = \mathsf{pk} + \tilde{\rho} \cdot G = \mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G + \tilde{\rho} \cdot G = \mathsf{pk}_{\mathcal{C}}$. Note that $\mathsf{pk}^*$ is *the only key* for which the forgery $\sigma^*$ is valid in game $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$. For any other key $\mathsf{pk}'$, the simulation of the signing oracle in $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$ requires to reprogram $\mathsf{H}_1$ on any message that is prefixed with $\mathsf{pk}'$. Since this involves a signing query on that very message to the signing oracle in $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$, the forgery would no longer be fresh in the latter game. This guessing on $\mathcal{C}$'s part is also the reason that our reduction is not tight.

We now provide the full formal proof.

*Proof.* For this proof, we consider an adversary $\mathcal{A}$ playing in the $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$ game relative to a random oracle $\mathsf{H}_1$. Below, we present a series of games $\boldsymbol{G}_0$ to $\boldsymbol{G}_6$ where the following holds.

$$\mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G_6}^{\mathcal{A}} = 1] + \frac{{q_{\mathsf{H}_1}}^2}{p}$$

**Game $\boldsymbol{G_0}$**: This game is equivalent to the original game, namely $\textbf{uf-cma-hrk1}^{\mathcal{A}}_{\mathsf{REC}[\mathsf{H}_1]}$. In particular, a key pair $(\mathsf{sk}, \mathsf{pk})$ is sampled as $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{REC}[\mathsf{H}_1].\mathsf{Gen}(\mathsf{par})$. The adversary $\mathcal{A}$ is given $\mathsf{pk}$ as the challenge public key and oracle access to `Rand`, `RSign` and random oracle $\mathsf{H}_1$. $\mathcal{A}$ can query `Rand` to receive a randomness $\rho$ and make a follow-up query to `RSign` to receive a signature on message $m$ with respect to the rerandomized key $\mathsf{pk}' \leftarrow \mathsf{pk} + \rho \cdot G$. In particular, $\mathcal{A}$ is allowed to query `RSign` on every input pair $(m, \rho)$ at most once. Additionally, $\mathcal{A}$ can make direct queries to the random oracle $\mathsf{H}_1$. Eventually, in order to win the game, $\mathcal{A}$ has to come up with a valid forgery $\sigma^*$ on a new message $m^*$ with respect to a randomness $\rho^*$. Since $\boldsymbol{G_0}$ proceeds as **uf-cma-hrk1** we have that $\Pr[\boldsymbol{G_0}^{\mathcal{A}} = 1] = \Pr[\textbf{uf-cma-hrk1}^{\mathcal{A}}_{\mathsf{REC}[\mathsf{H}_1]} = 1] = \mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}}$.

[3]An important aspect of this simulation is that $\mathcal{C}$ can program $\mathsf{H}_1$ whenever it observes a query $m$ to $\mathsf{H}_1$ that is prefixed with a previously rerandomized key. In particular, this can be done *before* $m$ is ever queried to the signing oracle in $\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}$.

**Game $\boldsymbol{G_1}$**: This game is similar to game $\boldsymbol{G_0}$ with the following modification. $\mathcal{A}$ is now given a public key $\widetilde{\mathsf{pk}}$ instead of $\mathsf{pk}$ (which served as the challenge public key in $\boldsymbol{G}_0$) as the challenge public key. $\widetilde{\mathsf{pk}}$ is derived as $\widetilde{\mathsf{pk}} \leftarrow \mathsf{pk} - \tilde{\rho} \cdot G$ with a freshly sampled randomness $\tilde{\rho} \xleftarrow{\$} \mathcal{R}$. The corresponding secret key is obtained as $\widetilde{\mathsf{sk}} = \mathsf{sk} - \tilde{\rho}$.

Due to the perfect rerandomizablity of keys of the rerandomizable signature scheme $\mathsf{REC}$, $\mathsf{pk}$ is indistinguishable from $\widetilde{\mathsf{pk}}$. Hence, we have $\Pr[\boldsymbol{G_0}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_1}^{\mathcal{A}} = 1]$.

**Game $\boldsymbol{G_2}$**: This game is similar to game $\boldsymbol{G_1}$ with the following modification in the $\mathtt{Rand}$ oracle. An index $j$ is sampled uniformly at random from the set $\{1, \ldots, q\}$, where $q$ is an upper bound on the number of queries to the oracle $\mathtt{Rand}$. The game returns $\tilde{\rho}$ at the $j^{\text{th}}$ $\mathtt{Rand}$ query. For all other queries, $\rho$ is sampled randomly as $\rho \xleftarrow{\$} \mathcal{R}$.

Since both $\tilde{\rho}$ and $\rho$ are sampled randomly from $\mathcal{R}$, the output distribution of the $\mathtt{Rand}$ oracle is the same in games $\boldsymbol{G_1}$ and $\boldsymbol{G_2}$. Hence, we have $\Pr[\boldsymbol{G_2}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_1}^{\mathcal{A}} = 1]$.

**Game $\boldsymbol{G_3}$**: This game behaves exactly like the game $\boldsymbol{G_2}$ with the following modifications: First, the game internally maintains a random oracle $\mathsf{H}_0$ (in addition to $\mathsf{H}_1$) in a straightforward manner, by storing a list $H_0$ of query/response pairs. Second, the game programs the oracle $\mathsf{H}_1$ by maintaining three lists $H_1$, $H_1'$ and $\Gamma$, where the first two will be used as possible replies to queries to $\mathsf{H}_1$, and $\Gamma$ stores pre-computed signatures. In the beginning of the game, $H_1$, $H_1'$ and $\Gamma$ are initially set to $\perp$ in each entry. Whenever $\mathcal{A}$ queries a message $m$ to $\mathsf{H}_1$, the values $H_1[m]$, $H_1'[m]$ and $\Gamma[m]$ are set in one of two ways depending on whether $m$ is prefixed with a public key $\mathsf{pk}'$ or not. Here, $\mathsf{pk}'$ is a rerandomized form of the public key $\widetilde{\mathsf{pk}}$ (i.e., $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ where $\rho \leftarrow \mathtt{Rand}$ is a previous answer to any $\mathtt{Rand}$ oracle query), where $\widetilde{\mathsf{pk}} = \mathsf{pk} - \tilde{\rho} \cdot G$ (see Game $\boldsymbol{G}_1$). Concretely, on query $m$ to $\mathsf{H}_1$, the lists $H_1$, $H_1'$ and $\Gamma$ are maintained in the following way:

- If $\mathsf{H}_1$ has already been programmed in a previous query, i.e., $H_1[m] \neq \perp$, return $H_1[m]$.
- Else $H_1[m] = \perp$, then sample uniformly at random $h \xleftarrow{\$} \mathbb{Z}_p$, set $H_1[m] = h$, and proceed as follows:
  - Case 1: $m$ is of the form $(\mathsf{pk}', m')$, where $\mathsf{pk}' = \widetilde{\mathsf{pk}} + \rho \cdot G = \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$, for $\rho \in \mathsf{RList}$. Derive a signature $\sigma$ as $\sigma \leftarrow \mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}', m')$ for $\mathsf{sk}' = \widetilde{\mathsf{sk}} + \rho = \mathsf{sk} + (\rho - \tilde{\rho}) \pmod p$ and parse $\sigma := (r, s)$. Then set $H_1'[m] = H_0[m] - r \cdot (\rho - \tilde{\rho}) \pmod p$ and $\Gamma[m] = \sigma$. Finally return $H_1[m]$.
  - Case 2: $m$ is not of the form $(\mathsf{pk}', m')$. Set $\Gamma[m] = \epsilon$ and return $H_1[m]$.

In both the cases, the output of $\mathsf{H}_1$ is uniformly distributed from $\mathcal{A}$'s point of view. It follows that $\Pr[\boldsymbol{G_2}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_3}^{\mathcal{A}} = 1]$.

**Game $\boldsymbol{G_4}$**: This game proceeds as the previous game with a modification in the $\mathtt{Rand}$ oracle. Upon $\mathcal{A}$ querying the $\mathtt{Rand}$ oracle, sample $\rho$ as before, then compute the rerandomized public key $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ and check if there exists a message $m$ with prefix $\mathsf{pk}'$ such that $\Gamma[m] = \epsilon$. In that case, the game aborts.

**Claim 3.4** Let $\mathsf{E}_1$ be the event that the game $\boldsymbol{G_4}$ aborts during a $\mathtt{Rand}$ query. Then, we have that $\Pr[\mathsf{E}_1] \leq \frac{{q_{\mathsf{H}_1}}^2}{p}$.

*Proof.* Event $\mathsf{E}_1$ can only occur if $\mathcal{A}$ has queried $\mathsf{H}_1$ on input $m$ with prefix $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ prior to making a query to `Rand` that returns $\rho$. Since $\mathcal{A}$ makes at most $q_{\mathsf{H}_1}$ queries to $\mathsf{H}_1$, for each query to `Rand` that the adversary $\mathcal{A}$ makes, we have that with probability $\frac{q_{\mathsf{H}_1}}{p}$ we receive a value $\rho$ such that $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ is a prefix of input $m$ that was earlier made to $\mathsf{H}_1$. Since there are at most $q$ such queries to `Rand` by taking the union bound over $q_{\mathsf{H}_1}$ we obtain $\Pr[\mathsf{E}_1] = \sum_{i=1}^{q_{\mathsf{H}_1}} \frac{q_{\mathsf{H}_1}}{p} = \frac{{q_{\mathsf{H}_1}}^2}{p}$. ■

From the above, we have that $\Pr[\boldsymbol{G_3}^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G_4}^{\mathcal{A}} = 1] + \frac{{q_{\mathsf{H}_1}}^2}{p}$.

**Game $\boldsymbol{G_5}$**: This game is similar to the game $\boldsymbol{G_4}$ except for a modification in the `RSign` oracle. Upon $\mathcal{A}'s$ query on input $(m, \rho)$, the game simulates the `RSign` oracle in the following manner. It computes the rerandomized public key $\mathsf{pk}' \leftarrow \widetilde{\mathsf{pk}} + \rho \cdot G$ and creates the public key prefixed message $\mathsf{pm} \leftarrow (\mathsf{pk}', m)$. The signature is implicitly derived via querying the simulated random oracle $\mathsf{H}_1$ (see Game $\boldsymbol{G_3}$ above) on input the prefixed message $\mathsf{pm}$. This results into $\Gamma[\mathsf{pm}] = \sigma = \mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}', m)$, which is returned as the response to the signature query.

Observe that all queries to `RSign` on input the tuple $(m, \rho)$ output the same signature. However, since ECDSA signatures are randomized, the output of `RSign` should be different with overwhelming probability for each query on the same input tuples. Here, we exploit that $\mathcal{A}$ is allowed to query `RSign` at most once for the same input pair $(m, \rho)$. Hence, the output distribution of `RSign` is identical to the distribution of the `RSign` oracle in the previous game and it holds that $\Pr[\boldsymbol{G_4}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_5}^{\mathcal{A}} = 1]$.

**Game $\boldsymbol{G_6}$**: This game is similar to game $\boldsymbol{G_5}$ except for the following changes: In the oracles `RSign` and $\mathsf{H}_1$ the game uses $\mathsf{EC}[H_0].\mathsf{Sign}$ instead of $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}$ to compute the signatures stored in $\Gamma$ (and in case of `RSign` this implicitly happens via $\mathsf{H}_1$). More precisely, when $\mathsf{H}_1$ is queried on $\mathsf{pm} = (\mathsf{pk}', m')$, where $\mathsf{pk}' = \widetilde{\mathsf{pk}} + \rho \cdot G = \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$ for $\rho \in \mathsf{RList}$, we derive $\sigma \leftarrow \mathsf{EC}[H_0].\mathsf{Sign}(\mathsf{sk}, \mathsf{pm})$, for $\mathsf{sk}' = \mathsf{sk} + (\rho - \tilde{\rho}) \pmod p$. Furthermore, upon $\mathsf{H}_1$ being queried on $m$, $\mathsf{H}_1$ returns $H_1'[m]$ instead of $H_1[m]$ whenever $\Gamma[m] \neq \bot$ and $\Gamma[m] \neq \epsilon$.

**Claim 3.5** It holds that $\Pr[\boldsymbol{G_5}^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G_6}^{\mathcal{A}} = 1]$.

*Proof.* First, note that in this game, $\mathsf{H}_1$ returns $H_0[m] - r \cdot (\rho - \tilde{\rho})$ on a message $m$ for which a signature is stored in $\Gamma$. We have to show now that when $\mathsf{H}_1$ is queried on $\mathsf{pm} = (\mathsf{pk}', m')$, where $\mathsf{pk}' = \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$ and $\mathsf{sk}' = \mathsf{sk} + (\rho - \tilde{\rho}) \pmod p$ for $\rho \in \mathsf{RList}$, we derive $\sigma \leftarrow \mathsf{EC}[H_0].\mathsf{Sign}(\mathsf{sk}, \mathsf{pm})$ (Game $\boldsymbol{G_6}$) instead of computing $\sigma \leftarrow \mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}(\mathsf{sk}', m')$ (Game $\boldsymbol{G_5}$).

To this end, we recall Lemma 3.2, which states that if $\sigma = (r, s)$ is a valid signature for $\mathsf{pm} \leftarrow (\mathsf{pk}', m')$ under $\mathsf{pk}$ w.r.t. $\mathsf{EC}[H_0]$, it is also a valid signature for $m'$ under $\mathsf{pk}' \leftarrow \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$ w.r.t. $\mathsf{REC}[\mathsf{H}_1]$, if it holds that $\mathsf{H}_1(\mathsf{pm}) = \mathsf{H}_0(\mathsf{pm}) - r \cdot (\rho - \tilde{\rho}) \pmod p$. Note that we replaced the $\mathsf{REC}[\mathsf{H}_1].\mathsf{Sign}$ procedure call on a message $m'$ in $\boldsymbol{G_5}$ by a $\mathsf{EC}[\mathsf{H}_0].\mathsf{Sign}$ procedure call on a prefixed message $\mathsf{pm} \leftarrow (\mathsf{pk}', m')$, where $\mathsf{pk}' = \mathsf{pk} + (\rho - \tilde{\rho}) \cdot G$. It remains to show that the condition $\mathsf{H}_1(\mathsf{pm}) = \mathsf{H}_0(\mathsf{pm}) - r \cdot (\rho - \tilde{\rho}) \pmod p$ holds. But since $H_1'[\mathsf{pm}] = H_0[\mathsf{pm}] - r \cdot (\rho - \tilde{\rho}) \pmod p$ is programmed accordingly (latest when `RSign` is queried), this follows directly. ■

Combining results from $\boldsymbol{G}_0$ to $\boldsymbol{G}_6$, we have that

$$\Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] + \frac{{q_{\mathsf{H}_1}}^2}{p}. \tag{1}$$

**Reduction to uf-cma1 security.** Having shown that the original $\textbf{uf-cma-hrk1}^{\mathcal{A}}_{\mathsf{REC}[\mathsf{H}_1]}$ game is indistinguishable from game $\boldsymbol{G_6}$, it remains to show that an adversary $\mathcal{A}$ winning in game $\boldsymbol{G}_6$ can be turned into an adversary $\mathcal{C}$ that wins $\textbf{uf-cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$ game with related success probability. To this end, we construct $\mathcal{C}$ that runs in the game $\textbf{uf-cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$ and simulates to $\mathcal{A}$ game $\boldsymbol{G}_6$. Thus, $\mathcal{C}$ proceeds as game $\boldsymbol{G}_6$ and leverages oracle access to its own signing oracle (with respect to its challenge public key) in the following way:

1. On input the challenge public key $\mathsf{pk}_{\mathcal{C}}$ from $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$, the adversary $\mathcal{C}$ sets $\mathsf{pk}$ to $\mathsf{pk}_{\mathcal{C}}$. Note that this implicitly sets the challenge public key in $\mathcal{C}$'s simulation of $\boldsymbol{G}_6$ to $\widetilde{\mathsf{pk}} = \mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G$. Hence, $\mathcal{C}$ runs $\mathcal{A}$ on input $\widetilde{\mathsf{pk}}$.

2. In case $\mathcal{A}$ returns a forgery $(m^*, \sigma^*, \rho^*)$ with $\rho^* \neq \tilde{\rho}$, $\mathcal{C}$ aborts.

$\mathcal{C}$ perfectly simulates $\boldsymbol{G}_6$ for $\mathcal{A}$ except in case where it aborts. Moreover, note that in case there is no abort, we have that

$$\mathsf{pk}^* = \widetilde{\mathsf{pk}} + \rho^* \cdot G = \mathsf{pk}_{\mathcal{C}} - \tilde{\rho} \cdot G + \tilde{\rho} \cdot G = \mathsf{pk}_{\mathcal{C}}.$$

From the above programming strategy, we conclude that for $\mathcal{A}$'s queries to $\mathsf{H}_1$ that are prefixed with $\mathsf{pk}^*$, the oracles $\mathsf{H}_0$ and $\mathsf{H}_1$ are identical. It remains to calculate the success probability of $\mathcal{C}$ in winnning the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game in case $\mathcal{A}$ returns a valid forgery.

**Claim 3.6** Let $\mathsf{E}_2$ be the event that $\mathcal{A}$ outputs $(m^*, \sigma^*, \rho^*)$ s.t. $(\mathsf{pm}^*, \sigma^*)$ constitutes a valid forgery in game $\textbf{uf-cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$. Then, we have that $\Pr[\mathsf{E}_2 | \boldsymbol{G_6}^{\mathcal{A}} = 1] \geq \frac{1}{q}$, where $q$ is the number of queries to the `Rand` oracle.

*Proof.* In order to prove this claim, we need to show that with probability $\frac{1}{q}$ it must hold that (1) $(\mathsf{pm}^*, \sigma^*)$ is a valid forgery in game $\textbf{uf-cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$ under public key $\mathsf{pk}_{\mathcal{C}}$ and (2) the `Sign` oracle of the $\textbf{uf-cma1}^{\mathcal{C}}_{\mathsf{EC}[\mathsf{H}_0]}$ game has not been queried on input $\mathsf{pm}^*$.
First, note that if $\sigma^*$ is a valid signature for message $(\mathsf{pk}^*, m^*)$ under the public key $\mathsf{pk}^*$ relative to $\mathsf{REC}[\mathsf{H}_1]$, then $\sigma^*$ is also a valid signature on $\mathsf{pm}^*$ under public key $\mathsf{pk}_{\mathcal{C}} = \mathsf{pk}^*$ relative to $\mathsf{EC}[\mathsf{H}_0]$, as $\mathsf{H}_0$ and $\mathsf{H}_1$ are identical for messages prefixed with $\mathsf{pk}^*$. Since there are at most $q$ possible values of $\rho^*$ and $\mathcal{C}$ chooses one of them uniformly at random, the probability that $\mathcal{C}$'s guess is correct is at least $\frac{1}{q}$. Note that from the adversary's perspective, the public key generated at index $j$ is no different than other public keys.
Second, since $(m^*, \sigma^*, \rho^*)$ is a valid forgery in $\textbf{uf-cma-hrk1}^{\mathcal{A}}_{\mathsf{REC}[\mathsf{H}_1]}$, $\mathcal{A}$ has not previously queried the `RSign` oracle on input $(m^*, \rho^*)$. Correspondingly, the `Sign` oracle of the $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$ game has also not been queried on message $\mathsf{pm}^*$ and hence, $(\mathsf{pm}^*, \sigma^*)$ is a valid forgery in $\textbf{uf-cma1}_{\mathsf{EC}[\mathsf{H}_0]}$.

■

From Eq. 1 we get the following.

$$\mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] + \frac{q_{\mathsf{H}_1}^2}{p}$$

$$\text{or,}\ \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \geq \mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} - \frac{q_{\mathsf{H}_1}^2}{p}$$

Since $\mathcal{C}$ can use a valid forgery by $\mathcal{A}$ in its own game whenever $\mathsf{E}_2$ occurs,

$$\begin{aligned}\mathsf{Adv}^{\mathcal{C}}_{\textbf{uf-cma}_{\mathsf{EC}}[\mathsf{H}_0]} &\geq \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \cdot \Pr[\mathsf{E}_2 \mid \boldsymbol{G}_6^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \cdot \frac{1}{q} \\ &\geq \left( \mathsf{Adv}^{\mathcal{A}}_{\textbf{uf-cma-hrk1}_{\mathsf{REC}[\mathsf{H}_1]}} - \frac{q_{\mathsf{H}_1}^2}{p} \right) \cdot \frac{1}{q}\end{aligned}$$

■

## 4 A Model for Hierarchical Deterministic Wallets

In this section, we introduce a formal model for hierarchical deterministic wallets. This model closely reflects the BIP32 specification [Wik18] with only minor differences which we list in Section 6. At a high level, a hierarchical deterministic wallet scheme can be visualized as a tree, where every node in the tree corresponds to a wallet. As is usual in a tree structure, the scheme originates from a root node, which contains a pair of master keys - a master public key $\mathsf{mpk}$ and a master secret key $\mathsf{msk}$ as well as a seed $\mathsf{ch}_{0,0}$ which we will refer to as chaincode from now on. We say that the root node is located at level 0 of the tree. The root can create a child node at level 1 and position $t$ by deriving a new key pair $(\mathsf{pk}_{1,t}, \mathsf{sk}_{1,t})$ and a chaincode $\mathsf{ch}_{1,t}$ from its master keys and chaincode $\mathsf{ch}_{0,0}$. This child node represents a new wallet that is initiated with the key pair $(\mathsf{pk}_{1,t}, \mathsf{sk}_{1,t})$ and chaincode $\mathsf{ch}_{1,t}$ and using these values it can in turn create a child node for level 2. This child creation process can continue recursively. Note, however, that a node at level $i$ can only create children for the *immediate* lower level, i.e., for level $i + 1$.

In our model, we distinguish between two different kinds of nodes, namely *non-hardened* and *hardened* nodes. Non-hardened nodes are, in essence, the nodes as discussed above, i.e., nodes that can be used for child creation at the next lower level. We assume that the public key and the chaincode of a non-hardened node can be corrupted by an adversary, whereas the secret key remains protected. One might think of non-hardened nodes as wallets in the hot/cold wallet setting, where the hot wallet stores the public key, the cold wallet stores the secret key and the chaincode is provided to both wallets. While the hot wallet is permanently online and thereby vulnerable to attacks, the cold wallet stays offline for the majority of the time and is therefore protected against attacks. To create a non-hardened child node at level $i$ and at position $t$, its parent must generate the child node's key pair $(\mathsf{pk}_{i,t}, \mathsf{sk}_{i,t})$ and chaincode $\mathsf{ch}_{i,t}$. We model the derivation of these values in such a way that the derivation process of $\mathsf{sk}_{i,t}$ involves the parent's secret key, while the derivation of $\mathsf{pk}_{i,t}$ and $\mathsf{ch}_{i,t}$ requires only the parent's public key and chaincode (i.e., it is independent of the parent's secret key).

Hardened nodes, on the other hand, represent the leaves of the tree, i.e., we do not consider any child derivation from hardened nodes[4]. However, in comparison to non-hardened nodes we allow secret key leakage, along with public key and chaincode leakage for hardened nodes. That is, we consider full corruption of hardened nodes. Our security goal is that the secret key leakage of a hardened node does not affect the security of any other node in the tree. As opposed to non-hardened nodes, the creation process

[4]We show in Appendix A, full version that child derivation of hardened nodes is possible under certain conditions.

of a hardened child node requires the secret key of the parent node, i.e., even for the derivation of the child's public key and chaincode . The tree structure of a hierarchical deterministic wallet scheme, containing hardened as well as non-hardened nodes can be found in Figure 3.

While hardened nodes clearly exhibit stronger security guarantees than non-hardened nodes, the advantage of non-hardened nodes lies in the child creation process. We will illustrate this advantage in the following example. In a company there might be trusted and untrusted employees. Trusted employees operate a non-hardened node, as they are trusted to properly protect their secret key, e.g., by storing it in a cold wallet. On the other hand, untrusted employees have to operate a hardened node as they might leak their secret key or simply get compromised. Assume a trusted employee maintains a non-hardened node with key pair $(\mathsf{pk}_{i,t}, \mathsf{sk}_{i,t})$ and chaincode $\mathsf{ch}_{i,t}$. Further assume that the node is operated in a hot/cold wallet setting, i.e., the tuple $(\mathsf{sk}_{i,t}, \mathsf{ch}_{i,t})$ is stored in a cold wallet and the tuple $(\mathsf{pk}_{i,t}, \mathsf{ch}_{i,t})$ is stored in a hot wallet. If the employee wishes to receive payments to different public addresses, it can simply generate these addresses by deriving non-hardened child public keys using only the information stored in its hot wallet. In particular, the cold wallet can remain offline during this process. Only when the employee wants to spend the coins it received, it has to use $\mathsf{sk}_{i,t}$ from the cold wallet to generate the secret keys corresponding to the public addresses it generated earlier.

Another example for the usefulness of non-hardened nodes is the following. Consider a company A that operates a non-hardened node with key pair $(\mathsf{pk}_{i,t}, \mathsf{sk}_{i,t})$ and chaincode $\mathsf{ch}_{i,t}$ only to receive payments from a company B. In this case, company A can simply share $\mathsf{pk}_{i,t}$ and $\mathsf{ch}_{i,t}$ with company B, which can then by itself generate non-hardened child public keys and make the payments to those addresses. Note that in this case, company A does not have to be involved in the payment process at all.

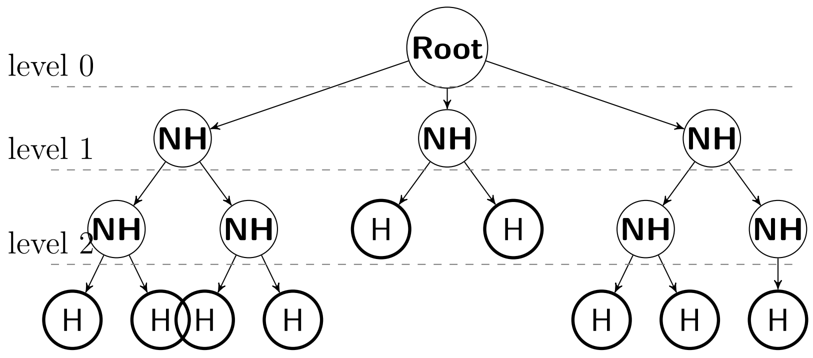

Figure 3: Tree structure of a hierarchical deterministic wallet scheme. Hardened nodes are denoted by H while non-hardened nodes are denoted by NH.

**Flat Vs Hierarchical Deterministic Wallets.** Let us now briefly discuss the main difference between the model for hierarchical deterministic wallets and the setting originally analyzed by Das et. al [DFL19] which we denote as *the flat model*. The key derivation process in the flat model works in the same way as the non-hardened key derivation in the hierarchical model with the difference that the flat model allows to derive keys only directly from the master key pair. Hardened nodes are not considered in the flat model. Therefore, the flat model basically represents a hierarchical wallet structure with non-hardened leaf nodes at level 1 (see Figure 4). Since the flat model allows only for non-hardened key derivation, the essential difference to the hierarchical model is that the flat model cannot allow for any secret key leakage as this would render the entire scheme insecure. Hierarchical wallets, on the other hand, introduce hardened nodes whose secret keys can be leaked without affecting the security of any other node in the tree.

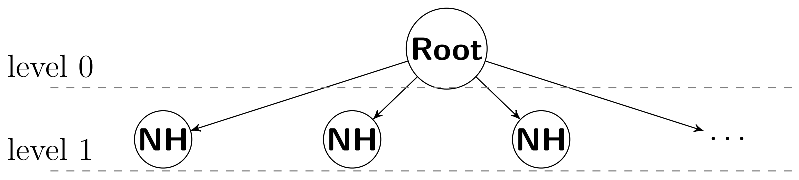

Figure 4: Tree structure of a deterministic wallet scheme in the flat setting.

In the following, we refer to a *tree* as a tuple $(h, n_{0,0}, \mathcal{N}, \mathcal{E})$ if $(\mathcal{N}, \mathcal{E})$ defines a tree of height $h$ with node set $\mathcal{N}$ and edge set $\mathcal{E}$, and a root node $n_{0,0} \in \mathcal{N}$. We denote a directed path $p_i^t$ of length $i$ from the root to a node $n_{i,t} \in \mathcal{N}$ at level $i$ and position $t$ in the tree as the corresponding ordered sequence of edges $p_i^t = (\mathsf{e}_1, \cdots, \mathsf{e}_i) \in \mathcal{E}^i$. A path of length 1 from a node $n_{i-1,s} \in \mathcal{N}$ to a node $n_{i,t} \in \mathcal{N}$ consists of only one edge which we denote as $\mathsf{e}_i^{s,t} \in \mathcal{E}$.

**Definition 4.1** (Address Structure). Let $\mathcal{T} = (h, n_{0,0}, \mathcal{N}, \mathcal{E})$ be a tree. Define a labeling of the nodes in $\mathcal{N}$ as follows.

- The root node $n_{0,0}$ is labeled by an address $\mathbf{addr}_{0,0}$.
- For $1 \leq t < |\mathcal{N}|$ and $0 \leq i \leq h$, a node $n_{i,t} \in \mathcal{N}$ is labeled by an address $\mathbf{addr}_{i,t} := (\mathbf{addr}_{0,0}, p_i^t)$.

A tuple $(\mathcal{T}, \mathbf{Addr})$ is said to be an *address structure (with respect to $\mathcal{T}$)* if $\mathbf{Addr}$ consists of a set of labels for the nodes in $\mathcal{N}$ that meets the above requirements. A prefix address $\mathbf{addr}_{i,t}^j$ for a node $n_{i,t} \in \mathcal{N}$ with $0 \leq j < i \leq h$ and $t < |N|$ is a vector of length $j+1$ consisting of the first $j+1$ components of $\mathbf{addr}_{i,t} \in \mathbf{Addr}$.

We are now ready to define hierarchical deterministic wallets. In short, these schemes consist of a $\mathsf{Setup}$ algorithm, which initializes the root node, hardened and non-hardened secret and public key derivation algorithms $\mathsf{SKDer_H}, \mathsf{PKDer_H}$ and $\mathsf{SKDer_{NH}}, \mathsf{PKDer_{NH}}$ and finally signing and signature verification algorithms $\mathsf{Sign}$ and and $\mathsf{Verify}$. We assume that public parameters $\mathsf{par}$ are known to all parties and we define appropriate secret and public key sets $\mathcal{SK}$ and $\mathcal{PK}$ respectively. We assume there exists a function $\mathsf{ToPubKey} : \mathcal{SK} \to \mathcal{PK}$ that on input a secret key from $\mathcal{SK}$ outputs the corresponding public key in $\mathcal{PK}$. Formally we have:

**Definition 4.2** (Hierarchical Deterministic Wallets). Let $\mathcal{T} = (h, n_{0,0}, \mathcal{N}, \mathcal{E})$ be a tree. A *hierarchical deterministic wallet* scheme is defined w.r.t. an address structure $(\mathcal{T}, \mathbf{Addr})$ and consists of seven algorithms $\mathsf{HDWal} = (\mathsf{Setup}, \mathsf{SKDer_H}, \mathsf{SKDer_{NH}}, \mathsf{PKDer_H}, \mathsf{PKDer_{NH}}, \mathsf{Sign}, \mathsf{Verify})$ which are defined as follows:

- $\mathsf{Setup}(1^\kappa)$: The probabilistic *setup* algorithm takes as input a security parameter $1^\kappa$ and outputs a non-hardened master key pair $(\mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0})$ with $\mathsf{msk}_{0,0} \in \mathcal{SK}$, $\mathsf{mpk}_{0,0} \in \mathcal{PK}$ and a chaincode $\mathsf{ch}_{0,0}$.

- $\mathsf{SKDer_H}(\mathsf{sk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$: The deterministic *hardened secret key derivation* algorithm takes as input a secret key $\mathsf{sk}_{i,s} \in \mathcal{SK}$, a chaincode $\mathsf{ch}_{i,s}$, an address $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ for level $i < h$, positions $s, t$, as well as an edge $\mathsf{e}_{i+1}^{s,t} \in \mathcal{E}$. It outputs a secret key $\mathsf{sk}_{i+1,t} \in \mathcal{SK}$, a chaincode $\mathsf{ch}_{i+1,t}$ and an address $\mathbf{addr}_{i+1,t} \in \mathbf{Addr}$ for level $i+1$ and position $t$.

- $\mathsf{SKDer_{NH}}(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$: The deterministic *non-hardened secret key derivation* algorithm takes as input a secret key $\mathsf{sk}_{i,s} \in \mathcal{SK}$, a public key $\mathsf{pk}_{i,s} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i,s}$, an address $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ for level $i < h$, positions $s, t$, as well as an edge $\mathsf{e}_{i+1}^{s,t} \in \mathcal{E}$. It outputs a secret key $\mathsf{sk}_{i+1,t} \in \mathcal{SK}$, a chaincode $\mathsf{ch}_{i+1,t}$ and an address $\mathbf{addr}_{i+1,t} \in \mathbf{Addr}$ for level $i+1$ and position $t$.

- $\mathsf{PKDer_H}(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$: The deterministic *hardened public key derivation* algorithm takes as input a secret key $\mathsf{sk}_{i,s} \in \mathcal{SK}$, a public key $\mathsf{pk}_{i,s} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i,s}$, an address $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ for level $i < h$, positions $s, t$, as well as an edge $\mathsf{e}_{i+1}^{s,t} \in \mathcal{E}$. It outputs a public key $\mathsf{pk}_{i+1,t} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i+1,t}$ and an address $\mathbf{addr}_{i+1,t} \in \mathbf{Addr}$ for level $i+1$ and position $t$.

- $\mathsf{PKDer_{NH}}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$: The deterministic *non-hardened public key derivation* algorithm takes as input a public key $\mathsf{pk}_{i,s} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i,s}$, an address $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ for level $i < h$, positions $s, t$, as well as an edge $\mathsf{e}_{i+1}^{s,t} \in \mathcal{E}$. It outputs a public key $\mathsf{pk}_{i+1,t} \in \mathcal{PK}$, a chaincode $\mathsf{ch}_{i+1,t}$ and an address $\mathbf{addr}_{i+1,t} \in \mathbf{Addr}$ for level $i+1$ and position $t$.

- $\mathsf{Sign}(\mathsf{sk}_{i,s}, m)$: The probabilistic *signing* algorithm takes as input a secret key $\mathsf{sk}_{i,s}$ and a message $m$. It outputs a signature $\sigma$.

- $\mathsf{Verify}(\mathsf{pk}_{i,s}, m, \sigma)$: The probabilistic *verification* algorithm takes as input a public key $\mathsf{pk}_{i,s}$, a message $m$ and a signature $\sigma$. It outputs 0 or 1.

A hierarchical deterministic wallet is *correct*, if a secret and public key pair is derived correctly using the algorithms $\mathsf{SKDer_H}, \mathsf{PKDer_H}$ or $\mathsf{SKDer_{NH}}, \mathsf{PKDer_{NH}}$, the keys represent a valid signing key pair.

We denote keys with subscript $\mathsf{nh}$ (e.g., $\mathsf{sk}_{\mathsf{nh},\cdot,\cdot}$ or $\mathsf{pk}_{\mathsf{nh},\cdot,\cdot}$) as *non-hardened* keys and keys with subscript $\mathsf{h}$ (e.g., $\mathsf{sk}_{\mathsf{h},\cdot,\cdot}$ or $\mathsf{pk}_{\mathsf{h},\cdot,\cdot}$) as *hardened* keys. A key without the subscript $\mathsf{nh}$ or $\mathsf{h}$ indicates that it can be both a non-hardened or hardened key.

**Definition 4.3** (Correctness of Hierarchical Deterministic Wallets). Let $\mathsf{HDWal}$ be a hierarchical deterministic wallet scheme with respect to an address structure $(\mathcal{T}, \mathbf{Addr})$. For any $\mathsf{e}_1^{0,s} \in \mathcal{E}$ and any $(\mathsf{ch}_{0,0}, \mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{mpk}_{\mathsf{nh},0,0}) \in \mathsf{Setup}(1^\kappa)$, we define tuples $(\mathsf{sk}_{\mathsf{h},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s})$ and $(\mathsf{pk}_{\mathsf{h},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s})$ as

$$(\mathsf{sk}_{\mathsf{h},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s}) := \mathsf{SKDer_H}(\mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,s})$$
$$(\mathsf{pk}_{\mathsf{h},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s}) := \mathsf{PKDer_H}(\mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,s})$$

and tuples $(\mathsf{sk}_{\mathsf{nh},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s})$ and $(\mathsf{pk}_{\mathsf{nh1},s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s})$ as

$$(\mathsf{sk}_{\mathsf{nh},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s}) := \mathsf{SKDer}_{\mathsf{NH}}(\mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,s})$$
$$(\mathsf{pk}_{\mathsf{nh},1,s}, \mathsf{ch}_{1,s}, \mathbf{addr}_{1,s}) := \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{mpk}_{\mathsf{nh},0,0}, \mathsf{ch}_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}_1^{0,s}).$$

Further, for any $\mathbf{addr}_{i-1,s} \in \mathbf{Addr}$, and any edge $\mathsf{e}_i^{s,t} \in \mathcal{E}$ we define the tuples $(\mathsf{sk}_{\mathsf{h},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t})$ and $\left(\mathsf{pk}_{\mathsf{h},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}\right)$ recursively as

$$(\mathsf{sk}_{\mathsf{h},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}) := \mathsf{SKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i-1,s}, \mathsf{ch}_{i-1,s}, \mathbf{addr}_{i-1,s}, \mathsf{e}_i^{s,t})$$
$$\left(\mathsf{pk}_{\mathsf{h},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}\right) := \mathsf{PKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i-1,s}, \mathsf{ch}_{i-1,s}, \mathbf{addr}_{i-1,t}, \mathsf{e}_i^{s,t})$$

and tuples $(\mathsf{sk}_{\mathsf{nh},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t})$ and $\left(\mathsf{pk}_{\mathsf{nh},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}\right)$ as

$$(\mathsf{sk}_{\mathsf{nh},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}) := \mathsf{SKDer}_{\mathsf{NH}}(\mathsf{sk}_{\mathsf{nh},i-1,s}, \mathsf{ch}_{i-1,s}, \mathbf{addr}_{i-1,s}, \mathsf{e}_i^{s,t})$$
$$\left(\mathsf{pk}_{\mathsf{nh},i,t}, \mathsf{ch}_{i,t}, \mathbf{addr}_{i,t}\right) := \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{\mathsf{nh},i-1,s}, \mathsf{ch}_{i-1,s}, \mathbf{addr}_{i-1,s}, \mathsf{e}_i^{s,t})$$

$\mathsf{HDWal}$ is *correct* if for all $m \in \{0,1\}^*$, all $1 \leq i \leq h$, all $1 \leq t \leq (1-d^{h+1})/(1-d)$, and all $(\mathsf{ch}_{0,0}, \mathsf{msk}_{\mathsf{nh},0,0}, \mathsf{mpk}_{\mathsf{nh},0,0}) \in \mathsf{Setup}(1^\kappa)$ it holds that

$$\Pr_{\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{h},i,t}, m)}[\mathsf{Verify}(\mathsf{pk}_{\mathsf{h},i,t}, \sigma, m) = 1] = 1$$
$$\wedge \Pr_{\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{nh},i,t}, m)}[\mathsf{Verify}(\mathsf{pk}_{\mathsf{nh},i,t}, \sigma, m) = 1] = 1.$$

## 4.1 Oracles

Let us now describe the general capability and influence that the adversary has over the hierarchical wallet schemes. An adversary is allowed to create new hardened and non-hardened nodes in the tree. Furthermore, the adversary can corrupt the hot wallet of all non-hardened nodes, thereby learning the public key and the chaincode of these nodes, as well as learning the secret key and chaincode of the hardened nodes. As we mentioned earlier, since hardened keys are given to untrustworthy nodes, the adversary is able to corrupt both their hot and cold wallets and as such, we do not consider the hardened nodes to derive new children. One way to look at hardened nodes, is that such nodes are the root of a new tree. We will later show in App. A, full version, that an adversary cannot distinguish hardened key pairs from freshly generated keys except with negligible probability. Therefore, our model can be recursively extended to consider settings where the hardened nodes can also derive new keys. Finally, the adversary can query any node on a freely chosen message $m$ and receive a signature for this message. To model the above mentioned capabilities, we describe the oracles which the adversary gets access to in the unlinkability game $\mathbf{unl}_{\mathsf{HDWal}}$ and the unforgeability game $\mathbf{wufcma1}_{\mathsf{HDWal}}$.

Initially, two lists $\mathsf{SK} = \emptyset$ and $\mathsf{CH} = \emptyset$ are initialized. These are used throughout the oracles to bookkeep which secret keys and chaincodes have been leaked to the adversary. In the following, we consider a fixed address structure $(\mathcal{T}, \mathbf{Addr})$.

- **Hardened Child Creation** `HChildO`**:** On inputs an address $\mathbf{addr}_{i,s}$ and an edge $\mathsf{e}_{i+1}^{s,t}$ from $\mathcal{A}$, return $\perp$ if the address $\mathbf{addr}_{i,s}$ belongs to a hardened node or the address $\mathbf{addr}_{i,s}$ is not valid (i.e., $\mathbf{addr}_{i,s} \notin \mathbf{Addr}$). Further, return $\perp$, if the address $\mathbf{addr}_{i+1,t}$ exists already. Otherwise, compute the keys and chaincode $(\mathsf{sk}_{\mathsf{h},i,s}, \mathsf{pk}_{\mathsf{h},i,s})$ and $\mathsf{ch}_{i,s}$ for the node $\mathbf{addr}_{i,s}$ by recursively deriving keys along the path in the tree, starting from the first node in the path that has already been assigned a key. Create a hardened child with address $\mathbf{addr}_{i+1,t}$ as follows. Generate keypair $(\mathsf{sk}_{\mathsf{h},i+1,t}, \mathsf{pk}_{\mathsf{h},i+1,t})$ by executing both secret and public key derivation algorithms.

$$(\mathsf{sk}_{\mathsf{h},i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t}) \leftarrow \mathsf{SKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$$
$$(\mathsf{pk}_{\mathsf{h},i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t}) \leftarrow \mathsf{PKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t}).$$

  Return $\mathsf{pk}_{\mathsf{h},i+1,t}$.

- **Non-Hardened Child Creation** `NHChildO`**:** On inputs an address $\mathbf{addr}_{i,s}$ and an edge $\mathsf{e}_{i+1}^{s,t}$ from $\mathcal{A}$, return $\perp$ if the address $\mathbf{addr}_{i,s}$ belongs to a hardened node or the address $\mathbf{addr}_{i,s}$ is not valid (i.e., $\mathbf{addr}_{i,s} \notin \mathbf{Addr}$). Further, return $\perp$, if the address $\mathbf{addr}_{i+1,t}$ exists already. Otherwise, compute the keys and chaincode $(\mathsf{sk}_{\mathsf{h},i,s}, \mathsf{pk}_{\mathsf{h},i,s})$ and $\mathsf{ch}_{i,s}$ for the node $\mathbf{addr}_{i,s}$ by recursively deriving keys along the path in the tree, starting from the first node in the path that has already been assigned a key. Create a non-hardened child with address $\mathbf{addr}_{i+1,t}$ as follows. Generate keypair $(\mathsf{sk}_{\mathsf{nh},i+1,t}, \mathsf{pk}_{\mathsf{nh},i+1,t})$ by executing both key derivation algorithms

$$(\mathsf{sk}_{\mathsf{nh},i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t}) \leftarrow \mathsf{SKDer}_{\mathsf{NH}}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t})$$
$$(\mathsf{pk}_{\mathsf{nh},i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t}) \leftarrow \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}_{i+1}^{s,t}).$$

  Return $\mathsf{pk}_{\mathsf{nh},i+1,t}$.

- **Signing** `HDSignO`**:** On input message $m$ and an address $\mathbf{addr}_{i,s}$ from $\mathcal{A}$, proceed as follows. Return $\perp$ if the address $\mathbf{addr}_{i,s}$ is not valid (i.e., $\mathbf{addr}_{i,s} \notin \mathbf{Addr}$). Further, check if $\mathbf{addr}_{i,s}$ has already been queried to either `NHChildO` or `HChildO` and return $\perp$ if this is not the case. Let $\mathsf{sk}_{i,s}$ be the secret key for the node with address $\mathbf{addr}_{i,s}$. Then compute a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{i,s}, m)$, add $m$ to the message list $\mathsf{SigList}[\mathbf{addr}_{i,s}]$ and return $\sigma$.[5]

- **Chaincode Leakage** `CHLeakO`**:** On input an address $\mathbf{addr}_{i,s}$ from $\mathcal{A}$, check if $\mathbf{addr}_{i,s}$ has already been queried to either `NHChildO` or `HChildO` and return $\perp$ if this is not the case. Set $\mathsf{CH}[\mathbf{addr}_{i,s}] = 1$ to denote that the chaincode $\mathsf{ch}_{i,s}$ of address $\mathbf{addr}_{i,s}$ has been leaked and return $(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s})$.

- **Secret Key Leakage (for hardened node)** `SKLeakO`**:** On input an address $\mathbf{addr}_{i,s}$ from $\mathcal{A}$, check if the address is that of the root, i.e., $\mathbf{addr}_{i,s} = \mathbf{addr}_{0,0}$ or if the address belongs to a non-hardened node; in this case, return $\perp$. Further, check if $\mathbf{addr}_{i,s}$ has already been queried to either `NHChildO` or `HChildO` and return $\perp$ if this is not the

[5]In case of one-per message unforgeability, the oracle aborts if it has been queried previously on input $(m, \mathbf{addr}_{i,s})$.

case. Else, set $\mathsf{SK}[\mathbf{addr}_{i,s}] = 1$ and $\mathsf{CH}[\mathbf{addr}_{i,s}] = 1$ to denote that the secret key $\mathsf{sk}_{h,i,s}$ and the chaincode $\mathsf{ch}_{i,s}$ of address $\mathbf{addr}_{i,s}$ have been leaked and return $(\mathsf{sk}_{h,i,s}, \mathsf{ch}_{i,s})$.

## 4.2 Unlinkability

Intuitively, the notion of unlinkability for hierarchical deterministic wallets guarantees that public keys in the tree, i.e., public keys that have been derived directly or indirectly from the master key of the tree root, cannot be distinguished from from a freshly generated public key. More concretely, the distribution of public keys from the tree should be computationally indistinguishable from a distribution of public keys that have been derived from an independently chosen master key. While this is a valuable privacy notion, it does not quite model practical scenarios in the hot/cold wallet setting. Recall that this setting assumes public keys and chaincodes to be stored in hot wallets, which are prone to corruptions. Therefore, we extend the unlinkability notion as described above in the following way. We consider hot wallet corruption upon which the public key and chaincode of the corrupted wallet are leaked. This extended notion gives more power to the adversary and is more close to the capabilities that an adversary has in real life scenarios. Naturally, the adversary can distinguish the distribution of keys derived from public keys of corrupted hot wallets from a distribution of public keys that have been derived from an independently chosen master key. Therefore, in our new unlinkability notion the adversary should not be able to distinguish the distribution of keys derived from non-compromised hot wallets and keys derived from independently chosen master keys.

In the following we describe the unlinkability game $\mathbf{unl}_{\mathsf{HDWal}}$ with respect to a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. In the first step of the game, the challenger generates a fresh master key pair and a chaincode via the execution of $\mathsf{Setup}(1^\kappa)$. The adversary receives the master public key as input and obtains access to all oracles as described in subsection 4.1. At some point, the adversary outputs an address $\mathbf{addr}_{i,s}$ and an edge $\mathsf{e}_{i+1}^{s,t}$ and receives a public key from the challenger. This public key is either the correct key for the node at address $\mathbf{addr}_{i,s}$ or a public key derived for a random address from a fresh master public key. $\mathcal{A}$ wins the game if it can successfully distinguish these two scenarios. In the following we give a detailed description of the game $\mathbf{unl}_{\mathsf{HDWal}}$:

<u>Game $\mathbf{unl}_{\mathsf{HDWal}}$:</u>

- **Setup Phase:** The challenger computes $(\mathsf{ch}_{0,0}, \mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0}) \leftarrow \mathsf{Setup}(1^\kappa)$ and sends $\mathsf{mpk}_{0,0}$ to $\mathcal{A}$.

- **Online Phase:** On input the security parameter and the master public key $\mathsf{mpk}_{0,0}$, the adversary $\mathcal{A}$ is allowed to make queries to the oracles as explained in subsection 4.1.

- **Output Phase:** Eventually, $\mathcal{A}$ chooses an address $\mathbf{addr}_{i,s}$, an edge $\mathsf{e}_{i+1}^{s,t}$ and a value $c \in \{\mathsf{h}, \mathsf{nh}\}$ and sends them to the challenger. Let $(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s})$ be the key pair and $\mathsf{ch}_{i,s}$ the chaincode of the node at address $\mathbf{addr}_{i,s}$. If the address $\mathbf{addr}_{i,s}$ belongs to a hardened node, $\mathcal{C}$ returns $\perp$. Otherwise, the challenger chooses a bit $b \xleftarrow{\$} \{0,1\}$ and generates a public key $\mathsf{pk}_{i+1,t}$ as follows:

  – If $b = 0$:

    * <u>If $c = \mathsf{h}$:</u> $\mathcal{C}$ computes $(\mathsf{pk}_{\mathsf{h},i+1,t}, \cdot, \cdot) \leftarrow \mathsf{PKDer}_{\mathsf{H}}(\mathsf{sk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$.
    * <u>If $c = \mathsf{nh}$:</u> If the chaincode for $\mathbf{addr}_{i,s}$ or any of its prefix addresses has been leaked, i.e., $\mathsf{CH}[\mathbf{addr}^{j}_{i,s}] = 1$, for any $j < i$, then $\mathcal{C}$ returns $\perp$. Else, $\mathcal{C}$ computes $(\mathsf{pk}_{\mathsf{nh},i+1,t}, \cdot, \cdot) \leftarrow \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{\mathsf{nh},i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$.
  - If $b = 1$: The challenger computes $(\mathsf{ch}'_{0,0}, \mathsf{msk}'_{0,0}, \mathsf{mpk}'_{0,0}) \leftarrow \mathsf{Setup}(1^{\kappa})$.
    * <u>If $c = \mathsf{h}$:</u> $\mathcal{C}$ derives a public key $\mathsf{pk}'_{\mathsf{h},1,t} \leftarrow \mathsf{PKDer}_{\mathsf{H}}(\mathsf{msk}'_{0,0}, \mathsf{ch}'_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}^{0,t}_{1})$.
    * <u>If $c = \mathsf{nh}$:</u> If the chaincode for $\mathbf{addr}_{i,s}$ or any of its prefix addresses has been leaked, i.e., $\mathsf{CH}[\mathbf{addr}^{j}_{i,s}] = 1$, for any $j < i$, then $\mathcal{C}$ returns $\perp$. Else $\mathcal{C}$ derives a public key $\mathsf{pk}'_{\mathsf{nh},1,t} \leftarrow \mathsf{PKDer}_{\mathsf{NH}}(\mathsf{mpk}'_{0,0}, \mathsf{ch}'_{0,0}, \mathbf{addr}_{0,0}, \mathsf{e}^{0,t}_{1})$.
  - Based on the value of $b$ and $c$, the challenger sends to the adversary either $\mathsf{pk}_{\mathsf{h},i+1,t}$ or $\mathsf{pk}_{\mathsf{nh},i+1,t}$ or $\mathsf{pk}'_{\mathsf{h},1,t}$ or $\mathsf{pk}'_{\mathsf{nh},1,t}$.

- The adversary can continue to make oracle queries under the restrictions as mentioned above.

- Eventually, $\mathcal{A}$ outputs a bit $b'$ and wins the game if $b = b'$.

We define the advantage of an adversary $\mathcal{A}$ in $\mathbf{unl}_{\mathsf{HDWal}}$ as

$$\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unl}_{\mathsf{HDWal}}} := \left| \Pr[\mathbf{unl}^{\mathcal{A}}_{\mathsf{HDWal}} = 1] - \frac{1}{2} \right| .$$

**On Forward Unlinkability.** The model of hierarchical wallets as defined in Definition 4.2 in Section 4 is stateless. In other words, each node in the tree maintains a fixed chaincode $\mathsf{ch}_{i,s}$ which is used as an input parameter for the child key derivation algorithms. If the (non-hardened) public key $\mathsf{pk}_{i,s}$ as well as the chaincode $\mathsf{ch}_{i,s}$ of a node are leaked (e.g., due to a hot wallet corruption of the node in the hot/cold wallet setting), then the adversary can as well compute the non-hardened keys in the entire sub-tree under $\mathsf{pk}_{i,s}$. Consequently, unlinkability of the sub-tree is lost. To enhance the unlinkability property, we can extend our model to a stateful variant where, each node maintains a state $\mathsf{St}^{t}_{i,s}$. On every child key derivation, the state of the node is refreshed to a new state $\mathsf{St}^{t+1}_{i,s}$. As a result of this modification, we can guarantee *forward unlinkability* for hierarchical wallets, which is similar to the standard notion of *forward security*. Precisely, on a hot wallet corruption, the adversary learns the *current state* $\mathsf{St}^{t}_{i,s}$ and the public key $\mathsf{pk}_{i,s}$ of a node. However, the existing children of this node were derived from earlier states $\mathsf{St}^{t'}_{i,s}$, for $t' < t$ - which are not known to the adversary. Thus it can no longer break the unlinkability of the existing child keys in the sub-tree under $\mathsf{pk}_{i,s}$. However, it would be able to link any future child keys derived from $\mathsf{pk}_{i,s}$.

## 4.3 Unforgeability

The notion of unforgeability for hierarchical deterministic wallets in the hot/cold wallet setting guarantees that an adversary cannot forge a signature of any uncorrupted node in the tree. In our model, non-hardened keys are always stored in hot/cold wallets, i.e., the secret keys are secured in the cold wallet storage, which cannot be corrupted by an adversary. Hardened keys, on the other hand, can be stored on any device and are thereby prone to corruption. Therefore, we allow an adversary to corrupt hardened secret keys, while non-hardened secret keys must remain uncorrupted.

In more detail, the unforgeability game proceeds as follows. The challenger generates a master key pair and a chaincode via the execution of $\mathsf{Setup}(1^\kappa)$. The adversary receives the master public key and obtains access to the oracles as described in subsection 4.1. Eventually, the adversary outputs a forgery, i.e., a message and a signature for a specific node in the tree. The adversary wins the game, if the signature is valid, the message has not been queried to the signing oracle `HDSignO` for this specific node before and the cold wallet of the node is uncorrupted. We note that a slightly weaker variant of unforgeability for hierarchical deterministic wallets is the notion of *one-per message unforgeability*, where the security game proceeds exactly as the game of the unforgeability notion with the difference that the adversary is allowed to query the `HDSignO` oracle only once for each message/address pair. We now give a detailed description of the unforgeability game $\mathbf{wufcma1}_{\mathsf{HDWal}}$.

Game $\mathbf{wufcma1}_{\mathsf{HDWal}}$:

- **Setup Phase:** The challenger computes $(\mathsf{ch}_{0,0}, \mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0}) \leftarrow \mathsf{Setup}(1^n)$ and sends $\mathsf{ch}_{0,0}$ and $\mathsf{mpk}_{0,0}$ to $\mathcal{A}$.
- **Online Phase:** On input the security parameter, the adversary $\mathcal{A}$ is allowed to make queries to the oracles as explained in subsection 4.1.
- **Output Phase:** Eventually, $\mathcal{A}$ outputs a public key $\mathsf{pk}_{i^*,s^*}$, a message $m^*$, an address $\mathbf{addr}_{i^*,s^*}$ and a signature $\sigma^*$. $\mathcal{A}$ wins if all of the following conditions hold,
  - $\mathsf{Verify}(pk_{i^*,s^*}, \sigma^*, m^*) = 1$
  - $m^* \notin \mathsf{SigList}[\mathbf{addr}_{i^*,s^*}]$
  - Either $\mathbf{addr}_{i^*,s^*}$ belongs to a non-hardened node or $\mathbf{addr}_{i^*,s^*}$ belongs to a hardened node and its secret key has not been corrupted, i.e., $\mathsf{SK}[\mathbf{addr}_{i^*,s^*}] = 0$.

We define the advantage of an adversary $\mathcal{A}$ in $\mathbf{wufcma1}_{\mathsf{HDWal}}$ as

$$\mathsf{Adv}^{\mathcal{A}}_{\mathbf{unl}_{\mathsf{HDWal}}} := \Pr[\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal}} = 1].$$

## 5 Generic Construction

In this section, we first show how to generically construct a hierarchical deterministic wallet scheme $\mathsf{HDWal}$ from a signature scheme with perfectly rerandomizable keys $\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify})$. We denote the construction of $\mathsf{HDWal}$ with respect to a signature scheme with rerandomizable keys $\mathsf{RSig}$ by $\mathsf{HDWal}[\mathsf{RSig}]$. Our generic construction $\mathsf{HDWal}[\mathsf{RSig}]$ uses internally a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{R} \times \{0,1\}^\kappa$. We detail our construction in Figure 5. Subsequently, we analyze the security of our generic construction by proving the unlinkability and the unforgeability properties of $\mathsf{HDWal}[\mathsf{RSig}]$. We defer the full proof for unlinkability of $\mathsf{HDWal}[\mathsf{RSig}]$ to Appendix A, full version. In the following subsection, we present the theorem that states that $\mathsf{HDWal}[\mathsf{RSig}]$ satisfies $\mathbf{wufcma1}_{\mathsf{HDWal}}$ security with a loss in the security reduction. We then show that this loss is indeed unavoidable which means that our security reduction is optimal.

```
Algorithm $\mathsf{HDWal[RSig].Setup(par)}$
00 $\mathsf{ch}_{0,0} \xleftarrow{\$} \{0,1\}^\kappa$
01 $(\mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0}) \xleftarrow{\$} \mathsf{RSig.Gen(par)}$
02 Return $(\mathsf{msk}_{0,0}, \mathsf{mpk}_{0,0}, \mathsf{ch}_{0,0})$

Algorithm $\mathsf{HDWal[RSig].Sign}(\mathsf{sk}_{i,s}, m)$
00 $\sigma \leftarrow \mathsf{RSig.Sign}(\mathsf{sk}_{i,s}, m)$
01 Return $\sigma$

Algorithm
$\mathsf{HDWal[RSig].Verify}(\mathsf{pk}_{i,s}, \sigma, m)$
00 $0/1 \leftarrow \mathsf{RSig.Verify}(\mathsf{pk}_{i,s}, \sigma, m)$
01 Return $0/1$

Algorithm $\mathsf{HDWal[RSig].SKDer_H}(\mathsf{sk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$
00 $(\omega, \mathsf{ch}_{i+1,t}) \leftarrow \mathsf{H}(\mathsf{sk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}^{s,t}_{i+1})$
01 $\mathsf{sk}_{i+1,t} \leftarrow \mathsf{RSig.RandSK}(\mathsf{sk}_{i,s}; \omega)$
02 $\mathbf{addr}_{i+1,t} \leftarrow \mathbf{addr}_{i,s} \parallel \mathsf{e}^{s,t}_{i+1}$
03 Return $(\mathsf{sk}_{i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t})$

Algorithm $\mathsf{HDWal[RSig].SKDer_{NH}}(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$
00 $(\omega, \mathsf{ch}_{i+1,t}) \leftarrow \mathsf{H}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}^{s,t}_{i+1})$
01 $\mathsf{sk}_{i+1,t} \leftarrow \mathsf{RSig.RandSK}(\mathsf{sk}_{i,s}; \omega)$
02 $\mathbf{addr}_{i+1,t} \leftarrow \mathbf{addr}_{i,s} \parallel \mathsf{e}^{s,t}_{i+1}$
03 Return $(\mathsf{sk}_{i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t})$

Algorithm $\mathsf{HDWal[RSig].PKDer_H}(\mathsf{sk}_{i,s}, \mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$
00 $(\omega, \mathsf{ch}_{i+1,t}) \leftarrow \mathsf{H}(\mathsf{sk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}^{s,t}_{i+1})$
01 $\mathsf{pk}_{i+1,t} \leftarrow \mathsf{RSig.RandPK}(\mathsf{pk}_{i,s}; \omega)$
02 $\mathbf{addr}_{i+1,t} \leftarrow \mathbf{addr}_{i,s} \parallel \mathsf{e}^{s,t}_{i+1}$
03 Return $(\mathsf{pk}_{i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t})$

Algorithm $\mathsf{HDWal[RSig].PKDer_{NH}}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$
00 $(\omega, \mathsf{ch}_{i+1,t}) \leftarrow \mathsf{H}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}^{s,t}_{i+1})$
01 $\mathsf{pk}_{i+1,t} \leftarrow \mathsf{RSig.RandPK}(\mathsf{pk}_{i,s}; \omega)$
02 $\mathbf{addr}_{i+1,t} \leftarrow \mathbf{addr}_{i,s} \parallel \mathsf{e}^{s,t}_{i+1}$
03 Return $(\mathsf{pk}_{i+1,t}, \mathsf{ch}_{i+1,t}, \mathbf{addr}_{i+1,t})$
```

Figure 5: Generic construction of a hierarchical deterministic wallet scheme HDWal[RSig] from a signature with perfectly rerandomizable keys RSig. HDWal[RSig] is defined w.r.t. an address structure $(\mathcal{T}, \mathbf{Addr})$, where $\mathcal{T} = (h, n_{0,0}, \mathcal{N}, \mathcal{E})$, such that $\mathbf{addr}_{i,s} \in \mathbf{Addr}$ and $\mathsf{e}^{s,t}_i \in \mathcal{E}$ for $0 \leq i \leq h$ and $1 \leq s, t \leq |\mathcal{N}|$. We denote by $(\mathsf{pk}_{i,s}, \mathsf{sk}_{i,s})$ and $\mathsf{ch}_{i,s}$ the public/secret key pair and chaincode of the node with address $\mathbf{addr}_{i,s}$. We denote by $\mathsf{H}$ a hash function $\mathsf{H}\colon \{0,1\}^* \to \mathcal{R} \times \{0,1\}^\kappa$.

## 5.1 Unforgeability of Generic Construction

We now analyze the unforgeability property of our generic construction HDWal[RSig] of a hierarchical wallet. We require the following properties from the underlying signature scheme RSig. RSig must satisfy (1) the definition of a signature scheme with rerandomizable keys as well as (2) a transitive property of the keys. We formally define the latter below.

**Definition 5.1** (Transitive Rerandomization). Let RSig = (RSig.Gen, RSig.Sign, RSig.Verify, RSig.RandSK, RSig.RandPK) be a signature scheme with perfectly rerandomizable keys. We say that RSig *transitively rerandomizes* if there exists an operation $\odot : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ s.t. for all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{RSig.Gen(par)}$ and all $(\rho, \rho') \in \mathcal{R} \times \mathcal{R}$, the values $(\mathsf{sk}', \mathsf{pk}'), (\mathsf{sk}'', \mathsf{pk}''), \tilde{\rho}$ which are defined as

$$
\begin{aligned}
&(\mathsf{sk}', \mathsf{pk}') \leftarrow (\mathsf{RSig.RandSK}(\mathsf{sk}; \rho), \mathsf{RSig.RandPK}(\mathsf{pk}; \rho))\\
&(\mathsf{sk}'', \mathsf{pk}'') \leftarrow (\mathsf{RSig.RandSK}(\mathsf{sk}'; \rho'), \mathsf{RSig.RandPK}(\mathsf{pk}'; \rho')),\\
&\tilde{\rho} = \rho \odot \rho' \text{ satisfy}\\
&(\mathsf{sk}'', \mathsf{pk}'') = (\mathsf{RSig.RandSK}(\mathsf{sk}; \tilde{\rho}), \mathsf{RSig.RandPK}(\mathsf{pk}; \tilde{\rho})).
\end{aligned}
$$

**Definition 5.2** (Invertible Rerandomization). Let $\mathsf{RSig} = (\mathsf{RSig.Gen}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK})$ be a signature scheme with perfectly rerandomizable keys. We say that $\mathsf{RSig}$ has *invertible rerandomization* if there exist (efficient) algorithms $\mathsf{RandSK}^{-1}$ and $\mathsf{RandPK}^{-1}$ s.t. for all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{RSig.Gen}(\mathsf{par})$ and all $\rho \in \mathcal{R}$ it holds

$$\mathsf{sk} = \mathsf{RandSK}^{-1}(\mathsf{RSig.RandSK}(\mathsf{sk}; \rho); \rho)$$
$$\mathsf{pk} = \mathsf{RandPK}^{-1}(\mathsf{RSig.RandPK}(\mathsf{pk}; \rho); \rho)$$

We note that the signature schemes with rerandomizable keys based on Schnorr [FKM+16], BLS [DFL19] and ECDSA (additive variant presented in Section 3 of this work and multiplicative variant presented in [DFL19]) all satisfy the properties of transitive rerandomization and invertible rerandmization as defined in Definitions 5.1, 5.2. For the Schnorr, BLS and additive ECDSA based schemes, the $\odot$ operation is a simple addition, while for the multiplicative ECDSA scheme it is a multiplication (modulo the group order $p$). Below we state our theorem for the one-per message unforgeability property of $\mathsf{HDWal}[\mathsf{RSig}]$.

**Theorem 5.3** *Let* $\mathsf{HDWal}[\mathsf{RSig}]$ *be the construction defined in Figure 5, let* $\mathsf{H}\colon \{0,1\}^* \to \mathcal{R} \times \{0,1\}^\kappa$ *be a hash function modeled as a random oracle and let* $\mathsf{RSig}$ *be a signature scheme with rerandomizable keys that satisfies the property of transitive rerandomization and invertible rerandomization as in Definitions 5.1, 5.2. Let* $\mathcal{A}$ *be an adversary playing in the game* $\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal}[\mathsf{RSig}]}$*, then there exists an algorithm* $\mathcal{C}$ *running in roughly the same time as* $\mathcal{A}$*, and that makes as many queries to the oracle* `Rand` *in* **uf-cma-hrk1** *as* $\mathcal{A}$ *makes queries to* `NHChildO`/`HChildO` *such that*

$$\mathsf{Adv}^{\mathcal{C}}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \frac{1}{4e(q_{sk}+1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}[\mathsf{RSig}]}}.$$

*where* $q_{\mathsf{sk}}$ *is the number of* `SKLeakO` *oracle queries from* $\mathcal{A}$*.*

We stated Theorem 5.3 w.r.t. the one-per message unforgeability notions of hierarchical deterministic wallet schemes and signature schemes with reradomizable keys, because these notions are sufficient in the setting of deterministic wallets. This is because wallets sign each unique transaction at most once. However, we note that we can likewise state and prove the above theorem with respect to the standard unforgeability notions, i.e., the notions that do not restrict the adversary to obtain at most one signature on a specific message.

In the following, we provide the full formal proof of Theorem 5.3.

*Proof.* The proof of Theorem 5.3 exhibits an adversary $\mathcal{C}$ who uses the adversary $\mathcal{A}$ who plays in game $\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal}[\mathsf{RSig}]}$ to win its own game $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}}_{\mathsf{RSig}}$. The main idea of our proof is that $\mathcal{C}$ guesses in advance which hardened nodes $\mathcal{A}$ might corrupt (i.e., calls the `SKLeakO` oracle on). In case the guess of $\mathcal{C}$ is wrong, $\mathcal{C}$ cannot answer all `SKLeakO` oracle queries from $\mathcal{A}$ and therefore has to abort. This leads to a polynomial loss in the number of `SKLeakO` oracle queries (i.e., $q_{\mathsf{sk}}$) in $\mathcal{C}$'s advantage in its $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}}_{\mathsf{RSig}}$ game. We use Coron's technique as presented in [Cor02] to bound this loss.

We now provide the formal proof via a series of games $\boldsymbol{G}_0$ to $\boldsymbol{G}_6$.

**Game $\boldsymbol{G}_0$**: This is the regular $\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}$ game at the beginning of which a key pair $(\mathsf{pk}, \mathsf{sk})$ is generated and the adversary $\mathcal{A}$ is given as input $\mathsf{pk}$ and oracle access to the following oracles: `HChildO`, `NHChildO`, `HDSignO`, `CHLeakO` and `SKLeakO` oracles and a random oracle $\mathsf{H}$. The random oracle $\mathsf{H}$ is internally programmed in a straight forward manner, by maintaining a list $H$. In particular, on input $s$, if $H[s] \neq \bot$, then return $H[s]$. Otherwise, sample a fresh randomness $\rho \xleftarrow{\$} \mathcal{R}$ and a fresh value as $\psi \xleftarrow{\$} \{0,1\}^\kappa$ and set $(\rho, \psi) =: H[s]$ and return $H[s]$. In addition, the game keeps a list $R$ in which it stores the randomness used to derive the keys at position $s$ and level $i$ at entry $R[i, s]$. We have that $\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}} = \Pr[\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal[RSig]}} = 1] = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1]$.

**Game $\boldsymbol{G}_1$**: Upon generating the key pair $(\mathsf{pk}, \mathsf{sk})$, the game chooses a fresh chaincode $\mathsf{ch}_{0,0} \xleftarrow{\$} \{0,1\}^\kappa$ and fresh randomness $\rho \xleftarrow{\$} \mathcal{R}$. Then it derives the root public key for the $\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}$ game as $\mathsf{mpk}_{0,0} \xleftarrow{\$} \mathsf{RSig.RandPK}(\mathsf{pk}; \rho)$, stores $\rho$ in a list as $R[0, 0] = \rho$. The game sends $\mathsf{ch}_{0,0}$ and $\mathsf{mpk}_{0,0}$ to $\mathcal{A}$.

Since the randomness $\rho$ is chosen uniformly at random from $\mathcal{R}$, the rerandomizability of keys property of the signature scheme $\mathsf{RSig}$ holds. This implies that the distributions of $(\cdot, \mathsf{mpk}_{0,0})$ and $(\cdot, \mathsf{mpk}'_{0,0}) \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par})$ are identical. Therefore, it holds that $\Pr[\boldsymbol{G}_1^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1]$.

**Game $\boldsymbol{G}_2$**: This game behaves like $\boldsymbol{G}_1$ with a modification in the `NHChildO` oracle. Upon an oracle query on input $(\mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$ the `NHChildO` oracle executes $\mathsf{PKDer}_{\mathsf{NH}}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$ and creates the public key $\mathsf{pk}_{\mathsf{nh},i+1,t}$ at level $i+1$ and position $t$ as $\mathsf{pk}_{\mathsf{nh},i+1,t} \leftarrow \mathsf{RandPK}(\mathsf{pk}; \omega \odot R[i, s])$, i.e., the public key $\mathsf{pk}_{\mathsf{nh},i+1,t}$ is derived directly from $\mathsf{pk}$ with randomness $\omega \odot R[i, s]$, where $(\omega, \cdot) \leftarrow \mathsf{H}(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathsf{e}^{s,t}_{i+1})$. The game then sets the list $R[i+1, t] = \omega \odot R[i, s]$. If any of the values $(\mathsf{pk}_{i,s}, \mathsf{ch}_{i,s}, \mathbf{addr}_{i,s}, R[i, s])$ is not defined yet, the game recursively derives the path from the root node up to $(\mathsf{pk}_{i,s}, \mathbf{addr}_{i,s})$ and updates the list up to $R[i, s]$.

Note that $\mathsf{RandPK}(\mathsf{pk}; \omega \odot R[i, s])$ and $\mathsf{RandPK}(\mathsf{pk}_{\mathsf{nh},i,s}; \omega)$ derive the same key $\mathsf{pk}_{\mathsf{nh},i+1,t}$, due to the transitive property of rerandomizable keys. Since $\omega$ and $R[i, s]$ are uniformly at random from $\mathcal{R}$, we have that $\Pr[\boldsymbol{G}_2^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_1^{\mathcal{A}} = 1]$.

**Game $\boldsymbol{G}_3$** : This game proceeds similarly to the previous game with a modification in the random oracle. The game aborts upon the adversary querying the random oracle on input $(\mathsf{sk}_{\mathsf{nh},i,s}, \cdot, \cdot)$ where $\mathsf{sk}_{\mathsf{nh},i,s}$ is either a non-hardened secret key that corresponds to a public key $\mathsf{pk}_{\mathsf{nh},i,s}$ previously output by the `NHChildO` oracle or $\mathsf{sk}_{\mathsf{nh},i,s}$ is the master secret key $\mathsf{msk}_{0,0}$ corresponding to $\mathsf{mpk}_{0,0}$.

**Claim 5.4** Let $\epsilon$ be the probability that game $\boldsymbol{G}_3$ aborts during a random oracle query. Then there exists an algorithm $\mathcal{C}_1$ playing in game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ such that $\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \epsilon$.

*Proof.* We prove this claim by providing a reduction to the $\mathbf{uf\text{-}cma\text{-}hrk1}$ security of $\mathsf{RSig}$. More concretely, we show that there exists an algorithm $\mathcal{C}_1$ with $\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \epsilon$ assuming $\mathcal{C}_1$ has access to an adversary $\mathcal{A}$ that causes $\boldsymbol{G}_3$ to abort with probability $\epsilon$. Initially, $\mathcal{C}_1$ receives as input a public key $\mathsf{pk}$ from the $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ game and chooses at random a chaincode $\mathsf{ch} \xleftarrow{\$} \{0,1\}^\kappa$. From $\mathsf{pk}$ and $\mathsf{ch}$, $\mathcal{C}_1$ can honestly simulate the `NHChildO` and `CHLeakO` oracles to $\mathcal{A}$. The simulation of the random oracle $\mathsf{H}$ works as described in $\boldsymbol{G}_3$ with the exception that instead of sampling the randomness $\rho \xleftarrow{\$} \mathcal{R}$ uniformly at random from $\mathcal{R}$, $\mathcal{C}_1$ calls the `Rand` oracle in game $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ to obtain the randomness $\rho$. A query from $\mathcal{A}$ to the `HDSignO` oracle on input $(m, \mathbf{addr}_{\cdot,\cdot})$ is forwarded to the `RSign` oracle on input $m$ and the randomness corresponding to $\mathbf{addr}_{\cdot,\cdot}$ of the $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}_1}_{\mathsf{RSig}}$ game. For a `HChildO` oracle query on input $(\mathbf{addr}_{i,s}, \mathsf{e}^{s,t}_{i+1})$, $\mathcal{C}_1$ chooses a fresh key pair (independently of $\mathsf{pk}$) $(\mathsf{sk}', \mathsf{pk}') \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par})$, assigns $(\mathsf{sk}_{\mathsf{h},i+1,t}, \mathsf{pk}_{\mathsf{h},i+1,t}) :- (\mathsf{sk}', \mathsf{pk}')$ and returns $\mathsf{pk}_{\mathsf{h},i+1,t}$. The `SKLeakO` oracle is then simulated by returning $\mathsf{sk}_{\mathsf{h},i+1,t}$ on input $\mathbf{addr}_{i+1,t}$. The simulation of the `HChildO` and `HDSignO` oracles cannot be distinguished by $\mathcal{A}$ from the oracles in $\boldsymbol{G}_3$ due to the rerandomizability of keys property of $\mathsf{RSig}$. The only way in which $\mathcal{A}$ could detect the difference between $\boldsymbol{G}_3$ and the reduction provided by $\mathcal{C}_1$ would be if the following event occurs. $\mathcal{A}$ makes a random oracle query of the form $(\mathsf{sk}_{\mathsf{nh},i,s}, \cdot, \cdot)$ where $\mathsf{sk}_{\mathsf{nh},i,s}$ is either a non-hardened secret key that corresponds to a public key $\mathsf{pk}_{\mathsf{nh},i,s}$ previously output by the `NHChildO` oracle or $\mathsf{sk}_{\mathsf{nh},i,s}$ is the secret key corresponding to $\mathsf{pk}$ (if $\mathsf{sk}_{\mathsf{nh},i,s}$ belongs to a public key $\mathsf{pk}_{\mathsf{nh},i,s}$ can be efficiently checked via the function $\mathsf{ToPubKey}(\mathsf{sk}_{\mathsf{nh},i,s})$). By Claim 5.4, this event happens with probability $\epsilon$. However, when this event occurs, $\mathcal{C}_1$ learns the secret key $\mathsf{sk}_{\mathsf{nh},i,s}$ which it can use to compute the secret key $\mathsf{sk}$ of the $\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}$ game. This is due to the transitivity and invertible rerandomization property of $\mathsf{RSig}$. $\mathcal{C}_1$ can then use $\mathsf{sk}$ to create a valid forgery in the $\mathbf{uf\text{-}cma\text{-}hrk1}^{\mathcal{C}_1}_{\mathsf{RSig}}$ game. Therefore, we have that $\mathsf{Adv}^{\mathcal{C}_1}_{\mathbf{uf\text{-}cma\text{-}hrk1}_{\mathsf{RSig}}} \geq \epsilon$. ∎

It follows that $\Pr[\boldsymbol{G}_2^{\mathcal{A}}] \leq \Pr[\boldsymbol{G}_3^{\mathcal{A}}] + \epsilon$.

**Game $\boldsymbol{G}_4$** : This game works like the previous game with a modification to the `HChildO` oracle which works as follows. Let $q_{\mathsf{sk}}$ be the number of hardened nodes that $\mathcal{A}$ corrupts via the `SKLeakO` oracle. Upon $\mathcal{A}$ querying the `HChildO` oracle, with probability $\frac{1}{q_{\mathsf{sk}}+1}$, the address of this node is added to a list $L$. Let $\mathsf{Bad}$ define the event that a node corresponding to an address in $L$ is corrupted.

Since the change in this game is only syntactical, $\mathcal{A}$'s winning probability is not affected by whether $\mathsf{Bad}$ occurs. It follows that $\Pr[\boldsymbol{G}_3^{\mathcal{A}}] = \Pr[\boldsymbol{G}_4^{\mathcal{A}}]$.

**Game $\boldsymbol{G}_5$** : This game works like the previous game with the only difference that $\boldsymbol{G}_5$ aborts in case event $\mathsf{Bad}$ occurs.

**Lemma 5.5** $\Pr[\boldsymbol{G}_4^{\mathcal{A}} = 1] \leq \Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] \cdot e$.

*Proof.* $\mathcal{A}$ can distinguish $\boldsymbol{G}_5$ from the previous game if the game aborts i.e., when the event $\mathsf{Bad}$ happens. This event happens for each `SKLeakO` query, independently, with probability $\frac{1}{q_{\mathsf{sk}}+1}$. With probability $(1 - \frac{1}{q_{\mathsf{sk}}+1})$, a `SKLeakO` oracle query does not lead to an abort. Hence, the overall probability with which the game does not abort on any `SKLeakO` oracle query can be lower bounded by $(1 - \frac{1}{q_{\mathsf{sk}}+1})^{q_{\mathsf{sk}}} \geq e^{-1}$, i.e., $\mathsf{Bad}$ occurs with probability at most $1 - e^{-1}$. As we have argued that $\mathsf{Bad}$ occurs in $\boldsymbol{G}_4$ independently of the event $\boldsymbol{G}_4 = 1$, we have that $\Pr[\boldsymbol{G}_4^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] \cdot 1/\Pr[\neg\mathsf{Bad}] \leq \Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] \cdot e$. ∎

**Game $\boldsymbol{G}_6$** : This game works like the previous game with a modification to the `HChildO` oracle which works as follows. For the nodes that are chosen to be added to the list $L$, the game derives the public key of that node as a public key of a non-hardened node. The rest of the hardened nodes are generated as $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{RSig.Gen}(\mathsf{par})$ and assigned $(\mathsf{sk}_{\mathsf{h},i+1,t}, \mathsf{pk}_{\mathsf{h},i+1,t}) := (\mathsf{sk}, \mathsf{pk})$.

**Lemma 5.6** $\Pr[\boldsymbol{G}_5^{\mathcal{A}} = 1] = \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1]$.

*Proof.* $\mathcal{A}$ can distinguish $\boldsymbol{G}_6$ from the previous game if it corrupts a hardened node which is simulated as a non-hardened node i.e., one of the nodes in the list $L$. The only other way for $\mathcal{A}$ to distinguish these two games would be if $\mathcal{A}$ was able to query the random oracle on input the secret key of a non-hardened node as this would allow to recursively compute the secret key of the corresponding child hardened node. This case is, however, has already been excluded in $\boldsymbol{G_3}$. As explained in game $\boldsymbol{G}_5$, upon $\mathcal{A}$ making a corruption query for a node in list $L$, the game aborts. Therefore, the adversary cannot distinguish this game from the previous game. ■

By the transition from game $\boldsymbol{G}_0$ to game $\boldsymbol{G}_6$, we get that

$$\mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}} = \Pr[\boldsymbol{G}_0^{\mathcal{A}} = 1] \leq \left(\Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \cdot e\right) + \epsilon$$
$$\text{or, } \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \geq \frac{1}{e} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}} - \frac{1}{e} \cdot \epsilon$$

**Reduction to uf-cma-hrk security.** Having shown that the transition from game $\mathbf{wufcma1}^{\mathcal{A}}_{\mathsf{HDWal[RSig]}}$ to the game $\boldsymbol{G}_6$ is indistinguishable, it remains to show that there exists a challenger $\mathcal{C}_2$ that simulates $\boldsymbol{G}_5$ and uses $\mathcal{A}$ to win its own game $\textbf{uf-cma-hrk1}_{\mathsf{RSig}}$. The challenger code is same as $\boldsymbol{G}_6$ with the following changes: (1) The sampling of $\rho \xleftarrow{\$} \mathcal{R}$ within the programming of $\mathsf{H}$ is replaced by a call to the oracle `Rand` (2) $\mathsf{pk}$ is replaced by the challenge public key $\mathsf{pk}_{\mathcal{C}_2}$ from the underlying game $\textbf{uf-cma-hrk1}^{\mathcal{C}_2}_{\mathsf{RSig}}$. Since the above changes are trivially indistinguishable to $\mathcal{A}$, we move on to analyze $\mathcal{C}_2$'s probability to win the $\textbf{uf-cma-hrk}^{\mathcal{C}_2}_{\mathsf{RSig}}$ game using the forgery of $\mathcal{A}$. There are two possibilities for $\mathcal{A}$; either to output a forgery for a non-hardened node or for a hardened node. We analyze each case separately and show that for both cases our simulator can win its game with non-negligible probability.

- Adversary outputs a forgery for a non-hardened node: If the adversary provides a forgery for a non-hardened node, $\mathcal{C}_2$ can always use this forgery to win the $\textbf{uf-cma-hrk}^{\mathcal{C}_2}_{\mathsf{RSig}}$ game. Therefore, the overall probability of $\mathcal{C}_2$ winning the game in case of $\mathcal{A}$ generating a forgery for a non-hardened node is:

$$\mathsf{Adv}^{\mathcal{C}_2}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}} \geq \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \geq \frac{1}{e} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal[RSig]}}} - \frac{\epsilon}{e}$$

- Adversary outputs a forgery for a hardened node: We now compute the probability that the game aborts in case the adversary generates a forgery for a hardened node.

Let $i^*$ be the index of the hardened node for which the adversary outputs a forgery. In this case $\mathcal{C}_2$ needs to abort if $i^*$ was sampled randomly. Recall, the probability that $i^*$ has been sampled at random is $1 - \frac{1}{q_{\mathsf{sk}}+1}$. Therefore, the overall probability of the simulator winning the game in case of $\mathcal{A}$ generating a forgery for a hardened node is:

$$\mathsf{Adv}^{\mathcal{C}_2}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}} \geq \Pr[\boldsymbol{G}_6^{\mathcal{A}} = 1] \cdot \frac{1}{q_{\mathsf{sk}}+1} \geq \left(\frac{1}{e} \cdot \mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal[RSig]}}} - \frac{\epsilon}{e}\right) \cdot \frac{1}{q_{\mathsf{sk}}+1}$$

We can now compose a challenger $\mathcal{C}$ from the challengers $\mathcal{C}_1$ of Claim 5.4 and $\mathcal{C}_2$, such that $\mathcal{C}$ uses adversary $\mathcal{A}$ to win in its game $\textbf{uf-cma-hrk1}^{\mathcal{C}}_{\mathsf{RSig}}$. $\mathcal{C}$ executes either of $\mathcal{C}_1$ and $\mathcal{C}_2$ with probability $\frac{1}{2}$. In order to compute $\mathcal{C}$'s advantage $\mathsf{Adv}^{\mathcal{C}}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}}$, we distinguish the following two cases:

- $\underline{\text{Case } \epsilon \geq \frac{1}{2}\mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal}}}:}$ In this case, we have by claim 5.4 that

$$\mathsf{Adv}^{\mathcal{C}_1}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}} \geq \epsilon \geq \frac{1}{2}\mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal}}}.$$

  Therefore we can lower bound $\mathcal{C}$'s advantage by

$$\mathsf{Adv}^{\mathcal{C}}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}} \geq \frac{1}{2}\mathsf{Adv}^{\mathcal{C}_1}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}} \geq \frac{\mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal}}}}{4}.$$

- $\underline{\text{Case } \epsilon < \frac{1}{2}\mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal}}}:}$ In this case, we can lower bound $\mathcal{C}_2$'s advantage by

$$\begin{aligned}
\mathsf{Adv}^{\mathcal{C}_2}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}} &\geq \left(\mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal[RSig]}}} \cdot \frac{1}{e} - \frac{\epsilon}{e}\right) \cdot \frac{1}{q_{\mathsf{sk}}+1} \\
&\geq \left(\mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal[RSig]}}} \cdot \frac{1}{e} - \frac{1}{2e} \cdot \mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal[RSig]}}}\right) \cdot \frac{1}{q_{\mathsf{sk}}+1} \\
&= \frac{1}{2e(q_{sk}+1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal[RSig]}}}
\end{aligned}$$

Hence, $\mathcal{C}$'s overall advantage can be lower bounded by

$$\begin{aligned}
\mathsf{Adv}^{\mathcal{C}}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}} &\geq \min\left(\frac{1}{2}\mathsf{Adv}^{\mathcal{C}_1}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}}, \frac{1}{2}\mathsf{Adv}^{\mathcal{C}_2}_{\textbf{uf-cma-hrk1}_{\mathsf{RSig}}}\right) \\
&\geq \frac{1}{4e(q_{sk}+1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\textbf{wufcma1}_{\mathsf{HDWal[RSig]}}}.
\end{aligned}$$

■

The proof of Theorem 5.3 incurs a polynomial loss in the number of `SKLeakO` oracle queries (i.e., $q_{\mathsf{sk}}$) in $\mathcal{C}$'s advantage in its $\textbf{uf-cma-hrk1}^{\mathcal{C}}_{\mathsf{RSig}}$ game. Interestingly, the following theorem states that this loss is inherent and that, in fact, there does not exist a tighter security reduction. In Appendix B, full version, we recall the security notion of *unforgeability under rerandomized keys* $\textbf{uf-cma-rk}_{\mathsf{RSig}}$ for a signature scheme with rerandomizable keys $\mathsf{RSig}$ as introduced in [FKM+16] and prove Theorem 5.7. Below, we denote as $\mathcal{A}_1^{\mathcal{A}_2}$ that $\mathcal{A}_1$ has black-box access to $\mathcal{A}_2$. In particular, it does not rewind $\mathcal{A}_2$.

**Theorem 5.7** *Let* $\mathsf{HDWal}$ *be an algorithm such that for any signature scheme with rerandomizable keys* $\mathsf{RSig}$*,* $\mathsf{HDWal}^{\mathsf{RSig}}$ *is a hierarchical deterministic wallet scheme. Moreover, suppose that there is a reduction* $\mathcal{R}$ *such that for every signature scheme with rerandomizable keys* $\mathsf{RSig}$ *and every adversary* $\mathcal{A}$ *running in time* $t_{\mathcal{A}}$ *with* $\epsilon_{\mathcal{A}} = \mathsf{Adv}^{\mathcal{A}}_{\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}}$*, it holds that* $\mathsf{Adv}^{\mathcal{R}^{\mathcal{A}}}_{\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}} \geq \epsilon_{\mathcal{R}}$ *and* $\mathcal{R}^{\mathcal{A}}$ *runs in time* $t_{\mathcal{R}}$*. Then there exists an algorithm* $\mathcal{M}$ *running in time* $t_{\mathcal{M}} \leq 2 \cdot t_{\mathcal{R}}$ *s.t.* $\mathsf{Adv}^{\mathcal{M}}_{\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}} \geq \epsilon_{\mathcal{R}} - \epsilon_{\mathcal{A}} \cdot \frac{2\exp(-1)}{q_{\mathsf{sk}}}$.

Theorem 5.7 implies that if there exists a reduction from $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ to $\mathbf{wufcma1}_{\mathsf{HDWal}^{\mathsf{RSig}}}$ for a signature scheme with rerandomizable keys $\mathsf{RSig}$ s.t. the reduction loses less than a factor proportional to $q_{\mathsf{sk}}$, then there exists an efficient algorithm $\mathcal{M}$ that can break the $\mathbf{uf\text{-}cma\text{-}rk}_{\mathsf{RSig}}$ security. We formulate and prove this result w.r.t. a reduction from the strongest possible security notion of signature schemes with rerandomizable keys (i.e., **uf-cma-rk**) to the restricted notion of one-per message wallet unforgeability (i.e., $\mathbf{wufcma1}_{\mathsf{HDWal}}$). Clearly, this implies that the result from Theorem 5.7 also holds for the weaker notion of **uf-cma-hrk1** for signature schemes with rerandomizable keys which we use in Theorem 5.3. We note that Theorem 5.7 can likewise be stated and proved with respect to the standard unforgeability notion of hierarchical deterministic wallet schemes, i.e., the notion that does not restrict the adversary to obtain at most one signature on a specific message.

## 6 Discussion

**On Security Parameters.** We instantiate our generic hierarchical deterministic wallet construction $\mathsf{HDWal}[\mathsf{RSig}]$ with two schemes, namely $\mathsf{REC}[\mathsf{H}_1]$ (Figure 7, full version) and $\mathsf{REC}'[\mathsf{H}_1]$ (Figure 6, full version). Note that $\mathsf{HDWal}[\mathsf{REC}[\mathsf{H}_1]]$ corresponds to the BIP32 wallet, while $\mathsf{HDWal}[\mathsf{REC}'[\mathsf{H}_1]]$ is instantiated from the multiplicatively rerandomized construction $\mathsf{REC}'[\mathsf{H}_1]$ from [DFL19], we will refer to it as BIP32-m.

First, let us recall, how to compute the bit security level of a scheme. A hierarchical wallet scheme $\mathsf{HDWal}$ is said to have a bit security level of $\kappa$ bits, if any algorithm $\mathcal{A}$ with running time $t$ and advantage $\epsilon$ in $\mathbf{wufcma1}_{\mathsf{HDWal}}$ takes *expected* running time $\frac{t}{\epsilon} \geq 2^{\kappa}$ to break the scheme for the first time. (The security level for a conventional signature scheme is defined analogously). From our Theorems C.1, C.2 (full version), we compute the bit security level of our schemes, considering an algorithm $\mathcal{A}$ with parameters $t', \epsilon'$ (in game $\mathbf{wufcma1}_{\mathsf{HDWal}}$), where $t' \approx t$ and $\epsilon' = \epsilon \cdot Q$ for some $Q \geq 1$ and where $t, \epsilon$ denote the runtime and advantage of the related forger $\mathcal{C}$ in game $\mathbf{uf\text{-}cma1}_{\mathsf{EC}}$. By assumption, $\mathsf{EC}$ satisfies $\kappa = 128$ bits of security, hence $\frac{t}{\epsilon} \geq 2^{128}$. Thus, we obtain $\frac{t'}{\epsilon'} = \frac{t}{\epsilon \cdot Q} \geq \frac{2^{128}}{Q} = 2^{\kappa - \log Q}$. Our results are reported in Table 1, where we took an estimate of the practical parameters as follows: the total number of keys is $q = 2^{20}$, the number of $q_{\mathsf{sk}}$ of secret keys leaked is roughly 1% of the total number of keys $q$, i.e., $q_{\mathsf{sk}} \approx 2^{14}$.

| Scheme | Theorem Ref. | Bit Security with $\kappa = 128$ |
|---|---|---|
| BIP32 | Thm C.1, full version | $\log(Q) = \log(q \cdot 4e \cdot q_{\mathsf{sk}}) \approx 37, \kappa - \log(Q) = 91$ |
| BIP32-m | Thm C.2, full version | $\log(Q) = \log(4e \cdot q_{\mathsf{sk}}) \approx 17, \kappa - \log(Q) = 111$ |

Table 1: Bit Security Level of BIP32 and BIP32-m, relying on **uf-cma1** of $\mathsf{EC}[\mathsf{H}_0]$

**On BIP32 Parameters.** Our construction of $\mathsf{HDWal}[\mathsf{REC}[\mathsf{H}_1]]$ gives us the BIP32 construction as specified in [Wik18]. Here we list the exact parameters used in BIP32 and minor differences of BIP32 with our construction $\mathsf{HDWal}[\mathsf{REC}[\mathsf{H}_1]]$.

- Each node can derive at most $2^{32}$ children nodes.
- $\mathsf{e}_{\cdot,\cdot}$ is chosen from $\{0,1\}^{32}$, which allows each non-hardened node to generate $2^{31}$ non-hardened and $2^{31}$ hardened child keys.
- A child key is derived as a hardened or a non-hardened node based on whether $\mathsf{e}_{\cdot,\cdot} \geq 2^{31}$ or $\leq 2^{31}$ respectively. However, this is syntactical, and does not affect our security analysis.
- Although at each level, the total number of derived keys can be at most $(2^{32}) \cdot p$, where $p$ is the number of parent nodes in the immediate upper level, we do not imagine that all of these keys are derived at every level. As can be seen, this would already exceed our parameter $q = 2^{20}$, as selected above.
- The chaincode $\mathsf{ch}_{\cdot,\cdot}$ is chosen from $\{0,1\}^{256}$.
- The input parameter $\mathbf{addr}_{\cdot,\cdot}$ to the key derivation algorithms is set to an empty string. We use this parameter to indicate the position in the tree, at which the child key is derived and to ensure that the actual BIP32 derivation algorithms are called on the proper inputs for this position.
- The input parameter $\mathbf{addr}_{\cdot,\cdot}$ to the key derivation algorithms is set to an empty string $\lambda$. Let us briefly explain this syntactical difference. In our Definition 4.2, $\mathbf{addr}_{\cdot,\cdot} \neq \lambda$ is provided as input. This makes the user aware of the position in the tree, at which the child key is derived and makes sure that the actual BIP32 derivation algorithms are called on the proper inputs for this position in the tree.

**Open Questions.** Finally, let us mention some interesting open questions that can be answered in future works:

- Is it possible to remove the one per-message restriction and prove the security of the additively rerandomizable ECDSA scheme in the **uf-cma-hrk** notion? Additionally, is there a tight reduction to **uf-cma-hrk**?
- Can we improve the tightness of **uf-cma1** security [FKP17] of ECDSA from the semi-logarithm problem?

## References

[ABFF09] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3), January 2009. (Cited on page 7.)

[ADE$^+$20] Nabil Alkeilani Alkadri, Poulami Das, Andreas Erwig, Sebastian Faust, Juliane Krämer, Siavash Riahi, and Patrick Struck. Deterministic wallets in a quantum world. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20: 27th Conference on Computer and Communications Security*, pages 1017–1031, Virtual Event, USA, November 9–13, 2020. ACM Press. (Cited on page 7.)

[AGKK19] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. A formal treatment of hardware wallets. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 426–445, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany. (Cited on page 7.)

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany. (Cited on page 7.)

[BH19] Joachim Breitner and Nadia Heninger. Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 3–20, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany. (Cited on page 7.)

[Bit18] BitcoinExchangeGuide. CipherTrace Releases Report Exposing Close to $1 Billion Stolen in Crypto Hacks During 2018. https://coinexchangeguide.com/ciphertrace-releases-report-exposing-close-to-1-billion-stolen-in_-crypto-hacks-during-2018/, 2018. (Cited on page 2.)

[Blo18] Bloomberg. How to Steal $500 Million in Cryptocurrency. http://fortune.com/2018/01/31/coincheck-hack-how/, 2018. (Cited on page 2.)

[BR18] Michael Brengel and Christian Rossow. Identifying key leakage of bitcoin users. In Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis, editors, *Research in Attacks, Intrusions, and Defenses*, pages 623–643, Cham, 2018. Springer International Publishing. (Cited on page 7.)

[But13] Vitalik Buterin. Deterministic Wallets, Their Advantages and their Understated Flaws. https://bitcoinmagazine.com/articles/deterministic-wallets-advantages-flaw-1385450276/, 2013. (Cited on page 2.)

[CEV14] Nicolas T. Courtois, Pinar Emirdag, and Filippo Valsorda. Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor RNG events. Cryptology ePrint Archive, Report 2014/848, 2014. https://eprint.iacr.org/2014/848. (Cited on page 7.)

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. (Cited on page 6, 28.)

[DFL19] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 651–668. ACM Press, November 11–15, 2019. (Cited on page 3, 4, 5, 7, 8, 10, 11, 19, 28, 33.)

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *2018 IEEE Symposium on Security and Privacy*, pages 980–997, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. (Cited on page 7.)

[Ele13] Version bytes for BIP32 extended public and private keys. https://electrum.readthedocs.io/en/latest/xpub_version_bytes.html, 2013. (Cited on page 2.)

[FF13] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of Schnorr signatures. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 444–460, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. (Cited on page 7.)

[FKM$^+$16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 301–330, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany. (Cited on page 4, 7, 28, 32.)

[FKP16] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (EC)DSA signatures. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1651–1662, Vienna, Austria, October 24–28, 2016. ACM Press. (Cited on page 7.)

[FKP17] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the one-per-message unforgeability of (ec)dsa and its variants. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 519–534, Cham, 2017. Springer International Publishing. (Cited on page 5, 9, 34.)

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*, volume 9696 of *Lecture Notes in Computer Science*, pages 156–174, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany. (Cited on page 7.)

[GS15] Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015: 19th International Conference on Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 497–504, San Juan, Puerto Rico, January 26–30, 2015. Springer, Heidelberg, Germany. (Cited on page 7.)

[KK18] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. *Journal of Cryptology*, 31(1):276–306, January 2018. (Cited on page .)

[KMOS19] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. Cryptology ePrint Archive, Report 2019/1328, 2019. https://eprint.iacr.org/2019/1328. (Cited on page 7.)

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 33–61, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. (Cited on page 7.)

[Led14] Ledger Support,Ledger Nano OS. https://support.ledger.com/hc/en-us/articles/115005297709-Export-your-accounts, 2014. (Cited on page 2.)

[LFA20] Adriano Di Luzio, Danilo Francati, and Giuseppe Ateniese. Arcula: A secure hierarchical deterministic wallet for multi-asset blockchains. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20: 19th International Conference on Cryptology and Network Security*, volume 12579 of *Lecture Notes in Computer Science*, pages 323–343, Vienna, Austria, December 14–16, 2020. Springer, Heidelberg, Germany. (Cited on page 7.)

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1837–1854, Toronto, ON, Canada, October 15–19, 2018. ACM Press. (Cited on page 7.)

[MPs19] Antonio Marcedone, Rafael Pass, and abhi shelat. Minimizing trust in hardware wallets with two factor signatures. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data*

*Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 407–425, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany. (Cited on page 7.)

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. (Cited on page 7.)

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. https://ia.cr/2004/332. (Cited on page 8.)

[Ske18] Rhys Skellern. Cryptocurrency Hacks: More Than $2b USD lost between 2011-2018. https://medium.com/ecomi/cryptocurrency-hacks-more-than-2b-usd-lost-between-2011-2018_-67054b342219, 2018. (Cited on page 2.)

[Tre14] Trezor Wiki,Cryptocurrency standards,Hierachical deterministic wallets. https://wiki.trezor.io/Cryptocurrency_standards, 2014. (Cited on page 2.)

[TVR16] Mathieu Turuani, Thomas Voegtlin, and Michaël Rusinowitch. Automated verification of electrum wallet. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 27–42, Christ Church, Barbados, February 26, 2016. Springer, Heidelberg, Germany. (Cited on page 7.)

[Wik18] Bitcoin Wiki. BIP32 proposal. https://en.bitcoin.it/wiki/BIP_0032, 2018. (Cited on page 2, 18, 34.)

[ZCC$^+$15] Zongyang Zhang, Yu Chen, Sherman S. M. Chow, Goichiro Hanaoka, Zhenfu Cao, and Yunlei Zhao. Black-box separations of hash-and-sign signatures in the non-programmable random oracle model. In Man Ho Au and Atsuko Miyaji, editors, *ProvSec 2015: 9th International Conference on Provable Security*, volume 9451 of *Lecture Notes in Computer Science*, pages 435–454, Kanazawa, Japan, November 24–26, 2015. Springer, Heidelberg, Germany. (Cited on page 7.)

# C. Deterministic Wallets in a Quantum World

This chapter corresponds to our published article in CCS 2020 [6], with minor edits. Our full version can be found in [7].

[6] N. A. Alkadri, P. Das, A. Erwig, S. Faust, J. Krämer, S. Riahi, and P. Struck. "Deterministic Wallets in a Quantum World". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020.* 2020, pp. 1017–1031. DOI: 10.1145/3372297.3423361. URL: https://doi.org/10.1145/3372297.3423361.

# Deterministic Wallets in a Quantum World

Nabil Alkeilani Alkadri[1], Poulami Das[2], Andreas Erwig[2], Sebastian Faust[2], Juliane Krämer[3],
Siavash Riahi[2], and Patrick Struck[3]

[1] CDC, Technische Universität Darmstadt, Germany
nabil.alkadri@tu-darmstadt.de
[2] CAC, Technische Universität Darmstadt, Germany
{poulami.das,andreas.erwig,sebastian.faust,siavash.riahi}@tu-darmstadt.de
[3] QPC, Technische Universität Darmstadt, Germany
{juliane,patrick}@qpc.tu-darmstadt.de

**Abstract.** Most blockchain solutions are susceptible to quantum attackers as they rely on cryptography that is known to be insecure in the presence of quantum adversaries. In this work we advance the study of quantum-resistant blockchain solutions by giving a quantum-resistant construction of a deterministic wallet scheme. Deterministic wallets are frequently used in practice in order to secure funds by storing the sensitive secret key on a so-called *cold wallet* that is not connected to the Internet. Recently, Das et al. (CCS'19) developed a formal model for the security analysis of deterministic wallets and proposed a generic construction from certain types of signature schemes that exhibit key rerandomization properties. We revisit the proposed classical construction in the presence of quantum adversaries and obtain the following results.

First, we give a generic wallet construction with security in the quantum random oracle model (QROM) if the underlying signature scheme is secure in the QROM. We next design the first post-quantum secure signature scheme with rerandomizable public keys by giving a construction from generic lattice-based Fiat-Shamir signature schemes. Finally, we show and evaluate the practicality by analyzing an instantiation of the wallet scheme based on the signature scheme qTESLA (ACNS'20).

## 1 Introduction

In the past decade cryptocurrencies such as Ethereum [eth15] and Bitcoin [Nak09] have gained huge popularity introducing a revolutionary payment paradigm. Cryptocurrencies do not rely on any central authority (i.e., banks) for the validation of transactions but instead use a consensus protocol to reach agreement on the validity of transactions in a decentralized network. As the name suggests the security of cryptocurrencies heavily relies on cryptographic building blocks – most importantly, on digital signature schemes. Digital signatures are used to authenticate money transfers between parties, where each party is identified by a public key with respect to the signature scheme. In a nutshell, a transfer of $v$ coins from sender $pk_{\mathrm{S}}$ to receiver $pk_{\mathrm{R}}$ is represented by a transaction $\mathsf{tx} := (pk_{\mathrm{S}}, pk_{\mathrm{R}}, v)$. The transaction $\mathsf{tx}$ is then sent together with a signature of $\mathsf{tx}$ with respect to the sender's public key $pk_{\mathrm{S}}$ to the network of miners who validate the transaction. Besides digital signatures many other (partially advanced) cryptographic building blocks are used by cryptocurrencies to achieve a variety of goals. This includes, for instance, non-interactive zero-knowledge proofs and ring signatures for privacy preserving transactions [Noe15,EZS+19], threshold signatures and deterministic wallets for securing funds [DFL19], aggregate signatures for scalability [HMW18], and many more [GS15, FTS+19, TVR16].

Unfortunately, most cryptographic primitives used by cryptocurrencies today can be broken by quantum adversaries. Most notably, the ECDSA signature scheme that is implemented by nearly all popular cryptocurrencies relies on the hardness of computing discrete logarithms, and hence can be broken by Shor's algorithm [Sho94]. Since quantum computers can have devastating consequences for the security of cryptocurrencies [ABL+18], several recent works design cryptocurrencies with post-quantum security features, i.e., they resist both classical and quantum attacks, but run on classical machines. Cryptocurrency projects such as "Bitcoin Post Quantum" [Bit] or QRL [QRL] replace ECDSA with hash-based post-quantum signatures. Other examples include a Monero-based cryptocurrency with privacy guarantees that hold against quantum adversaries [EZS+19], or a security analysis of the proof of work consensus protocol in the quantum random oracle model [CGK+19]. In this work, we follow this line of work and investigate the post-quantum security of deterministic wallet schemes, and propose the first construction that provably resists quantum adversaries.

*Deterministic wallets.* In cryptocurrencies, secret keys are a particular attractive target for attackers. Indeed, the most devastating attacks in the cryptocurrency space have typically targeted secret keys of users resulting in billions of dollars worth of cryptocurrency being stolen [Ske18, Blo18, Bit18]. To protect keys against theft, one of the most prominent solutions is the concept of a *deterministic wallet*. A deterministic wallet scheme consists of two components: a hot wallet that is permanently connected to the Internet, and a cold wallet, which comes online only rarely (e.g., when a large amount of money has to be transferred). Das et al. [DFL19] formalized the concept of deterministic wallets and defined its security goals. The first security goal is *wallet unforgeability* which states that funds sent to the cold wallet must remain secure even if the hot wallet is corrupted. Second, *wallet unlinkability*, which guarantees that individual transactions that sent money to the same wallet are unlinkable despite being publicly available on the blockchain.

At a high-level a hot/cold wallet scheme works as follows. In an initialization phase, it generates a master key pair $(msk, mpk)$, where the master secret key $msk$ is stored on the cold wallet, while the hot wallet keeps the corresponding master public key $mpk$. The main ingredient of a deterministic wallet scheme is a deterministic key derivation procedure, which allows both, cold and hot wallet, to derive matching secret and public session keys without interacting with each other. To this end, in addition to the master secret/public key, the hot and cold wallet share a state $St$. From this state each wallet can derive the corresponding session key by combining the master key with a deterministically derived value $\mathsf{H}(St, ID)$, where $ID$ is an arbitrary key identifier and $\mathsf{H}$ is a cryptographic hash function. More concretely, consider a simplified version of the BIP32 deterministic wallet scheme [BIP17] used for Bitcoin. The master secret/public key pair consists of a valid ECDSA key pair $(msk, mpk) := (x, x \cdot G)$, where $G$ is a generator of the ECDSA elliptic curve. The session key pair for identity $ID$ is computed as $pk_{ID} := mpk + w \cdot G$ and $sk := msk + w$ with $w := \mathsf{H}(St, ID)$.

*Post-quantum security of deterministic wallets.* While deterministic wallet schemes offer an elegant solution to increase the security of users' funds, they are particularly susceptible to attacks by quantum adversaries. To illustrate this, let us first consider how quantum attacks against the underlying signature scheme of a cryptocurrency such as Bitcoin work. Recall that in Bitcoin (as in most other cryptocurrencies) an address for transferring funds to is not represented by the public key itself but by its hash value. More concretely, when a party transfers $v$ coins to some receiver R with public key $pk_{\mathrm{R}}$, then the transaction will store $h = \mathsf{H}(pk_{\mathrm{R}})$. Only when R wants to spend these coins he reveals $pk_{\mathrm{R}}$ together with a signature with respect to $pk_{\mathrm{R}}$. This leaves a quantum adversary that wants to steal $v$ coins from R, with two options: either he tries to find $pk'$ such that $\mathsf{H}(pk') = h$, or he waits until $pk_{\mathrm{R}}$ is revealed by R and computes the corresponding secret key $sk_{\mathrm{R}}$. The first type of attack is believed to be hard because common cryptographic hash functions such as SHA3-512 are known to be preimage resistant even under quantum attacks when appropriately choosing their parameters. While in the second case a quantum adversary can indeed efficiently attack the signature scheme, he has only a very small window of time to carry out this attack[4]. In particular, he has to frontrun the transaction published by R, which is unlikely, assuming that the majority of miners is following the protocol.

A quantum attacker can have more devastating consequences against a deterministic wallet scheme. More concretely, unlike for normal addresses (hashes of public keys), in a deterministic wallet all session keys are related, and in particular efficiently computable from $(msk, mpk)$. Hence, if the adversary manages to learn $mpk$ then he can recover the corresponding master secret key $msk$ and from that recover *all* session secret keys. Hence, all the money that was ever transferred to the cold wallet is at stake.

### 1.1 Our Contributions

Our main contribution in this work is to give the first construction of a post-quantum secure deterministic wallet. Our scheme is intended to be used on classical computers, and to remain secure even in the presence of quantum adversaries. To achieve our goal we extend the security model of Das et al. [DFL19] to the quantum setting and prove that certain standard post-quantum secure signature schemes can be used to construct post-quantum secure wallets. Concretely, our contributions are as follows:

– We extend the security model for deterministic wallets introduced by Das et al. [DFL19] to the quantum world. In particular, we show that if the underlying signature scheme satisfying the property of honestly rerandomizable keys is post-quantum secure, then it can be used to build post-quantum secure deterministic wallets. We relax the notion of rerandomizable keys as given by [DFL19] to

[4] In Bitcoin in most cases transactions are considered to be final after 60 minutes.

consider only rerandomization of public keys. Subsequently, we show that this relaxed notion is sufficient for the security of wallets, and hence we are able to prove post-quantum security based on this relaxed notion.
- We design the first post-quantum secure signature scheme with rerandomizable public keys. This is achieved by giving a generic construction from a Fiat-Shamir signature scheme based on lattice assumptions.
- We discuss optimizations of our post-quantum secure signature scheme with rerandomizable public keys and evaluate its feasibility for blockchains.

### 1.2 Our Techniques

Signature schemes with rerandomizable keys [FKM+16] are the main building block of the wallet scheme presented in [DFL19]. At a high-level besides the standard algorithms of a digital signature scheme for key generation, signing, and verification, a signature scheme with rerandomizable keys has two additional algorithms, namely `RSig.RandSK` and `RSig.RandPK`. These algorithms take as input the secret key $sk$, respectively public key $pk$ and randomness $\rho$, and output fresh keys $sk'$, respectively $pk'$. Moreover, the unforgeability property of the signature scheme must hold even if the adversary sees signatures that are generated using rerandomized secret keys.

We show that certain post-quantum secure signature schemes support rerandomization of keys and satisfy the security notion of unforgeability under honestly rerandomized keys in the quantum world. In [FKM+16] it was shown that Schnorr's signature scheme [Sch91] has rerandomizable keys with unforgeability in the random oracle model (ROM). This motivates to study post-quantum secure Schnorr-like signature schemes. More concretely, we investigate if lattice-based, Schnorr-like signature schemes can have rerandomizable keys with unforgeability in the quantum random oracle model (QROM). Lattice-based schemes are particularly suitable for constructing post-quantum secure rerandomizable signature schemes because (a) lattice-based assumptions are conjectured to be secure under quantum computer attacks; and (b) unlike hash-based signature schemes, lattice-based schemes exhibit an algebraic structure, which enables rerandomization of keys.

The key pair $(pk, sk)$ of such Schnorr-based lattice schemes consists of an instance of a hard lattice problem, where the secret key $sk$ typically follows either the discrete Gaussian distribution or the uniform distribution over a small set. The first idea that comes to mind when rerandomizing keys in the lattice setting is the following: Given $(pk, sk)$ and randomness $\rho$, $sk$ is rerandomized additively by computing $sk' = sk + \rho$ (as carried out in [FKM+16] for Schnorr's scheme). In the lattice setting however, we must ensure that the sum $sk'$ follows the correct distribution, e.g., the Gaussian or uniform distribution. If this is not the case, one can sample a new randomness from $\rho$ in a deterministic way until a valid $sk'$ is generated. Naturally, the same (correct) $\rho$ must be used when rerandomizing $pk$. This approach satisfies (under a specified distribution of $sk$) the original definition of signature schemes with rerandomizable keys (see Definition 3), as the initially generated key pair and any rerandomization of it are identically distributed. However, this approach cannot be used for building hot/cold wallets, because the hot and cold wallet must agree on the correct $\rho$ for each session key generation. This contradicts the main goal of using hot/cold wallets, which requires that the cold wallet stays off-line, and hence cannot frequently communicate with the hot wallet to synchronize on $\rho$. In Section 4.3 we give more details on this approach as well as others, and argue why they are not suitable in the wallet setting.

In this work we show that the key pair $(pk, sk)$ can still be rerandomized additively in a way that fits to the setting of hot/cold wallets. The main observation that we exploit is that the sum of two Gaussians is also Gaussian distributed (see Lemma 3). Based on this observation, our approach works as follows. Let $sk$ be Gaussian distributed. Given randomness $\rho$, $sk$ is rerandomized additively by adding to $sk$ a freshly Gaussian distributed secret key $sk^*$. The key $sk^*$ is deterministically sampled using the randomness $\rho$, i.e., we use $\rho$ as the randomness required in the Gaussian sampler algorithm. We obtain a rerandomized secret key that is Gaussian distributed, but with a slightly larger standard deviation than the one of the original secret key (cf. Lemma 3). Consequently, we can construct a signature scheme with rerandomizable keys, in which the distribution of rerandomized public keys is identical to the distribution of the original public key, while rerandomized secret keys follow a different distribution than a freshly sampled secret key. We formally define such relaxed notion in Section 2 and call it a *signature scheme with rerandomizable public keys*. We then show in Section 3 that this notion is sufficient for post-quantum secure wallets and present a lattice-based construction of such a scheme in Section 4 with a security proof in the QROM. Finally, we show in Section 5 that our construction can be instantiated with state-of-the-art lattice-based signature schemes such as qTESLA [ABB+20]. Hence, it can use their proposed parameters and enjoy their performance and efficiency.

We emphasize that the post-quantum security model considers the adversary to be quantum while the challenger - representing the honest user in a real-world application - remains classical. As a result, every oracle that is provided by the challenger can be accessed only classically, while oracles that can be accessed by the adversary directly can be accessed using quantum computing power, i.e., in superposition. This describes a threat model where an adversary can use its quantum power to locally access the random oracle, while he observes signatures created by a user on a classical machine. In our work, we consider this standard post-quantum security model since it entails that the cryptographic scheme is still used on classical computers. This is, in contrast to the (fully-) quantum setting, where the scheme itself is implemented on quantum computers as well. While this is a stronger security model, it is more of theoretical interest as it requires users to have access to quantum computers as well.

### 1.3 Related Work

The concept of hot/cold wallets is used in many cryptocurrencies in order to provide stronger security guarantees to its users. Various works have proposed formal security models for analyzing the security of wallet schemes. Gutoski and Stebila [GS15] have discussed flaws in the BIP32 construction [BIP17] and possible countermeasures. However, they do not consider the standard notion of unforgeability but rather a restricted model where the adversary can corrupt the cold wallet and recover secret keys. Other works worth mentioning are "privilege escalation attacks" by Fan et al. [FTS+19] which unfortunately lacks any formal security analysis, and the analysis of the Bitcoin Electrum wallet in the Dolev-Yao model by Turuani et al. [TVR16]. The latter considers cryptographic primitives (e.g., signature schemes and encryption schemes) as idealized objects, hence fails to capture potential vulnerabilities such as related key attacks which are relevant in case of hot/cold wallets.

As mentioned earlier, we closely follow the model introduced by Das et al. [DFL19], where the notion of a *stateful deterministic wallet* is introduced and two desirable security properties called *wallet unlinkability* and *wallet unforgeability* are considered. The first property ensures that the session public keys generated by $\mathtt{SW.RandPK}$ are unlinkable to the master public key. This property is guaranteed as long as the hot wallet has not been corrupted. The second property ensures unforgeability of signatures signed by the secret keys of the cold wallet even when the hot wallet is corrupted.

According to [DFL19] a *stateful deterministic wallet* $\mathtt{SW}$ consists of two components – a hot wallet and a cold wallet which share a common state $St$. $\mathtt{SW}$ is given by a tuple of algorithms ($\mathtt{SW.KGen}$, $\mathtt{SW.RandSK}$, $\mathtt{SW.RandPK}$, $\mathtt{SW.Sign}$, $\mathtt{SW.Verify}$), where the session public key and secret key derivation algorithms $\mathtt{SW.RandPK}$ and $\mathtt{SW.RandSK}$ are run respectively within the hot and cold wallet to deterministically derive matching session (public/secret) keys from the (public/secret) master keys. Unlike deterministic wallets in use (e.g., the BIP32 construction), the state $St$ of the wallet scheme of [DFL19] is refreshed within the (hot/cold) wallets with each key derivation. This approach allows to show forward unlinkability, which intuitively means that even upon leakage of the state, all session keys derived *before* the state leakage remain unlinkable. The second security property – *wallet unforgeability* – is achieved by a reduction to standard $\mathsf{EUF\text{-}CMA}$ security of a concrete signature scheme (such as ECDSA and Schnorr) in a modularized fashion. As the intermediary step the authors show that these signature schemes satisfy the properties of signature schemes with rerandomizable keys.

We note that the model by Das et al. [DFL19] only considers adversaries in the classical setting and does not protect against quantum adversaries. Our work fills this gap by designing the first *post-quantum secure deterministic wallet.*

Many prior works have investigated lattice-based Fiat-Shamir signatures, e.g., [Lyu09, DDLL13, BG14, DKL+18, ABB+20], and in particular, their security was analyzed in the QROM, e.g., by [KLS18, Unr17, LZ19, DFMS19]. To the best of our knowledge we propose the first work on lattice-based signature schemes with honestly rerandomizable keys and prove its security in the QROM. Inspired by Das et al. [DFL19] and Fleischhacker et al. [FKM+16], we use the abstraction of signature schemes with rerandomizable keys but transfer this concept to the post-quantum setting.

As mentioned in the introduction, various works build blockchains with security features against quantum adversaries. Most recently, Esgin et al. [EZS+19] have proposed a new ring signature scheme based on lattice assumptions for the blochchain setting which focus on similar anonymity guarantees to Monero [Noe15]. In Monero-like cryptocurrencies the sender of a transaction can hide her identity in a set of transactions using ring signatures. In particular, the public key related to the sender's signature is never revealed explicitly in the blockchain network, hence remains unlinkable to the sender. We note that this notion of unlinkability is different from our notion of session key unlinkability.

Blockchain initiatives such as the "Bitcoin Post-Quantum" [Bit] and QRL [QRL] replace ECDSA with hash-based signature schemes which are post-quantum secure. Despite the hash-based schemes being quite efficient, the underlying hash function does not permit to construct a signature scheme with rerandomizable keys which plays a key role in our wallet scheme.

## 2 Preliminaries

### 2.1 Basic Notations

We let $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ denote the set of natural numbers, integers, and real numbers, respectively. For any positive integer $k$ we write $[k]$ to denote the set of integers $\{1, \ldots, k\}$. For a positive integer $q$ we let $\mathbb{Z}_q$ denote the set of integers in the range $[-\frac{q}{2}, \frac{q}{2}) \cap \mathbb{Z}$. We define the ring $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$ and its quotient $R_q = R/qR$, where $n$ is a power of 2. Elements in $R$ and $R_q$ (including $\mathbb{Z}$ and $\mathbb{Z}_q$) are denoted by regular font letters. Column vectors and matrices with entries from $R$ or $R_q$ are denoted by bold lower-case letters and bold upper-case letters, respectively. We define the $\ell_2$ and $\ell_\infty$ norms of $v = \sum_{i=0}^{n-1} v_i x^i \in R$ by $\|v\| = (\sum_{i=0}^{n-1} |v_i|^2)^{1/2}$ and $\|v\|_\infty = \max_i |v_i|$, respectively. For $\mathbf{w} = (w_1, \ldots, w_k) \in R^k$ we define $\|\mathbf{w}\| = (\sum_{i=1}^{k} \|w_i\|^2)^{1/2}$ and $\|\mathbf{w}\|_\infty = \max_i \|w_i\|_\infty$. We let $\mathbb{T}_\kappa^n$ denote the set of all $(n-1)$-degree polynomials with coefficients from $\{-1, 0, 1\}$ and Hamming weight $\kappa$. We always denote the security parameter by $\lambda \in \mathbb{N}$, and $o(\lambda)$ denotes a linear function in $\lambda$. A function $f : \mathbb{N} \longrightarrow \mathbb{R}$ is called *negligible* if there exists an $n_0 \in \mathbb{N}$ such that for all $n > n_0$, it holds $f(n) < \frac{1}{p(n)}$ for any polynomial $p$. With $\mathrm{negl}(\lambda)$ we denote a negligible function in $\lambda$. A probability is called overwhelming if it is at least $1 - \mathrm{negl}(\lambda)$. The *statistical distance* between two distributions $X, Y$ over a countable domain $D$ is defined by $\frac{1}{2}\sum_{n \in D} |X(n) - Y(n)|$. We write $x \leftarrow D$ to denote that $x$ is sampled according to a distribution $D$. We let $x \leftarrow_\$ S$ denote choosing $x$ uniformly random from a finite set $S$. Unless specified otherwise, every adversary is considered to be an efficient quantum polynomial time algorithm.

### 2.2 Quantum Random Oracle Model

In this section, we recall the quantum random oracle model and existing results that we will use. Since quantum computation is only necessary in the proofs of these results, we do not provide information on quantum computation here, but refer to [NC11] for a detailed discussion on the topic.

In [BR93], Bellare and Rogaway introduced the *random oracle model* (ROM). In this model every party has access to an oracle implementing a random function. Upon being queried on some input $x$, the oracle answers with a random output $y$. Every further invocation on input $x$, even by other parties, results in the same $y$. In security proofs, one often models a hash function as a random oracle. Since hash functions are public, Boneh et al. [BDF+11] observed that the ROM is not appropriate in the post-quantum setting. In the real world an adversary equipped with a quantum computer is able to implement the hash function and evaluate it in superposition. Thus, Boneh et al. introduced the *quantum random oracle model* (QROM). In this model, parties with quantum computing power get access to the oracle $|\mathsf{H}\rangle$, where $|\mathsf{H}\rangle : |x, y\rangle \mapsto |x, y \oplus \mathsf{H}(x)\rangle$. In our proofs we will also consider reprogrammed random oracles. For a random oracle $\mathsf{H}$ we write $\mathsf{H}_{x\to y}$ for the random oracle that is reprogrammed on input $x$ to $y$. Further on, we denote the classical random oracle by the symbol $\mathsf{H}$ and the quantum random oracle by the notation $|\mathsf{H}\rangle$.

Nowadays, the QROM is considered the de facto standard for post-quantum security proofs of cryptographic primitives which rely on random oracles. Below we describe some results for quantum random oracles that are required for our proofs.

The one-way to hiding (O2H) lemma [Unr15] is an important tool for security proofs in the quantum random oracle model. It gives bounds on the advantage of an adversary in distinguishing between different random oracles when the adversary is allowed to query them in superposition. Below we state the lemma using the reformulation by Ambainis et al. [AHU19].

**Lemma 1 (One-way to hiding** (O2H) **[AHU19]).** *Let* $\mathsf{G}, \mathsf{H} \colon \mathcal{X} \to \mathcal{Y}$ *be random functions, let* $z$ *be a random value, and let* $\mathcal{S} \subset \mathcal{X}$ *be a random set such that* $\forall x \notin \mathcal{S}$, $\mathsf{G}(x) = \mathsf{H}(x)$. $(\mathsf{G}, \mathsf{H}, \mathcal{S}, z)$ *may have arbitrary joint distribution. Furthermore, let* $\mathcal{A}^{|\mathsf{H}\rangle}$ *be a quantum oracle algorithm which queries* $|\mathsf{H}\rangle$ *at most* $q$ *times. Let* $\mathsf{Ev}$ *be an arbitrary classical event. Define an oracle algorithm* $\mathcal{B}^{|\mathsf{H}\rangle}$ *as follows: Pick* $i \leftarrow_{\$} [q]$. *Run* $\mathcal{A}^{|\mathsf{H}\rangle}(z)$ *until just before its* $i$*-th round of queries to* $|\mathsf{H}\rangle$. *Measure the query in the computational basis, and output the measurement outcome. It holds that*

$$\left|\Pr[\mathsf{Ev} \colon \mathcal{A}^{|\mathsf{H}\rangle}(z)] - \Pr[\mathsf{Ev} \colon \mathcal{A}^{|\mathsf{G}\rangle}(z)]\right| \leq 2q\sqrt{\Pr[x \in \mathcal{S} \colon \mathcal{B}^{|\mathsf{H}\rangle}(z) \Rightarrow x]}\,.$$

Another tool that we will use are Zhandry's small range distributions, defined below. These are distributions where the set of possible outputs is limited.

**Definition 1 (Small-range distributions [Zha12a]).** *Let* $\mathcal{X}$, $\mathcal{Y}$ *be sets,* $r$ *be an integer,* $D$ *be a distribution on* $\mathcal{Y}$, $P$ *be a random function from* $\mathcal{X}$ *to* $[r]$, *and* $\vec{y} = (y_1, \ldots, y_r)$ *be* $r$ *samples of* $D$. *Define a function* $\mathsf{H} \colon \mathcal{X} \to \mathcal{Y}$ *by* $\mathsf{H}(x) \mapsto y_{P(x)}$. *The distribution of* $\mathsf{H}$, *induced by* $P$ *and* $\vec{y}$, *is called a* small-range distribution *with* $r$ *samples of* $D$.

The following lemma provides a bound on the distinguishing advantage between a random oracle and an oracle drawn from a small-range distribution when superposition access is granted.

**Lemma 2** **([Zha12a]).** *There is a universal constant $C$ such that, for any set $\mathcal{X}$ and $\mathcal{Y}$, distribution $D$ on $\mathcal{Y}$, integer $l$, and any quantum algorithm $\mathcal{A}$ making $q$ queries to an oracle $\mathsf{H}\colon \mathcal{X} \to \mathcal{Y}$, the following two cases are indistinguishable, except with probability less than $\frac{Cq^3}{l}$:*

- *$\mathsf{H}(x) = y_x$ where $\vec{y}$ is a list of samples of $D$ of size $|\mathcal{X}|$.*
- *$\mathsf{H}$ is drawn from the small-range distribution with $l$ samples of $D$.*

## 2.3 Cryptographic Primitives

**Definition 2 (Signature Scheme).** *Let $\lambda$ be a security parameter. A signature scheme Sig with key space $\mathcal{K}$, message space $\mathcal{M}$, and signature space $\mathcal{S}$ is a tuple of polynomial-time algorithms* $(\mathtt{KGen}, \mathtt{Sign}, \mathtt{Verify})$ *such that*

$\mathtt{KGen}(1^\lambda)$ *is the key generation algorithm that outputs a key pair $(pk, sk) \in \mathcal{K}$, where $pk$ is a public key and $sk$ is a secret key.*

$\mathtt{Sign}(sk, m)$ *is the signing algorithm that takes as input a secret key $sk$ and a message $m \in \mathcal{M}$. It outputs a signature $\sigma \in \mathcal{S}$.*

$\mathtt{Verify}(pk, m, \sigma)$ *is the verification algorithm that takes as input a public key $pk$, a message $m$ with a signature $\sigma$. It outputs $1$ if $\sigma$ is valid and $0$ otherwise.*

**Correctness:** *A signature scheme is correct if for all $\lambda \in \mathbb{N}$, $m \in \mathcal{M}$, $(pk, sk) \leftarrow \mathtt{KGen}(1^\lambda)$, and all $\sigma \leftarrow \mathtt{Sign}(sk, m)$, it holds that* $\Pr[\mathtt{Verify}(pk, m, \mathtt{Sign}(sk, m)) = 1] \geq 1 - \mathrm{negl}(\lambda)$.

We will use the notion of signature schemes with rerandomizable keys [FKM+16]. In the following we recall its definition.

**Definition 3 (Signature Scheme with Rerandomizable Keys).** *A signature scheme with perfectly rerandomizable keys* $\mathtt{RSig}$ *is given by a tuple of algorithms:*

$$(\mathtt{RSig.KGen}, \mathtt{RSig.RandSK}, \mathtt{RSig.RandPK}, \mathtt{RSig.Sign}, \mathtt{RSig.Verify}),$$

*where* $\mathtt{RSig.KGen}$, $\mathtt{RSig.Sign}$, *and* $\mathtt{RSig.Verify}$ *satisfy the definition of a standard signature scheme (cf. Definition 2). For randomness space $\mathcal{R}$,* $(\mathtt{RSig.RandSK}, \mathtt{RSig.RandPK})$ *are two polynomial-time algorithms such that*

$\mathtt{RSig.RandSK}(sk, \rho)$ *is a secret key rerandomization algorithm that takes as input the secret key $sk$ and a randomness $\rho \in \mathcal{R}$ and outputs a randomized secret key $sk'$.*

$\mathtt{RSig.RandPK}(pk, \rho)$ *is a public key rerandomization algorithm that takes as input the public key $pk$ and a randomness $\rho \in \mathcal{R}$ and outputs a randomized public key $pk'$.*

$\mathtt{RSig}$ *satisfies the following properties:*

**Rerandomizability of keys:** *For all $(sk, pk) \in \mathtt{RSig.KGen}(1^\lambda)$ and all $\rho \in \mathcal{R}$, the distributions of $(sk', pk')$ and $(sk'', pk'')$ are identical, where $(sk'', pk'') \leftarrow \mathtt{RSig.KGen}(1^\lambda)$ and*
*$sk' \leftarrow \mathtt{RSig.RandSK}(sk, \rho)$, $pk' \leftarrow \mathtt{RSig.RandPK}(pk, \rho)$.*

**Correctness under rerandomizable keys:**

1. *For all* $\lambda \in \mathbb{N}$, $(pk, sk) \leftarrow \texttt{RSig.KGen}(1^\lambda)$, $m \in \mathcal{M}$, *and all* $\sigma \leftarrow \texttt{RSig.Sign}(sk, m)$, *it holds that*

$$\Pr[\texttt{RSig.Verify}(pk, m, \texttt{RSig.Sign}(sk, m)) = 1] \geq 1 - \text{negl}(\lambda)\,.$$

2. *For all* $(pk, sk) \leftarrow \texttt{RSig.KGen}(1^\lambda)$, *all* $\rho \in \mathcal{R}$, $m \in \mathcal{M}$, *and for a pair of rerandomized keys* $sk' \leftarrow \texttt{RSig.RandSK}(sk, \rho)$ *and* $pk' \leftarrow \texttt{RSig.RandPK}(pk, \rho)$, *it holds*

$$\Pr[\texttt{RSig.Verify}(pk', m, \texttt{RSig.Sign}(sk', m)) = 1] \geq 1 - \text{negl}(\lambda)\,.$$

We also consider a relaxed version of Definition 3. In the following definition we introduce the notion of *signature schemes under rerandomizable public keys*, where the property of rerandomizability of keys holds only for the generated public keys, but not in case of secret keys. We present a concrete instantiation of such a scheme in Section 4.

**Definition 4 (Signature Scheme with Rerandomizable Public Keys).** *A signature scheme with perfectly rerandomizable public keys* $\texttt{RSig}'$ *is given by a tuple of algorithms* $(\texttt{RSig}'.\texttt{KGen}, \texttt{RSig}'.\texttt{RandSK}, \texttt{RSig}'.\texttt{RandPK}, \texttt{RSig}'.\texttt{Sign}, \texttt{RSig}'.\texttt{Verify})$, *which are defined as in Definition 3.* $\texttt{RSig}'$ *satisfies the following properties:*

**Rerandomizability of public keys:** *For all public keys* $(\cdot, pk) \leftarrow \texttt{RSig}'.\texttt{KGen}(1^\lambda)$ *and* $\rho \in \mathcal{R}$, *the distributions of* $pk'$ *and* $pk''$ *are computationally indistinguishable, where* $pk' \leftarrow \texttt{RSig}'.\texttt{RandPK}(pk, \rho)$, *and* $pk'' \leftarrow \texttt{RSig}'.\texttt{KGen}(1^\lambda)$.

**Correctness under rerandomizable keys:** *This property is defined as the property of correctness for signature schemes under rerandomizable keys in Definition 3.*

In our work, we require a signature scheme with rerandomizable public keys $\texttt{RSig}'$ to additionally satisfy the following property:

**Simulatibility:** For all $\lambda \in \mathbb{N}$, all $(sk, pk) \leftarrow \texttt{RSig}'.\texttt{KGen}(1^\lambda)$, and all $m \in \mathcal{M}$, there exists a polynomial time algorithm $\mathcal{T}$ which on input $pk$ and $m$ outputs a signature $\sigma \in \mathcal{S}$. It must hold for $\kappa \in \text{poly}(\lambda)$ the distributions $\{\sigma_1, \ldots, \sigma_\kappa\}$ and $\{\sigma'_1, \ldots, \sigma'_\kappa\}$ are computationally indistinguishable where $\sigma_i \leftarrow \mathcal{T}(pk, m)$ and $\sigma'_i \leftarrow \texttt{RSig}'.\texttt{Sign}(sk, m)$ for $i \in [\kappa]$.

## 2.4 Security Notions

Security of signature schemes is captured by the standard security notion of *existential unforgeability under adaptive chosen-message attacks* (EUF-CMA), presented below.

```
Game $\mathsf{EUF\text{-}CMA}^{\mathcal{A}}_{\Sigma}$
1: $\mathcal{Q} := \varnothing$
2: $H := \varnothing$
3: $(pk, sk) \leftarrow \mathtt{KGen}(1^{\lambda})$
4: $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{O}}(pk)$
5: if $(m^* \in \mathcal{Q})$ then
6:     return 0
7: return $\mathtt{Verify}(pk, m^*, \sigma^*)$

$\mathsf{H}(m')$
1: if $(m' \in H)$ then
2:     return $\mathsf{H}(m') \in H$
3: $\mathsf{H}(m') \leftarrow_{\$} \{0,1\}^{o(\lambda)}$
4: $H := H \cup \{(m', \mathsf{H}(m'))\}$
5: return $\mathsf{H}(m')$

$\mathsf{O}(m)$
1: $\mathcal{Q} := \mathcal{Q} \cup \{m\}$
2: $\sigma \leftarrow \mathtt{Sign}(sk, m)$
3: return $\sigma$
```

**Fig. 1.** The security game $\mathsf{EUF\text{-}CMA}$ of signature schemes.

**Definition 5 ($\mathsf{EUF\text{-}CMA}$ Security).** *Let $H : \{0,1\}^* \to \{0,1\}^{o(\lambda)}$ be a hash function modeled as (quantum) random oracle. A signature scheme $\Sigma$ is called $(t, q_{\mathtt{Sign}}, q_H, \varepsilon)$-EUF-CMA in the (quantum) random oracle model if for any adversary $\mathcal{A}$ running in time at most $t$ and making at most $q_{\mathtt{Sign}}$ signature queries and at most $q_H$ (superposition) queries to H, the game EUF-CMA$^{\mathcal{A}}_{\Sigma}$ depicted in Figure 1 outputs 1 with probability at most $\varepsilon$, i.e., $\Pr[\mathsf{EUF\text{-}CMA}^{\mathcal{A}}_{\Sigma} = 1] \leq \varepsilon$.*

In the following we present the notion of *EUF-CMA-HRK security under honestly rerandomizable keys* due to [DFL19]. This notion is similar to *EUF-CMA-RK security under rerandomizable keys* due to [FKM+16], however with certain differences which makes it a weaker notion. In the EUF-CMA-RK game, an adversary $\mathcal{A}$ gets access to a signing oracle. The signing oracle takes a message and a randomness as input and provides a signature on this message under the rerandomized key as an answer. Note that the rerandomized key was derived from the randomness input by $\mathcal{A}$. This means that $\mathcal{A}$ can obtain signatures under keys with randomness of $\mathcal{A}$'s choice. $\mathcal{A}$ can win the EUF-CMA-RK game if it can produce a valid forgery under a rerandomized key of its choice (note that the randomness can also be null).

In the EUF-CMA-HRK game, we restrict $\mathcal{A}$'s capabilities in the following way. In addition to the signing oracle, in the EUF-CMA-HRK game $\mathcal{A}$ is given access to a Rand oracle to derive a fresh randomness. This randomness can be later used to get a signature under the rerandomized key by querying the signing oracle. Here, $\mathcal{A}$ can only win the EUF-CMA-HRK game if it produces a valid forgery under a rerandomized key, where the underlying randomness was obtained honestly by querying the Rand oracle. We formally present *EUF-CMA-HRK security under honestly rerandomizable (public) keys* below.

**Definition 6 ($\mathsf{EUF\text{-}CMA\text{-}HRK}$ Security under Honestly Rerandomized (Public) Keys).** *Let $H : \{0,1\}^* \to \{0,1\}^{o(\lambda)}$ be a hash function modeled as (quantum) random oracle. A signature scheme with honestly rerandomizable (public) keys $\mathtt{RSig}$ is called $(t, q_{\mathtt{Sign}}, q_H, \varepsilon)$-EUF-CMA-HRK in the (quantum) random oracle model if for any adversary $\mathcal{A}$ running in time at most $t$ and making at most $q_{\mathtt{Sign}}$ signature queries and at most $q_H$ (quantum) random oracle queries to H, the game EUF-CMA-HRK$^{\mathcal{A}}_{\mathtt{RSig}}$ depicted in Figure 2 outputs 1 with probability at most $\varepsilon$, i.e., $\Pr[\mathsf{EUF\text{-}CMA\text{-}HRK}^{\mathcal{A}}_{\mathtt{RSig}} = 1] \leq \varepsilon$.*

```
Game EUF-CMA-HRK$^{\mathcal{A}}_{\texttt{RSig}}$
 1: RList := $\varnothing$
 2: $\mathcal{Q} := \varnothing$
 3: $H := \varnothing$
 4: $(pk, sk) \leftarrow \texttt{RSig.KGen}(1^\lambda)$
 5: $(m^*, \sigma^*, \rho^*) \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{Rand},\mathsf{OHR}}(pk)$
 6: if $(m^* \in \mathcal{Q})$ then
 7:    return 0
 8: if $(\rho^* \neq \mathsf{NULL})$ then
 9:    if $(\rho^* \notin \mathsf{RList})$ then
10:       return 0
11:    $pk \leftarrow \texttt{RSig.RandPK}(pk, \rho^*)$
12: return $\texttt{RSig.Verify}(pk, m^*, \sigma^*)$
```

```
H$(m')$ (see Figure 1)
Rand
 1: $\rho \leftarrow_{\$} \mathcal{R}$
 2: $\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$
 3: return $\rho$
OHR$(m, \rho)$
 1: $\mathcal{Q} := \mathcal{Q} \cup \{m\}$
 2: if $(\rho \neq \mathsf{NULL})$ then
 3:    if $(\rho \notin \mathsf{RList})$ then
 4:       return $\perp$
 5:    $sk \leftarrow \texttt{RSig.RandSK}(sk, \rho)$
 6: $\sigma \leftarrow \texttt{RSig.Sign}(sk, m)$
 7: return $\sigma$
```

**Fig. 2.** The security game EUF-CMA-HRK of signature schemes with rerandomizable (public) keys.

## 2.5 Lattice-Based Fiat-Shamir Signatures

In this section we review a generic construction of lattice-based Fiat-Shamir signatures. We first define the discrete Gaussian distribution and recall a lemma, which shows that the sum of Gaussian distributed random variables is also Gaussian distributed. This property is crucial for our analysis.

**Definition 7 (Discrete Gaussian Distribution).** *The discrete Gaussian distribution $D_{\mathbb{Z}^n,\sigma,\mathbf{c}}$ over $\mathbb{Z}^n$ with standard deviation $\sigma > 0$ and center $\mathbf{c} \in \mathbb{R}^n$ is defined as follows: For every $\mathbf{x} \in \mathbb{Z}^n$ the probability of $\mathbf{x}$ is given by $D_{\mathbb{Z}^n,\sigma,\mathbf{c}}(\mathbf{x}) = \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(\mathbb{Z}^n)$, where $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp(\frac{-\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2})$ and $\rho_{\sigma,\mathbf{c}}(\mathbb{Z}^n) = \sum_{\mathbf{x}\in\mathbb{Z}^n} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. The subscript $\mathbf{c}$ is taken to be $\mathbf{0}$ when omitted.*

**Lemma 3 ([BF11, Theorem 9]).** *Let $\mathcal{L} \subseteq \mathbb{Z}^m$ be a lattice and $\sigma \in \mathbb{R}$. For $i = 1, \ldots, n$ let $\mathbf{t}_i \in \mathbb{Z}^m$ and let $X_i$ be mutually independent random variables sampled from $D_{\mathcal{L}+\mathbf{t}_i,\sigma}$. Let $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{Z}^n$ and define $d = \gcd(c_1, \ldots, c_n)$, $\mathbf{t} = \sum_1^n c_i \mathbf{t}_i$. Suppose that $\sigma > \|\mathbf{c}\| \cdot \eta_\varepsilon(\mathcal{L})$, where $\eta_\varepsilon(\mathcal{L})$ is the smoothing parameter [MR04] for some negligible $\varepsilon$. Then $Z = \sum_1^n c_i X_i$ is statistically close to $D_{d\mathcal{L}+\mathbf{t},\|\mathbf{c}\|\sigma}$.*

Next, we describe two functions used in the signature scheme:

- $\mathsf{E} : \{0,1\}^* \longrightarrow \{0,1\}^*$ is a function that expands given strings to any desired length. It is used to extract the randomness used for signing.
- $\mathsf{H} : \{0,1\}^* \longrightarrow \mathbb{T}^n_\kappa$ is a hash function modeled as a (quantum) random oracle and used for signing and verification.

The signature scheme is formally described in Figure 3. It makes use of a uniformly random matrix $\mathbf{A} \in R_q^{k_1 \times k_2}$, which is publicly known and shared among all users in a multi-user setting. We assume that $\mathbf{A}$ is an implicit input to all algorithms of the scheme in addition to all algorithms in Section 4. In order to save bandwidth it can also be generated by expanding a uniformly random seed using the function $\mathsf{E}$, and including the seed in the secret and public key rather than storing the whole matrix $\mathbf{A}$. In this case, $\mathsf{E}$ is modelled as a random oracle. We note that this setting makes sense in the context of blockchains, since the randomly chosen seed can be included in the first block known as the genesis block, which is assumed to be honestly generated. Furthermore, since $\mathbf{A}$ is computed as the output of the random oracle on input the seed, $\mathbf{A}$ is truly random and cannot have a trapdoor embedded as shown in [MP12].

Basically, the key generation algorithm generates an instance of a computationally hard lattice problem called *Module Learning with Errors* (MLWE) [LS15] (or a special variant of it such as Ring Learning with Errors (RLWE) [LPR10]). The secret of this instance is chosen from some distribution $\chi$. In the state-of-the-art lattice-based signature schemes, e.g., Dilithium [DKL$^+$18] and qTESLA [ABB$^+$20], the distribution of the secrets is either the discrete Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$ or the distribution $R_d$ that outputs uniformly random polynomials from $R$ whose $\ell_\infty$ norm is bounded by some integer $d \geq 1$.

A signature consists of a tuple $(\mathbf{z}_1, \mathbf{z}_2, c)$, where the pair $(\mathbf{z}_1, \mathbf{z}_2)$ is uniformly random over a subset of $R^{k_2} \times R^{k_1}$ and $c \in \mathbb{T}^n_\kappa$ is output from the random oracle $\mathsf{H}$. The vectors $\mathbf{z}_1, \mathbf{z}_2$ are each generated by adding a masking term to a term related to the secret key and $c$. More precisely, we have $\mathbf{z}_1 = \mathbf{y}_1 + \mathbf{s}c$ and $\mathbf{z}_2 = \mathbf{y}_2 + \mathbf{e}c$, where the secret masking pair $(\mathbf{y}_1, \mathbf{y}_2)$ is uniformly random over $R_Y^{k_2} \times R_Y^{k_1}$ and $R_Y \subset R$ for some predefined positive integer $Y$. The signature is only output after verifying that the pair $(\mathbf{z}_1, \mathbf{z}_2)$ lies in $R_{B_1}^{k_2} \times R_{B_2}^{k_1}$, i.e., $\|\mathbf{z}_1\|_\infty \leq B_1$ and $\|\mathbf{z}_2\|_\infty \leq B_2$, where the bounds $B_1, B_2$ are defined depending on the distribution of the secret key. This ensures that signatures are uniformly distributed over $R_{B_1}^{k_2} \times R_{B_2}^{k_1} \times \mathbb{T}^n_\kappa$ and do not leak information about the secret key. If this is not the case, the algorithm restarts with a fresh masking pair $(\mathbf{y}_1, \mathbf{y}_2)$. The average number of repetitions is denoted by $M = O(1)$. Valid signatures are generated with probability $\left(\frac{2B_1+1}{2Y+1}\right)^{k_2 n} \cdot \left(\frac{2B_2+1}{2Y+1}\right)^{k_1 n}$, which is usually chosen such that it is at least $1/M$. We note that this generic construction can be optimized by either following the technique due to Bai and Galbraith [BG14] (adopted in qTESLA) or the approach used in Dilithium. The first one optimizes the signature size, while the second one optimizes the total size of public key and signature.

Finally, the EUF-CMA security of lattice-based Fiat-Shamir signatures in the quantum random oracle model was analyzed in several works, e.g., in [KLS18, DKL$^+$18, ABB$^+$20, Unr17, LZ19, DFMS19].

## 3 The Stateful Model for Wallets

Our formal security model for post-quantum secure stateful deterministic wallets is based on the model of [DFL19]. In this section, we recall the formal definition of a stateful wallet and the security properties that we want to guarantee for such a wallet. A stateful deterministic wallet scheme consists of two entities, a cold wallet and a hot wallet, that can respectively derive a valid pair of secret and public

```
$\boxed{\mathsf{LB}.\texttt{KGen}(1^\lambda)}$
1: $\mathbf{s} \leftarrow \chi^{k_2}$, $\mathbf{e} \leftarrow \chi^{k_1}$
2: $\mathbf{b} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e} \pmod q$
3: $sk := (\mathbf{s}, \mathbf{e})$, $pk := \mathbf{b}$
4: return $(sk, pk)$
$\boxed{\mathsf{LB}.\texttt{Verify}(pk, m, (\mathbf{z}_1, \mathbf{z}_2, c))}$
1: $\mathbf{w} \leftarrow \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{b}c \pmod q$
2: if $\big((\mathbf{z}_1, \mathbf{z}_2) \in R_{B_1}^{k_2} \times R_{B_2}^{k_1} \wedge$
       $\mathsf{H}(\mathbf{w}, m) = c\big)$ then
3:     return 1
4: return 0
```

```
$\boxed{\mathsf{LB}.\texttt{Sign}(sk, m)}$
1: $\mathbf{r} \leftarrow_\$ \{0,1\}^{o(\lambda)}$
2: $\mathsf{ctr} \leftarrow 1$
3: $(\mathbf{y}_1, \mathbf{y}_2) \in R_Y^{k_2} \times R_Y^{k_1} \leftarrow \mathsf{E}(\mathbf{r}, \mathsf{ctr})$
4: $\mathbf{v} \leftarrow \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2 \pmod q$
5: $c \leftarrow \mathsf{H}(\mathbf{v}, m)$
6: $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + \mathbf{s}c$
7: $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + \mathbf{e}c$
8: if $\big((\mathbf{z}_1, \mathbf{z}_2) \notin R_{B_1}^{k_2} \times R_{B_2}^{k_1}\big)$ then
9:     $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$
10:    goto 3
11: return $(\mathbf{z}_1, \mathbf{z}_2, c)$
```

**Fig. 3.** A formal description of a generic (non-optimized) Fiat-Shamir signature scheme from lattice assumptions.

keys without the need for any interaction among each other. In more detail, upon initialization of the scheme, the cold wallet generates a master key pair $(msk, mpk)$ and some initial state information $St_0$ and forwards $(mpk, St_0)$ to the hot wallet. After this initial setup, the idea is that an arbitrary number of valid session key pairs can be generated by using the session secret/public key derivation algorithms within the respective wallets without further interaction. More precisely the public key derivation algorithm takes as input the current state and the master public key to generate a session public key. While the secret key derivation takes as inputs the current state and the master secret key and generates a session secret key. Since both public key and secret key derivation algorithms are deterministic, and the two wallets share the same current state, the key derivation algorithms output a valid session key pair. In order to keep track of which key has been derived with which state, each session key is indexed by a parameter $ID$, which is given as input into the key derivation procedures. In the following we recall the definition of a deterministic stateful wallet scheme and its correctness.

**Definition 8 (Stateful Wallet).** *A* stateful wallet scheme *is a tuple of algorithms* $\mathsf{SW} := (\texttt{SW.KGen}, \texttt{SW.RandSK}, \texttt{SW.RandPK}, \texttt{SW.Sign}, \texttt{SW.Verify})$, *which are defined as follows:*

<u>`SW.KGen`</u>**:** *The master key generation algorithm takes as input public parameters param and outputs a master key pair $(msk, mpk)$ as well as an initial state $St_0$.*

<u>`SW.RandSK`</u>**:** *The secret key derivation algorithm takes as input a master secret key msk, a state St and an identity ID and outputs a session secret key $sk_{ID}$ and the state St.*

<u>`SW.RandPK`</u>**:** *The public key derivation algorithm takes as input a master public key mpk, a state St and an identity ID and outputs a session secret key $pk_{ID}$ and the state St.*

<u>`SW.Sign`</u>**:** *The probabilistic signing algorithm takes as input a session secret key $sk_{ID}$ for some ID and a message m and outputs a signature $\sigma$.*

```
$\boxed{\texttt{SW.KGen}(1^\lambda)}$
1: $St \leftarrow\$ \{0,1\}^\lambda$
2: $(mpk, msk) \leftarrow \texttt{RSig.KGen}(1^\lambda)$
3: return $(St, mpk, msk)$

$\boxed{\texttt{SW.Sign}(sk, pk, m)}$
1: $m' \leftarrow (m, pk)$
2: $\sigma \leftarrow \texttt{RSig.Sign}(sk, m')$
3: return $\sigma$

$\boxed{\texttt{SW.Verify}(pk, m, \sigma)}$
1: $m' \leftarrow (m, pk)$
2: return $\texttt{RSig.Verify}(pk, m', \sigma)$

$\boxed{\texttt{SW.RandSK}(msk, ID, St)}$
1: $(\rho, St) \leftarrow \mathsf{H}(St, ID)$
2: $sk_{ID} \leftarrow \texttt{RSig.RandSK}(msk, \rho)$
3: return $(sk_{ID}, St)$

$\boxed{\texttt{SW.RandPK}(mpk, ID, St)}$
1: $(\rho, St) \leftarrow \mathsf{H}(St, ID)$
2: $pk_{ID} \leftarrow \texttt{RSig.RandPK}(mpk, \rho)$
3: return $(pk_{ID}, St)$
```

**Fig. 4.** Generic Construction of a stateful deterministic wallet SW from a signature scheme with honestly rerandomizable keys RSig and a random oracle H.

<u>SW.Verify</u>**:** *The verification algorithm takes as input a session public key $pk_{ID}$ for some ID, a message $m$, and a signature $\sigma$ and outputs 1 if $\sigma$ is a valid signature for $m$ under public key $pk_{ID}$. It outputs 0 otherwise.*

**Definition 9 (Correctness of Stateful Wallets).** *For $n \in \mathbb{N}$, any $(St_0, msk, mpk) \in \texttt{SW.KGen}(param)$, and any $\vec{ID} := (ID_1, ..., ID_n) \in \{0,1\}^*$, we define the sequence $(sk_i, St_i)$ and $(pk_i, St_i)$ for $1 \leq i \leq n$ recursively as*

$$(sk_i, St_i) := \texttt{SW.RandSK}(msk, ID_i, St_{i-1}),$$
$$(pk_i, St_i) := \texttt{SW.RandPK}(mpk, ID_i, St_{i-1}).$$

SW *is* correct *if for all $m \in \{0,1\}^*$ and $i$ with $1 \leq i \leq n$ it holds that*

$$\Pr_{\sigma \leftarrow\$ \texttt{SW.Sign}(sk_i, m)}[\texttt{SW.Verify}(pk_i, \sigma, m) = 1] \geq 1 - \text{negl}(\lambda)\,.$$

A generic construction of a stateful deterministic wallet scheme SW := (SW.KGen, SW.RandSK, SW.RandPK, SW.Sign, SW.Verify) from a signature scheme with honestly rerandomizable keys RSig following Definition 8 is presented in Figure 4. Such a scheme should satisfy the following two security properties - *wallet unlinkability* and *wallet unforgeability* - which are described below.

## 3.1 Wallet Unlinkability

Intuitively, the unlinkability property guarantees that session public keys that have been derived from the same master public key are computationally indistinguishable from the distribution of session public keys that have been derived from a different, independently chosen master public key. However, considering that hot wallet corruptions reveal the state and hence trivially break the unlinkability property, [DFL19] introduces the notion of forward unlinkability. This notion guarantees unlinkability prior to any hot wallet corruption.

```
Game WUNL
 1: $(mpk, msk, St) \leftarrow \texttt{SW.KGen}()$
 2: $b \leftarrow_{\$} \{0,1\}$
 3: $ID^* \leftarrow \mathcal{A}_1^{\mathsf{WalSign},\mathsf{PK}}(mpk)$
 4: if $(Keys[ID^*] \neq \bot)$ then
 5:     return 0
 6: $(pk^0_{ID^*}, St^*) \leftarrow \texttt{SW.RandPK}(mpk, ID^*, St)$
 7: $(sk^0_{ID^*}, St^*) \leftarrow \texttt{SW.RandSK}(msk, ID^*, St)$
 8: $St \leftarrow St^*$
 9: $(\overline{mpk}, \overline{msk}, \overline{St}) \leftarrow \texttt{SW.KGen}()$
10: $(pk^1_{ID^*}, St^*) \leftarrow \texttt{SW.RandPK}(\overline{mpk}, ID^*, \overline{St})$
11: $(sk^1_{ID^*}, St^*) \leftarrow \texttt{SW.RandSK}(\overline{msk}, ID^*, \overline{St})$
12: $Keys[ID^*] \leftarrow (pk^b_{ID^*}, sk^b_{ID^*})$
13: $b' \leftarrow \mathcal{A}_2^{\mathsf{WalSign},\mathsf{PK},\mathsf{getState}}(mpk, pk^b_{ID^*})$
14: return $b' = b$
```

```
Oracle PK($ID$)
1: $(pk_{ID}, St^*) \leftarrow \texttt{SW.RandPK}(mpk, ID, St)$
2: $(sk_{ID}, St^*) \leftarrow \texttt{SW.RandSK}(msk, ID, St)$
3: $Keys[ID] \leftarrow (pk_{ID}, sk_{ID})$
4: $St \leftarrow St^*$
5: return $pk_{ID}$
```

```
Oracle getState()
1: return $St$
```

```
Oracle WalSign($m, ID$)
1: if $(Keys[ID] = \bot)$ then
2:     return $\bot$
3: $(pk_{ID}, sk_{ID}) \leftarrow Keys[ID]$
4: $\sigma \leftarrow \texttt{SW.Sign}(sk_{ID}, m)$
5: return $\sigma$
```

**Fig. 5.** Unlinkability game WUNL for stateful wallets.

The formal security game for unlinkability is defined in Figure 5 and proceeds as follows: Upon the initialization of the wallet scheme by executing $(mpk, msk, St) \leftarrow_{\$} \texttt{SW.KGen}()$, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ obtains $mpk$ and runs its subprocedure $\mathcal{A}_1$ on input $mpk$, where $\mathcal{A}_1$ has access to oracles WalSign and PK. These oracles represent the adversary's capability to observe signatures with corresponding session public keys of the wallet on the ledger. More concretely, $\mathcal{A}_1$ can call WalSign on an arbitrary message $m$ and any $ID$ and receives a valid signature for $m$ under public key $pk_{ID}$. Further, $\mathcal{A}_1$ can query the PK oracle on any $ID$ and receives the session public key $pk_{ID}$.

Finally, $\mathcal{A}_1$ outputs an $ID^*$. If neither WalSign nor PK has been queried on $ID^*$ before, the game proceeds to the challenge phase, in which two session key pairs $(pk^0_{ID^*}, sk^0_{ID^*})$ and $(pk^1_{ID^*}, sk^1_{ID^*})$ are generated, where $(pk^0_{ID^*}, sk^0_{ID^*})$ are derived from $mpk$ and $msk$ respectively, while $(pk^1_{ID^*}, sk^1_{ID^*})$ are derived from a freshly generated master key pair. After a uniformly random bit $b$ is chosen, the subprocedure $\mathcal{A}_2$ is executed on input $mpk$ and $pk^b_{ID^*}$. $\mathcal{A}_2$ gets access to oracles WalSign, PK and getState, where getState returns the current state of the wallet scheme. $\mathcal{A}$ wins the game, if its subprocedure $\mathcal{A}_2$ returns a bit $b'$, such that $b' = b$. We define the advantage of an adversary $\mathcal{A}$ as its winning probability in game WUNL over random guessing.

**Definition 10 (Unlinkability).** *Let* SW *be a stateful wallet scheme (cf. Definition 8). We say that* SW *is pq-unlinkable if for any quantum adversary $\mathcal{A}$, the advantage in game* WUNL *(cf. Figure 5) is negligible.*

## 3.2 Wallet Unforgeability

At a high level, unforgeability for stateful wallets ensures that funds held by the cold wallet remain secure even in case an adversary corrupts the hot wallet and/or observes transactions on the ledger signed by the cold wallet. In order to model this property, we define the game WUF, in which the adversary $\mathcal{A}$ obtains a master

```
Game WUF
1: $(mpk, msk, St) \leftarrow \texttt{SW.KGen}()$
2: $b \leftarrow_\$ \{0,1\}$
3: $(m^*, \sigma^*, ID^*) \leftarrow \mathcal{A}^{\mathsf{WalSign},\mathsf{PK}}(mpk, St)$
4: if $(Keys[ID^*] \neq \perp)$ then
5:     return 0
6: $(pk_{ID^*}, sk_{ID^*}) \leftarrow Keys[ID^*]$
7: if $(m^* \in M[ID^*])$ then
8:     return 0
9: if $(\texttt{SW.Verify}(pk_{ID^*}, m^*, \sigma^*) = 0)$ then
10:    return 0
11: return 1
```

```
Oracle PK(ID)
1: $(pk_{ID}, St^*) \leftarrow \texttt{SW.RandPK}(mpk, ID, St)$
2: $(sk_{ID}, St^*) \leftarrow \texttt{SW.RandSK}(msk, ID, St)$
3: $Keys[ID] \leftarrow (pk_{ID}, sk_{ID})$
4: $St \leftarrow St^*$
5: return $pk_{ID}$
```

```
Oracle WalSign(m, ID)
1: if $(Keys[ID] = \perp)$ then
2:     return $\perp$
3: $(pk_{ID}, sk_{ID}) \leftarrow Keys[ID]$
4: $\sigma \leftarrow \texttt{SW.Sign}(sk_{ID}, m)$
5: $M \leftarrow_\cup \{m\}$
6: return $\sigma$
```

**Fig. 6.** Unforgeability game $\mathsf{WUF}$ for stateful wallets.

public key $mpk$ and the initial state $St_0$ as input. This models the situation in which an adversary corrupts the hot wallet right after initialization of the wallet scheme. Further, $\mathcal{A}$ gets access to the oracles $\mathsf{PK}$ and $\mathsf{WalSign}$, which are defined in the same way as in the game $\mathsf{WUL}$, with the difference that $\mathsf{WalSign}$ now additionally keeps track of all queried messages. Eventually, $\mathcal{A}$ outputs a forgery consisting of a message $m^*$, a signature $\sigma^*$ and an $ID^*$. $\mathcal{A}$ wins the game if (1) $m^*$ has not been queried to $\mathsf{WalSign}$ before, (2) $\mathsf{PK}$ has been previously queried on $ID^*$ and (3) $\sigma^*$ is a valid signature for $m^*$ under public key $pk_{ID^*}$.

Note that the adversary knows $mpk$ and $St_0$ and hence can generate any session public key for any $ID$ itself, which seems to make the $\mathsf{PK}$ oracle redundant. However, $\mathsf{PK}$ is still needed for bookkeeping purposes, i.e., to ensure that the session key pair for $\mathcal{A}$'s forgery has been created before $\mathcal{A}$ outputs its forgery. We define the advantage of an adversary $\mathcal{A}$ as its probability of winning the game $\mathsf{WUF}$.

As mentioned in [DFL19], the fact that the adversary can derive arbitrary session public keys makes the wallet scheme vulnerable to *related key attacks*, in case the underlying signature scheme is prone to such attack. Intuitively, upon an adversary learning a signature $\sigma_{ID}$ and a corresponding session public key $pk_{ID}$, a related key attack allows the adversary to transform $\sigma_{ID}$ into a valid signature $\sigma_{ID^*}$ under public key $pk_{ID^*}$. This attack may have a severe impact on the security guarantees of our wallet scheme, since it may allow an adversary to steal all funds of a cold wallet. One common counter measure against related key attack used in [DFL19, MSM+15] is called *public key prefixing*, i.e., a signature on a message $\mu$ is computed as $\texttt{Sign}(sk, (pk, \mu))$. In many signature schemes the signature is computed on the hash of the message and not the message itself. Therefore by prefixing the public key an adversary not only has to transform $\sigma_{ID}$ into a valid signature $\sigma_{ID^*}$ under public key $pk_{ID^*}$ but also find a collision for the hash function in order to mount a related key attack.

**Definition 11 (Unforgeability).** *Let* $\mathtt{SW}$ *be a stateful wallet scheme (cf. Definition 8). We say that* $\mathtt{SW}$ *is pq-unforgeable if for any quantum adversary* $\mathcal{A}$*, the advantage in game* $\mathsf{WUF}$ *(cf. Figure 6) is negligible.*

### 3.3 Relevance of Our Relaxed Notions

In Section 2 we defined the notions of *signature schemes with rerandomizable public keys* (cf. Definition 4) and $\mathsf{EUF\text{-}CMA\text{-}HRK}$ (cf. Definition 6). While these notions deviate from the ones used in previous works [FKM+16, DFL19], it turns out that our relaxed notions are sufficient for building deterministic wallets as we discuss below.

**Rerandomizable public keys.** One of the benefits of using deterministic wallets is that individual payments to the wallet are unlinkable (cf. Figure 5). To satisfy the unlinkability definition, Das et al. [DFL19] require that the underlying signature scheme must have rerandomizable secret and public keys. However, as it can be observed in the unlinkability game, the adversary gets access only to the public key, while the secret key is never revealed to the adversary (as revealing it would trivially break the security of the scheme). Hence, it is sufficient to use our relaxed notion of rerandomizable public keys in order to achieve the unlinkability property. While the post-quantum secure signature scheme that we consider in this work does not offer rerandomizable public and secret keys as required by [DFL19], it fortunately achieves our relaxed notion of rerandomizable public keys. Thus, it is sufficient to instantiate a wallet scheme that achieves the unlinkability property.

**EUF-CMA-HRK.** As in [DFL19], we use the notion of *EUF-CMA under honestly rerandomizable keys*, where unforgeability holds if the randomness used to derive the keys is *honestly* generated. This is in contrast to the stronger notion of $\mathsf{EUF\text{-}CMA\text{-}RK}$ as defined in [FKM+16], where unforgeability must hold for *adversarial* chosen randomness. Stateful deterministic wallet schemes, however, derive the randomness deterministically from the state (see Figure 4), which is generated initially during a trusted setup when the master keys are created. Hence, the adversary has no influence on the randomness used during the rerandomization procedures. To conclude, the notion of $\mathsf{EUF\text{-}CMA\text{-}HRK}$ is not only suitable but also sufficient in the wallet setting.

### 3.4 Post-Quantum Security of Wallets

In this section we show that the generic construction achieves both unlinkability and unforgeability against quantum adversaries. Recall that since we are in the post-quantum setting, the oracles provided by the challenger in the unlinkability game ($\mathsf{PK}$, $\mathsf{getState}$, $\mathsf{WalSign}$) and in the unforgeability game ($\mathsf{PK}$, $\mathsf{WalSign}$) are run on a classical computer. Hence, also the (quantum) adversary gets only classical access to these oracles. However, the adversary can use its quantum computing power to access the quantum random oracle $|\mathsf{H}\rangle$, i.e., querying the random oracle in superposition.

The following theorem shows the unlinkability.

**Theorem 1.** *Let* `RSig` *be a signature scheme with rerandomizable public keys (cf. Definition 4) and* $\mathsf{H}$ *a random oracle. Then the stateful wallet scheme* `SW` *built from* `RSig` *and* $\mathsf{H}$ *(cf. Figure 4) is pq-unlinkable according to Definition 10, i.e., against quantum adversaries which have access to* $|\mathsf{H}\rangle$.

First, we provide a proof intuition of Theorem 1. Let us first recall how the unlinkability property is proven in the classical ROM setting (cf. [DFL19]). Note that the wallet public keys are derived from the wallet state, which is stored within the wallet, hidden from the adversary. The classical adversary can then try to guess one of the states of the wallet and make a "problematic query" to the random oracle $\mathsf{H}$ on such a state, in order to derive one of the session public keys generated by the wallet. If the adversary guesses the wallet's state correctly, it can distinguish a public key generated by the wallet from a randomly generated one, and hence the adversary will be able to win the unlinkability game. The classical proof consists of two steps: (1) showing the probability that the adversary makes the above mentioned *problematic* query to the random oracle is negligible, and (2) showing that the adversary has no advantage in winning the unlinkability game conditioned on the event that it does not make any problematic query. Finally, note that while this proof uses the stronger notion of rerandomizable public and secret keys (cf. Definition 3), it is easy to see that it also works with our relaxed definition of rerandomizable public keys (cf. Definition 4). This is because the unlinkability game requires the adversary to distinguish a public key generated by the wallet from a randomly generated public key.

Our proof in the QROM follows the same approach, however, the first step requires a different technique. Recall that the wallet state gets refreshed with every public key query. In [DFL19], the challenger keeps a list of the states of the wallet scheme – starting from the initial state till the one obtained during last public key query. In the analysis a simple comparison allows to check whether a query by the classical adversary is problematic, i.e., whether it coincides with one of the states of the wallet. Since the adversary can access the random oracle $\mathsf{H}$ only classically (it can query on exactly one input at a time), hence the challenger can store all these queries in a list. In the QROM, however, we can not keep such a list as the adversary now has quantum computation power, hence can query the random oracle on several, and even all, inputs in superposition.[5] Instead, we consider a game hop which we can bound by the advantage of the adversary in distinguishing two random oracles, which in turn can be bound using the O2H lemma. For the resulting game, we can show via a reduction to the rerandomizability property of public keys of the RSig scheme, using the simulatability property of RSig, that the adversary has only negligible advantage in winning the game.

[5] We note that, to some extent, the *compressed oracle technique* by Zhandry [Zha19] allows the recording of superposition queries.

*Proof (of Theorem 1).* Throughout, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary which makes $q$ and $q_{\mathsf{PK}}$ queries to its oracles $|\mathsf{H}\rangle$ and $\mathsf{PK}$, respectively. To prove the theorem we use the following two games.

**Game $\mathsf{G}_0$:** This game is the game $\mathsf{WUNL}$ instantiated with $\mathtt{SW}$ (cf. Figure 4).

**Game $\mathsf{G}_1$:** This game is the same as $\mathsf{G}_0$, except that the randomness $\rho$ and the new state $St^*$, prior to running $\mathcal{A}_2$ (i.e., Line 13 in Figure 5), are sampled at random, independent of the random oracle. In both games, the randomness and new state are distributed identical. The only difference lies in the random oracle. From the point of view of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the random oracle in game $\mathsf{G}_1$ is $|\mathsf{H}_{\mathcal{S}\to\$}\rangle$, i.e., the random oracle that is reprogrammed to random values for every $x \in \mathcal{S}$, where $\mathcal{S}$ contains all pairs of states and IDs prior to running $\mathcal{A}_2$. Hence, we can bound the advantage in distinguishing $\mathsf{G}_0$ and $\mathsf{G}_1$ by the advantage in distinguishing the random oracles $|\mathsf{H}\rangle$ and $|\mathsf{H}_{\mathcal{S}\to\$}\rangle$. Applying the O2H Lemma (cf. Lemma 1) yields

$$\left|\Pr\left[\mathcal{A}^{|\mathsf{H}\rangle} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{|\mathsf{H}_{\mathcal{S}\to\$}\rangle} \Rightarrow 1\right]\right| \leq 2q\sqrt{\Pr[x \in \mathcal{S} \colon \mathcal{B}^{|\mathsf{H}\rangle} \Rightarrow x]}\,.$$

where $\mathcal{B}$ is the adversary specified in Lemma 1. Note that $\mathcal{A}$ has no information about the states in the set $\mathcal{S}$ until it queries $\mathsf{getState}$ to which only $\mathcal{A}_2$ has access. Furthermore, we have $|\mathcal{S}| \leq q_{\mathsf{PK}} + 2$ at any point in time, $q_{\mathsf{PK}}$ from $\mathcal{A}_1$'s queries and 2 from the challenge phase. This yields

$$\Pr\left[x \in \mathcal{S} \colon \mathcal{B}^{|\mathsf{H}\rangle} \Rightarrow x\right] \leq \frac{|\mathcal{S}|}{2^\lambda} \leq \frac{q_{\mathsf{PK}} + 2}{2^\lambda}.$$

Combining the above equations yields that the advantage in distinguishing $\mathsf{G}_0$ and $\mathsf{G}_1$ is negligible in the security parameter $\lambda$.

It remains to bound the advantage of $\mathcal{A}$ in game $\mathsf{G}_1$, where the same argument from the classical proof applies. In $\mathsf{G}_1$, the challenge public key $pk^b_{ID^*}$ given to $\mathcal{A}_2$ is independent of the random oracle (as the random oracle is not used in $\mathsf{G}_1$ anymore for deriving keys). Hence, it is irrelevant whether the adversary makes any query (classical or quantum) to the random oracle. We can show via a reduction to the rerandomizability of public keys property of $\mathtt{RSig}$ that the challenge public keys $pk^0_{ID^*}$ and $pk^1_{ID^*}$ are computationally indistinguishable, due to the simulatibility property of $\mathtt{RSig}$. This yields that the adversarial advantage is negligible. Combining the above proves the theorem. $\square$

The following theorem shows that the generic construction is unforgeable in the presence of quantum attackers.

**Theorem 2.** *Let* $\mathtt{RSig}$ *be a signature scheme with rerandomizable public keys (cf. Definition 4) and* $\mathsf{H}$ *a random oracle. Then the stateful wallet scheme* $\mathtt{SW}$ *built from* $\mathtt{RSig}$ *and* $\mathsf{H}$ *(cf. Figure 4) is pq-unforgeable according to Definition 11, i.e., against quantum adversaries which have access to* $|\mathsf{H}\rangle$*.*

We briefly recap the classical proof in the ROM (cf. [DFL19]), thereby highlighting the challenge when switching to a quantum adversary. Note again, that the classical proof uses the stronger notion of rerandomizable keys (cf. Definition 3) but also holds for the weaker notion of rerandomizable public keys (cf. Definition 4). The proof consists of a game hop in which the adversary loses the game if there is a collision of session keys for different identities. Due to the construction, this occurs if the random oracle outputs a collision which is bound by a simple counting argument. The advantage of an adversary in the resulting game is bound by the security of the underlying signature scheme using a reduction. The crucial part is that the reduction simulates the random oracle $\mathsf{H}$ for the adversary using its oracle $\mathsf{Rand}$ from the EUF-CMA-HRK game. More precisely, for each query to $\mathsf{H}$ by the adversary, the reduction makes a query to $\mathsf{Rand}$.

Our proof in the QROM follows the same idea, however additionally needs to take care of the use of the quantum random oracle $|\mathsf{H}\rangle$ by the adversary. The first part works exactly as in [DFL19], since the access to the oracle $\mathsf{PK}$ remains classical even for a quantum adversary. The second part, however, does not work as in [DFL19]. While the adversary can query the quantum random oracle $|\mathsf{H}\rangle$ in superposition, the reduction can query its oracle $\mathsf{Rand}$ only classical as it is provided by its (classical) challenger. By querying $|\mathsf{H}\rangle$ on an equal superposition of all (i.e., exponentially many) inputs, the reduction would need exponentially many queries to $\mathsf{Rand}$ in order to simulate $|\mathsf{H}\rangle$ for the adversary. Clearly, this would render the reduction useless as it would not be efficient. To tackle this issue, we do an additional game hop in which we switch from a random oracle to an oracle drawn from a small-range distribution. While this affects the advantage of the adversary only negligibly, it allows us to construct a reduction which can simulate the quantum random oracle for the adversary by making a polynomial number of (classical) queries to its oracle $\mathsf{Rand}$.

*Proof (of Theorem 2.).* Let $\mathcal{A}$ be an adversary which makes $q_{\mathsf{H}}$ queries to $|\mathsf{H}\rangle$. The proof consists of the following three games.

**Game $\mathsf{G}_0$:** This game is the game $\mathsf{WUF}$ instantiated with $\mathtt{SW}$ (cf. Figure 4). Assume that $\mathcal{A}$ has non-negligible advantage $\epsilon = \epsilon(\lambda)$ in winning $\mathsf{G}_0$. This means that there exists a polynomial $p = p(\lambda)$ such that $p(\lambda) > \frac{1}{\epsilon(\lambda)}$.

**Game $\mathsf{G}_1$:** This game is the same as $\mathsf{G}_0$, except the adversary loses when there is a collision of keys for different identities. To detect the change, the adversary has to make queries to $\mathsf{PK}$ which result in colliding keys. Note that the adversary only has classical access to $\mathsf{PK}$ as it is provided by the classical challenger. Hence the bound from [DFL19] is applicable, which is a simple counting argument over the number of queries to $\mathsf{PK}$. This yields that the advantage of $\mathcal{A}$ in $\mathsf{G}_1$ is $\epsilon - \mathrm{negl}(\lambda)$, i.e., it is negligibly close to its advantage $\epsilon$ in $\mathsf{G}_0$.

**Game $\mathsf{G}_2$:** In this game the adversaries queries to $|\mathsf{H}\rangle$ is simulated using Definition 1 and the Lemma 2. Let $l = 2Cq_{\mathsf{H}}^3 p$ with $C$ being the constant from Lemma 2 and $p$ being the polynomial described above. At the start of the game, the challenger will generate $l$ random values and draw the first output (the randomness $\rho$) of the quantum random oracle $|\mathsf{H}\rangle$ from a small-range distribution using these $l$ samples. The second output (the new state $St$) is generated just as in $\mathsf{G}_1$. According to Lemma 2, $\mathcal{A}$ can only distinguish this game from the previous one with probability less than $\frac{1}{2p}$. Therefore, Lemma 2 yields that the advantage of $\mathcal{A}$ in this game is at least $\epsilon - \text{negl}(\lambda) - \frac{1}{2p}$.

**Bounding the advantage of $\mathcal{A}$ in Game $\mathsf{G}_2$:** We now show how to transform an adversary $\mathcal{A}$ playing $\mathsf{G}_2$ into an adversary $\mathcal{B}$ playing $\mathsf{EUF\text{-}CMA\text{-}HRK}$ (where, the underlying signature scheme is $\mathtt{RSig}$). W.l.o.g., we assume that $\mathcal{A}$ never makes a query which results in $\perp$ and that there are no collisions. At the start, $\mathcal{B}$ receives a public key $pk$. It performs $l = 2Cq_{\mathsf{H}}^3p$ queries to its oracle $\mathsf{Rand}$ and samples an initial state $St_0$. It invokes $\mathcal{A}$ on input $(mpk = pk, St_0)$.

*Simulation of Quantum Random Oracle* $|\mathsf{H}\rangle$. $\mathcal{B}$ simulates the first output (the randomness $\rho$), by using the $l$ samples from $\mathsf{Rand}$ drawn from a small-range distribution. Note that $\mathsf{Rand}$ internally stores the output $\rho$ in its list $\mathsf{RList}$. For the second output (the new state $St$), $\mathcal{B}$ simulates it using a $2q_{\mathsf{H}}$-wise independent function which is indistinguishable for an adversary making $q_{\mathsf{H}}$ queries [Zha12b].

*Simulation of* $\mathsf{PK}$ *oracle.* When $\mathcal{A}$ queries its oracle $\mathsf{PK}$ on $ID$, $\mathcal{B}$ computes $pk_{ID} \leftarrow \mathtt{RSig.RandPK}(pk; \omega_{ID})$, where $(\omega_{ID}, St^*) \leftarrow \mathsf{H}(St, ID)$, sets $Keys[ID] \leftarrow (pk_{ID}, \omega_{ID})$, and sends $pk_{ID}$ to $\mathcal{A}$.

*Simulation of* $\mathsf{WalSign}$ *oracle.* When $\mathcal{A}$ makes a query $(m, ID)$ to its oracle $\mathsf{WalSign}$, $\mathcal{B}$ obtains the $(pk_{ID}, \omega_{ID}) = Keys[ID]$, sets $m' \leftarrow (m, pk_{ID})$, queries its own oracle $\mathsf{OHR}$ on $(m', \omega_{ID})$, and forwards the response to $\mathcal{A}$. When $\mathcal{A}$ outputs a forgery $(m^*, \sigma^*, ID^*)$, $\mathcal{B}$ obtains $(pk_{ID^*}, \omega_{ID^*}) = Keys[ID^*]$, sets $\hat{m^*} \leftarrow (m^*, pk_{ID^*})$, and outputs $(\hat{m^*}, \sigma^*, \omega_{ID^*})$.

*If $\mathcal{A}$'s forgery $(m^*, \sigma^*, ID^*)$ is valid in Game $\mathsf{G}_2$, then $\mathcal{B}$'s forgery $(\hat{m^*}, \sigma^*, \omega_{ID^*})$ is also valid in* $\mathsf{EUF\text{-}CMA\text{-}HRK}$. We now show that the output of $\mathcal{B}$ is a valid forgery whenever the output of $\mathcal{A}$ is. First, since $(m^*, \sigma^*, ID^*)$ is a valid forgery by $\mathcal{A}$, we know that $\mathcal{A}$ never queried $(m^*, ID^*)$ to $\mathsf{WalSign}$. Recall that, for every $\mathsf{WalSign}$ query by $\mathcal{A}$ on any message $(m)$, $\mathcal{B}$ made a $\mathsf{OHR}$ query on public key prefixed message $(m' \leftarrow \{m, pk\})$. Since $\mathcal{A}$ never queried $\mathsf{WalSign}$ on input $(m^*, ID^*)$, $\mathcal{B}$ never queried $(\hat{m^*}, \omega_{ID^*})$ to its oracle $\mathsf{OHR}$, where $\hat{m^*} \leftarrow \{m^*, pk\}$. Second, it holds that $\omega_{ID^*} \in \mathsf{RList}$. This follows from the simulation of the quantum random oracle where, for every possible output $(\rho, St)$, $\rho$ is in $\mathsf{RList}$. Third, validity of the forgery by $\mathcal{A}$ yields validity of the forgery by $\mathcal{B}$.

Recall that $l = 2Cq_{\mathsf{H}}^3p$ and as discussed at the beginning of this game, according to Lemma 2, the advantage of the adversary in this game is equal to $\epsilon - \text{negl}(\lambda) - \frac{1}{2p}$. Assuming the security of the underlying signature scheme $\mathtt{RSig}$, we have that this advantage must be negligible. Combined with $\epsilon > \frac{1}{p}$ (see description of $\mathsf{G}_0$), this yields that $\frac{1}{2p}$ is negligible, resulting in a contradiction. Hence, we conclude that $\epsilon$, the advantage of $\mathcal{A}$, is negligible. $\square$

## 4 PQ Signatures with Honestly Rerandomizable Public Keys

In this section we propose a lattice-based construction of a signature scheme with honestly rerandomizable public keys (cf. Definition 4). In such a signature scheme, the distribution of honestly rerandomized public keys is identical to the distribution of original public key, while honestly rerandomized secret keys are allowed to be distributed differently from the original secret key. The scheme extends the generic construction of lattice-based signatures from Section 2.5. We analyze the security of our scheme in Section 4.2. In Section 4.3 we discuss alternative ways of key rerandomization in a lattice-based signature scheme and argue why they fall short in building practical hot/cold wallets.

### 4.1 Description of the Scheme

Let $\mathsf{LB}.\Sigma = (\mathsf{LB}.\mathtt{KGen}, \mathsf{LB}.\mathtt{Sign}, \mathsf{LB}.\mathtt{Verify})$ be the lattice-based signature scheme given in Section 2.5, Figure 3, and let $\mathbf{A} \in R_q^{k_1 \times k_2}$ be a uniformly random matrix as defined in Section 2.5, i.e., $\mathbf{A}$ is publicly known and an implicit input to all algorithms. Furthermore, we define the following functions and algorithms:

- $\mathsf{Max}_j$ is a function that on input $a \in R$, it outputs the $j^{\text{th}}$ largest absolute coefficient of $a$. This function is used for bounding the secret-related terms, and hence the signatures generated by the algorithm $\mathsf{LB}.\mathtt{Sign}$ (cf. line 6–7 in Figure 3).
- $\mathsf{GenG}$ is an algorithm that on input $(\mathsf{dim}, \sigma, \mathsf{bnd}, \mathsf{rnd})$, it outputs a vector $\mathbf{x} = (x_1, \ldots, x_{\mathsf{dim}})$, where $x_i \in R$ are sampled from $D_{\mathbb{Z}^n,\sigma}$ such that $\sum_{j=1}^{\kappa} \mathsf{Max}_j(x_i) \leq \mathsf{bnd}$ by using a randomness $\mathsf{rnd}_i$ that is extracted from $\mathsf{rnd}$, e.g., via the function $\mathsf{E}$.
- $\mathsf{F} : \{0,1\}^* \longrightarrow \{0,1\}^{o(\lambda)}$ is a collision resistant hash function. It is used to hash the public key in order to prevent related key attacks [MSM+15].

In this section we set the distribution used in the algorithm $\mathsf{LB}.\mathtt{KGen}$ for the secret key to $\chi = D_{\mathbb{Z}^n,\sigma}$. More precisely, we assume that $sk = (\mathbf{s}, \mathbf{e}) \in D_{\mathbb{Z}^n,\sigma}^{k_2} \times D_{\mathbb{Z}^n,\sigma}^{k_1}$, where $\mathbf{s} \leftarrow \mathsf{GenG}(k_2, \sigma, S/2, \mathsf{rnd}_s)$ and $\mathbf{e} \leftarrow \mathsf{GenG}(k_1, \sigma, E/2, \mathsf{rnd}_e)$ for two predefined positive numbers $S, E$ and randomnesses $\mathsf{rnd}_s, \mathsf{rnd}_e$. Setting $\chi = D_{\mathbb{Z}^n,\sigma}$ is essential for rerandomizing the secret key in the construction introduced in this section because the sum of Gaussian distributed elements with standard deviation $\sigma$ is also Gaussian distributed with standard deviation $\sqrt{2}\sigma$ (cf. Lemma 3).

In the following we describe our signature scheme with honestly rerandomizable public keys. The respective algorithms are formalized in Figure 7. In order to simplify the construction, we first define the algorithm $\mathsf{RandG}$ (see Figure 7 for a formal description). This algorithm takes as input a randomness $\rho = (\rho_s, \rho_e) \in \{0,1\}^{o(\lambda)} \times \{0,1\}^{o(\lambda)}$, and outputs two vectors $\mathbf{r}, \mathbf{u}$, which are generated by running the algorithm $\mathsf{GenG}$ on input $(k_2, \sigma, S/2, \rho_s)$, $(k_1, \sigma, E/2, \rho_e)$, respectively.

```
RandG(ρ)
1: $\rho := (\rho_s, \rho_e) \in \{0,1\}^{2o(\lambda)}$
2: $\mathbf{r} \leftarrow \mathsf{GenG}(k_2, \sigma, S/2, \rho_s)$
3: $\mathbf{u} \leftarrow \mathsf{GenG}(k_1, \sigma, E/2, \rho_e)$
4: return $(\mathbf{r}, \mathbf{u})$

RSig.KGen(1^λ)
1: $(sk, pk) \leftarrow \mathsf{LB}.\mathtt{KGen}(1^\lambda)$
2: $\mathsf{hpk} \leftarrow \mathsf{F}(pk)$
3: $sk \leftarrow (\mathsf{hpk}, sk)$
4: return $(sk, pk)$

RSig.RandPK(pk, ρ)
1: $(\mathbf{r}, \mathbf{u}) \leftarrow \mathsf{RandG}(\rho)$
2: $\mathbf{b}' \leftarrow \mathbf{b} + \mathbf{Ar} + \mathbf{u} \pmod q$
3: $pk' := \mathbf{b}'$
4: return $pk'$

RSig.RandSK(sk, ρ)
1: $(\mathbf{r}, \mathbf{u}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma} \leftarrow \mathsf{RandG}(\rho)$
2: $\mathbf{s}' \in D^{k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma} \leftarrow \mathbf{s} + \mathbf{r}$
3: $\mathbf{e}' \in D^{k_1}_{\mathbb{Z}^n,\sqrt{2}\sigma} \leftarrow \mathbf{e} + \mathbf{u}$
4: $\mathbf{b}' \leftarrow \mathbf{As}' + \mathbf{e}' \pmod q$
5: $\mathsf{hpk}' \leftarrow \mathsf{F}(\mathbf{b}')$
6: $sk' := (\mathsf{hpk}', \mathbf{s}', \mathbf{e}')$
7: return $sk'$

RSig.Sign(sk, m)
1: $\mu \leftarrow (m, \mathsf{hpk})$
2: $(\mathbf{z}_1, \mathbf{z}_2, c) \leftarrow \mathsf{LB}.\mathtt{Sign}(sk, \mu)$
3: return $(\mathbf{z}_1, \mathbf{z}_2, c)$

RSig.Verify(pk, m, (z_1, z_2, c))
1: $\mu \leftarrow (m, \mathsf{F}(pk))$
2: return $\mathsf{LB}.\mathtt{Verify}(pk, \mu, (\mathbf{z}_1, \mathbf{z}_2, c))$
```

**Fig. 7.** Construction of lattice-based signature scheme with honestly rerandomizable public keys.

**RSig.KGen:** The key generation algorithm runs the algorithm LB.KGen to obtain a key pair $(sk, pk)$, where $sk = (\mathbf{s}, \mathbf{e}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma}$ and $pk = \mathbf{b} \in R_q^{k_1}$. Then, it computes $\mathsf{hpk} = \mathsf{F}(pk)$, prepends hpk to $sk$, and returns the updated $(sk, pk)$.

**RSig.RandPK:** Given a public key $pk = \mathbf{b}$ and an honestly generated randomness $\rho$, the algorithm RSig.RandPK runs $\mathsf{RandG}(\rho)$ to generate a pair of Gaussian distributed vectors $(\mathbf{r}, \mathbf{u})$. Then, it computes $\mathbf{b}' = \mathbf{b} + \mathbf{Ar} + \mathbf{u} \pmod q$ and outputs the honestly rerandomized public key $pk' = \mathbf{b}'$.

**RSig.RandSK:** Given a secret key $sk = (\mathsf{hpk}, \mathbf{s}, \mathbf{e})$ and an honestly generated randomness $\rho \in \{0,1\}^{2o(\lambda)}$, the algorithm RSig.RandSK runs RandG to obtain $(\mathbf{r}, \mathbf{u}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma}$. Then, it computes $\mathbf{s}' = \mathbf{s} + \mathbf{r}$ and $\mathbf{e}' = \mathbf{e} + \mathbf{u}$. Note that by Lemma 3, the pair $(\mathbf{s}', \mathbf{e}')$ is distributed as $D^{k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sqrt{2}\sigma}$. Finally, the algorithm computes $\mathsf{hpk}' = \mathsf{F}(\mathbf{b}')$ and outputs the honestly rerandomized secret key $sk' = (\mathsf{hpk}', \mathbf{s}', \mathbf{e}')$.

**RSig.Sign:** The algorithm RSig.Sign returns the signature obtained by calling the algorithm LB.Sign on message $\mu = (m, \mathsf{hpk})$. Signing messages together with the hash value of (honestly rerandomized) public keys ensures security under related key attacks [MSM+15].

**RSig.Verify:** The algorithm RSig.Verify returns the bit obtained by running $\mathsf{LB}.\mathtt{Verify}(pk, \mu)$, where $\mu = (m, \mathsf{F}(pk))$.

We note that rerandomizing $sk$ must be carried out only with the original secret key, i.e., a rerandomized secret key cannot be used to generate a new rerandomized one. This ensures that all honestly rerandomized secret keys have identical distribution, i.e., $D^{k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sqrt{2}\sigma}$. Furthermore, signatures generated using honestly rerandomized keys have different distribution from signatures generated using the original key pair. More precisely, the pair $(\mathbf{z}_1, \mathbf{z}_2)$ is distributed uniformly at random

```
Reduction $\mathcal{D}(pk)$
1: $\mathsf{RList} := \varnothing$
2: $\mathcal{Q} := \varnothing$
3: $(m, ((\mathbf{z}_1, \mathbf{z}_2, c), \rho)) \leftarrow \mathcal{A}^{\mathsf{H}', \mathsf{Rand}, \mathsf{OHR}}(pk)$
4: if $(\rho = \mathsf{NULL})$ then
5:     $\mathsf{hpk} \leftarrow \mathsf{F}(pk)$
6:     $\mu \leftarrow (m, \mathsf{hpk})$
7:     return $(\mu, (\mathbf{z}_1, \mathbf{z}_2, c))$
8: if $(\rho \neq \mathsf{NULL})$ then
9:     $pk' \leftarrow \mathtt{RSig.RandPK}(pk, \rho)$
10:    $\mathsf{hpk}' \leftarrow \mathsf{F}(pk')$
11:    $\mu' \leftarrow (m, \mathsf{hpk}')$
12:    $(\mathbf{r}, \mathbf{u}) \in D^{k_2}_{\mathbb{Z}^n,\sigma} \times D^{k_1}_{\mathbb{Z}^n,\sigma} \leftarrow \mathsf{RandG}(\rho)$
13:    $\mathbf{z}'_1 \leftarrow \mathbf{z}_1 - \mathbf{r}c$
14:    $\mathbf{z}'_2 \leftarrow \mathbf{z}_1 - \mathbf{u}c$
15: return $(\mu', (\mathbf{z}'_1, \mathbf{z}'_2, c))$
```

**Fig. 8.** Reduction from the EUF-CMA security of LB.$\Sigma$ (Figure 3) to EUF-CMA-HRK security of signature scheme with honestly rerandomizable public keys (Figure 7). Queries to OHR, H′, and Rand are answered as shown in Figure 9.

over $R^{k_2}_{B_1} \times R^{k_1}_{B_2}$, where

$$B_1 = \begin{cases} Y - S/2 & \text{if } sk \leftarrow \mathsf{LB}.\mathtt{KGen} \\ Y - S & \text{if } sk \leftarrow \mathtt{RSig.RandSK} \end{cases}$$
$$B_2 = \begin{cases} Y - E/2 & \text{if } sk \leftarrow \mathsf{LB}.\mathtt{KGen} \\ Y - E & \text{if } sk \leftarrow \mathtt{RSig.RandSK} \end{cases}$$

The bound $Y$ of the masking pair $(\mathbf{y}_1, \mathbf{y}_2)$ is chosen such that the probability of generating valid signatures (cf. Section 2.5) is at least $1/M$, i.e.,

$$\left(\frac{2B_1 + 1}{2Y + 1}\right)^{k_2 n} \cdot \left(\frac{2B_2 + 1}{2Y + 1}\right)^{k_1 n} \geq 1/M,$$

where $M = O(1)$ is the repetition rate of the signing algorithm.

## 4.2 Security Analysis

In this section we analyze the EUF-CMA-HRK security of the scheme introduced in Section 4.1 in the QROM. More precisely, we reduce its EUF-CMA-HRK security to the EUF-CMA security of the lattice-based signature scheme LB.$\Sigma$ = (LB.$\mathtt{KGen}$, LB.$\mathtt{Sign}$, LB.$\mathtt{Verify}$) described in Section 2.5. The correctness of the scheme directly follows from the correctness of LB.$\Sigma$. Note that rerandomizability of public keys (see Definition 4) follows from the MLWE assumption [LS15]. That is, for any public key $\mathbf{b}$ and any honestly rerandomized public key $\mathbf{b}'$ both pairs $(\mathbf{A}, \mathbf{b})$, $(\mathbf{A}, \mathbf{b}')$ are indistinguishable from the uniform distribution over $R_q^{k_1 \times k_2} \times R_q^{k_1}$. Therefore, $\mathbf{b}$ and $\mathbf{b}'$ are identically distributed.

**Theorem 3 (EUF-CMA-HRK Security).** *The signature scheme with honestly rerandomizable public keys depicted in Figure 7 is EUF-CMA-HRK secure in the* QROM *if scheme $LB.\Sigma = (LB.\texttt{KGen}, LB.\texttt{Sign}, LB.\texttt{Verify})$ described in Figure 3 is EUF-CMA secure in the* QROM*.*

*Proof.* Let $\mathcal{A}$ be an adversary that is able to generate valid forgeries under the signature scheme with honestly rerandomizable public keys, i.e., $\mathcal{A}$ is able to win the game $\mathsf{EUF\text{-}CMA\text{-}HRK}^{\mathcal{A}}_{\texttt{RSig}}$ (cf. Definition 6). We construct an algorithm $\mathcal{D}$ that runs $\mathcal{A}$ as subroutine in order to win the game $\mathsf{EUF\text{-}CMA}^{\mathcal{D}}_{\mathsf{LB}.\Sigma}$ (see Definition 5) against the scheme $\mathsf{LB}.\Sigma$. According to the security model, $\mathcal{A}$ has quantum access to a random oracle $\mathsf{H}'$ and classical access to a random oracle $\mathsf{Rand}$ in addition to classical access to the signing oracle $\mathsf{OHR}$. The reduction $\mathcal{D}$ has quantum access to the random oracle $\mathsf{H}$ and classical access to the signing oracle $\mathsf{O}$, which returns to $\mathcal{D}$ signatures generated by $\mathsf{LB}.\Sigma$. The algorithm $\mathcal{D}$ is described in Figure 8. $\mathcal{D}$ initializes two empty lists $\mathsf{RList}, \mathcal{Q}$. These are used by $\mathcal{D}$ to store queries to $\mathsf{Rand}$ and $\mathsf{OHR}$, respectively. Simulation of $\mathsf{OHR}, \mathsf{H}'$, and $\mathsf{Rand}$ is given in Figure 9.

*Analysis.* Let $(m, ((\mathbf{z}_1, \mathbf{z}_2, c), \rho))$ be a valid forgery output by the adversary $\mathcal{A}$. This means that $m \notin \mathcal{Q}$ and $\texttt{RSig.Verify}(pk, m, (\mathbf{z}_1, \mathbf{z}_2, c)) = 1$. Moreover, $\rho \in \mathsf{RList}$ in case randomness $\rho \neq \mathsf{NULL}$.

We first analyze the case that $\rho = \mathsf{NULL}$. The signature satisfies $(\mathbf{z}_1, \mathbf{z}_2) \in R^{k_2}_{Y-\frac{S}{2}} \times R^{k_1}_{Y-\frac{E}{2}}$ and $c = \mathsf{H}'(\mathbf{w}, m, \mathsf{hpk}) = \mathsf{H}(\mathbf{w}, m, \mathsf{hpk})$, where $\mathbf{w} = \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{b}c \pmod q$. Hence, this forgery constitutes a valid signature under $\mathsf{LB}.\Sigma$ on message $\mu = (m, \mathsf{hpk})$. Note that if $c$ was not queried by some input, then $\mathcal{A}$ produces such $c$ only with negligible probability, i.e., $1/|\mathbb{T}^n_\kappa|$. Thus, with probability of $1 - 1/|\mathbb{T}^n_\kappa|$, the value $c$ must be a random oracle answer to a query made by $\mathcal{A}$, where $|\mathbb{T}^n_\kappa| = 2^\kappa \binom{n}{\kappa}$ and $\kappa$ is chosen such that $|\mathbb{T}^n_\kappa| \geq 2^{2\lambda}$. This ensures that the probability of mapping two different values to the same output of $\mathsf{H}$ is at most $2^{-2\lambda}$.

Next, we assume that $\mathcal{A}$ outputs a valid forgery $(m, (\mathbf{z}_1, \mathbf{z}_2, c), \rho)$ under honestly rerandomized public key $\mathbf{b}'$ and $\rho \neq \mathsf{NULL}$. This means that $(\mathbf{z}_1, \mathbf{z}_2) \in R^{k_2}_{Y-S} \times R^{k_1}_{Y-E}$. In this case $\mathcal{D}$ transforms this signature into a forgery under the original public key $\mathbf{b}$ as follows: $\mathcal{D}$ runs $\mathsf{RandG}(\rho)$ to obtain $(\mathbf{r}, \mathbf{u})$. Then, it computes the vectors $\mathbf{z}_1' = \mathbf{z}_1 - \mathbf{r}c$ and $\mathbf{z}_2' = \mathbf{z}_2 - \mathbf{u}c$. Note that

$$\begin{aligned}
\|\mathbf{z}_1'\|_\infty &\leq \|\mathbf{z}_1\|_\infty + \|\mathbf{r}c\|_\infty \leq\ Y - S + S/2 = Y - S/2, \\
\|\mathbf{z}_2'\|_\infty &\leq \|\mathbf{z}_2\|_\infty + \|\mathbf{u}c\|_\infty \leq Y - E + E/2 = Y - E/2\,.
\end{aligned}$$

Hence, $(\mathbf{z}_1', \mathbf{z}_2') \in R^{k_2}_{Y-\frac{S}{2}} \times R^{k_1}_{Y-\frac{E}{2}}$. Furthermore, we have

$$\mathbf{w} = \mathbf{A}\mathbf{z}_1' + \mathbf{z}_2' - \mathbf{b}c = \mathbf{A}(\mathbf{z}_1 - \mathbf{r}c) + \mathbf{z}_2 - \mathbf{u}c - \mathbf{b}c = \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{b}'c \pmod q\,.$$

Therefore, it holds that $c = \mathsf{H}'(\mathbf{w}, m, \mathsf{hpk}') = \mathsf{H}(\mathbf{w}, m, \mathsf{hpk}')$. Hence, the forgery output by $\mathcal{A}$ can be turned into a valid forgery under the original public key $\mathbf{b}$ for message $\mu' = \mathsf{F}(m, \mathsf{hpk}')$, i.e., it is a forgery under $\mathsf{LB}.\Sigma$.

```
Sim(pk, m, ρ)
1: if ($\rho = \mathsf{NULL}$) then
2:     $\mathcal{Q} := \mathcal{Q} \cup \{m\}$
3:     return $\mathsf{SimNoR}(pk, m)$
4: if ($\rho \neq \mathsf{NULL}$) then
5:     if ($\rho \notin \mathsf{RList}$) then
6:         return $\perp$
7:     $\mathcal{Q} := \mathcal{Q} \cup \{m\}$
8:     return $\mathsf{SimR}(pk, m, \rho)$

SimNoR(pk, m)
1: $\mathsf{hpk} \leftarrow \mathsf{F}(\mathbf{b})$
2: $\mu \leftarrow (m, \mathsf{hpk})$
3: $(\mathbf{z}_1, \mathbf{z}_2, c) \leftarrow \mathsf{O}(\mu)$
4: return $(\mathbf{z}_1, \mathbf{z}_2, c)$

H'(·)
1: return $\mathsf{H}(\cdot)$

SimR(pk, m, ρ)
1: $(\mathbf{r}, \mathbf{u}) \leftarrow \mathsf{RandG}(\rho)$
2: $pk' := \mathbf{b}' \leftarrow \mathtt{RSig.RandPK}(pk, \rho)$
3: $\mathsf{hpk}' \leftarrow \mathsf{F}(\mathbf{b}')$
4: $\mu' \leftarrow (m, \mathsf{hpk}')$
5: $(\mathbf{z}'_1, \mathbf{z}'_2, c) \leftarrow \mathsf{O}(\mu')$
6: $\mathbf{z}_1 \leftarrow \mathbf{z}'_1 + \mathbf{r}c$
7: $\mathbf{z}_2 \leftarrow \mathbf{z}'_2 + \mathbf{u}c$
8: if $\left((\mathbf{z}_1, \mathbf{z}_2) \notin R^{k_2}_{Y-S} \times R^{k_1}_{Y-E}\right)$ then
9:     goto 5
10: return $(\mathbf{z}_1, \mathbf{z}_2, c)$

Rand()
1: $\rho \leftarrow_{\$} \{0,1\}^{2o(\lambda)}$
2: $\mathsf{RList} \leftarrow \mathsf{RList} \cup \{\rho\}$
3: return $\rho$
```

**Fig. 9.** Description of algorithm Sim, which simulates signing queries to OHR. The algorithms SimNoR, SimR are subroutines used by Sim. The first one is called when signing query does not include randomness $\rho$, while the latter one is called when signing query includes honestly generated randomness $\rho \neq \mathsf{NULL}$. Queries to $\mathsf{H}'$ made by adversary $\mathcal{A}$ are redirected to the random oracle $\mathsf{H}$, to which reduction $\mathcal{D}$ has access. Queries to Rand are answered locally by $\mathcal{D}$.

Finally, we note that the environment of $\mathcal{A}$ is perfectly simulated, and whenever $\mathcal{A}$ wins the game $\mathsf{EUF\text{-}CMA\text{-}HRK}^{\mathcal{A}}_{\mathtt{RSig}}$, $\mathcal{D}$ wins the game $\mathsf{EUF\text{-}CMA}^{\mathcal{D}}_{\mathsf{LB}.\Sigma}$. The number of signing queries made by $\mathcal{D}$ is at most $M \cdot Q$, where $M = O(1)$ is the repetition rate[6] of $\mathsf{LB}.\Sigma$ and $Q$ is the number of signing queries made by $\mathcal{A}$. □

## 4.3 Alternative Methods for Rerandomization

In this section we describe alternative approaches for rerandomizing keys in the lattice setting and show why our scheme introduced in Section 4.1 is the most suitable option in the context of hot/cold wallets. First, we recall that our construction from the previous section assumes that the distribution of the secrets used in the key generation algorithm are from the Gaussian distribution, i.e., $\chi = D_{\mathbb{Z}^n,\sigma}$. This allows us to use Lemma 3 in order to obtain rerandomized secret keys that are also Gaussian distributed but with a slightly different standard deviation, i.e., $D_{\mathbb{Z}^n,\sqrt{2}\sigma}$. The key generation of our scheme cannot use uniformly distributed secrets over a small subset $R_d$ from $R$, where $R_d$ is the set of all polynomials from $R$ with $\ell_\infty$ norm bounded by some integer $d \geq 1$. This is because the sum of two uniformly random polynomials over $R_d$ does not yield a polynomial that follows the uniform distribution over a subset $S \subseteq R_d$. Using a uniformly random *sk* for rerandomization would yield rerandomized secret keys with unknown distribution, and hence the hardness of the computational assumption underlying the rerandomized key pairs would be unclear. Let us now discuss the alternative approaches.

[6] In practice, the repetition rate of the signing algorithm of standard lattice-based signature schemes is strictly smaller than 4 (e.g., see [DKL+18, ABB+20]).

**Rerandomizability of Gaussian distributed secret keys.** It is (theoretically) possible to rerandomize key pairs such that the rerandomized secret keys have the same distribution as the original secret key. More precisely, assume that $sk$ is Gaussian distributed with standard deviation $\sigma$. Given a randomness $\rho$, a rerandomized secret key is computed as $sk' = sk + \rho$. Due to [GKPV10, Lemma 3], $sk'$ is Gaussian distributed with the same $\sigma$ when $\sigma$ is of a super-polynomial size in the security parameter $\lambda$. In other words, we must select $\sigma$ large enough in order to make the statistical distance between the distribution of $sk$ and $sk'$ negligible in $\lambda$. This value of $\sigma$ gives secret keys of very large size, and requires to increase the size of the masking vectors used in the signing algorithm. Hence, we obtain signatures of very large size, which rules out using the resulting scheme in practice.

**Rerandomizability of uniform distributed secret keys.** In theory, it is possible to use uniformly distributed rather than Gaussian distributed secret keys as follows. Assume that $\chi = R_d$ and $\rho \in R_1$. The rerandomized secret key $sk' = sk + \rho$ is uniformly distributed over $R_{d-1}$ with probability $\left(\frac{2d-1}{2d+1}\right)^{(k_1+k_2)n}$, where $(k_1 + k_2)n$ is the dimension of $sk'$. Therefore, for a very large $d$ this probability would be overwhelming in $\lambda$.

*Example 1.* By considering the parameters of Dilithium [DKL+18] proposed for $\lambda = 128$, we have $k_1 = 5$, $k_2 = 4$, and $n = 256$. Hence, we have to set $d \approx 2^{139}$ in order to make the previously stated probability at least $1 - 2^{-128}$. This value of $d$ yields a secret key of size $\approx 2^{147}$ Bytes.

The above given example shows that this approach is merely of theoretical interest only and is not suitable for practical applications as it requires huge sizes of keys, and hence signatures.

**Allowing rerandomization algorithms to communicate.** Consider an application, in which the rerandomization algorithms (i.e., `RSig.RandSK` and `RSig.RandPK`) synchronize after each invocation of `RSig.RandSK`. Given $sk$ and $\rho$, the algorithm `RSig.RandSK` uses $\rho$ together with a counter $\mathsf{ctr}$ in order to deterministically generate a randomness $\rho'$, e.g., by using the function $\mathsf{E}$ on input $(\rho, \mathsf{ctr})$. Then, it computes $sk' = sk + \rho'$ and outputs the rerandomized secret key $sk'$ only after verifying that it has the correct distribution. If this is not the case, it increases $\mathsf{ctr}$ by 1 and repeats this process. The algorithm `RSig.RandPK` needs to receive the corresponding $\mathsf{ctr}$ from `RSig.RandPK` in order to generate the rerandomized public key related to $sk'$. Note that if $sk$ is Gaussian distributed, then we even obtain a scheme with rerandomizable public and secret keys as defined in [FKM+16]. While this method is practical and may be applicable in the construction of sanitizable signatures proposed in [FKM+16], it cannot be used in the setting of hot/cold wallets due to the fact that in each signing process `RSig.RandPK` must obtain the correct $\mathsf{ctr}$ that were used to generate $sk'$. This synchronization requirement undermines the main concept of hot/cold wallets, namely the fact that hot and cold wallets do not communicate with each other (except when they are being initialized).

## 5 Practical Instantiation

In this section we present an efficiency analysis of the wallet scheme introduced in Section 3. To this end, we instantiate the signature scheme presented in Section 4 with a concrete lattice-based signature scheme. The most recent Fiat-Shamir constructions of lattice-based signatures are Dilithium [DKL+18] and qTESLA [ABB+20]. We consider the latter scheme, since the hard lattice problem underlying its key generation algorithm uses Gaussian distributed secrets. This is essential for rerandomizing the secret key in our setting, and hence is sufficient for our scheme with honestly rerandomizable public keys described in Figure 7. On the other hand, Dilithium's key generation uses uniformly distributed secrets for the underlying lattice problem, instead of Gaussian distributed secrets, which is not suitable in our wallet setting (see Section 4.3). Employing the Gaussian distribution in the key generation algorithm of Dilithium instead, requires to adjust the security analysis of Dilithium and to choose new parameters. The resulting scheme would be similar to qTESLA, with slight differences in how signatures are compressed. We choose not to modify Dilithium's original design but stick to qTESLA, which does not need any modification for our setting and is well-studied in comparison to a modified version of Dilithium.

### 5.1 An Instantiation with qTESLA

In this section we show how the signature scheme with honestly rerandomizable public keys introduced in Section 4 can be instantiated with qTESLA. We note that the parameters of qTESLA were selected according to the security reduction from the RLWE problem. This approach has two different aspects: On the one hand, it guarantees that qTESLA has the security level as long as the underlying RLWE instance is hard enough. On the other hand, this approach affects the performance and sizes of keys and signatures, because larger parameters are required to achieve the desired security level. The main goal of our choice is to demonstrate that our wallet scheme can be instantiated with state-of-the-art lattice-based signature schemes without taking into account any of the two different aspects mentioned above.

The design of our scheme is based on lattices over modules. In order to employ qTESLA in our construction we set $k_2 = 1$ to obtain a variant based on lattices over ideals, and security based on the hardness of RLWE. The (master) secret key includes polynomials $s, e_1, \ldots, e_{k_1}$ sampled from $D_{\mathbb{Z}^n,\sigma}$. The polynomial $s$ is bounded by $S/2$ using the function $\mathsf{Max}_j$ defined in Section 2.5, while $e_1, \ldots, e_{k_1}$ are each bounded by $E/2$ using $\mathsf{Max}_j$. In qTESLA the bounds are $S$ and $E$, respectively. However, our wallet scheme uses the master key pair only for rerandomization, and signatures are generated using honestly rerandomized key pairs, which already satisfy the bounds $S$ and $E$. Therefore, we can use exactly the same parameters proposed for qTESLA in [ABB+20, Table 4]).

Note that in comparison to the generic signature scheme shown in Figure 3, Section 2.5, the signature scheme qTESLA [ABB+20] compresses signatures by employing the technique of [BG14]. In this technique the signer proves knowledge of only the secret polynomial $s$ rather than $s$ and $e_1, \dots, e_{k_1}$. Therefore, signatures are of the form $(z_1, c) \in R_Y \times \mathbb{T}^n_\kappa$ rather than $(z_1, \mathbf{z}_2, c) \in R_Y \times R_Y^{k_1} \times \mathbb{T}^n_\kappa$. This approach does not affect the EUF-CMA-HRK security of the signature scheme with honestly rerandomizable public keys. That is, the reduction given in Figure 8 remains the same. Only simulating the signing oracle (cf. Figure 9) requires to include an additional check to ensure the correctness of simulated signatures. More concretely, after step 9 of algorithm SimR (see Figure 9) we add the last **for** loop of qTESLA's signature generation algorithm [ABB+20, Algorithm 4]. However, we have in our setting

$$w_i = a_i z_1 - b'_i c - r_i c \; (\mathsf{mod}^{\pm} q) \quad \text{for all} \quad i = 1, \dots, k_1,$$

where $a_i$, $b'_i$, and $r_i$ are the entries of the public vector $\mathbf{a}$ (replaced by the matrix $\mathbf{A}$, since $k_2 = 1$), rerandomized public key $\mathbf{b}'$, and the vector $\mathbf{r}$, respectively.

## 5.2 Deploying PQ Wallets over Blockchains

In this section we give an overview of the transaction throughput that can be achieved in a cryptocurrency system using our signature scheme instantiated with qTESLA.

A simple transaction in most cryptocurrency networks transfers coins from one party to another. Such transactions must usually include the public key $pk$ and the signature $\sigma$ of the sender such that the validity of the transaction can be verified. In order to give an estimated transaction throughput, we use the raw transaction size of a regular Bitcoin transaction (i.e., without the size of $pk$ and $\sigma$) and then add the size of $pk$ and $\sigma$ of our scheme to it. The raw transaction size of a Bitcoin is roughly 100 Bytes (B) [Bit19]. Hence, when instantiating our wallet scheme with qTESLA, we can take the corresponding signature size (2,592 B) and public key size (14,880 B) [ABB+20, Table 4] for a post-quantum security level of 95 bits[7] and add those to the rough raw transaction size of 100 B. The size of a transaction would then result in $100 \text{ B} + 14,880 \text{ B} + 2,592 \text{ B} \approx 17.5$ KB. We note that it is possible for a party to send coins to multiple receivers in a single transaction which would essentially allow for transactions to be aggregated and increase efficiency.

Many cryptocurrencies (including Bitcoin and Ethereum) currently use the classical signature scheme ECDSA. For the sake of drawing a comparison, note that the size of the ECDSA public key and signature in Bitcoin is approximately 65 B and 73 B [ECD19], respectively, which results in more compact transactions (minimum size of a transaction being $100B + 65B + 73B \approx 240B$), and hence higher transaction throughput.

Naturally, there are various ways to improve the transaction throughput such as increasing block size and the rate at which blocks are produced. For example, in a Bitcoin-like currency new blocks are created roughly every 10 minutes, which tremendously limits the throughput and scalability of the network. In contrast, one

[7] Note that qTESLA proposes only two parameter sets, chosen with respect to a conservative cost model: qTESLA-p-I with 95 bits of post-quantum security and qTESLA-p-III with 160 bits of post-quantum security [ABB+20, Section 4.3].

can consider a system with a block rate of a few seconds, say 15-20 seconds (e.g., this is the case for the Ethereum blockchain). This significantly increases transaction throughput, and hence compensates for larger sizes of $pk$ and $\sigma$. Yet these solutions are ad-hoc, while a more interesting direction for future work is to design further efficient post-quantum secure signature schemes with rerandomizable keys.

## References

ABB+20. Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Juliane Krämer, Patrick Longa, and Jefferson E. Ricardini. The lattice-based digital signature scheme qTESLA. In *Applied Cryptography and Network Security - 18th International Conference, ACNS 2020*, Lecture Notes in Computer Science, 2020. 5, 6, 13, 27, 29, 30

ABL+18. Divesh Aggarwal, Gavin Brennen, Troy Lee, Miklos Santha, and Marco Tomamichel. Quantum attacks on bitcoin, and how to protect against them. *Ledger*, 3, 2018. 2

AHU19. Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, 2019*, volume 11693 of *Lecture Notes in Computer Science*, pages 269–295. Springer, 2019. 8

BDF+11. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security*, volume 7073, pages 41–69. Springer, 2011. 8

BF11. Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Public Key Cryptography - PKC 2011*, pages 1–16. Springer, 2011. 12

BG14. Shi Bai and Steven D Galbraith. An improved compression technique for signatures based on learning with errors. In *Cryptographers' Track at the RSA Conference*, pages 28–47. Springer, 2014. 6, 13, 29

BIP17. Bitcoin bip32 specification. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, Feb. 24 2017. Accessed: 2020-09-15. 2, 5

Bit. Bitcoin post-quantum. https://bitcoinpq.org/. 2, 7

Bit18. BitcoinExchangeGuide. CipherTrace Releases Report Exposing Close to $1 Billion Stolen in Crypto Hacks During 2018. https://bitcoinexchangeguide.com/ciphertrace-releases-report-exposing-close-to-1-billion-stolen-in_-crypto-hacks-during-2018/, 2018. 2

Bit19. Bitcoin wiki transaction format. https://en.bitcoin.it/wiki/Transaction, Dec. 2019. Accessed: 2020-05-04. 30

Blo18. Bloomberg. How to Steal $500 Million in Cryptocurrency. http://fortune.com/2018/01/31/coincheck-hack-how/, 2018. 2

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, 1993*, pages 62–73. ACM, 1993. 8

CGK+19. Alexandru Cojocaru, Juan A. Garay, Aggelos Kiayias, Fang Song, and Petros Wallden. The bitcoin backbone protocol against quantum adversaries. Cryptology ePrint Archive, Report 2019/1150, 2019. https://eprint.iacr.org/2019/1150. 2

DDLL13. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In *Advances in Cryptology–CRYPTO 2013*, pages 40–56. Springer, 2013. 6

DFL19. Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In *ACM SIGSAC Conference on Computer and Communications Security - CCS 2019*, pages 651–668. ACM, 2019. 2, 3, 4, 6, 11, 13, 15, 17, 18, 19, 21

DFMS19. Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11693, pages 356–383. Springer, 2019. 6, 13

DKL+18. Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *Transactions on Cryptographic Hardware and Embedded Systems - TCHES 2018*, (1):238–268, 2018. 6, 13, 27, 28, 29

ECD19. Bitcoin wiki: Elliptic curve digital signature algorithm, Nov. 2019. https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm. 30

eth15. ethereum.org. Ethereum. https://ethereum.org/, 2015. 1

EZS+19. Muhammed F Esgin, Raymond K Zhao, Ron Steinfeld, Joseph K Liu, and Dongxi Liu. Matrict: Efficient, scalable and post-quantum blockchain confidential transactions protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 567–584, 2019. 1, 2, 6

FKM+16. Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *Public Key Cryptography - PKC 2016*, pages 301–330. Springer, 2016. 4, 6, 9, 11, 18, 28

FTS+19. Chun-I Fan, Yi-Fan Tseng, Hui-Po Su, Ruei-Hau Hsu, and Hiroaki Kikuchi. Secure hierarchical bitcoin wallet scheme against privilege escalation attacks. *International Journal of Information Security*, pages 1–11, 2019. 2, 6

GKPV10. Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *Innovations in Computer Science - ICS 2010*, pages 230–240. Tsinghua University Press, 2010. 28

GS15. Gus Gutoski and Douglas Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015*, volume 8975, pages 497–504. Springer, 2015. 2, 5

HMW18. Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018. 2

KLS18. Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In *Advances in Cryptology–EUROCRYPT 2018*, pages 552–586. Springer, 2018. 6, 13

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010*, pages 1–23. Springer, 2010. 13

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015. 13, 25

Lyu09. Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Advances in Cryptology–ASIACRYPT 2009*, pages 598–616. Springer, 2009. 6

LZ19. Qipeng Liu and Mark Zhandry. Revisiting post-quantum Fiat-Shamir. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11693 of *Lecture Notes in Computer Science*, pages 326–355. Springer, 2019. 6, 13

MP12. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012*, pages 700–718. Springer, 2012. 13

MR04. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *Symposium on Foundations of Computer Science (FOCS 2004)*, pages 372–381. IEEE Computer Society, 2004. 12

MSM+15. Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. On the security of the schnorr signature scheme and DSA against related-key attacks. In *Information Security and Cryptology - ICISC 2015*, volume 9558, pages 20–35. Springer, 2015. 17, 23, 24

Nak09. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. http://bitcoin.org/bitcoin.pdf. 1

NC11. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011. 7

Noe15. Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. http://eprint.iacr.org/2015/1098. 1, 7

QRL. Quantum resistant ledger (qrl). https://github.com/theQRL/Whitepaper/blob/master/QRL_whitepaper.pdf. 2, 7

Sch91. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991. 4

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society, 1994. 2

Ske18. Rhys Skellern. Cryptocurrency Hacks: More Than $2b USD lost between 2011-2018. https://medium.com/ecomi/cryptocurrency-hacks-more-than-2b-usd-lost-between-2011-2018_-67054b342219, 2018. 2

TVR16. Mathieu Turuani, Thomas Voegtlin, and Michael Rusinowitch. Automated verification of electrum wallet. In *International Conference on Financial Cryptography and Data Security*, pages 27–42. Springer, 2016. 2, 6

Unr15. Dominique Unruh. Revocable quantum timed-release encryption. *J. ACM*, 62(6):49:1–49:76, 2015. 8

Unr17. Dominique Unruh. Post-quantum security of Fiat-Shamir. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, volume 10624, pages 65–95. Springer, 2017. 6, 13

Zha12a. Mark Zhandry. How to construct quantum random functions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012*, pages 679–687. IEEE Computer Society, 2012. 8, 9

Zha12b. Mark Zhandry. Secure identity-based encryption in the quantum random oracle model. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*, volume 7417, pages 758–775. Springer, 2012. 22

Zha19. Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*, volume 11693, pages 239–268. Springer, 2019. 19

# D. Round Efficient Byzantine Agreement from VDFs

This chapter corresponds to our work on Graded PKI and Byzantine Agreement. Full version of this work can be found here: [41].

[41] P. Das, L. Eckey, S. Faust, J. Loss, and M. Maitra. *Round Efficient Byzantine Agreement from VDFs.* Cryptology ePrint Archive, Paper 2022/823. `https://eprint.iacr.org/2022/823`. 2022.

# Round Efficient Byzantine Agreement from VDFs

Poulami Das[1], Lisa Eckey[1], Sebastian Faust[1], Julian Loss[2], and Monosij Maitra[1]

[1] Technical University of Darmstadt, Germany
{firstname.lastname}@tu-darmstadt.de
[2] CISPA Helmoltz Center for Information Security, Germany
loss@cispa.de

**Abstract.** Byzantine agreement (BA) is a fundamental primitive in distributed systems and has received huge interest as an important building block for blockchain systems. Classical byzantine agreement considers a setting where $n$ parties with fixed, known identities want to agree on an output in the presence of an adversary. Motivated by blockchain systems, the assumption of fixed identities is weakened by using a *resource-based model.* In such models, parties do not have fixed known identities but instead have to invest some expensive resources to participate in the protocol. Prominent examples for such resources are computation (measured by, e.g., proofs-of-work) or money (measured by proofs-of-stake). Unlike in the classical setting where BA without trusted setup (e.g., a PKI or an unpredictable beacon) is impossible for $t \geq n/3$ corruptions, in such resource-based models, BA can be constructed for the optimal threshold of $t < n/2$. In this work, we investigate BA without a PKI in the model where parties have restricted computational resources. Concretely, we consider sequential computation modeled via computing a verifiable delay function (VDF) and establish the following results:

1. *Positive result:* We present the *first* protocol for BA with expected constant round complexity and termination under adaptive corruption, honest majority and without a PKI. Earlier work achieved round complexity $O(n\kappa^2)$ (CRYPTO'15) or $O(\kappa)$ (PKC'18), where $\kappa$ is the security parameter.
2. *Negative result:* We give the *first* lower bound on the communication complexity of BA in a model where parties have restricted computational resources. Concretely, we show that a multicast complexity of $\mathrm{O}(\sqrt{n})$ is necessary even if the parties have access to a VDF-oracle.

**Keywords:** Byzantine Agreement, Proof of Work, Verifiable Delay Functions, Impossibility

## 1 Introduction

In the Byzantine agreement (BA) problem, a set of $n$ parties jointly run a distributed protocol to agree on a common output in the presence of some minority of $t$ malicious parties. BA is a well-studied and fundamental problem in distributed computing and has recently garnered renewed interest in the context of blockchain protocols [Nak08,CP19,KKKZ19,But13]. Traditionally, most existing protocols for BA assume a setting in which the parties' identities are fixed and known at the beginning of the protocol. In the fixed identity setting, two types of protocols are studied: the first type requires setup, e.g., a public key infrastructure (PKI) or some form of correlated randomness. These protocols typically tolerate the (optimal) corruption threshold of $t < n/2$. The second type does not require such assumptions but can tolerate only $t < n/3$ corruptions.

More recently, a third type of protocol has emerged [AD15,KMS14] that gives up on the fundamental assumption that parties know each other's identities at the beginning of the protocol. Moreover, these protocols do not require setup in the classical sense, yet still achieve the optimal corruption tolerance of $t < n/2$. Note that if identities are not fixed then without further measures, every party could pose as many parties and easily obtain a dishonest majority; this is commonly referred to as a *sybil attack* [Dou02]. To avoid such attacks, parties must instead invest some expensive resources, such as computation or money, to participate in this type of protocol. A prominent example is the Proof-of-Work model (PoW) initially introduced by Bitcoin, where parties have limited access to a computational resource which they are forced to continuously expend in order to participate in the protocol.

In this work, we refine the PoW model by considering the effort it takes to evaluate *verifiable delay functions (VDFs)* [BBBF18,CP19] as the main computational resource. VDFs can be seen as a special type of proof-of-work whose computation cannot be sped up by much. This is in stark contrast to the typical lottery-type proofs-of-work, whose computation can be sped up almost arbitrarily, given sufficient parallel computation resources. We explore, for the first time, the implications of bounding the number of VDF evaluations that an (adaptive) adversary can compute in parallel: 1) We show an expected constant-round BA protocol that tolerates $t < n/2$ corrupted parties and does not rely on a PKI or known identities; 2) we give the first non-trivial communication lower bound by showing that any BA protocol in this setting requires at least $O(\sqrt{n})$ send-to-all steps.

## 1.1 The VDF Model

Our work introduces the VDF model as a refinement of the common PoW model to replace trusted setups and protect against Sybil attacks in permissionless consensus. Similar to the PoW model, we assume that the adversary only controls less than 1/2 of a computational resource to invest in producing *proofs of computation*. In contrast to the PoW model, however, we require a lower bound on the time it takes to create such proofs. This differs from the PoW model, where proofs can be computed almost arbitrarily fast, given sufficient parallel resources. We believe that the VDF model is a realistic alternative for the PoW model. Indeed, there exists various different constructions of VDFs [Wes19,Pie19,FMPS19] that leverage inherently sequential computation, and are used (or envisioned to be used) by blockchain projects for their consensus protocol (albeit not as an anti Sybil countermeasure as in our work). Examples include Chia, which relies on VDFs for leader election [CP19], and ETH 2.0, which plans to leverage VDFs for constructing a random beacon [But13]. To make our model more realistic, we follow Wan et al. [WXDS20], and allow the adversary a small speedup in evaluating the VDF compared to the honest parties.

Let us describe our model with a concrete example. Suppose that the total amount computational power (over all protocol participants) over a fixed time period of length $t$ is 1000 VDF evaluations. Then, we demand that the total number of proofs produced by the adversary be at most 500 in the same time span. This is similar to the case of PoW model, where it is assumed that the majority of computational power belongs to honest parties. As mentioned above, we additionally give the adversary a small *speedup*, meaning that it can compute proofs a little bit faster than the honest parties.

**The $\mathsf{VDF}_\delta$ Oracle.** We now explain our formalization of the VDF model in some more detail. At the center of our model, we introduce the oracle $\mathsf{VDF}_\delta$, which parties can query on an input $s$ to obtain one evaluation $\phi$ of the VDF after $\delta$ time. Thus, $\delta$ denotes the difficulty parameter, specifying the number of sequential steps to be computed for one VDF evaluation. To make the model more realistic, we allow corrupted parties a *$\varkappa$-speedup* (where $\varkappa \geq 1$), meaning that they can obtain an output from $\mathsf{VDF}_\delta$ after $\delta/\varkappa$ time. An adversary $\mathcal{A}$ in our model controls some $q$ number of parties, where each party has $\varkappa$-speed-up. For some $i > 1$, let us discuss how many proofs an adversary is able to compute within time $t$, where $(i-1)\cdot\delta < t < i\cdot\delta$. We can express $t$ more concretely as $t = (i-1)\cdot\delta + r\cdot\delta$, where $r \in (0,1)$. With a corruption budget of $q$ parties, $\mathcal{A}$ can invoke the $\mathsf{VDF}_\delta$ oracle $q$ times concurrently (once per party). Since each proof is obtained after time $\frac{\delta}{\varkappa}$, at time $(i-1)\cdot\delta$, each party computes $(i-1)\cdot\varkappa$ proofs. In the remaining $r\cdot\delta$ time, each party can compute (*at most*) $\lfloor\frac{r.\delta}{\delta/\varkappa}\rfloor = \lfloor r\cdot\varkappa\rfloor$ proofs. Thus, in total $\mathcal{A}$ obtains at most $((i-1)\cdot\varkappa + \lfloor r\cdot\varkappa\rfloor)\cdot q$ proofs at time $t$. Figure 1 illustrates our model with a small example. We refer to this property of the $\mathsf{VDF}_\delta$ oracle as its *sequentiality*

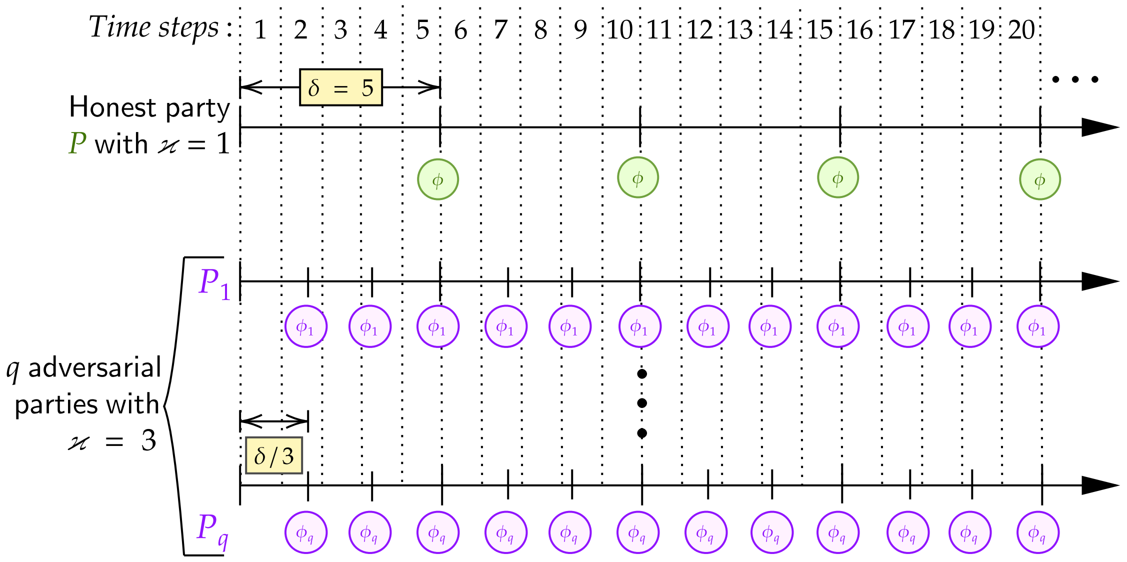

**Fig. 1.** Consider $i = 3, \delta = 5$. We have $i\cdot\delta = 15, (i-1)\cdot\delta = 10$. Say an adversary $\mathcal{A}$ controls $q$ parties $\{P_1\}_{i\in[q]}$ with a speed-up $\varkappa = 3$ compared to an honest party with $\varkappa = 1$. Consider two time steps: $t = 11$ and $t = 14$. In both cases, each $P_i$ can compute $(i-1)\varkappa = 6$ proofs in time $10 < i\cdot\delta$. For the remaining time $r\cdot\delta = 1$ (for $t = 11$) and $r\cdot\delta = 4$ (for $t = 14$), no extra proofs can be computed in the first case, whereas $\lfloor\frac{4}{5/3}\rfloor = 2$ extra proofs can be computed in the second case. Thus, $\mathcal{A}$ can compute in total $6q$ and $8q$ proofs for $t = 11$ and $t = 14$ respectively.

and give a formal definition in Section 2.

Such oracle abstraction of the VDF computation allows us to give a cleaner and more modular analysis of our main protocols. In support of our modelling approach, we conjecture that the $\mathsf{VDF}_\delta$ oracle can be instantiated in the standard model (full version, Appendix A, Lemma 20) and we prove that it can be instantiated in the strong algebraic group model for constructions of Wesolowski [Wes19] and Pietrzak [Pie19] (full version, Appendix A, Lemma 23).

## 1.2 Byzantine Agreement in the VDF Model.

As our main technical contribution, we show how to obtain an expected constant-round Byzantine agreement protocol without any additional trusted setup in the VDF+random oracle model. This is of particular significance, given that we can also instantiate the VDF model without any trusted setup, using Wesolowski's construction. Given an upper bound $n$ on the number of parties, our protocol tolerates $q$ adaptively corrupted parties with $\varkappa$-speed-up, where $q(\lfloor\varkappa\rfloor + 1) < n$. In particular, our protocols tolerate up to $\frac{n}{2}$ corruptions when $\varkappa = 1$. Our protocol combines several ideas from previous works in a novel way, as we now describe.

– *Step 1: Establishing a Graded Public key Infratsructure (GPKI).* We adopt the idea of Andrychowicz and Dziembowski [AD15] and start by setting up a precursor to a full PKI called *graded PKI* (GPKI) among the parties. Roughly speaking, a GPKI differs from a full PKI in that the keys of the parties are additionally associated with grades. These grades can differ between parties, but not by too much. As a first step, to reduce the round complexity of their protocol to $O(1)$ from $O(n\kappa^2)$ (where $\kappa$ is a security parameter), we make two modifications: 1) We set up a *much weaker* GPKI with only two possible grades, whereas [AD15] sets up $n$ possible grades. 2) We borrow a technique from Katz et al. [KMS14][3] and rely on VDFs to make the round complexity of our protocol independent of $\kappa$. As a second step, the difficulty parameter $\delta$ of the $\mathsf{VDF}_\delta$ must be adjusted to tolerate adversarial speedup in the $\mathsf{VDF}_\delta$ model.

– *Step 2: Graded Consenus from GPKI.* One of the ingredients needed in our BA protocol is a graded consensus protocol. Graded consensus is similar to a BA, with the difference that at the end of the protocol, every party outputs a value with some grade, where the grades of different parties must satisfy some consistency properties. We build upon the graded consensus protocol of Micali and Vaikuntanathan [MV17] where we modify their protocol such that it requires only a GPKI instead of a full PKI.

– *Step 3: Leader election protocol from VDFs.* It is well known that expected constant round protocols are inherently randomized, e.g., by electing a random leader in every protocol iteration. However, electing a random leader efficiently is challenging without prior setup. We overcome this issue by presenting a novel leader election protocol that leverages the oracle $\mathsf{VDF}_\delta$ to efficiently elect a random leader that all honest parties agree on with high probability. Our protocol is inspired by

[3] The construction in [KMS14] uses only a proof of sequential as opposed to a VDF.

leader elections based on verifiable random functions [ACD+19,ADD+19] and implements a leader election lottery with unique tickets. This makes the tickets hard to bias from the perspective of the adversary.

– *Step 4: BA protocol.* Finally, we combine all of the above components to adopt the expected constant round protocol of Katz and Koo [KK06a] to our setting. We informally state our overall result in the following theorem.

**Theorem 1 (Informal).** *Let $n$ denote an upper bound on the number of parties. Then, there exists an expected-constant round BA protocol in the VDF model that is secure against any adversary $\mathcal{A}$ that controls at most $q$ parties with $\varkappa$-speed-up, where $q \cdot (\lfloor \varkappa \rfloor + 1) < n$.*

### 1.3 A Lower Bound on Communication Complexity for BA

As a third contribution, we give the *first* lower bound for communication complexity of BA, assuming parties have bounded computational resources. Concretely, we show that in the VDF model without additional trusted setup, no protocol can realize BA with overwhelming probability by multicasting fewer than $O(\sqrt{n})$ messages in the presence of adaptive corruptions. In the multicast model of communication, honest parties are restricted to sending messages to *all parties at once*, whereas the adversary can send to only a subset of the parties. This models a setting in which parties communicate via a *gossip network* [GKPS18,AD15]. The cost of running the same protocol from the multicast model in the bilateral channel model [DS83] would be $O(n^{3/2})$. We remark, however, that the multicast restriction is crucially used in the lower bound, and thus, a better communication complexity might be possible in the bilateral channels model. Our lower bound builds on ideas of Abraham et al. [ACD+19] who show a bound for Byzantine broadcast in the multicast model without setup.

Our bound has to overcome several technical challenges that arise when parties have limited computational resources. The adversary in our attack has to carry out a simulation of the protocol in its head (this is a standard technique used in lower bounds), which may require to query the VDF oracle. At the same time, the adversary must also participate in an actual execution of the protocol it is attacking, which, of course, also results in queries to the oracle. Thus, the key difficulty in our lower bound is to carefully balance the adversary's limited budget of queries to VDF over the two executions of the protocol (real and simulated).

Although our lower bound is relatively weak compared to most existing lower bounds in this area (which are quadratic, or of the form $O(n)$ in the multicast model, respectively), we argue that it is still meaningful. Namely, protocols that require significantly above $O(\kappa)$ multicasts are deemed impractical for large-scale settings with millions or even billions of users. This means that our bound essentially rules out efficient solutions in the VDF model unless further setup is assumed among the parties. Second, we point out that our lower bound actually holds in the relatively weak $\mathsf{VDF}$-*model* and can likely be carried over to a less restrictive model (e.g., to the PoW model used by Bitcoin). It also leaves room for a tighter bound in such more general models.

### 1.4 Implications of Our Results and Related Work

Our model can be instantiated using Wesolowski's VDF, which does not require trusted setup. Thus, our results show, for the first time, how to perform expected-constant round BA in a permissionless model with a simple honest majority and no trusted setup (beyond a random oracle). This has many important implications. For example, one could use our protocol to efficiently agree on a random string in a permissionless setting. This string could be used as a genesis block or as a uniform common reference string to perform an MPC protocol. Our results also significantly improves over the result of Andrychowicz and Dziembowski, who presented a protocol that achieves essentially the same, but requires $O(n \cdot \kappa^2)$ rounds to do so [AD15] . We also improve over a similar (slightly more efficient) version of this idea shown by Katz et al. [KMS14]. Another closely related work is that of Garay et al. [GKLP18] who show how to bootstrap the classical Nakamoto consensus protocol [Nak08,GKL15,PSs17] in the PoW model without trusted setup. However, it requires $O(\kappa)$ rounds, and therefore also does not constitute an expected constant round protocol.

**Further Related Work.** There is a large body of research on BA and related problems (sometimes colloquially referred to as "consensus"), and we focus here on the most closely related works. We have already mentioned the works of [AD15,KMS14] and [GKLP18] who achieve BA in the PoW model without setup and run in $O(n\kappa^2)$, $O(n)$, and $O(\kappa)$ rounds, respectively. It should also be noted that we require stronger assumptions (namely a VDF and the random oracle model (ROM)) than the protocols in [AD15,GKLP18,GKO$^+$20], who require only the ROM that can be queried at a bounded rate by any party, but (possibly) weaker assumptions than required in the work of Katz et al. [KMS14], who also require some form of an unpredictable beacon in their protocol. The more recent work of Aggarwal et al. [AMSZ19] presents a setup-free solution in the PoW model that also runs in expected $O(1)$ rounds, but assumes a static adversary (while we consider the much stronger adversarial model of adaptive corruptions). Another related work that focuses on the PoW model is by Garay et al. [GKO$^+$20]. They show how to achieve UC-secure BA and multi-party computation (MPC) protocols in the PoW model. Similar to [AD15], their BA takes $O(n\kappa^2)$ rounds.

Although the above-mentioned prior works [AD15,KMS14,GKLP18] achieve BA without a PKI, their techniques are not what we need to achieve $O(1)$-round BA. In fact, we notice that achieving $O(1)$-round BA protocols requires very particular techniques that have been studied [Rab83,FM88,KK06a,MV17,Mic17,ADD$^+$19,ACD$^+$19] in the classical setting for many years and this round reduction is very challenging to achieve.

## 2 Definitions and Model

Notation. We write for the set of positive real numbers greater than some number $n \in \mathbb{R}$ as $\mathbb{R}_{>n}$, the set of natural numbers as $\mathbb{N}$. Throughout this paper, $\kappa$ will denote the security parameter. We write $x \leftarrow S$ to denote that $x$ is sampled uniformly at random from set $S$. Similarly, we write $y \leftarrow \mathcal{A}(x)$ to denote that the output of a probabilistic algorithm $\mathcal{A}$ on input $x$ is $y$.

### 2.1 Model

We consider a setting in which $n$ parties $P_1, \ldots, P_n$ engage in a distributed protocol $\Pi$. We assume that the exact number of parties is unknown but that there is some known upper bound $n$. Additionally, we assume that the majority of the parties follow the protocol honestly, and the remainder of the parties can be corrupted by the adversary (whose capabilities we will describe in the following). We emphasize that no public key infrastructure (PKI) needs to be shared among the parties, i.e., the parties do not initially know each other's public keys. Moreover, the parties are assumed to have synchronized clocks.

Communication Model. Inspired by [ACD+19], we consider a communication setting where parties *multicast* messages to other parties. In other words, a party may send the same message to *everyone in the network* (as opposed to possibly sending $n$ messages separately to $n$ parties). We say that a protocol has multicast complexity $\theta$, if the total number of multicasts (i.e., by all honest parties) in the protocol is at most $\theta$. This implies that in the classical communication model with bilateral channels, the (same) protocol requires sending $n\theta$ messages. We consider the synchronous model, where any message sent by an honest party over the multicast channel is received by all honest parties after at most time $\Delta$. As is usual in this model, any of our protocols proceeds in rounds of duration $\Delta$, where round $r$ of the protocol starts at time $(r-1) \cdot \Delta$ (assuming parties run the protocol at time 0).

Random Oracle Model (ROM). We model hash functions as random oracles [BR93]. The code of a hash function $\mathsf{H}$ is defined as follows. On input $x$ from the domain of the hash function, $\mathsf{H}$ checks whether $\mathsf{H}(x)$ has been previously defined. If so, it returns $\mathsf{H}(x)$. Else, it sets $\mathsf{H}(x)$ to a uniformly random element from the range of $\mathsf{H}$ and then returns $\mathsf{H}(x)$.

Model of Computation. We consider the running time of parties in some fixed but unspecified model of computation, e.g., the Turing machine model or the arithmetic circuit model. This allows us to both simplify and generalize our results, as we discuss below. In addition to $\mathsf{H}$, parties in our model have access to oracle $\mathsf{VDF}_\delta$, which is used to restrict parties to performing certain computations *sequentially.* $\mathsf{VDF}_\delta$ has the following properties.

– **Evaluation:** On input $(\mathsf{Eval}, S, \varkappa)$, $\varkappa \geq 1$ from party $P$ at time $t$, $\mathsf{VDF}_\delta$ generates a *proof of computation* $\phi$ according to some (fixed) output distribution $\mathcal{D}$ and returns $\phi$ at time $t + \frac{\delta}{\varkappa}$. It ignores any further inputs of the form $(\mathsf{Eval}, S, \varkappa)$ from

$P$ before time $t + \frac{\delta}{\varkappa}$. We call the maximum value of $\varkappa$ over all queries of $P$ to $\mathsf{VDF}_\delta$ the *speedup of* $P$.

– **Verification:** On input $(\mathsf{Verify}, \phi, S)$, $\mathsf{VDF}_\delta$ (immediately) returns 1, if $\phi$ is a valid proof with respect to input $S$ and 0 otherwise.

Intuitively, $\mathsf{VDF}_\delta$ corresponds to a verifiable delay function (VDF) that takes $\delta$ time to evaluate. Here, $\delta$ can be set conservatively s.t. any honest party is able to compute an evaluation of the VDF within this timeframe. However, since not all parties run at the same speed, we allow for some speedup when evaluating the VDF. In our protocols, honest parties will always call $\mathsf{VDF}$ with $\varkappa = 1$. The adversary, on the other hand, may set $\varkappa$ to some value above 1. We summarize this in the following definition.

**Definition 1** (**$(q, t_p, \varkappa)$-Algorithm**). *$\mathcal{A}$ is a $(q, t_p, \varkappa)$-*adversary *if it has full control over $q$ parties with speedup at most $\varkappa$ and runs for at most $t_p$ steps.*

**Sequentiality of $\mathsf{VDF}_\delta$.** For a $(q, t_p, \varkappa)$-adversary $\mathcal{A}$, we now define a natural notion of *sequentiality*. Intuitively, it should be impossible for $\mathcal{A}$ to compute more evaluations of the VDF than its allotted budget of computation over a certain period of time. Here, $\mathcal{A}$'s budget spans over all $q$ parties it controls and includes speedups quantified by $\varkappa$. Therefore, we expect $\mathcal{A}$ to be unable to compute much more that $(i-1) \cdot \varkappa \cdot q$ proofs in less than $i \cdot \delta$ time.

In a bit more detail, for any $i \geq 1$, when $\mathcal{A}$ has less than $i \cdot \delta$ time, with an $\varkappa$-speedup it can compute exactly $\lfloor (i-1) \cdot \varkappa \rfloor \cdot q$ proofs at time $(i-1) \cdot \delta$. The remaining time left is less than $\delta$ $(i \cdot \delta - (i-1) \cdot \delta)$. $\mathcal{A}$ can compute only a few more proofs in this time. In particular, denoting this remaining fractional time as $r \cdot \delta$, where $r \in (0, 1)$, $\mathcal{A}$ can only compute at most $\lfloor r \cdot \varkappa \rfloor \cdot q$ proofs. Thus $\mathcal{A}$ computes in total $\lfloor (i-1+r) \cdot \varkappa \rfloor \cdot q$ proofs at time $(i-1+r) \cdot \delta$.

Of course, such a notion only makes sense if $\mathcal{A}$ can not start computing before a certain time $T$. Hence, we begin by defining what it means for a random variable to be unpredictable in $\mathcal{A}$'s view. For convenience, we will say that $\mathbf{S}$ is *determined at time $T$* if its value is fixed in the view of at least one honest party at time $T$.

**Definition 2** (**$(k, T, \epsilon)$-Unpredictable**). *Let $\mathbf{S} = (S_1, \ldots, S_k)$ be a vector of $k$ random variables whose outcome is determined at time $T$. We say that* $\mathbf{S}$ *is $(k, T, \epsilon)$-*unpredictable*, if for all $(q, t_p, \varkappa)$-adversaries $\mathcal{A}$, $\Pr_{\hat{S}_1, \ldots, \hat{S}_k \leftarrow \mathcal{A}_T^{\mathsf{H}, \mathsf{VDF}_\delta}}[\exists j \in [k] \colon S_j = \hat{S}_j] \leq \epsilon$, where the probability is over the random coins of $\mathcal{A}$,* $\mathsf{H}$, $\mathsf{VDF}_\delta$ *and the random choice of* $\mathbf{S}$.

We say, that $S'$ *depends on* $S$ if $S' = \mathsf{H}(\cdot||S||\cdot)$, or if $S' = \mathsf{H}(\cdot||S''||\cdot)$ with $S'' = \mathsf{H}(\cdot||S||\cdot)$. We now define the sequentiality property of $\mathsf{VDF}_\delta$ oracle relative to a sequence of inputs $\{S'_1, \ldots, S'_\tau\}$, where each $S'_j$ depends on some $(k, T, \epsilon)$-unpredictable vector $\mathbf{S}$.

**Definition 3** (**$(i, r, \beta)$-Sequentiality**). *Let $\mathcal{A}$ be a $(q, t_p, \varkappa)$-adversary and fix $T > 0$. Let $\mathbf{S}$ be $(k, T, \epsilon)$-unpredictable for some $\epsilon > 0$. For any $i \in \mathbb{N}, r \in (0, 1)$, let $T_{i,r} := T + (i-1+r) \cdot \delta$ and $\tau_{i,r} := \lfloor (i-1+r) \cdot \varkappa \rfloor \cdot q$. Let $\{S'_1, ..., S'_{\tau_{i,r}+1}\}$ be a* *sequence of length $\tau_{i,r}+1$, where each $S'_j$ depends on at least one component of* $\mathbf{S}$. *We say that* $\mathsf{VDF}_\delta$ *is* $(i,r,\beta)$-sequential*, if for all* $i \in \mathbb{N}, r \in (0,1)$ *s.t.* $\mathcal{A}$ *outputs* $\hat{\phi}_1,...,\hat{\phi}_{\tau_{i,r}+1}$ *before or at time* $T_{i,r}$*, we have*

$$\Pr_{\hat{\phi}_1,...,\hat{\phi}_{\tau_{i,r}+1} \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{VDF}_\delta}_{T_{i,r}}}[\forall j \in [\tau_{i,r}+1] : \mathsf{VDF}_\delta(\mathsf{Eval}, S'_j, \varkappa) = \hat{\phi}_j] \leq \epsilon + \beta.$$

*The above probability holds over the random coins of* $\mathcal{A}$*,* $\mathsf{H}$*,* $\mathsf{VDF}_\delta$ *and the random choice of* $\mathbf{S}$.

**Relevance of our $\mathsf{VDF}_\delta$ oracle.** As mentioned earlier, we consider the VDF construction due to Wesolowski [Wes19], which is based on solving the classical RSW time-lock puzzle over class groups of an imaginary quadratic field. It uses a hash function $\mathsf{H}_G \colon \{0,1\}^* \to G$, where $G$ is a class group of an imaginary quadratic field. On input $s \in \{0,1\}^*$, the construction computes $h := \mathsf{H}_G(s)$ and outputs the value $h^{2^\delta}$. Verification can be done using $\frac{\delta}{\log(\delta)}$ group elements and using a storage of $\sqrt{\delta}$ group elements. This concrete instantiation would translate to our model by setting the output distribution $\mathcal{D}$ to the distribution that first draws $s \leftarrow S$, computes $h = \mathsf{H}_G(s)$, and then evaluates the repeated squaring as described above. Note that the adversary $\mathcal{A}$ may guess the outcome of the random variable $\mathcal{D}$ before time $T+\delta$, and we compensate for this in the definition via the factor $\beta$. Moreover, the adversary may guess correctly the outcome $s$ of $S$ and try to compute $\mathsf{VDF}_\delta$ before time $T+\delta$. However, this is only possible with probability $\epsilon$ due to the unpredictability of $S$.

We remark that for concrete constructions of VDFs such as Pietrzak's or Wesolowski's VDF [Pie19,Wes19], a simpler (and less idealized) way of modeling might be to consider adversaries $\mathcal{A}$ in the sequentiality definition from the class of arithmetic circuits of depth at most $\delta$. Indeed, this would not require to model the VDF as an oracle, as one could simply bound the adversary's probability of computing the VDF on unpredictable inputs according to the computational model. Since the focus of this work is on constructions and lower bounds of BA, we choose to model VDFs more abstractly to give a cleaner and more modular analysis of our main protocols. Moreover, as we use the VDF as a anti-Sybil protection, we need some way of quantifying the adversary's resource budget, which is most commonly done via bounding oracle access. To support our modeling approach, we prove in full version, Appendix A, the sequentiality of our $\mathsf{VDF}_\delta$ oracle in the strong algebraic group model [vBS21,KLX20]. Although our proof holds for both constructions from [Pie19,Wes19], we choose to analyze Wesolowski's VDF to obtain our results without additional trusted setup assumptions.

**Protection against Homomorphic Computation of a $\mathsf{VDF}_\delta$.** Some concrete constructions of VDFs (or of the related primitive time-lock encryption) may have an homomorpism, where for any two inputs $s_1$, $s_2$, $\mathsf{VDF}(s_1)\cdot\mathsf{VDF}(s_2) = \mathsf{VDF}(s_1\cdot s_2)$. This may enable an adversary to speed-up multiple evaluations of $\mathsf{VDF}_\delta$ on related inputs. In case of Wesolowski's VDF [Wes19], it already prevents such an homomorphism due to the use of the hash function $\mathsf{H}_G\colon \{0,1\}^* \to G$. Recall that here, first, a group element $h_1 = \mathsf{H}_G(s_1)$ is computed, then output $\phi_1$ is computed as $\phi_1 = {h_1}^{2^\delta}$ in $\delta$ steps. Interestingly, this issue is also inherently prevented in our $\mathsf{VDF}$ model due to the following reason. Recall that in our definition of sequentiality, we require that inputs to the oracle go via a hash function $\mathsf{H}(\cdot)$. Even if the VDF itself exhibits a homomorphism, this effectively prevents any homomorphic evaluation for our concrete application.

Adversary Model. The adversary in our protocol is modelled as a $(q, t_p, \varkappa)$-algorithm $\mathcal{A}$ as defined above. $\mathcal{A}$ can control at most $q$ parties each with a maximum speedup of $\varkappa$, such that $q < \frac{n}{\lfloor\varkappa\rfloor+1}$ holds. In particular, the number of adversarial parties can be at most $< \frac{n}{2}$ (this is the case for $\varkappa = 1$). We consider an adaptive adversary which can corrupt a party at any point during the protocol execution. Once a party has been corrupted, it can arbitrarily deviate from the protocol execution. Furthermore, it can deliver a message over the multicast channel only to a subset of honest parties. In this way, it can send different messages to different subsets of honest parties over the multicast channel. However, the adversary can not drop the messages of honest parties from the channel or delay them for longer than $\Delta$. Our adversary is *rushing*, which means it can observe all the messages that the honest parties send in any round of the protocol, and then choose its own messages for that round adaptively. We notice that we consider the standard notion of an adaptive, rushing adversary, as opposed to the stronger notion of a *strongly rushing* (or strongly adaptive) adversary (see for e.g., [ADD+19,ACD+19,CGZ21]) who can adaptively corrupt parties and then delete messages that they sent in the same round (prior to corruption).

**Lifting the Assumption on Clock Synchronization.** Here, we discuss how to overcome the assumption that parties have synchronized clocks. Since we focus on a setting, where parties do not share any PKI, it is crucial to relax this assumption, so that parties can work with their respective local clocks. Lets say, local clocks of individual parties can differ by at most time interval $\alpha$. When parties are instructed to send some messages in any round, they can always send messages at one particular time of the day, say 10:00 AM, local time. The duration of each round $\Delta$ can be set as $\Delta := 2\alpha$, to make sure that every party received messages sent back from all other parties before the current round ends and the next round begins.

### 2.2 Definitions

In this work, we focus on $n$-party protocols that reach consensus a.k.a. Byzantine agreement when up to $q$ out of $n$ parties are corrupted. All the following definitions consider a synchronous setting of communication.

**Definition 4 (Byzantine Agreement).** *A protocol executed among $n$ parties where each party $P_i$ holds initial input $x_i$ and parties output upon terminating $y_i$ achieves* Byzantine Agreement *if the following properties hold whenever at most $q$ parties are corrupted:*

- ***Consistency:*** *If two honest parties $P_i$, $P_j$ output values $y_i$, $y_j$ respectively, then $y_i = y_j$.*
- ***Validity:*** *If all honest parties $P_i$ have the same input $x_i = x$, then all honest parties output $y_i = x$.*
- ***Termination:*** *All honest parties terminate.*

We also study the sender-centric version of Byzantine Agreement, which is called Byzantine Broadcast.

**Definition 5 (Byzantine Broadcast).** *A protocol executed among $n$ parties, where a designated sender $S$ holds an input $x$ at the beginning of the protocol and parties output $y_i$ upon terminating, achieves* Byzantine Broadcast *if the following holds whenever at most $q$ parties are corrupted:*

- ***Consistency:*** *If two honest parties $P_i$, $P_j$ output values $y_i$, $y_j$ respectively, then $y_i = y_j$.*
- ***Validity:*** *If the sender $S$ is honest, then all honest parties output $y_i = x$.*
- ***Termination:*** *All honest parties terminate.*

Graded byzantine agreement (also known as graded consensus) is a weaker variant of byzantine agreement, while graded broadcast (or gradecast, for short) is a weaker variant of byzantine broadcast. They have been used in various previous works (cf. graded byzantine agreement in [FM00] and [FLL21], gradecast in [MV17] and [KK06b]). Both of them are very useful building blocks for byzantine agreement and byzantine broadcast protocols.

**Definition 6 (Graded Byzantine Agreement).** *A protocol $\Pi$ executed among $n$ parties, where each party holds initial input $x_i$ and parties output upon terminating a value $y_i$ and a grade $\zeta_i$, achieves* Graded Byzantine Agreement *if the following holds whenever at most $q$ parties are corrupted:*

- ***Consistency:*** *If there is an honest party that outputs $y_i$ with grade $\zeta_i = 2$, then every other honest party outputs $y_j = y_i$ with grade $\zeta_j \geq 1$.*
- ***Validity:*** *If all honest parties input $x_i = x$, then every honest party outputs $y_i = x$ with grade $\zeta_i = 2$.*
- ***Termination:*** *All honest parties terminate.*

**Definition 7 (Graded Broadcast).** *A protocol $\Pi$ among $n$ parties, where a designated sender $S$ holds an input $x$ at the beginning of the protocol and parties output a value $y_i$ and a grade $\zeta_i$ upon terminating, achieves* Graded Broadcast *if the following holds whenever at most $q$ parties are corrupted.*

- ***Consistency:*** *If there is an honest party that outputs $y_i$ with grade $\zeta_i = 2$, then every other honest party outputs $y_j = y_i$ with grade $\zeta_j \geq 1$.*
- ***Validity:*** *If $S$ is honest, then every honest party $P_i$ outputs $y_i = x$ with grade $\zeta_i = 2$.*
- ***Termination:*** *All honest parties terminate.*

For the following definition, let $\mathsf{KeySet}$ be the local key set of a party $P$, which keeps tuples of the form $(\mathsf{pk}_j, \zeta_j)$. $\mathsf{pk}_j$ is the public key of some party $P_j$ and $\zeta_j \in \{1, 2\}$ is the grade assigned to this key.

**Definition 8 (Graded Public Key Infrastructure).** *A protocol $\Pi$ among $n$ parties, where parties output a set* $\mathsf{KeySet}$ *upon terminating achieves* Graded PKI *(tolerating $q$ corrupted parties) if the following holds for any two honest parties $P_i, P_j$.*

- ***Graded Validity:*** *$P_i$'s public key $\mathsf{pk}$ is assigned grade $2$ by $P_j$ (i.e., $(\mathsf{pk}_i, 2) \in \mathsf{KeySet}_j$).*
- ***Graded Consistency:*** *If $P_i$ assigns grade $2$ to a public key $\mathsf{pk}$, i.e., $(\mathsf{pk}, 2) \in \mathsf{KeySet}_i$), then $P_j$ assigns at least grade $1$ to the same key, i.e., $(\mathsf{pk}, \zeta) \in \mathsf{KeySet}_j$, where $\zeta \geq 1$.*
- ***Bounded Number of Identities:*** *Let $\mathsf{N} := |\bigcup_{i\in[n]_{P_i\in\mathcal{H}}} \mathsf{KeySet}_i|$ be the total number of keys in the combined key sets of all honest parties $\mathcal{H}$, then the total number of keys that belong to corrupted parties is $< \frac{1}{2}\mathsf{N}$.*
- ***Termintaion:*** *All honest parties terminate.*

We remark that we require the properties in the above definitions to hold with probability $1 - 2^{-\kappa}$ (where $\kappa$ is the security parameter), but we omit this detail from the definition for ease of presentation.

Signature Schemes. We use the following standard definition of digital signature schemes.

**Definition 9 (Signature Schemes).** *A* digital signature scheme *is a triple of algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{VrfySig})$ *with the following properties. On input the security parameter $\kappa$, the randomized* key generation algorithm $\mathsf{KeyGen}$ *outputs a key pair* $(\mathsf{sk}, \mathsf{pk})$. *On input a message $m \in \{0, 1\}^*$ and a secret key $\mathsf{sk}$, the randomized* signing algorithm $\mathsf{Sign}$ *outputs a signature $\sigma$. On input a public key $\mathsf{pk}$, a message $m \in \{0, 1\}^*$, and a signature $\sigma$, the deterministic* verification algorithm $\mathsf{VrfySig}$ *outputs $1$ if the signature is correct or $0$ otherwise.*

Assume for convenience of notation that to each signed message implicitly has the public key of the signer and a fresh nonce appended. We require that the scheme satisfies *(perfect) correctness*, i.e., for all $m \in \{0, 1\}^*$ and $(\mathsf{sk}, \mathsf{pk})$ output by $\mathsf{KeyGen}$ it holds that: $\mathsf{VrfySig}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, m), m) = 1$. In line with most works in this area, we treat signature schemes as idealized objects satisfying perfect unforgeability. This means that it is information-theoretically impossible to forge a signature under some public key $\mathsf{pk}$ without knowing the corresponding secret key $\mathsf{sk}$. It is easy to instantiate our schemes with a concrete scheme satisfying unforgeability under chosen message attack [GMR84].

## 3 Graded Public Key Infrastructure (GPKI)

In a public key infrastructure (PKI) setting, parties have a *globally consistent* view of a *keyset*, containing $n$ public keys, where $n$ is the total number of parties. As a starting point, we present a construction which is similar to such a PKI. The challenge that we face without an a-priori setup is that we can not easily achieve a globally consistent view on the protocol participants. As discussed in the last Section 2.1, we do not know the exact number of parties, but a known upper bound $n$. We address this problem by building a so-called *graded PKI*. A graded PKI differs from a "real" PKI since the *local* views of parties $P_i, P_j$ on their respective key sets $\mathsf{KeySet}_i, \mathsf{KeySet}_j$ can differ. Here we present $\Pi_{\mathsf{KeyGrade}}$ (c.f. Figure 2), which achieves a graded PKI in the VDF + random oracle model. Our protocol builds on earlier works [KMS14,AD15] which achieve a stronger notion of GPKI with $n$ grades but require $O(n\kappa^2)$ rounds to do so. In contrast, we run a 2-graded protocol requiring only a constant number of rounds. We begin by giving some intuition about our protocol and identifying the main challenges that it has to overcome. We describe the protocol from the view of a (honest) party $P$ with $\varkappa = 1$.

**Challenge Phase.** The protocol begins with a two-round challenge phase. For $i \in \{1, 2\}$, we denote $i$-th challenge message as $(\mathsf{chal}||i, \cdot)$.

- First round: $P$ samples a uniform challenge $c \leftarrow \{0,1\}^{\kappa+\log(t_p)+2\log(\kappa)+1}$. $P$ then multicasts the message $(\mathsf{chal}||1, c)$. Let $\mathbf{c} := (c_1, \ldots, c_\theta)$, where $\theta \leq n$, denote the vector of challenges that $P$ receives from parties (including its own) by the end of the first round.
- Second round: $P$ computes its second round challenge $d$ as $d := \mathsf{H}(\mathbf{c})$ and multicasts the message $(\mathsf{chal}||2, d)$. Denote $\mathbf{d} := (d_1, \ldots, d_{\theta'})$, where $\theta' \leq n$, as the challenges that $P$ receives over the course of the second round (including its own) and denote $\chi := \mathsf{H}(\mathbf{d})$ as their hash.

**Proof of Computation Phase.** The challenge phase is immediately followed by a proof of computation phase. At the beginning of this round, $P$ generates a fresh pair of keys as $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. $P$ then calls $\mathsf{VDF}_\delta$ on input $(\mathsf{Eval}, s := \{\chi, \mathsf{pk}\}, 1)$ to compute the proof $\phi$.

**Key Grading Phase.** After computing $\phi$ in the proof of computation phase, a party $P$ runs a phase of key grading, during which the public key of each other party will be assigned a grade in $\{1, 2\}$. This phase consists of three rounds, where parties send around messages of the form $(\mathsf{rank}||i, \cdot), i \in \{1, 2\}$, for assignment of the $i$-th grade.

- **First round:** $P$ multicasts the message $(\mathsf{rank}||2, \mathsf{pk}, \chi, \phi, \mathbf{d})$.
- **Second round:** For each such message $(\mathsf{rank}||2, \mathsf{pk}_j, \chi_j, \phi_j, \mathbf{d}_j)$ that was received by time $3\Delta + \delta$ from some party $P_j$, $P$ checks whether it was correctly formed. More precisely, $P$ first checks that $\phi_j$ is a proof that passes verification for the input $\{\chi_j, \mathsf{pk}_j\}$, i.e., $\mathsf{VDF}_\delta(\mathsf{Verify}, \phi_j, \{\chi_j, \mathsf{pk}_j\}) = 1$. Next, it checks that $\chi_j$ is consistent with $\mathbf{d}_j$, i.e., $\chi_j = \mathsf{H}(\mathbf{d}_j)$. Lastly, $P$ checks if its second round challenge $d \in \mathbf{d}_j$. If all these checks verify, $P$ is convinced that $P_j$ could not have started computing $\phi_j$ earlier than at round $2\Delta$, as $\phi_j$ depends on a

value that $P$ chose uniformly at random at this time. Hence, it assigns the highest grade 2 to $\mathsf{pk}_j$, i.e., it adds $(\mathsf{pk}_j, 2)$ to $\mathsf{KeySet}$. To make sure that every other honest party assigns $\mathsf{pk}_j$ at least grade 1, $P$ multicasts the message $(\mathsf{rank}||1, \mathsf{pk}_j, \chi_j, \phi_j, \mathbf{d}_j, \mathbf{c})$. Note that, party $P_j$ also follows the same steps as above for each received message $(\mathsf{rank}||2, \mathsf{pk}_l, \chi_l, \phi_l, \mathbf{d}_l)$ from $P_l$ and accordingly sends around $(\mathsf{rank}||1, \mathsf{pk}_l, \chi_l, \phi_l, \mathbf{d}_l, \mathbf{c}_j)$, if it has assigned grade 2 to $P_l$.

- **Third round:** Let $(\mathsf{rank}||1, \mathsf{pk}_l, \chi_l, \phi_l, \mathbf{d}_l, \mathbf{c}_j)$ (for $l \neq j$), be the message received by the end of the previous round from $P_j$, where $P_j$'s key $\mathsf{pk}_j$ was already assigned grade 2 by $P$ in the first grading step. It assigns grade 1 to $\mathsf{pk}_l$ (i.e., it adds $(\mathsf{pk}_l, 1)$ to $\mathsf{KeySet}$), if it has not assigned grade 2 to $\mathsf{pk}_l$ in the previous round but is convinced from $P_j$'s perspective (i.e., from the messages received from $P_j$) that $\mathsf{pk}_l$ should have been marked with grade 2. In addition, $P$ needs to be convinced that the proof $\phi_l$ depended on some unpredictable value and could not have been computed earlier than at time $2\Delta$. To make sure of this, $P$ does the following. It first checks that the proof $\phi_l$ passes verification for the input $\{\chi_l, \mathsf{pk}_l\}$, i.e., $\mathsf{VDF}_\delta(\mathsf{Verify}, \phi_l, \{\chi_l, \mathsf{pk}_l\}) = 1$. Note, however, that since $P_l$ is a *dishonest* party (otherwise, $P$ would have already graded $\mathsf{pk}_l$ in the first grading step), it may have computed $\chi_l = \mathsf{H}(\mathbf{d}_l)$ independently of $P$'s second round challenge $d$. In this case, it verifies instead that $\chi_l$ depends on $c$, by checking that both $\mathsf{H}(\mathbf{c}_j) \in \mathbf{d}_l$ and $c \in \mathbf{c}_j$. If all these checks pass, $P$ adds $(\mathsf{pk}_l, 1)$ to $\mathsf{KeySet}$. Once all messages of this form have been processed, $P$ outputs $\mathsf{KeySet}$.

It is easy to verify that each honest party's key is assigned grade 2 by every honest party, thus proving graded validity. Additionally, the second step of the key grading ensures any key that an honest party has assigned grade 2 is assigned a grade of at least 1 by all honest parties – this implies graded consistency. Finally, an honest party only accepts keys if it is convinced that its corresponding proof could not have been precomputed prior to the beginning of the proof of computation phase. The duration of the proof of computation phase is set to $\delta$ according to the $\mathsf{VDF}_\delta$ oracle. While an honest party (with *no speed-up*) computes one proof within $\delta$ time, an adversarial party with $\varkappa$-speed-up of computational power computes $\lfloor \varkappa \rfloor$ proofs within the same time $\delta$. Below, we will set the parameter $\delta$ such that no adversarial party can make more than $\lfloor \varkappa \rfloor$ calls to $\mathsf{VDF}_\delta$ for the entire duration of the protocol, given that it calls $\mathsf{VDF}_\delta$ earliest at the beginning of the Challenge phase. Taken together, the total number of adversarial keys accepted by honest parties can not exceed half of the total number of keys. This implies bounded number of identities.

The following sequence of lemmas prove the security of $\Pi_{\mathsf{KeyGrade}}$. Lemmas 1 to 3, respectively, show that $\Pi_{\mathsf{KeyGrade}}$ satisfies the properties of graded validity, graded consistency and bounded number of identities as per Definition 8.

**Lemma 1.** *Protocol $\Pi_{\mathsf{KeyGrade}}$ achieves graded validity.*

*Proof.* It is easy to verify that all honest parties' keys are assigned the highest grade 2 by all other honest parties from the protocol description. Hence, the protocol satisfies graded validity (for any number of corrupted parties). □

**Protocol $\Pi_{\mathsf{KeyGrade}}$**

We describe the protocol from the view of an honest party $P$ with *no speed-up* of computational power, where $P$ has access to a random oracle $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\kappa+\log(t_p)+2\log(\kappa)+1}$ and an oracle $\mathsf{VDF}_\delta$. $P$ executes the following phases.

**Challenge Phase.**

– At time 0: Sample $c \xleftarrow{\$} \{0,1\}^{\kappa+\log(t_p)+2\log(\kappa)+1}$ and multicast the message $(\mathsf{chal}||1, c)$.
– At time $\Delta$: Add all $\theta(\leq n)$ challenges received by time $\Delta$ to the set $\mathbf{c}_i = (c_1,\dots,c_\theta)$. Compute $d = \mathsf{H}(\mathbf{c})$ and multicast the message $(\mathsf{chal}||2, d)$.

**Proof of Computation Phase.**

– At time $2\Delta$: Add all $\theta'(\leq n)$ challenges received in round 2 to the set $\mathbf{d} = (d_1,\dots,d_{\theta'})$ and compute its hash $\chi = \mathsf{H}(\mathbf{d})$. Then, sample key pair $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and compute $\phi = \mathsf{VDF}_\delta(\mathsf{Eval}, s := \{\chi, \mathsf{pk}\}, 1)$.

**Key Ranking Phase.** Define the following Boolean conditions:

– $C_2(j)$: $\mathsf{VDF}_\delta(\mathsf{Verify}, \phi_j, \{\chi_j, \mathsf{pk}_j\}) = 1$, $\chi_j = \mathsf{H}(\mathbf{d}_j)$, and $d \in \mathbf{d}_j$.
– $C_1(j,l)$: $(\mathsf{pk}_j, 2) \in \mathsf{KeySet}$, $(\mathsf{pk}_l, \cdot) \notin \mathsf{KeySet}$, $\mathsf{VDF}_\delta(\mathsf{Verify}, \phi_l, \{\chi_l, \mathsf{pk}_l\}) = 1$, $\chi_l = \mathsf{H}(\mathbf{d}_l)$, $\mathsf{H}(\mathbf{c}_j) \in \mathbf{d}_l$, and $c \in \mathbf{c}_j$.

Do:

– At time $2\Delta + \delta$: Multicast $(\mathsf{rank}||2, \mathsf{pk}, \chi, \phi, \mathbf{d})$.
– At time $3\Delta+\delta$: For each received message $(\mathsf{rank}||2, \mathsf{pk}_j, \chi_j, \phi_j, \mathbf{d}_j)$ from $P_j$, such that $C_2(j)$, add $(\mathsf{pk}_j, 2)$ to $\mathsf{KeySet}$ and multicast the message $(\mathsf{rank}||1, \mathsf{pk}_j, \chi_j, \phi_j, \mathbf{d}_j, \mathbf{c})$.
– At time $4\Delta + \delta$: For each received message $(\mathsf{rank}||1, \mathsf{pk}_l, \chi_l, \phi_l, \mathbf{d}_l, \mathbf{c}_j)$ from $P_j$ such that $C_1(j,l)$, add $(\mathsf{pk}_l, 1)$ to $\mathsf{KeySet}$. After processing each message, output $\mathsf{KeySet}$.

**Fig. 2.** A protocol for establishing a GPKI with 2 grades run between $n$ parties, where parties have access to a random oracle $\mathsf{H}$ and an oracle $\mathsf{VDF}_\delta$. Duration of the $\Pi_{\mathsf{KeyGrade}}$ protocol is $(5\Delta+\delta)$. For the subsequent Sections we set $\delta := 11\Delta$, which sets the duration of the protocol to $16\Delta$.

**Lemma 2.** *Protocol $\Pi_{\mathsf{KeyGrade}}$ achieves graded consistency.*

*Proof.* Let $P_i$ and $P_j$ be honest parties. We show that if $P_j$ assigns grade 2 to some party $P_l$'s public key $\mathsf{pk}_l$, then $P_i$ assigns $\mathsf{pk}_l$ at least grade 1. Since $(\mathsf{pk}_l, 2) \in \mathsf{KeySet}_j$, in the second round of the Key Grading Phase, $P_j$ must have received a correctly formed message $(\mathsf{rank}||2, \mathsf{pk}_l, \chi_l, \phi_l, \mathbf{d}_l)$ from $P_l$, i.e., such that $\mathsf{VDF}_\delta(\mathsf{Verify}, \phi_l, \{\chi_l, \mathsf{pk}_l\})$ $= 1$, and that $\chi_l$ depends on $P_j$'s first round challenge $d_j$, i.e., $\chi_l = \mathsf{H}(\mathbf{d}_l)$ and $d_j \in \mathbf{d}_l$. Thus, $P_j$ multicasts the message $(\mathsf{rank}||1, \mathsf{pk}_l, \chi_l, \phi_l, \mathbf{d}_l, \mathbf{c}_j)$ to all parties at the beginning of the third round of the Key Grading Phase. $P_i$ receives this message by the end of the third round of the Key Grading Phase and has, at that point, already assigned $\mathsf{pk}_j$ grade 2. It is then able to verify that $P_j$'s message is correctly formed and depends on $c_i$ (by performing $P_j$'s checks and the additional checks $c_i \in \mathbf{c}_j$ and $d_j = \mathsf{H}(\mathbf{c}_j)$). Hence, it assigns $\mathsf{pk}_l$ the grade 1 at this point, unless it has previously assigned it the grade 2. □

**Lemma 3.** *Suppose that $\mathcal{A}$ is a $(q, t_p, \varkappa)$-algorithm where $q < \frac{n}{\lfloor\varkappa\rfloor+1}$, For $k \in \mathbb{N}$, let $\delta = k\Delta$. If $k > 5\varkappa$, and $\mathsf{VDF}_\delta$ is $(2, \frac{5}{k}, \beta)$-sequential, then $\Pi_{\mathsf{KeyGrade}}$ achieves bounded number of identities with probability at least $1 - (2^{-\kappa-2\log(\kappa)-1} + \beta)$. In particular, if $\mathsf{N} := |\bigcup_{i\in[n]_{P_i\in\mathcal{H}}} \mathsf{KeySet}_i|$ be the total number of keys in the combined key sets of* *all honest parties $\mathcal{H}$, then the total number of keys that belong to corrupted parties is* $< \frac{1}{2}\mathsf{N}$.

*Proof.* Suppose that parties run $\Pi_{\mathsf{KeyGrade}}$ at time $T$. Let $E$ be the event that $\mathcal{A}$ predicts the first round challenge value $c_i$ of at least one honest party $P_i$ at time $T' < T$. Since $\mathcal{A}$ runs for at most $t_p$ steps and each honest party $P_i$ samples $c_i$ uniformly from the space $\{0,1\}^{\kappa+\log(t_p)+2\log(\kappa)+1}$, $E$ occurs with probability at most $2^{-\kappa-2\log(\kappa)-1}$. By Definition 2, this means that the sequence of honest challenges $\{c\}_{i,P_i\in\mathcal{H}}$ is $((n-q), T, 2^{-\kappa-2\log(\kappa)-1})$-unpredictable. Since the duration of $\Pi_{\mathsf{KeyGrade}}$ is $\delta + 5\Delta$, the protocol terminates at $T+\delta+5\Delta$. From $(2, \frac{5}{k}, \beta)$-sequentiality of $\mathsf{VDF}_\delta$, we have

$$T_{2,\frac{5}{k}} = T + (\delta + 5\Delta) = T + (1 + \frac{5\Delta}{\delta}) \cdot \delta = T + (1 + \frac{5}{k}) \cdot \delta$$
$$\tau_{2,\frac{5}{k}} = \tau = \lfloor (1 + \frac{5}{k}) \cdot \varkappa \rfloor \cdot q = \lfloor \varkappa + \frac{5\varkappa}{k} \rfloor \cdot q = \lfloor \varkappa \rfloor \cdot q \text{ (Since, } k > 5\varkappa)$$

Let $\chi_1, \ldots, \chi_{\tau+1}$ depend on $\{c\}_{i,P_i\in\mathcal{H}}$ and let $\mathsf{pk}_1, \ldots, \mathsf{pk}_{\tau+1}$ be pairwise distinct. $\mathcal{A}$ is able to compute at most $q \cdot \lfloor \varkappa \rfloor$ VDFs within time $(T+\delta+5\Delta)$ via honest invocations of the $\mathsf{VDF}_\delta$ oracle. It follows from $(2, \frac{5}{k}, \beta)$-sequentiality of $\mathsf{VDF}_\delta$ that $\mathcal{A}$ can output $\{\phi_i\}_{i\in[\tau+1]}$ before or at time $(T+\delta+5\Delta)$ such that for all $j \in [\tau+1]$, $\mathsf{VDF}_\delta(\mathsf{Verify}, \phi_j, \{\chi_j, \mathsf{pk}_j\}) = 1$ with probability at most $2^{-\kappa-2\log(\kappa)-1} + \beta$. (Note that if $\chi_j$ does not depend on $\{c\}_{i,P_i\in\mathcal{H}}$, no honest party accepts $\mathsf{pk}_j$). As a result, the number of identities produced by $\mathcal{A}$ is bounded by $\tau$ with probability at least $1 - (2^{-\kappa-2\log(\kappa)-1} + \beta)$. The total number of keys from honest and corrupted parties can be given by $\mathsf{N} = (n-q) + \tau = (n-q) + q \cdot \lfloor \varkappa \rfloor$, and the total number keys belonging to corrupted parties is $\tau = q \cdot \lfloor \varkappa \rfloor < (n-q)$ (given that $q < \frac{n}{\lfloor \varkappa \rfloor + 1}$) or, $q\lfloor \varkappa \rfloor < \frac{1}{2}\mathsf{N}$. □

**Corollary 1.** *Since $\delta > 5\varkappa \cdot \Delta \geq (5\varkappa + 1) \cdot \Delta$, the duration of $\Pi_{\mathsf{KeyGrade}}$ is set as $\delta + 5\Delta := (5\varkappa + 6) \cdot \Delta$.*

**Corollary 2.** *If $n$ be the number of parties in the $\Pi_{\mathsf{KeyGrade}}$ protocol, $q$ be the number of adversarial parties, each with speedup at most $\varkappa$, then the total number of keys in the combined keysets of all honest parties is given as*

$$\mathsf{N} := |\bigcup_{i\in[n]_{P_i\in\mathcal{H}}} \mathsf{KeySet}_i| = (n-q) + q \cdot \lfloor \varkappa \rfloor = n + q \cdot (\lfloor \varkappa \rfloor - 1).$$

**Parameter Selection.** For the remaining sections of the paper we set $\varkappa = 2$, which gives us: $\delta > 10 \cdot \Delta \geq 11\Delta$. We assign $\delta := 11\Delta$, and accordingly duration of $\Pi_{\mathsf{KeyGrade}} := 16\Delta$. To tolerate a higher speed-up of adversarial parties the parameters need to be adjusted accordingly.

## 4 Construction of Graded Broadcast (GBC) from GPKI

As a next step, we construct a graded broadcast protocol that is an adaptation of the protocol in [MV17]. The resulting protocol $\Pi_{\mathsf{GBC}}$ only requires a graded PKI whereas the original required a full PKI. In the GPKI setting, parties neither have a global consistent view of all parties' keys, nor do they know the exact number of parties (only an upper bound $n$). Henceforth, parties rely on their local $\mathsf{KeySet}$ obtained as an output of the $\Pi_{\mathsf{KeyGrade}}$ protocol. The total number of keys obtained after running the $\Pi_{\mathsf{KeyGrade}}$ protocol is set to $\mathsf{N} = n + q \cdot (\lfloor \varkappa \rfloor - 1)$. Before describing protocol $\Pi_{\mathsf{GBC}}$ in Figure 3, we introduce some nomenclatures that will help to present our protocol succinctly. In the following, we assume a fixed sender $S$.

- A signature $\langle x \rangle_i$ is *valid* in the view of party $P$, if (i) $P$ has assigned grade 2 for $\mathsf{pk}_i$, i.e., $(\mathsf{pk}_i, 2) \in \mathsf{KeySet}_P$ and (ii) $\mathsf{VrfySig}(\langle x \rangle_i, \mathsf{pk}_i) = 1$.
- A signature $\langle x \rangle_i$ is *weakly valid* in the view of party $P$, if (i) $(\mathsf{pk}_i, g_i) \in \mathsf{KeySet}_P$, where $g_i \geq 1$ and (ii) $\mathsf{VrfySig}(\langle x \rangle_i, \mathsf{pk}_i) = 1$.
- We refer to an iterated signature of the form $\langle\langle x \rangle_S\rangle_j$ (i.e., by $S$ and some party $P_j$) as a *countersignature* and say that $P_j$ is the *outer signer*.
- A countersignature $\langle\langle x \rangle_S\rangle_j$ is said to be *valid* in the view of party $P$ if the following holds. (i) $(\mathsf{pk}_S, 2) \in \mathsf{KeySet}_P$, (ii) $\mathsf{VrfySig}(\langle x \rangle_S, \mathsf{pk}_S) = 1$, (iii) $(\mathsf{pk}_j, 2) \in \mathsf{KeySet}_P$, and (iv) $\mathsf{VrfySig}(\langle\langle x \rangle_S\rangle_j, \mathsf{pk}_j) = 1$.
- A countersignature $\langle\langle x \rangle_S\rangle_j$ is said to be *weakly valid* in the view of party $P$ if the following holds. (i) $(\mathsf{pk}_S, g_S) \in \mathsf{KeySet}_P, g_S \geq 1$ (ii) $\mathsf{VrfySig}(\langle x \rangle_S, \mathsf{pk}_S) = 1$, (iii) $(\mathsf{pk}_j, g_j) \in \mathsf{KeySet}_P, g_j \geq 1$, and (iv) $\mathsf{VrfySig}(\langle\langle x \rangle_S\rangle_j, \mathsf{pk}_j) = 1$.
- A set of signatures $\psi(x)$ is said to be *consistent for* $x$ in the view of party $P$, if it contains valid countersignatures on $x$ from at least $\frac{\mathsf{N}}{2}$ distinct outer signers.
- A set of signatures $\psi(x)$ is said to be *weakly consistent for* $x$ in the view of party $P$, if it contains weakly valid countersignatures on $x$ from at least $\frac{\mathsf{N}}{2}$ distinct outer signers.

The following sequence of lemmas proves security of $\Pi_{\mathsf{GBC}}$.

**Lemma 4.** *If the sender $S$ is honest and inputs $x$, then every honest party $P$ multicasts a set of signatures $\psi_P(x)$ at time $2\Delta$ which is consistent for $x$ in the view of every honest party.*

*Proof.* If the sender $S$ is honest, then it multicasts $x$ with a valid signature $\langle x \rangle_S$ at time 0. Let $P$ be an honest party. From the protocol description, $P$ generates a valid countersignature as $\langle\langle x \rangle_S\rangle_P$ and multicasts it at time $\Delta$. At time $2\Delta$, $P$ collects valid countersignatures of the form $\langle\langle x \rangle_S\rangle_k$ from all parties $P_k$ with $g_k = 2$ into its set of signatures $\psi_P(x)$. Note that due to the graded validity property of the underlying graded PKI, if $P_k$ is honest, then $P_k$'s public key has grade 2 for $P$, i.e., $((\mathsf{pk}_k, 2) \in \mathsf{KeySet}_P)$. As there are at least $\frac{\mathsf{N}}{2}$ honest parties, this implies that $P$'s signature set $\psi_P(x)$ contains at least $\frac{\mathsf{N}}{2}$ countersignatures from honest parties. Moreover, since honestly generated countersignatures are valid in the view of each

**Protocol $\Pi_{\mathsf{GBC}}$**

We describe the protocol from the view of party $P$ with a sender $S$. $P$ executes the following three phases.

- At time 0: If $P$ is the sender $S$, it computes signature on message $x$ as $\langle x \rangle_S$ and multicasts $(x, \langle x \rangle_S)$ to all parties.
- At time $\Delta$: If $P$ sees $(x, \langle x \rangle_S)$, where $\langle x \rangle_S$ is a valid signature on message $x$, then $P$ multicasts $(x, \langle \langle x \rangle_S \rangle_P)$ to all parties.
- At time $2\Delta$: Party $P$ collects *all* valid countersignatures of the form $(x', \langle \langle x' \rangle_S \rangle_k)$ from parties $P_k$ with same input $x'$. $P$ constructs the signature set $\psi(x')$ with message $x'$ as

$$\left(\psi(x') := \left\{ \langle \langle x' \rangle_S \rangle_k \right\}_k\right).$$

  If $\psi(x')$ is a consistent signature set from $P$'s view and it has not received a countersignature of the form $\langle \langle x'' \rangle_S \rangle_k$, where $x'' \neq x'$ then $P$ multicasts $\psi(x')$. (Otherwise, $P$ multicasts nothing).
- At time $3\Delta$: Let $\psi_k(x')$ be a weakly consistent signature set received from party $P_k$ (note that this includes consistent sets).To determine its output, $P$ does as follows:
  - If $P$ sees at least $\frac{\mathsf{N}}{2}$ consistent signature sets of the form $\psi_k(x')$ (for distinct $k$) on $x'$, then $P$ outputs $(x', 2)$.
  - If $P$ sees at least one weakly consistent signature set $\psi_k(x')$ for some $x'$, and no weakly consistent signature set for a different value $x'' \neq x'$, then it outputs $(x', 1)$.
  - Else, $P$ outputs $(\perp, 0)$.

**Fig. 3.** Gradecast protocol $\Pi_{\mathsf{GBC}}$, where each party has a local $\mathsf{KeySet}$ from $\Pi_{\mathsf{KeyGrade}}$ protocol. Duration of $\Pi_{\mathsf{GBC}}$ is $4\Delta$. $\mathsf{N} = n + q \cdot (\lfloor \varkappa \rfloor - 1)$ (c.f Corollary 2).

honest party, this implies that $\psi_P(x)$ is a consistent signature set in the view of every honest party. Due to the unforgeability of underlying digital signature scheme, there is no valid countersignature of the form $\langle \langle x' \rangle_S \rangle_k$, where $g_k = 2$ and $x' \neq x$. Hence, $P$ multicasts a set $\psi_P(x)$ at time $2\Delta$ which consistent in the view of every honest party. □

**Lemma 5.** *If, at time $3\Delta$, an honest party receives at least $\frac{\mathsf{N}}{2}$ sets of signatures which are all consistent with (the same) $x$, no honest party receives a set of signatures that is weakly consistent with $x' \neq x$.*

*Proof.* Let us assume that at time $3\Delta$, honest party $P_i$ receives at least $\frac{\mathsf{N}}{2}$ signature sets which are all consistent with some value $x$. This implies that among them, there is at least one set $\psi_P(x)$ which was sent by an honest party $P$ at time $2\Delta$. Now, towards contradiction, suppose honest party $P_j$ receives the set $\psi_{P'}(x')$ of countersignatures from party $P'$ such that $\psi_{P'}(x')$ is weakly consistent with $x' \neq x$. By definition, $\psi_{P'}(x')$ contains at least $\frac{\mathsf{N}}{2}$ weakly valid countersignatures on $x'$ from distinct outer signers. Among these $\frac{\mathsf{N}}{2}$ outer signers, there is at least one honest party $P^*$. Since $P^*$ would have sent the countersignature $\langle \langle x' \rangle_S \rangle_{P^*}$ at time $\Delta$, all honest parties would have received it by time $2\Delta$. Hence, no honest party would have sent a consistent set of signatures for $x$ at time $2\Delta$. This contradicts that $P$ sends such a set at time $2\Delta$. □

**Lemma 6.** *Protocol $\Pi_{\mathsf{GBC}}$ achieves graded validity.*

*Proof.* If $S$ is honest, then by Lemma 4 every honest party $P$ multicasts a signature set $\psi_P(x)$ at time $2\Delta$ which is consistent with $x$. At time $3\Delta$, every honest party hence receives at least $\frac{\mathsf{N}}{2}$ such signature sets. Moreover, by Lemma 5, no honest party receives a set of signatures that is weakly consistent with some $x' \neq x$. This implies that all honest parties output $(x, 2)$. □

**Lemma 7.** *Protocol $\Pi_{\mathsf{GBC}}$ achieves graded consistency.*

*Proof.* Let $P$ be a honest party that outputs $(x, 2)$. We want to show that every honest party $P'$ outputs $(x, g' \geq 1)$. Since $P$ outputs grade 2 for value $x$, it receives at least $\frac{\mathsf{N}}{2}$ signature sets from distinct parties which are all consistent with $x$. One of these sets must have been sent by an honest party $P^*$ and received by time $3\Delta$. Therefore, $P'$ must have also received this set $\psi_{P^*}(x)$ from $P^*$ by time $3\Delta$. From the graded consistency property of the underlying graded PKI, each countersignature that is valid in $P^*$'s view, is weakly valid in each honest parties view. This implies that $\psi_{P^*}(x)$ is a weakly consistent signature set in the view of party $P'$. By Lemma 5, $P'$ receives no signature set that is weakly consistent with $x' \neq x$. Hence, $P'$ outputs $(x, 2)$ or $(x, 1)$. □

## 4.1 Construction of a Graded Byzantine Agreement from GBC

We define a simple graded byzantine agreement protocol $\Pi_{\mathsf{GBA}}$ in fig. 4 from $\Pi_{\mathsf{GBC}}$. As a first step, each party runs $\Pi_{\mathsf{GBC}}$ as the sender with their input. In the following step, a value is output with grade 2 or 1 depending on the message/grade outputs of the $\Pi_{\mathsf{GBC}}$ initiated by other parties. The security of $\Pi_{\mathsf{GBA}}$ follows from the following sequence of lemmas.

**Protocol $\Pi_{\mathsf{GBA}}$**

We describe the protocol from the view of party $P$, where $P$ executes the following two steps.

- At time 0: Initiate $\Pi_{\mathsf{GBC}}$ as the sender with input $x$.
- At time $4\Delta$: Let $(x_j, g_j)$ be the message/grade that $P$ outputs in the gradecast initiated by $P_j$. For all $v$ that were received in at least one invocation of $\Pi_{\mathsf{GBC}}$, $P$ sets $\mathcal{S}^v := \{j : x_j = v \wedge g_j = 2\}$ and $\tilde{\mathcal{S}}^v := \{j : x_j = v \wedge g_j \geq 1\}$.
  - If there exists $v$ s.t. $|\tilde{\mathcal{S}}^v| \geq \frac{\mathsf{N}}{2}$, then prepare a tuple $t$ of value, grade pair s.t. $t := (v, 1)$, otherwise $t := (\perp, 0)$.
  - If $|\mathcal{S}^v| \geq \frac{\mathsf{N}}{2}$, then $t := (v, 2)$.
  - output $t$.

**Fig. 4.** Graded Byzantine Agreement protocol $\Pi_{\mathsf{GBA}}$, where each party has a local $\mathsf{KeySet}$ from $\Pi_{\mathsf{KeyGrade}}$ protocol. Duration of $\Pi_{\mathsf{GBA}}$ is $4\Delta$. $\mathsf{N} = n + q \cdot (\lfloor \varkappa \rfloor - 1)$ (c.f Corollary 2).

**Lemma 8.** *Protocol $\Pi_{\mathsf{GBA}}$ achieves graded validity.*

*Proof.* If $P_i$ output $v_i$ with grade 2, this means $P_i$ must have set $|S_{v_i}| \geq \frac{\mathsf{N}}{2}$. Due to the graded consistency property of $\Pi_{\mathsf{GBC}}$, $P_j$ must have $|S_{v_j}| \supseteq |S_{v_i}| \geq \frac{\mathsf{N}}{2}$, such that $v_j = v_i$. This implies $P_j$ outputs value $v_j = v_i$ with grade at least 1. □

**Lemma 9.** *Protocol $\Pi_{\mathsf{GBA}}$ achieves graded consistency.*

*Proof.* Due to the graded validity property of $\Pi_{\mathsf{GBC}}$, the result follows immediately. □

## 5 Achieving Byzantine Agreement

As the final step, we build our Byzantine agreement protocol $\Pi_{\mathsf{BA}}$ (c.f. Figure 5). Though our protocol is an adaptation of the same from [KK06a], it uses as main ingredients the graded PKI protocol $\Pi_{\mathsf{KeyGrade}}$ (Figure 2) and the graded byzantine agreement protocol $\Pi_{\mathsf{GBA}}$ (Figure 4) from Sections 3 and 4.1 respectively. One subtle difference because of the use of a graded PKI is that each party $P$ relies on their local $\mathsf{KeySet}_P$. To make the BA protocol work, we need one more ingredient – a leader election protocol. More precisely, as one of the steps of the $\Pi_{\mathsf{BA}}$ protocol, it is necessary to elect one of the parties as a leader in an unpredictable manner such that an honest party will be elected with constant probability. For the description of $\Pi_{\mathsf{BA}}$, we assume a protocol $\Pi_{\mathsf{Leader}}$ that is run as a subroutine to $\Pi_{\mathsf{BA}}$, in parallel with protocol $\Pi_{\mathsf{KeyGrade}}$. As we will explain later, the subprotocols $\Pi_{\mathsf{KeyGrade}}$ and $\Pi_{\mathsf{Leader}}$ share a common state. For our purposes, we will require that the subprotocol $\Pi_{\mathsf{Leader}}$ has the following properties for all $k \in \mathbb{N}$ and all pairs of honest parties $P, P'$, assuming that parties initiate $\Pi_{\mathsf{BA}}$ at time 0:

- $\Pi_{\mathsf{Leader}}$ outputs a value $\ell_P$ to $P$ at time $(12 \cdot k + 27) \cdot \Delta$.
- With probability at least $\frac{n-q}{\mathsf{N}} \geq \frac{1}{2}$, $\ell_P = \ell_{P'} = \ell$ and $P_\ell$ is an honest party at round $(12 \cdot k + 24) \cdot \Delta$.

We present an instantiation $\Pi_{\mathsf{Leader}}$ based on VDF in Section 5.1. The following theorem summarizes the properties of our $\Pi_{\mathsf{BA}}$ protocol.

**Theorem 2.** *Suppose that $\Pi_{\mathsf{KeyGrade}}$ be a graded public key infrastructure protocol, $\Pi_{\mathsf{GBA}}$ be a graded byzantine agreement protocol and $\Pi_{\mathsf{Leader}}$ be a leader election protocol run between $n$ parties. Then $\Pi_{\mathsf{BA}}$ is a Byzantine agreement protocol tolerating $q$ corrupted parties with at most 2-speedup of computational power, where $q < \frac{n}{3}$. Moreover, it terminates within $O(1)$ rounds in expectation for all honest parties, and with probability $1 - 2^{-\kappa}$ within $16 + 12 \cdot (\kappa + 1)$ rounds.*

Our proof for Theorem 2 is almost identical to [KK06a, Theorem 13] which is why we defer it to full version, Appendix E. After an initial setup phase in which parties agree on a graded PKI via $\Pi_{\mathsf{KeyGrade}}$ and run the setup steps for $\Pi_{\mathsf{Leader}}$, the $\Pi_{\mathsf{BA}}$ protocol proceeds in iterations. In every such iteration, we invoke two graded byzantine agreement subroutines and one multicast subroutine, and update the values $\mathsf{lock}$ and $m$ until they converge on a common value for $m$. Here, the variable $\mathsf{lock}$ indicates the number of iterations after which the protocol may terminate. In particular, $\mathsf{lock} = \infty$ means that the protocol could keep running for an unbounded number of iterations, $\mathsf{lock} = 1$ means that termination will be reached after one more iteration and $\mathsf{lock} = 0$ means that the protocol terminates in the next iteration. Overall, each iteration requires $12\Delta$ time.

To enter termination mode, i.e., to set $\mathsf{lock} = 1$, parties rely on a randomly elected leader who is chosen in the last round of an iteration via $\Pi_{\mathsf{Leader}}$. In our construction of $\Pi_{\mathsf{Leader}}$, honest parties learn the identity of the leader for the $k$-th iteration at time $(12 \cdot k + 27) \cdot \Delta$. On the other hand, dishonest parties learn it at round $(12 \cdot k + 25) \cdot \Delta$ – this is because dishonest parties, controlled by a rushing adversary can start the proof of computation phase of the $\Pi_{\mathsf{KeyGrade}}$ at time 0 instead of at time $2\Delta$. Hence, the idea is to have all parties perform a multicast instruction in the protocol at round $(12 \cdot k + 24) \cdot \Delta$, i.e., one round prior to the adversary learning about the identity of the the leader. If the elected leader executed this instruction honestly (which happens with probability at least $\frac{1}{2}$), the protocol enters termination mode, meaning that all honest parties set $\mathsf{lock} = 1$. Overall, the protocol enters the termination routine after at most $\kappa$ many iterations with probability all but $2^{-\kappa}$.

**Generalization of Theorem 2.** Our Theorem 2 can be expressed more generically in the following Theorem 3, for any speedup parameter $\varkappa \in \mathbb{R}_{>0}$, where the number of rounds of $\Pi_{\mathsf{KeyGrade}}$ is $(5\varkappa + 6) \cdot \Delta$ (c.f Corollary 1). In particular, for the lowest value of $\varkappa = 1$, our $\Pi_{\mathsf{BA}}$ protocol tolerates at most $< \frac{n}{2}$ corrupted parties.

**Theorem 3.** *Suppose that $\Pi_{\mathsf{KeyGrade}}$ be a graded public key infrastructure protocol, $\Pi_{\mathsf{GBA}}$ be a graded byzantine agreement protocol and $\Pi_{\mathsf{Leader}}$ be a leader election protocol run between $n$ parties. Then $\Pi_{\mathsf{BA}}$ is a Byzantine agreement protocol tolerating $q$ corrupted parties with at most $\varkappa$-speedup of computational power, where $q < \frac{n}{(\lfloor \varkappa \rfloor + 1)}$. Moreover, it terminates within $O(1)$ rounds in expectation for all honest parties, and with probability $1 - 2^{-\kappa}$ within $(5\varkappa + 6) + 12 \cdot (\kappa + 1)$ rounds.*

Proof of Theorem 3 follows similar to that of Theorem 2.

## 5.1 The Protocol $\Pi_{\mathsf{Leader}}$

In this section, we present our construction of the leader election protocol $\Pi_{\mathsf{Leader}}$ (cf. Figure 6) that relies on a $\mathsf{VDF}$ accessed as an oracle. The idea of our protocol is for each party to compute a proof of (sequential) computation on some value that could not have been predicted before commencing the current run of $\Pi_{\mathsf{Leader}}$. More concretely, parties are asked to present an evaluation of $\mathsf{VDF}$ at periodic steps in $\Pi_{\mathsf{Leader}}$ and spend the time interval between these steps in computing of the next evaluation. Such a chain of sequential evaluations of $\mathsf{VDF}$ is only accepted in round $r$ if it accounts for $r$ sequential steps of computation. In addition, it must be possible to ensure that the computation could not have started before the onset of the first round.

The key observation is that per party, there can be at most one such sequence of evaluations that accounts for the entire duration of $\Pi_{\mathsf{Leader}}$ up to that round (since evaluations of a $\mathsf{VDF}$ can not be parallelized). Moreover, the adversary can not predict the sequences produced by honest parties due to its limited budget of computation. Hence, parties can use (hashes of) these sequences as unique (per party) and unpredictable sources of randomness, which can be verified by everyone efficiently. In each election, the party who produces the *smallest* hash, computed according to

```
Protocol $\Pi_{\mathsf{BA}}$

We describe the following protocol from the view of party $P$ with input $m_P$.

Setup Phase.

– At time 0: Participate in a run of $\Pi_{\mathsf{KeyGrade}}$. In parallel, participate in a run of $\Pi_{\mathsf{Leader}}$.
– At time $16\Delta$: Denote $\mathsf{KeySet}_P$ the set of keys output in $\Pi_{\mathsf{KeyGrade}}$. (Hereafter, it is assumed
  that parties share a graded PKI.)

Agreement Phase. Initialize $\mathsf{lock} := \infty$, $m := m_P$ and $k := 0$. Repeat the following loop
forever:

– At time $(12k + 16) \cdot \Delta$: Initiate $\Pi_{\mathsf{GBA}}$ on input $m$.
– At time $(12k + 20) \cdot \Delta$:
    • Let $(v, g)$ be the output of $P$. If $\mathsf{lock} = \infty$ then:
        ∗ If $g = 2$, set $\mathsf{lock} := 1$.
        ∗ If $g = 1$, then set $m := v$.
        ∗ If $g = 0$, then set $m = \bot$.
    • Initiate $\Pi_{\mathsf{GBA}}$ on input $m$.
– At time $(12k + 24) \cdot \Delta$:
    • Let $(v, g)$ be the output of $P$. If $\mathsf{lock} = \infty$ then:
        ∗ If $g = 1$, then set $m := v$.
        ∗ If $g = 0$, then set $m = \bot$.
    • Multicast $m$.
– At time $(12k + 27) \cdot \Delta$:
    • Set $k := k + 1$ and denote $m_j$ the message received from $P_j$. Let $\ell$ denote the output
      obtained from $\Pi_{\mathsf{Leader}}$.
    • If $\mathsf{lock} = \infty$ and $m = \bot$, then $P$ sets $m := m_\ell$.
    • If $\mathsf{lock} = 0$, $P$ outputs $m$ and terminates.
    • If $\mathsf{lock} = 1$, then $P$ sets $\mathsf{lock} := 0$.
```

**Fig. 5.** A Byzantine agreement protocol $\Pi_{\mathsf{BA}}$, where parties share a graded PKI by running the $\Pi_{\mathsf{KeyGrade}}$ protocol. $\Pi_{\mathsf{BA}}$ internally invokes two other subprotocols: graded byzantine agreement protocol $\Pi_{\mathsf{GBA}}$ and a leader election protocol $\Pi_{\mathsf{Leader}}$. $\mathsf{N} = n + q \cdot (\lfloor \varkappa \rfloor - 1)$ (c.f Corollary 2). We set $\mathsf{N} = n + q$ (for adversarial speedup $\varkappa = 2$). All parties have access to random oracles $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{\kappa + \log(t_p) + 2\log(\kappa) + 1}$, $\mathsf{H}_{\mathsf{N}} : \{0,1\}^* \to \mathsf{N}$ and an oracle $\mathsf{VDF}_\delta$.

hash function $\mathsf{H}_{\mathsf{N}} \colon \{0,1\}^* \to \mathsf{N}$, will be elected as the leader in $\Pi_{\mathsf{Leader}}$. Since an honest party sends its hash to everyone and each party produces the smallest hash with the same (uniform) probability, parties agree on the hash of an honest leader as the minimal hash with probability at least $\frac{1}{2}$ in every election We note that this idea closely mimics the standard approach of electing leaders in Byzantine agreement protocols using verifiable random functions (VRFs). However, to use VRFs, it is required that a trusted dealer generates and distributes the keys or some unpredictable random string at the beginning of the protocol. Since we can not rely on either of these setup assumptions, we choose to instead rely on the above approach that uses VDFs.

Our $\Pi_{\mathsf{Leader}}$ protocol is formally described in Figure 6. Below, we elaborate on the two phases of the protocol from the view of (an honest) party $P$ in a bit more detail and give some intuition about them.

**Setup Phase.** The protocol begins at time 0 with a one-time setup phase which is executed in parallel with a run of $\Pi_{\mathsf{KeyGrade}}$. Recall that $\Pi_{\mathsf{KeyGrade}}$ begins with two

**Protocol $\Pi_{\mathsf{Leader}}$**

We describe the protocol from the view of an honest party $P$ with $\varkappa = 1$. Initialize a set $L := \{\}$. $P$ participates (in parallel) in a run of $\Pi_{\mathsf{KeyGrade}}$. (We make this explicit below).

**Setup Phase.**

- At time 0: In parallel, participate in a run of $\Pi_{\mathsf{KeyGrade}}$.
- At time $13\Delta$: Denote $\phi^0 = \mathsf{VDF}_{11\Delta}(\mathsf{Eval}, \chi, 1)$ the proof computed during the proof of computation phase of $\Pi_{\mathsf{KeyGrade}}$. Call $\mathsf{VDF}_{13\Delta}(\mathsf{Eval}, \mathsf{H}_{\mathsf{N}}(\phi^0), 1)$ to compute $\phi^1$.
- At time $16\Delta$: Denote $\mathsf{KeySet}$ the set of keys output in $\Pi_{\mathsf{KeyGrade}}$.

**Leader Election Phase.** Initialize $k := 1$ and $\mathsf{bad}_j := \mathsf{false}$, for all $j \in [n]$ s.t. there exists $(\mathsf{pk}_j, g_j) \in \mathsf{KeySet}$. (We refer to the owner of $\mathsf{pk}_j$ as $P_j$ below). Denote $\phi_j^0$ the proof of computation received together with $\mathsf{pk}_j$ during $\Pi_{\mathsf{KeyGrade}}$. Repeat the following sequence of steps forever:

- At time $(26 + 12(k-1)) \cdot \Delta$: Upon completing computation of $\phi^k$, call $\mathsf{VDF}_{12\Delta}(\mathsf{Eval}, \mathsf{H}_{\mathsf{N}}(\phi^k), 1)$ to compute $\phi^{k+1}$. Multicast $\phi^k$.
- At time $(27 + 12(k-1)) \cdot \Delta$: For all $j \in [n]$, denote $\phi_j^k$ the proof received from party $P_j$ and do:
  - Set $\mathsf{bad}_j := \mathsf{true}$ if nothing was received from $P_j$ in this iteration.
  - If $k = 1$ and $\mathsf{VDF}_{13\Delta}(\mathsf{Verify}, \phi_j^k, \mathsf{H}_{\mathsf{N}}(\phi_j^0)) = 0$, set $\mathsf{bad}_j := \mathsf{true}$.
  - If $k = 1$ and $\mathsf{VDF}_{13\Delta}(\mathsf{Verify}, \phi_j^k, \mathsf{H}_{\mathsf{N}}(\phi_j^0)) = 1$, and $\neg\mathsf{bad}_j$ then $L := L \cup \{\mathsf{H}_{\mathsf{N}}(\phi_j^0)\}$.
  - If $k > 1$ and $\mathsf{VDF}_{12\Delta}(\mathsf{Verify}, \phi_j^k, \mathsf{H}_{\mathsf{N}}(\phi_j^{k-1})) = 0$, set $\mathsf{bad}_j := \mathsf{true}$.
  - If $k > 1$ and $\mathsf{VDF}_{12\Delta}(\mathsf{Verify}, \phi_j^k, \mathsf{H}_{\mathsf{N}}(\phi_j^{k-1})) = 1$ and $\neg\mathsf{bad}_j$ then $L := L \cup \{\mathsf{H}_{\mathsf{N}}(\phi_j^k)\}$.
- If $k = 1$: Output $\ell$, s.t. $\mathsf{H}_{\mathsf{N}}(\phi_\ell^0) = \min L$. Set $k := 2$.
- If $k > 1$: Output $\ell$, s.t. $\mathsf{H}_{\mathsf{N}}(\phi_\ell^k) = \min L$. Set $k := k + 1$.

**Fig. 6.** Leader election protocol $\Pi_{\mathsf{Leader}}$, where all parties have access to random oracles $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\kappa+\log(t_p)+2\log(\kappa)+1}$, $\mathsf{H}_{\mathsf{N}} : \{0,1\}^* \longrightarrow \mathsf{N}$ and an oracle $\mathsf{VDF}_\delta$. $\Pi_{\mathsf{Leader}}$ is run for $k \geq 1$ iterations when the byzantine agreement protocol $\Pi_{\mathsf{BA}}$, that invokes $\Pi_{\mathsf{Leader}}$ internally terminates after $k$ iterations. $\mathsf{N} = n + q \cdot (\lfloor \varkappa \rfloor - 1)$ (c.f Corollary 2). We set $\mathsf{N} = n + q$ (for adversarial speedup $\varkappa = 2$).

rounds of exchanging (unpredictable) challenges among parties. At the end of the second round of exchanges, $P$ creates a hash $\chi$ from these values. During the subsequent proof of computation phase, it then computes $\phi^0 = \mathsf{VDF}_{11\Delta}(\mathsf{Eval}, \{\chi || \mathsf{pk}\}, 1)$. When $\phi^0$ becomes available at time $13\Delta$, $P$ immediately starts computing $\phi^1$ as $\mathsf{VDF}_{13\Delta}(\mathsf{Eval}, \mathsf{H}_{\mathsf{N}}(\phi^0), 1)$. At time $16\Delta$, $P$ outputs $\mathsf{KeySet}$ in $\Pi_{\mathsf{KeyGrade}}$.

**Leader Election Phase.** The setup phase is followed by a leader election phase, which begins at time $26\Delta$ and is repeated until $P$ terminates $\Pi_{\mathsf{Leader}}$ from within the invocation of $\Pi_{\mathsf{BA}}$. (The time interval between these two phases is spent computing $\mathsf{VDF}_{13\Delta}(\mathsf{Eval}, \mathsf{H}_{\mathsf{N}}(\phi^0))$,1). $P$ keeps a flag $\mathsf{bad}_j$ (initialized to $\mathsf{false}$) for each key $\mathsf{pk}_j$ that it has previously accepted during $\Pi_{\mathsf{KeyGrade}}$. The purpose of $\mathsf{bad}_j$ is to indicate whether $P_j$ (the owner of $\mathsf{pk}_j$) has ever stopped investing computational effort during this run of $\Pi_{\mathsf{Leader}}$. Note that since $P$ has accepted $\mathsf{pk}_j$ in $\Pi_{\mathsf{KeyGrade}}$, it has already received a proof $\phi_j^0$ associated with $\mathsf{pk}_j$. This ensures that computational effort was invested with respect to $\mathsf{pk}_j$ up to time $13\Delta$. Below, $k$ denotes the current iteration of the protocol $\Pi_{\mathsf{BA}}$ and is initialized as $k := 1$. The leader election phase now proceeds in following two rounds:

– During the first round, $P$ completes the computation of $\phi^k$ and immediately commences computation of $\phi^{k+1}$ by calling $\mathsf{VDF}_{12\Delta}(\mathsf{Eval}, \mathsf{H_N}(\phi^k), 1)$. It multicasts $\phi^k$.
– In the second round, upon receiving $\phi_j^k$ from party $P_j$, $P$ verifies that $P_j$ has continuously been investing computational effort in $\Pi_{\mathsf{Leader}}$. To do so, it checks that $\neg\mathsf{bad}_j$ holds and that $\mathsf{VDF}_{12\Delta}(\mathsf{Verify}, \phi_j^k, \mathsf{H_N}(\phi_j^{k-1})) = 1$. If either of these conditions is violated (or nothing was received from $P_j$), $P$ sets $\neg\mathsf{bad}_j$ to indicate that $P_j$ has broken the chain of continuous computation from the beginning of $\Pi_{\mathsf{Leader}}$ and can never again be trusted as an honest leader.
– $P$ completes the iteration by computing the hash of every proof $\phi_j^k$ that it has received for which $\neg\mathsf{bad}_j$ still holds. It elects the party $P_\ell$ to be the leader if $\mathsf{H_N}(\phi_\ell^k)$ was the minimal value among all hashes. (Ties can be resolved by fixing some arbitrary rule in the protocol description).

We say that a party, upon receiving a proof $\phi_j^k$ from another party $P_j$, accepts it, provided it does not set $\mathsf{bad}_j = \mathsf{true}$. With this, we proceed to state Lemmas 10 to 12 below formally. To prove our results, we consider $(2, \frac{5}{11}, \beta)$, $(2, \frac{1}{6}, \beta)$, $(2, \frac{2}{13}, \beta)$ sequentiality of $\mathsf{VDF}_{11\Delta}$, $\mathsf{VDF}_{12\Delta}$ and $\mathsf{VDF}_{13\Delta}$ respectively in presence of a $(q, t_p, 2)$ adversary.

**Lemma 10.** *Let $\mathcal{A}$ be a $(q, t_p, 2)$-algorithm, where $q < \frac{n}{3}$ and suppose that $\mathsf{VDF}_{11\Delta}$, $\mathsf{VDF}_{12\Delta}$ and $\mathsf{VDF}_{13\Delta}$ are respectively $(2, \frac{5}{11}, \beta)$, $(2, \frac{1}{6}, \beta)$ and $(2, \frac{2}{13}, \beta)$ sequential. If $\Pi_{\mathsf{Leader}}$ is run at time 0, then with probability at least $1 - (2^{-\kappa-2\log(\kappa)-1} + \beta)$, for all $k \geq 1$, $\mathcal{A}$ outputs at most $\tau = 2q$ proofs $\phi_1^k, ..., \phi_\tau^k$ within time $(12 \cdot (k-1) + 26) \cdot \Delta$, such that for all $i \in [\tau]$, $\phi_i^k$ is accepted by at least one honest party in $\Pi_{\mathsf{Leader}}$.*

*Proof.* From the arguments of Lemma 3 and due to $(2, \frac{5}{11}, \beta)$ sequentiality of $\mathsf{VDF}_{11\Delta}$, $\mathcal{A}$ computes at most $\lfloor(2 + \frac{5\cdot 2}{11})\rfloor q = 2q$ proofs of the form $\phi_i^0 = \mathsf{VDF}_{11\Delta}(\mathsf{Eval}, \chi_i, \varkappa)$, $(\varkappa \leq 2)$, earliest at time $11\Delta$. Next, $\mathcal{A}$ can make at most $2q$ different calls of the form $\mathsf{VDF}_{13\Delta}(\mathsf{Eval}, \mathsf{H_N}(\phi_i^0), \varkappa)$ at $11\Delta$, which returns $2q$ proofs of the form $\phi_i^1$ at $24\Delta$. An honest party computes its $\phi^0$ and $\phi^1$ at time $13\Delta$ and $26\Delta$ respectively. Due to the $(2, \frac{2}{13}, \beta)$ sequentiality of $\mathsf{VDF}_{13\Delta}$, $\mathcal{A}$ cannot complete any additional proof $\phi_i^1$ within the remaining time $(26 - 24)\Delta = 2\Delta$. Applying the same argument inductively for $k \geq 2$ and $\mathsf{VDF}_{12\Delta}(\mathsf{Eval}, \mathsf{H}(\phi_i^{k-1}), \varkappa)$ completes the proof. □

Using the above lemma, we can now view the entire set of proofs that are accepted by at least one honest party in any iteration $k$ of $\Pi_{\mathsf{Leader}}$ as a vector of random variables $\mathbf{h}_k = (\phi_1^k, ..., \phi_p^k)$, where $p \leq n$. We now prove that $\mathbf{h}_k$ is unpredictable from the view of $\mathcal{A}$ before time $(25 + 12 \cdot (k-1)) \cdot \Delta$, for all $k \geq 1$.

**Lemma 11.** *Suppose that $\mathsf{VDF}_{11\Delta}, \mathsf{VDF}_{12\Delta}$, and $\mathsf{VDF}_{13\Delta}$ are $(2, \frac{5}{11}, \beta)$, $(2, \frac{1}{6}, \beta)$, $(2, \frac{2}{13}, \beta)$ sequential respectively. Then for all $k \geq 1$, the vector $\mathbf{h}_k$ is $(n - q, (25 + 12(k-1)) \cdot \Delta, 2^{-\kappa-2\log(\kappa)-1} + (k+1) \cdot \beta)$-unpredictable.*

*Proof.* We prove this statement by induction on $k$. From the arguments of Lemma 3, we know that the probability of a $(q, t_p, 2)$-algorithm guessing any element of $\mathbf{h}_0$ is at most $2^{-\kappa-2\log(\kappa)-1}+\beta$ before (earliest) time $11\Delta$. Thus, $\mathbf{h}_0$ is $(n-q, 11\Delta, 2^{-\kappa-2\log(\kappa)-1}+\beta)$-unpredictable and the base case $k=1$ follows directly from combining $(2, \frac{2}{13}, \beta)$-sequentiality of $\mathsf{VDF}_{13\Delta}$ with $(n-q, 11\Delta, 2^{-\kappa-2\log(\kappa)-1}+\beta)$-unpredictability of $\mathbf{h}_0$. For the step case, assume that $\mathbf{h}_k$ is $(n-q, (25+12(k-1))\cdot\Delta, 2^{-\kappa-2\log(\kappa)-1}+(k+1)\cdot\beta)$-unpredictable. Combining this with $(2, \frac{1}{6}, \beta)$-sequentiality of $\mathsf{VDF}_{12\Delta}$ immediately yields that $\mathbf{h}_{k+1}$ is $(n-q, (25+12(k-1))\cdot\Delta, 2^{-\kappa-2\log(\kappa)-1}+(k+2)\cdot\beta)$-unpredictable. $\square$

**Lemma 12.** *Assume that the conditions of Lemma 11 hold. Set* $\beta = 2^{-2\log\kappa-\kappa-2}$*. Then with probability at least* $\frac{1}{2}$ *and for all* $1 \le k \le \kappa$*, all honest parties output* $\ell$ *in* $\Pi_{\mathsf{Leader}}$ *such that* $P_\ell$ *is honest at time* $(12(k-1)+24)\cdot\Delta$*.*

*Proof.* For $k \ge 1$ denote $E_k$ the event that $\mathcal{A}$ queries $\mathsf{H}$ on an element of $\mathbf{h}_k$ before time $(12(k-1)+25)\cdot\Delta$. By Lemma 11, $E_k$ occurs with probability at most $2^{-\kappa-2\log(\kappa)-1}+(k+1)\cdot\beta$. By a union bound, $E := \bigcup_{1\le k\le\kappa} E_k$ occurs with probability at most $2\kappa^2\cdot\beta + \kappa\cdot 2^{-\kappa-2\log(\kappa)-1} \le 2\cdot 2^{-\kappa-1} \le 2^{-\kappa}$. Unless $E$ occurs, the values $\mathsf{H}_\mathsf{N}(\mathbf{h}_{k,i})$ are uniformly random values in the range $[\mathsf{N}]$ from the view of $\mathcal{A}$ for all $1 \le i \le \mathsf{N}$ at time $(12(k-1)+25)\cdot\Delta$. Hence, conditioned on $\neg E$, the component $\ell$ which minimizes $\mathsf{H}_\mathsf{N}(\mathbf{h}_{k,\ell})$ corresponds to an honest party with probability at least $\frac{n-q}{\mathsf{N}}$, where $\mathsf{N} = n+q$. Overall, $\ell$ corresponds to an honest party at time $(12(k-1)+24)\cdot\Delta$ with probability at least $\frac{n-q}{\mathsf{N}} = \frac{n-q}{n+q} \ge \frac{1}{2}$ (since $q < \frac{n}{3}$). $\square$

## 6 Communication Complexity in the VDF Model

In this section, we provide the first lower bound for the communication complexity of Byzantine broadcast in the multicast model (c.f. Definition 10) in presence of a VDF oracle. More concretely, inspired by [ACD+19, Section 7], we consider a setting without a PKI where parties are connected via multicast channels. In addition – and in contrast to previous work [ACD+19] – we assume that parties have access to a VDF oracle. This adds additional technical challenges to the analysis. To state our theorem, we refine our definition of Byzantine broadcast to make failure probabilities and communication complexity of the protocol explicit.

**Definition 10 (Byzantine Broadcast in the Multicast Model).** *Consider a protocol that is executed between* $n$ *parties, where a designated sender* $S$ *holds an input* $x_S$ *at the beginning of the protocol and all parties output upon terminating. We call this a* $(q,p)$-secure protocol for Byzantine broadcast *with multicast complexity* $\Theta$*, if the following properties hold (simultaneously) with probability at least* $p$ *when at most* $q$ *parties are adaptively corrupted:*

- ***Consistency:*** *Every honest party* $P_i$ *outputs the same value* $x_i = x$*.*
- ***Validity:*** *If the sender* $S$ *is honest, then all honest parties output* $x_i = x_S$*.*
- ***Termination:*** *All parties terminate.*
- ***Multicast Complexity:*** *The multicast complexity of the protocol is at most* $\Theta$*.*

**Theorem 4.** *Let $c = O(1)$ and $n \geq (64c^2 + 2c)$. Then there is no $(q, p)$-secure protocol for Byzantine broadcast (among $n$ parties) with $\left\lfloor \frac{\sqrt{2q}}{8} \right\rfloor$ multicast complexity (relative to a VDF oracle $\mathsf{VDF}$), when $p > \frac{19}{20}$ and $q = \frac{n}{2} - c$.*

Our proof is inspired by the lower bound of [ACD+19] (Section 7) for Byzantine broadcast without a PKI in the multicast model. (Note that our lower bound for Byzantine broadcast implies a lower bound for Byzantine agreement.) The goal of our proof is to show that the view of a special (honest) party $P$ that is *not* the sender $S$ can be made identical in a protocol execution where the input bit of $S$ is either 0 or 1. This leads to a violation of the consistency property of Byzantine agreement. For the formal proof, we define four worlds: $\mathsf{World}_{c,b}$, $\mathsf{World}_{c,1-b}$, $\mathsf{World}_{h,b}$ and $\mathsf{World}_{h,1-b}$. In $\mathsf{World}_{c,*}$, we consider a protocol execution where the special party $P$ is statically corrupted, whereas in $\mathsf{World}_{h,*}$ this party remains honest throughout the protocol. Moreover, in $\mathsf{World}_{*,b}$, the sender has input bit $b$.

To show the violation of the consistency property, we then proceed as follows. In both worlds, $\mathsf{World}_{c,b}$ and $\mathsf{World}_{h,b}$, we show that honest parties have the same view. Moreover, the special party $P$ acts in both worlds as if it receives messages according to world $\mathsf{World}_{*,b}$ and $\mathsf{World}_{*,1-b}$. Notice that in world $\mathsf{World}_{c,b}$, this can be done since $P$ is statically corrupted, and hence it can be instructed by the adversary to behave accordingly. In world $\mathsf{World}_{h,b}$, this is done by *adaptively* corrupting parties that multicast messages in a certain round, and instructing the freshly corrupted party to also multicast messages according to the (honest) party $P$'s view of world $\mathsf{World}_{h,1-b}$. (This is also the reason for restricting the number of multicasts in the theorem, since each multicast requires to corrupt the party that multicast). This confuses $P$ as to which world it is actually being run in, and hence the honest $P$ in worlds $\mathsf{World}_{h,1-b}$ and $\mathsf{World}_{h,b}$ behaves as the malicious party in worlds $\mathsf{World}_{c,1-b}, \mathsf{World}_{c,b}$. It is now possible to show that with high probability, $P$'s confusion leads to it outputting an inconsistent bit in one of these worlds.

A crucial difference between our setting and the setting of [ACD+19] is the way in which the adversary collapses the views of worlds $\mathsf{World}_{h,1-b}$ and $\mathsf{World}_{h,b}$ from that of $P$. In a nutshell, this requires the adversary to simulate the execution of the protocol in one of these worlds. Unfortunately, when parties have access to a $\mathsf{VDF}$ oracle, a simple simulation strategy ceases to work. At a high level, in $\mathsf{VDF}$-based protocols, the simulation depends on oracle queries to the $\mathsf{VDF}$ oracle, and hence can only be completed if the adversary has sufficient query budget for the $\mathsf{VDF}$. In our simulation, we achieve this by letting the adversary statically corrupt some set of parties which do not participate in the protocol (the adversary crashes them in every one of the worlds). We can then use their $\mathsf{VDF}$ oracle budget to complete the simulation for those parties who do participate.

*Proof (Of Theorem 4).* Suppose for the sake of contradiction, there exists a $(q, p)$-secure protocol $\Pi$ for Byzantine broadcast with $\frac{\sqrt{2q}}{8}$ multicast complexity such that $p > \frac{19}{20}$, $q = \frac{n}{2} - c$, and $n \geq (64c^2 + 2c)$. We proceed by presenting the strategy of an adversary $\mathcal{A}$ that violates consistency of $\Pi$ with probability at least $\frac{1}{20}$. Throughout the rest of the description, we denote $\varrho = \left\lfloor \frac{\sqrt{2q}}{8} \right\rfloor$.

We explain $\mathcal{A}$'s strategy separately for each of the worlds $\mathsf{World}_{c,b}$, $\mathsf{World}_{h,b}$ as introduced above. In each of the worlds, the adversary statically corrupts an arbitrary set of $(q-\varrho)$ parties $\mathcal{R}$ that does not include $P$ or the sender $S$ at the beginning of the execution of $\Pi$ in that world. These parties behave as if they are crashed (i.e., they never send any messages). We remark that $\mathcal{R}$ is fixed through all worlds. Furthermore, let us denote $C$ as the event that two distinct (but possibly dependent) executions of $\Pi$ satisfy validity and consistency and have multicast complexity at most $\varrho$. The following lemma lower bounds the probability of the event $C$.

**Lemma 13.** *Let the event $C$ be defined as above. Then* $\Pr[C] \geq 2p-1$.

*Proof.* Let $A_1$ and $A_2$ denote the events that two (possibly dependent) executions (labeled one and two for the purpose of this lemma) of protocol $\Pi$ (in any of the worlds) achieve Byzantine Broadcast and have $\varrho$ multicast complexity. By assumption, $\Pr[A_1] \geq p$ and $\Pr[A_2] \geq p$, and hence $\Pr[C] = \Pr[A_1 \cap A_2] \geq \Pr[A_1] + \Pr[A_2] - 1 = 2p - 1$. $\square$

**Behavior in $\mathsf{World}_{c,b}$:** $\mathcal{A}$ statically corrupts the parties in $\mathcal{R}$ and uses their computational resources in the simulation of $\mathsf{World}_{c,1-b}$. It corrupts one additional special party $P$ (which is not the sender $S$) and directs $P$ to behave honestly in $\mathsf{World}_{c,b}$ and the simulation as if it were receiving messages from two executions of the protocol in which the sender holds either 0 or 1. $P$'s precise strategy is described below. The remaining $(n+\varrho-q)-1$ parties remain honest throughout the execution $\mathsf{World}_{c,b}$ (including the sender $S$). Denote this set of parties as $\mathcal{L}$.

In more detail, $\mathcal{A}$'s strategy is as follows.

- $\mathcal{A}$ chooses random coins for all parties in $\mathcal{L}$ and simulates an execution of $\Pi$ where the sender holds input $1-b$, the parties in $\mathcal{R}$ are crashed throughout the execution of $\Pi$, and the only other corrupted party is $P$. (In other words, $\mathcal{A}$ simulates $\mathsf{World}_{c,1-b}$).
- It selects $(q-\varrho)$ parties in the set $\mathcal{L}$ uniformly at random.
- If the simulation directs a party $Q \in \mathcal{L}$ to query $\mathsf{VDF}$, $\mathcal{A}$ instructs a party $Q' \in \mathcal{R}$ to make the same query (unless that party is already waiting $\mathsf{VDF}$ to reply to a prior query). When $\mathsf{VDF}$ returns $\phi$ to $Q'$, the adversary returns $\phi$ to $Q$ in the simulation.
- The party $P$ behaves as if it receives messages from both $\mathsf{World}_{c,b}$ and the simulation that it is running in its head. It reacts to these messages as an honest party $P$ would do in an execution of $\Pi$ where everybody holds input $b$.
- If $P$ sends a message in $\mathsf{World}_{c,b}$ or in the simulation, $\mathcal{A}$ delivers this message to all honest parties in $\mathsf{World}_{c,b}$ and in the simulation.

Observe that for a small set of $(n-2q+2\varrho)$ parties in $\mathcal{L}$, $\mathcal{A}$ is not able to simulate the $\mathsf{VDF}$ calls in the simulation (it can only simulate such calls for $|\mathcal{R}|$ many parties in $\mathcal{L}$). Denote this set of parties by $\mathcal{U}$. Clearly, the simulation of the adversary fails if any party from $\mathcal{U}$ attempts to multicast a message in $\Pi$ within the first $\varrho$ multicasts. Let $F_1$ denote the event that simulation of the adversary fails in $\mathsf{World}_{c,b}$, (conditioned on the event $C$). Lemma 14 below bounds the probability $\Pr[F_1|C]$.

**Lemma 14.** *Let $F_1$ denote the event that $\mathcal{A}$'s simulation fails in* $\mathsf{World}_{c,b}$. *Then* $\Pr[F_1|C] < \frac{1}{6}$.

*Proof.* To bound the probability of event $F_1$ (conditioned on $C$), we first define the following events. Let $S$ denote the event that a uniformly chosen party $P_i \in \mathcal{L}$ ever multicasts in the simulation. Conditioned on $C$, the simulation directs parties to multicast at most $\varrho$ many times. Since party $P_i$ is chosen uniformly from the set $\mathcal{L}$, $\Pr[S|C] = \frac{\varrho}{|\mathcal{L}|}$. Now, observe that the set $\mathcal{U}$ is a uniformly created subset of set $\mathcal{L}$ chosen by the adversary according to its simulation strategy. By the previous calculation, the probability that any particular party in set $\mathcal{U}$ ever multicasts in the simulation coincides with the probability of event $S$. Now, let $T_1$ denote the event that at least party uniformly chosen from set $\mathcal{U}$ ever multicasts in the protocol. By a union bound, we see that

$$\Pr[T_1|C] \le \sum_{i=1}^{|\mathcal{U}|} \Pr[S|C].$$

Hence, with multicast complexity up to $\varrho$, $\Pr[T_1|C] \le \sum_{i=1}^{|\mathcal{U}|} \Pr[S|C]$. Since $F_1$ can only occur as a result of an unsimulated party attempting to multicast when we have conditioned on $C$, we can infer that $\Pr[F_1|C] = \Pr[T_1|C]$.

Now,

$$\Pr[F_1|C] \le \sum_{i=1}^{|\mathcal{U}|}(\Pr[S|C]) = \frac{\varrho \cdot |\mathcal{U}|}{n-q+\varrho} = (n-2q+\frac{\sqrt{2q}}{4}) \cdot \frac{\lfloor \frac{\sqrt{2q}}{8} \rfloor}{(n-q+\frac{\sqrt{2q}}{8})})$$
$$\le \frac{\frac{\sqrt{2q}}{8}(n-2t+\frac{\sqrt{2q}}{4})}{(n-q+\frac{\sqrt{2q}}{8})}.$$

By setting $t = \frac{n}{2} - c$, for $c = O(1)$, we bound $\frac{(n-2t+\frac{\sqrt{2t}}{4})}{(n-t+\frac{\sqrt{2t}}{8})}$ as

$$\frac{(2c+\frac{\sqrt{n-2c}}{4})}{\frac{n}{2}+c+\frac{\sqrt{n-2c}}{8}} = \frac{(2c+\frac{\sqrt{n-2c}}{4})}{\frac{n}{2}-c+\frac{\sqrt{n-2c}}{8}+2c} = \frac{\frac{\sqrt{n-2c}}{4}(\frac{8c}{\sqrt{n-2c}}+1)}{\frac{n-2c}{2}+\frac{\sqrt{n-2c}}{8}+2c}$$
$$= \frac{\frac{\sqrt{n-2c}}{4}(\frac{8c}{\sqrt{n-2c}}+1)}{\frac{\sqrt{n-2c}}{8}(4\sqrt{n-2c}+1+\frac{16c}{\sqrt{n-2c}})} = \frac{2(\frac{8c}{\sqrt{n-2c}}+1)}{(4\sqrt{n-2c}+1+\frac{16c}{\sqrt{n-2c}})} < \frac{2\cdot 2}{3\sqrt{n}} = \frac{4}{3\sqrt{n}},$$

where the last inequality holds for $n \ge (64c^2 + 2c)$. By substituting $q = (\frac{n}{2} - c)$ in $\frac{\sqrt{2q}}{8}$, we finally obtain $\Pr[F_1|C] \le \frac{\sqrt{n-2c}}{8} \cdot \frac{4}{3\sqrt{n}} = \frac{\sqrt{n-2c}}{6\sqrt{n}} < \frac{1}{6}$. □

Note that since the number of corrupted parties is strictly less than $\frac{n}{2}$, by the validity property of Byzantine broadcast, all the honest parties in $\mathsf{World}_{c,b}$ output bit $b$ with probability greater than $p$ in case the failure event $F_1$ does not occur.

**Behaviour in $\mathsf{World}_{h,b}$:** Initially, $\mathcal{A}$ statically corrupts the parties in $\mathcal{R}$ which will be used as the resource to simulate $\mathsf{World}_{h,1-b}$. The remaining $(n + \varrho - q)$ parties (excluding crashed parties in $\mathcal{R}$) is denoted by set $\mathcal{L}'$. Note that the special party $P$ (which is not the sender $S$) is not among the aforementioned statically corrupted parties and remains honest throughout the protocol, i.e., $P \in \mathcal{L}'$. Note that the sender $S \in \mathcal{L}' \setminus P$.

$\mathcal{A}$ now simulates the world $\mathsf{World}_{h,1-b}$ for all parties in set $\mathcal{L}' \setminus \{P\}$ as follows.

- $\mathcal{A}$ chooses random coins for all parties in $\mathcal{L}' \setminus \{P\}$ and simulates $\mathsf{World}_{h,1-b}$. More precisely, it simulates an execution of $\Pi$ where the sender holds input $1-b$ and the parties in $\mathcal{R}$ are crashed throughout the execution of $\Pi$ (the remaining parties act honestly).
- It selects $(q - \varrho)$ parties in set $\mathcal{L}'$ uniformly at random.
- When the simulation directs a party $Q \in \mathcal{L}' \setminus \{P\}$ to multicast in some round $r$, $\mathcal{A}$ adaptively corrupts $Q$ in round $r$ of the real execution of $\Pi$ (i.e., in $\mathsf{World}_{h,b}$), unless $Q$ is already corrupted. Note that the designated sender $S$ might get corrupted in this step.
- A corrupted party $Q$ continues to send honest messages in $\mathsf{World}_{h,b}$ to all remaining parties in $\mathcal{L}'$ but it forwards to $P$ all messages from both $\mathsf{World}_{h,b}$ and $\mathsf{World}_{h,1-b}$.
- To produce the simulated messages of $\mathsf{World}_{h,1-b}$, when the simulation directs a party $Q \in \mathcal{L}' \setminus P$ to query $\mathsf{VDF}$, $\mathcal{A}$ acts as follows: it instructs a party $Q' \in \mathcal{R}$ to make the same query (unless that party is already waiting for the $\mathsf{VDF}$ to reply to a prior query). When the $\mathsf{VDF}$ returns $\phi$ to $Q'$, the adversary returns $\phi$ to $Q$ in the simulation.
- When $P$ multicasts a message in $\mathsf{World}_{h,b}$, that message is also multicast in the simulation.

Observe that there is a gap of $(n - 2q + 2\varrho) - 1$ parties for which the adversary could not simulate, but it might be the case that one of these parties want to speak in the protocol. Denote the set of these parties as $\mathcal{U}'$. Let $F_2$ denote the event that the simulation of the adversary fails. For $q = (\frac{n}{2} - c)$, Lemma 15 below proves that $\Pr[F_2|C] < \frac{1}{6}$.

**Lemma 15.** *Let $F_2$ denote the event that $\mathcal{A}$'s simulation fails in $\mathsf{World}_{h,b}$. Then $\Pr[F_2|C] < \frac{1}{6}$.*

The proof of Lemma 15 is similar to that of 14.

We now state two technical lemmas below. We first give an indistinguishability lemma about the worlds $\mathsf{World}_{c,b}$ and $\mathsf{World}_{h,b}$ for forever honest parties.

**Lemma 16.** *Conditioned on the events $C$ and $\neg F_1$, $\mathsf{World}_{c,b}$ is indistinguishable from $\mathsf{World}_{h,b}$ for parties that are forever honest in both $\mathsf{World}_{c,b}$ and $\mathsf{World}_{h,b}$.*

*Proof.* The statement holds due to the following reasons. 1) In $\mathsf{World}_{c,b}$, the forever honest parties always receive honest messages. 2) In $\mathsf{World}_{c,b}$, parties that are adaptively corrupted by $\mathcal{A}$ send correct messages to the honest parties except for party $P$. 3) The behavior of the corrupted party $P$ in $\mathsf{World}_{c,b}$ is exactly like that of honest party $P$ in $\mathsf{World}_{h,b}$. 4) The multicast complexity in both worlds $\mathsf{World}_{c,b}$ and $\mathsf{World}_{h,b}$ is at most $\varrho$, according to the definition of the event $C$. Therefore, conditioned on events $C$ and $\neg F_1$, the views of the parties that are forever-honest in $\mathsf{World}_{c,b}$ and $\mathsf{World}_{h,b}$ are identically distributed. □

Next, we give an indistinguishability lemma about the worlds $\mathsf{World}_{h,b}$ and $\mathsf{World}_{h,1-b}$ for party $P$.

**Lemma 17.** *Conditioned on $C$ and $\neg F_2$, $\mathsf{World}_{h,b}$ is indistinguishable from $\mathsf{World}_{h,1-b}$ for party $P$.*

*Proof.* The statement of the lemma holds due to the following reasons. 1) In $\mathsf{World}_{h,b}$, $P$ always receives messages of both worlds ($\mathsf{World}_{h,0}$ and $\mathsf{World}_{h,1}$) from adaptively corrupted parties. 2) Both the worlds $\mathsf{World}_{h,0}$ and $\mathsf{World}_{h,1}$ have multicast complexity at most $\varrho$ according to the definition of the event $C$. Therefore, conditioned on events $C$ and $\neg F_2$, the views of the party $P$ in the worlds $\mathsf{World}_{h,0}$ and $\mathsf{World}_{h,1}$ are identically distributed. □

In $\mathsf{World}_{c,b}$, since the sender $S$ is honest, the validity property of Byzantine broadcast implies that all honest parties output $b$. Using the indistinguishability between $\mathsf{World}_{c,b}$ and $\mathsf{World}_{h,b}$ guaranteed by Lemma 16, we can use consistency to ensure that party $P$ which is honest in $\mathsf{World}_{h,b}$, outputs $b$ in world $\mathsf{World}_{h,b}$. We formalize this intuition in the following Lemma.

**Lemma 18.** *Let $Y$ denote the event that the forever honest parties in $\mathsf{World}_{h,b}$ output $b$. Then $\Pr[Y|C \cap \neg F_1] = p$.*

*Proof.* Since the sender $S$ is honest in $\mathsf{World}_{c,b}$, by the validity property of the Byzantine Broadcast, all the forever honest parties output $b$ with at least probability $p$ in $\mathsf{World}_{c,b}$. From Lemma 16, $\mathsf{World}_{c,b}$ is indistinguishable from $\mathsf{World}_{h,b}$ for forever honest parties. We infer that $\Pr[Y|C \cap \neg F_1] \geq p$. □

Similarly, using indistinguishability between $\mathsf{World}_{h,b}$ and $\mathsf{World}_{h,1-b}$ guaranteed by Lemma 17, we obtain:

**Lemma 19.** *Let $X$ denote the event that $P$ does not output 1 in $\mathsf{World}_{h,1}$. Then $\Pr[X|C \cap \neg F_2] = \frac{1}{2}$.*

*Proof.* By Lemma 17, conditioned on events $C$ and $\neg F_2$, the views of the party $P$ in the worlds $\mathsf{World}_{h,0}$ and $\mathsf{World}_{h,1}$ are identically distributed. Therefore, conditioned on the events $C$ and $\neg F_2$, the probability of $P$ not outputting 1 in $\mathsf{World}_{h,1}$ is given as $\Pr[X|C \cap \neg F_2] \geq \frac{1}{2}$. □

Thus, the probability that consistency of the Byzantine broadcast is violated in $\mathsf{World}_{h,1}$ is at least $\Pr[X \cap Y]$ which we bound from below as follows:

$$\begin{aligned}\Pr[X \cap Y] &= \Pr[X] + \Pr[Y] - 1 \\ &\geq \Pr[X \cap C \cap \neg F_2] + \Pr[Y \cap C \cap \neg F_1] - 1 \\ &= \Pr[X|C \cap \neg F_2] \cdot \Pr[C \cap \neg F_2] + \Pr[Y|C \cap \neg F_1] \cdot \Pr[C \cap \neg F_1] - 1 \\ &= \Pr[X|C \cap \neg F_2] \cdot \Pr[\neg F_2|C] \cdot \Pr[C] + \Pr[Y|C \cap \neg F_1] \cdot \Pr[\neg F_1|C] \cdot \Pr[C] - 1 \\ &= \frac{1}{2} \cdot \frac{5}{6} \cdot (2p-1) + p \cdot \frac{5}{6} \cdot (2p-1) - 1 = \frac{20p^2 - 17}{12} \\ &= \frac{7}{80} > \frac{1}{20} = 1 - p.\end{aligned}$$

This contradicts the supposition that $\Pi$ achieves consistency with probability more than $\frac{19}{20}$ and within complexity $\varrho$. □

## References

ACD+19. Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 317–326. ACM, July / August 2019.

AD15. Marcin Andrychowicz and Stefan Dziembowski. PoW-based distributed cryptography with no trusted setup. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 379–399. Springer, Heidelberg, August 2015.

ADD+19. Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 320–334. Springer, Heidelberg, February 2019.

AMSZ19. Abhinav Aggarwal, Mahnush Movahedi, Jared Saia, and Mahdi Zamani. Bootstrapping public blockchains without a trusted setup. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 366–368. ACM, July / August 2019.

BBBF18. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

But13. Vitalik Buterin. Ethereum white paper. 2013.

CGZ21. Ran Cohen, Juan Garay, and Vassilis Zikas. Adaptively secure broadcast in resource-restricted cryptography. Cryptology ePrint Archive, Report 2021/775, 2021. https://eprint.iacr.org/2021/775.

CP19. Bram Cohen and Krzysztof Pietrzak. The chia network blockchain. Technical report, Chia Network, 2019.

Dou02. John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

DS83. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

FLL21. Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. A new way to achieve round-efficient byzantine agreement. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 355–362. ACM, 2021.

FM88. Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, May 1988.

FM00. Matthias Fitzi and Ueli M. Maurer. From partial consistency to global broadcast. In *32nd ACM STOC*, pages 494–503. ACM Press, May 2000.

FMPS19. Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 248–277. Springer, 2019.

GKL15. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

GKLP18. Juan A. Garay, Aggelos Kiayias, Nikos Leonardos, and Giorgos Panagiotakos. Bootstrapping the blockchain, with applications to consensus and fast PKI setup. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 465–495. Springer, Heidelberg, March 2018.

GKO$^+$20. Juan A. Garay, Aggelos Kiayias, Rafail M. Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2020.

GKPS18. Chaya Ganesh, Yashvanth Kondi, Arpita Patra, and Pratik Sarkar. Efficient adaptively secure zero-knowledge from garbled circuits. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 499–529. Springer, Heidelberg, March 2018.

GMR84. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical" solution to the signature problem (extended abstract). In *25th FOCS*, pages 441–448. IEEE Computer Society Press, October 1984.

KK06a. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Heidelberg, August 2006.

KK06b. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. Cryptology ePrint Archive, Report 2006/065, 2006. https://eprint.iacr.org/2006/065.

KKKZ19. Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 157–174. IEEE, 2019.

KLX20. Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 390–413. Springer, Heidelberg, November 2020.

KMS14. Jonathan Katz, Andrew Miller, and Elaine Shi. Pseudonymous broadcast and secure computation from cryptographic puzzles. Cryptology ePrint Archive, Report 2014/857, 2014. https://eprint.iacr.org/2014/857.

Mic17. Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 6:1–6:1, 67, January 2017. LIPIcs.

MV17. Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. 2017.

Nak08. Bitcoin: A peer-to-peer electronic cash system. 2008.

Pie19. Krzysztof Pietrzak. Proofs of catalytic space. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 59:1–59:25. LIPIcs, January 2019.

PSs17. Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, April / May 2017.

Rab83. Michael O. Rabin. Randomized byzantine generals. In *24th FOCS*, pages 403–409. IEEE Computer Society Press, November 1983.

vBS21. Aron van Baarsen and Marc Stevens. On time-lock cryptographic assumptions in abelian hidden-order groups. *IACR Cryptol. ePrint Arch.*, page 1184, 2021.

Wes19. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

WXDS20. Jun Wan, Hanshen Xiao, Srinivas Devadas, and Elaine Shi. Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC,* *USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 412–456. Springer, 2020.

# E. Distributed Password-Authenticated Symmetric-key Encryption

This chapter corresponds to our published article in AsiaCCS 2022 [48], with minor edits. Our full version can be found in [47].

[48] P. Das, J. Hesse, and A. Lehmann. “DPaSE: Distributed Password-Authenticated Symmetric-Key Encryption, or How to Get Many Keys from One Password”. In: *ASIA CCS ’22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*. 2022, pp. 682–696. DOI: 10.1145/3488932.3517389. URL: https://doi.org/10.1145/3488932.3517389.

# DPaSE: Distributed Password-Authenticated Symmetric-Key Encryption, or How to Get Many Keys from One Password

Poulami Das[1], Julia Hesse[2], and Anja Lehmann[3]

[1] TU Darmstadt, Germany
[2] IBM Research Zurich, Switzerland
[3] Hasso-Plattner-Institute, University of Potsdam, Germany

**Abstract.** Cloud storage is becoming increasingly popular among end users that outsource their personal data to services such as Dropbox or Google Drive. For security, uploaded data should ideally be encrypted under a key that is controlled and only known by the user. Current solutions that support user-centric encryption either require the user to manage strong cryptographic keys, or derive keys from weak passwords. While the former has massive usability issues and requires secure storage by the user, the latter approach is more convenient but offers only little security since encrypted data is susceptible to offline attacks. The recent concept of password-authenticated secret-sharing (PASS) enables users to securely derive strong keys from weak passwords by leveraging a distributed server setup, and has been considered a promising step towards secure and usable encryption. However, using PASS for encryption is not as suitable as originally thought: it only considers the (re)construction of a *single*, static key – whereas practical encryption will require the management of *many*, object-specific keys. Using a dedicated PASS instance for every key makes the solution vulnerable against online attacks, inherently leaks access patterns to the servers and poses the risk of permanent data loss when an incorrect password is used at encryption. We therefore propose a new protocol that directly targets the problem of boostrapping encryption from a single password: distributed password-authenticated symmetric encryption (DPaSE).
DPaSE offers strong security and usability, such as protecting the user's password against online and offline attacks, and ensuring message privacy and ciphertext integrity as long as at least one server is honest. We formally define the desired security properties in the UC framework and propose a provably secure instantiation. The core of our protocol is a new type of Oblivious Pseudorandom Function (OPRF) that allows to extend a previous partially-blind query with a follow-up request and will be used to blindly carry over passwords across evaluations and avoid online attacks. Our (proof-of-concept) implementation of DPaSE uses 10 exponentiations at the user, 4 exponentiations and 2 pairings at each server, and has a server throughput of 76 account creations and 37 (user authentication followed by) encryptions per second, when run between a user and 2-10 servers.

## 1 Introduction

Outsourcing storage to cloud providers is not only a common approach in enterprise settings, but is also widely appreciated by end users relying on services such as Dropbox, Google Drive, iCloud or Microsoft OneDrive to manage their personal data. With data breaches happening on a daily basis, it is essential that personal data kept in such cloud storage must be protected accordingly. The prevalent approach is to trust the cloud with properly encrypting the data, where the service provider controls access to the respective encryption keys via standard user authentication, mostly relying on username-password authentication. Clearly, such a solution crucially relies on the honesty of the service provider who can otherwise gain plaintext access to the users' data.

A different approach is let the user already encrypt the data before storing it in the cloud, which is offered e.g., by Tresorit [4] or Mega [3]. Therein a user client is locally encrypting the data and only uploads ciphertexts to the cloud. The cryptographic keys are either generated and stored directly by the user client, or (re)-derived from a human-memorizable password that the user enters into the local client. The former provides strong security guarantees, but is cumbersome to use as it relies on users' being able to manage and securely store cryptographic keys. The latter provides (roughly) the same convenience and usability as standard cloud provides as it does not require secure storage on the user side, but is inherently vulnerable to so-called offline attacks: Since encryption keys are derived from a low-entropy password, a corrupt service provider or an attacker gaining access to the ciphertexts, can attempt to decrypt the files by guessing the user's password.

While recently some service providers have moved away from password and deploy solutions where users are required to store key material (e.g., [2]), password-based systems remain the only truly device-independent solution at our disposal. In this paper, we investigate how users can password-encrypt their cloud data *without* storing any key material, and *without* making their encrypted data prone to offline password-guessing attacks.

***Known approaches to password-based encryption.*** One way to avoid the two aforementioned issues is to use a *distributed password-based key management* system: a user retrieves her encryption key from a set of servers, using only a password as input. This does not require the user to store any cryptograhpic material, since the servers take over this role, and the distribution of keys among servers thwarts of offline attacks on the password. There exist various cryptographic primitives suitable to implement such password-protected key retrieval, for example Password-Authenticated Secret Sharing (PASS/PPSS) [6] and Oblivious Key Management [19], which we discuss more in related work below.

All aforementioned schemes allow users to turn a password into an encryption key. In practice, this means that users either encrypt all their data with *the same key*, or they must memorize *as many passwords as keys* that they want to use. For optimal usability *and* security, in a password-based key management scheme, we want to ask the user to remember only *few but strong* passwords, and "behind the scenes" still use different encryption keys for every piece of data she wants to encrypt. Varying encryption keys is desirable to mitigate the effect of security breaches of the user's device, or of irresponsible handling of secret keys on the user side. We note that there exist other ways to mitigate the effect of such attacks, for example allowing for efficient updates of the encryption key, which however provide no protection in case the attacker is already in posession of ciphertexts. In this work, we prefer one-time usage of encryption keys over updatability, since then revelance of one key upon compromise does not impact the confidentiality of more than one encrypted piece of data.

## 1.1 Our Contributions

In this work we develop usable yet strongly secure distributed password-based symmetric encryption (DPaSE). DPaSE allows users to securely and conveniently encrypt and decrypt their data with different encryption keys while relying only on a single password and the assistance of $n$ servers. We provide an efficient realization based on a new type of Oblivious Pseudorandom Function (OPRF) that supports correlated evaluations of blind inputs, which we believe to be of independent interest. DPaSE is not a mere key

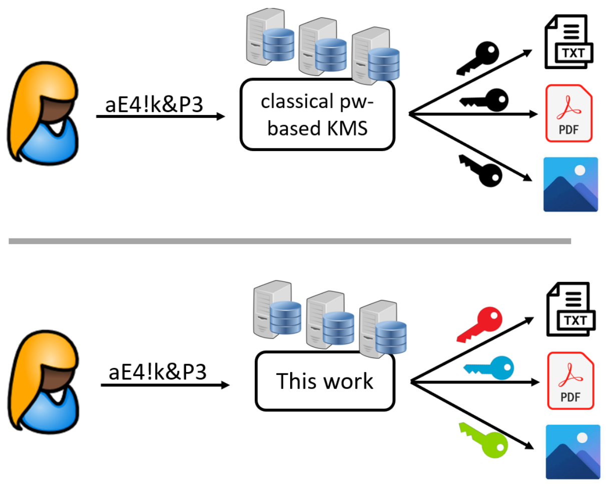

**Fig. 1.** Classical password-based server-assisted KMS yields one key per password to encrypt all the different user data. In this work we develop a server-assisted encryption scheme that allows to derive different encryption keys from only one password.

management system, but has built-in encryption of data with the retrieved keys already. Encryption and decryption is carried out locally by the user using the retrieved key. This integrated modeling allows us to demand the following strong security and functionality from a DPaSE system, covering guarantees with respect to both passwords *and* encryption of data.

**Correct Encryption:** If a user types an incorrect password upon encryption, her data is not encrypted and the user instead obtains an error message. This property is important to avoid that a user accidentally encrypts her data with unrecoverable secret keys.

**No Reuse of Keys:** Every ciphertext is created with an individual key. Hence, in case a user loses one of her encryption keys, all but one of her encrypted files remain confidential.

**Security against Offline Attacks:** As long as at least one server is honest, the encrypted data (or rather the underlying password) cannot be offline attacked. And even if eventually *all* servers are corrupted, they cannot decrypt the data immediately but must still perform an offline attack on the password – thus when users have chosen strong passwords, their data remains secure.

**Security against Online Attacks:** To detect and prevent online guessing attacks, the servers learn which user is trying to encrypt or decrypt, and whether her entered password was correct. In particular, we require that every file access/decryption requires explicit approval of all servers. When an honest server has recognized suspicious behaviour or was alerted by the user herself, it can enforce user-specific rate limiting or even fully block a certain account.

**Obliviousness:** Servers do not learn anything about the files (plain- or ciphertext) the user wants to access[4]. It was demonstrated [15, 24] that such leakage would have devastating effects on the user's privacy.

**Authenticated Encryption:** An adversary cannot plant wrong information into the outsourced storage. Thus, unless the adversary knows the user's password (and is assisted by all servers) it must be infeasible to create valid ciphertexts.

***Security Model in the UC Framework.*** We formally define these properties by means of an ideal functionality $\mathcal{F}_{\mathsf{DPaSE}}$ using the Universal Composability (UC) framework [12], which is known to allow for the most realistic modeling for how users (mis)handle passwords. In game-based security models, users choose their passwords at random from known distributions and are assumed to behave perfectly, i.e., never make a typo when using a password. This clearly does not reflect reality, where users share or re-use passwords, and make mistakes when typing them. The UC framework models that much more naturally as therein the environment provides the passwords. Thus, a UC security notion guarantees the desired security properties without making any assumptions regarding the passwords' distributions or usages. Our modeling also ensures that any DPaSE protocol is secure when executed concurrently with other systems, thanks to the strong composability guarantees of the UC framework.

However, these desirable features come at a cost. In order to end up with a manageable and understandable security definition (i.e., UC functionality), we need to make compromises and protect against some attacks that might not be of high relevance to DPaSE in practise[5]. For example, we need to prevent servers from intentionally deriving encryption keys from wrong passwords, which makes our protocol a bit more costly and restricted to security against semi-honest servers. There exist many ways of protecting against such attacks, each with different trade-offs. For example, we could use client-side caching of password-dependent inputs, leaving it up to the client to use the correct password. Such a solution would however not suffice for our purpose of achieving a concise UC definition (a malicious client could simply mess up the caching then, introducing valid encryptions under wrong passwords to the system). Hence, in this paper, we opt for a stronger and cleaner definition, at the cost of slightly worse efficiency and slightly weaker corruption model.

***Efficient* DPaSE *Protocol.*** We present an efficient protocol that provably realizes our functionality $\mathcal{F}_{\mathsf{DPaSE}}$. The high-level idea of the protocol is very simple and follows the known paradigm of password-based protocols to turn the password into cryptographic key material using an OPRF [20, 7]. More detailed, to create an account, the user derives a signing key $(upk, usk) \leftarrow \mathsf{OPRF}(K, uid, pw)$ from her username and password, where the OPRF key $K$ is split among the $n$ servers and the evaluation reveals the username to

[4] We do not want to go further and hide the identity of the user in his requests, since otherwise we would not be able to protect against online guessing attacks.

[5] A UC functionality needs to "list" *all* potential attacks that can be mounted against a protocol. While some attacks might be benign in practise and we might be okay with the threat they are imposing on us, every such attack still shows in the functionality. It is one of the main challenges in using the UC framework to find a mid-way between a not overly strong notion that still allows for efficient instantiations, and one that is not overly cluttered with such benign attacks.

the servers to later allow for user-specific rate limiting. The servers store $(uid, upk)$ upon registration.

To encrypt a file, the user again enters $uid, pw'$ and starts by re-running the steps from account creation to recover her signing key pair $(upk, usk)$. She then signs a fresh nonce with $usk$ and sends it to the servers who verify it against the stored $upk$, thereby verifying that $pw = pw'$. If the password is correct, the user and server engage in a follow-up OPRF evaluation where an object-specific encryption key is derived. The OPRF evaluation thereby "reuses" the previously entered $uid, pw'$ to ensure that the actual encryption keys are also bound to the user' identity and correct password. This prevents users from accidentally encrypting data under a wrong password. To ensure obliviousness, the object for which the key is derived is hidden in the evaluation.

Decryption works almost analogously to encryption, verifying the password and – if correct – recovering the object-specific encryption key via the distributed OPRF. The generated ciphertexts and decryption proceeds also include checks to guarantee the desired ciphertext integrity.

***Extendable Distributed Partially-Oblivious PRF.*** The core of our DPaSE protocol is a new type of OPRF that we believe to be of independent interest for many password-based applications. So far, OPRFs have been designed as *single*-evaluation primitives[6] that can either be fully or partially-blind. Thus, the user sends a (partially) blind query, and receives a single output related to that input. What we need for DPaSE though is an OPRF that "remembers" the blindly provided password from a previous query and re-uses it in a follow-up evaluation: we need to perform a dedicated password check and also want to ensure that encryption is done with the same password that was verified. We model that as an extension query, where a second OPRF query re-uses the blinded input from a previous request. This extension feature is required on top of *partial*-blindness (as the $uid$'s must be a known input to all parties) and the *distributed* setting. We formalize the desired properties of such an extendable OPRF in the UC framework and propose a secure instantiation. We believe that this is a contribution of independent interest. Namely, using an extendable OPRF instead of a single-evaluation OPRF could generally add secure password verification to protocols that deploy an OPRF to bootstrap cryptographic material from passwords.

Our OPRF construction is based on the classical double-hash DH scheme, basically combining all tricks that have been used in this context into a single scheme. The challenge thereby is that our second OPRF call which blindly carries over the input from the first call now has *three* inputs: the non-blind part ($x_{\mathsf{pub}} = uid$), and two blinded values, namely the blinded ($x_{\mathsf{priv1}} = pw$) from the previous evaluation and the new input ($x_{\mathsf{priv2}} = oid$). Previous partially-blind OPRFs deal with two inputs only $x_{\mathsf{pub}}$ and $x_{\mathsf{priv}}$ which are mostly combined through a pairing [14, 7], with the final PRF being of the form $\mathsf{H}_T(e(\mathsf{H}_1(x_{\mathsf{priv}}), x_{\mathsf{pub}})^K, x_{\mathsf{priv}})$. In our construction, we will already need both "slots" of the pairing to combine the two blinded inputs, and therefore must find a different place to include the public input. We take inspiration from [19] and replace the direct use of the server's secret key $K$ by $K' \leftarrow \mathsf{F}(K, uid)$ where $\mathsf{F}$ is a standard PRF. Thus, overall our new OPRF computes the output for an extended query as

[6] With the exception of OPRF with batch evaluations under several keys [23, 26]. This is orthogonal to our problem since we have a single OPRF key.

$\mathsf{H}_T(e(\mathsf{H}_1(x_{\mathsf{priv1}}), \mathsf{H}_2(x_{\mathsf{priv2}}))^{\mathsf{F}(K, x_{\mathsf{pub}})}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}})$. The first (non-extended) query, just consisting of $x_{\mathsf{pub}}$ and $x_{\mathsf{priv1}}$ has the same form and simply sets $x_{\mathsf{priv2}} = 1$.

This construction allows us to combine three values into a single evaluation, but this extendability feature comes for a price. First, relying on exponents that are derived from a standard PRF $K' \leftarrow \mathsf{F}(K, uid)$ only allows for a distributed, but not threshold protocol. The distributed version simply considers the additive combination of all $K'$ as the implicit overall secret key (per $x_{\mathsf{pub}}$). Second, there are currently no efficient proofs that allow to check whether the servers have computed the second evaluation correctly – which again stems from the use of the standard PRF to derive the OPRF secret key share. As we will require correct computation of OPRF outputs in our DPaSE protocol, we must assume that the servers in the OPRF are at most honest-but-curious. We stress that considering honest-but-curious servers already captures the main threat to passwords: an adversary stealing the password database (or other offline-attackable information). To our knowledge, DPaSE is currently the only protocol being secure in the presence of such attacks.

Lastly, we note that extendability is a property that could as well be ensured on the application level by, e.g., caching the user's password on the client machine. While this would enable using DPaSE with a standard, i.e., single-evaluation OPRF and make our protocol simpler and more efficient, it allows for a „benign“ attack which prevents a security proof. Namely, an adversary knowing the password of an honest user could produce encryption keys under bogus passwords. If the honest user later tries to decrypt such a maliciously crafted ciphertext, decryption would fail – yet the adversary can decrypt using the bogus password again. While this attack is rather harmless in practice, to prove the password-caching version secure one would have to include this imperfection into the security definition, with a different set of „shadow passwords“ for each (!) ciphertext that the adversary could use (even for honest accounts). With the extendability property, we enforce password consistency *on the protocol level* and hence avoid cluttering the security definition of DPaSE with attacks resulting from inconsistent usage of passwords.

***Implementation and Evaluation.*** Instantiating DPaSE with our OPRF yields an efficient scheme that requires 10 exponentiations at the user, 4 exponentiations and 2 pairings at each server. We further provide a proof-of-concept implementation of $\Pi_{\mathsf{DPaSE}}$ which respectively takes 13 ms for an account creation and 27 ms for each encryption and decryption on the server side, when run between an user and any number of servers; currently the implementation has a server throughput of 76 account creation and 37 (user authentication followed by) encryption or decryption requests per second.

## 1.2 Related Work

Password-authenticated secret sharing (PASS/PPSS) allows a user to recover a strong secret that is shared among $n$ servers when she can enter the correct password [6, 16–18]. In contrast to end users, servers can easily maintain strong cryptographic keys which is leveraged by PASS to thwart offline attacks against the password (and consequently on the shared secret key) if at least one, or a certain threshold, of the servers is not compromised. While this concept is shared between PASS and DPaSE, PASS can only be used to derive one encryption key per password, while DPaSE is required to encrypt each

| Properties\Schemes | **Key Management Schemes (KMS)** | | **Encryption Schemes** | | | |
|---|---|---|---|---|---|---|
| | PASS scheme [18] | PASS scheme Memento [10] | OKMS [19] | DiSE [5] | (Threshold) PHE [25], [8] | DPaSE this work |
| Password correctness ensured | - | - | | | ✓ | ✓ |
| Can derive multiple keys per password | - | - | | | - | ✓ |
| Security against online attacks | - | ✓ | | | ✓ | ✓ |
| Security against offline attacks | ✓ | ✓ | | | ✓ | ✓ |
| Password remains private | ✓ | ✓ | | | - | ✓ |
| Access pattern remains private | | | - | - | ✓ | ✓ |
| Authenticated encryption | | | ✓ | ✓ | ✓ | ✓ |
| Who encrypts? (U=User, S=Server) | | | U | S | S | U |
| Mitigation of compromised encryption keys | | | no reuse & key rotation | - | key rotation | no reuse |
| Secure in concurrent settings | ✓ | ✓ | - | - | - | ✓ |

**Table 1.** Properties of server-assisted encryption and encryption key retrieval (KMS) schemes. Gray cells are not applicable. More precisely, password properties do not apply to OKMS and DiSE schemes, as they rely on strong user authentication. Likewise, encryption properties do not apply to the KMS schemes, since their purpose is to recover an encryption key from a password.

piece of data under different keys, yet enabling the user to encrypt all her data under the same password.

Password-hardened encryption (PHE) [25] targets a related setting, where a user outsources key management, encryption and decryption to a so-called rate limiter. The user can send encryption/decryption requests through a server, but needs to provide a correct password. The rate limiter can be implemented in a threshold version [8] to further enhance PHE's security. The scheme allows a mechanism of key rotation, to mitigate against compromises or simply as a routine process. Key rotation involves the server and the rate limiter updating their respective keys as well as the ciphertexts accordingly. In PHE the frontend server is fully trusted, as it learns the user's password and keys. PHE schemes are very efficient (no OPRF is required!) and a good option in settings where the client fully trusts the server, since both password and access pattern on user's data are shared with the server. In our work, we do not want to assume such trust and hence opt for client-side encryption of data to hide access patterns.

Updatable Oblivious Key Management [19] also relies on a OPRF to derive file-specific encryption keys with the help of a (single) external server for increased security. Their work focuses on an enterprise setting for storage systems though, i.e., it relies on strong authentication between the client (that wants to encrypt or decrypt) and the server that holds the OPRF key. This is a first difference to DPaSE: our scheme achieves oblivious key management without strong client-authentication. Second, their system uses key rotation – similar to PHE and the general concept of *updatable encryption* with post-compromise security [27, 22] – to update encryption keys and the corresponding ciphertexts as a measure to mitigate the effect of security breaches. The approach of our DPaSE protocol is orthogonal: we mitigate the risk of data breaches by using a distributed setting instead of key rotation: the information to recover the encryption keys is split across $n$ servers, and the file-specific encryption keys are secure as long as one server remains uncompromised.

The DiSE protocol [5] and its improvements [28, 13] for distributed symmetric encryption consider strong authentication only. In these protocols, a group of $n$ parties jointly controls encryption keys under which ciphertexts for the group get encrypted. The secret key material is split among the group and *any* member of the group can request decryption of ciphertexts which is again done jointly by all member. DiSE implicitly – yet crucially – relies on strong authentication to ensure that only valid members of the group can make such requests, whereas we want only a single user to encrypt or decrypt her files from a password. Nevertheless, the authenticity checks in the encryption/decryption process of our protocol are build upon the ideas of the DiSE protocol.

Finally, the PESTO protocol [7] for distributed single sign-on (SSO) relies on a similar idea of first deriving a strong key pair from a distributed OPRF in order to let a user authenticate to a number of servers. The overall application is different though, SSO vs. encryption, and consequently also the desired functionality and security are different. PESTO is in one aspect stronger than $\mathsf{DPaSE}$ since it features *proactive security*, meaning that a once corrupted server can be sanitized to be honest again. This strong aspect comes at a cost that is much more critical for the targeted encryption use case than in SSO: PESTO guarantees no security whatsoever when all servers are corrupt. In $\mathsf{DPaSE}$, even in case of a full corruption of all servers, the user's data still remains confidential unless all servers jointly mount a successful offline attack on her password. Thus, a dedicated offline attack for each user (on top of corrupting all servers) would be required, and the encrypted files of users with reasonably strong passwords can remain secure.

We give a detailed comparison of properties of the schemes that are closely related to $\mathsf{DPaSE}$ in Table 1.

## 2 Preliminaries

**Definition 1 (Signature Schemes).** *A* signature scheme $\mathsf{SIG}$ *is a triple of algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *with the following properties. On input the security parameter* $\lambda$*, the randomized* key generation algorithm $\mathsf{Gen}$ *outputs a key pair* $(pk, sk)$*. On a message* $m \in \{0,1\}^*$ *and a secret key sk, the randomized* signing algorithm $\mathsf{Sign}$ *outputs a signature* $\sigma$*. On input a public key pk, a message* $m \in \{0,1\}^*$*, and a signature* $\sigma$*, the deterministic* verification algorithm $\mathsf{Verify}$ *outputs* 1 *if the signature is correct or* 0 *otherwise.*

We require that the scheme satisfies *correctness*, i.e., for all $m \in \{0,1\}^*$ and $(pk, sk)$ output by $\mathsf{Gen}$ it holds that: $\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = 1$. Additionally, the scheme needs to satisfy *unforgeability under chosen message attacks (UF-CMA-security)*, i.e., after learning signatures for $q$ number of adaptively chosen messages $\{m_1, ..., m_q\} \in \mathcal{M}$, it should be impossible to find a signature/message pair $(\sigma, m)$ s.t. $\mathsf{Verify}(pk, m, \sigma) = 1$ and $m \notin \{m_1, ..., m_q\}$.

### 2.1 Bilinear Groups

**Definition 2 (Asymmetric Pairing).** *Let* $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ *be cyclic groups of order p with generators* $g_1, g_2, g_T$*, respectively. Let* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *be an efficiently computable non-degenerate function such that* $\forall a, b \in \mathbb{Z}_p : \; e(g_1^a, g_2^b) = g_T^{ab}$*. Then e is called an* asymmetric pairing*.* $\mathbb{G} = (p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ *is called an* asymmetric bilinear group setting*, or* bilinear group *for short.*

**Definition 3 (Gap One-More BDH Assumption).** *Let* $\lambda \in \mathbb{N}$ *be a security parameter and* $\mathbb{G} = (p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ *be a bilinear group with* $\log(p) = poly(\lambda)$*, then we then say that the Gap One-More Bilinear Diffie-Hellman (Gapom-BDH) assumption holds for* $\mathbb{G}$ *if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}\cdot$ *such that* $\Pr[\mathsf{Exp}^{\mathbb{G}}_{\mathcal{A},\mathsf{Gapom\text{-}BDH}}(\lambda) = 1] \leq \mathsf{negl}\lambda$*.*

The underlying experiment is defined as follows.

```
*Experiment* $\mathsf{Exp}^{\mathbb{G}}_{\mathcal{A},\mathsf{Gapom\text{-}BDH}}(\lambda)$:
  $k \xleftarrow{\$} \mathbb{Z}_p$, $q_C \leftarrow 0$, $X_1 \leftarrow \emptyset$, $X_2 \leftarrow \emptyset$.
  $\{(x_i, y_i, z_i)\}_{i\in[\ell]} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathbb{G}\text{-}1},\mathcal{O}_{\mathbb{G}\text{-}2},\mathcal{O}_{\mathsf{D\text{-}help}},\mathcal{O}_{\mathsf{C\text{-}help}}}(\mathbb{G}, g_2^k)$
  return 0 if
    $0 \leq \ell - 1 < q_C$ or
    $\exists i \in [\ell] : (x_i \notin X_1 \vee y_i \notin X_2)$ or
    $\exists i, j \in [\ell], i < j : (x_i = x_j \wedge y_i = y_j)$
  return 1 if $\forall i \in [\ell] : e(x_i, y_i)^k = z_i$ and 0 otherwise.
```

where the experiment uses the following oracles

```
$\mathcal{O}_{\mathbb{G}\text{-r}}()$
  return $\bot$ if $r \notin \{1, 2\}$
  $x \xleftarrow{\$} \mathbb{G}_r$
  $X_r \leftarrow X_r \cup \{x\}$
  return $x$

$\mathcal{O}_{\mathsf{C\text{-}help}}(m)$
  return $\bot$ if $m \notin \mathbb{G}_T$.
  $q_C \leftarrow q_C + 1$
  return $m^k$

$\mathcal{O}_{\mathsf{D\text{-}help}}(m, w, m', w')$
  return $\bot$ if either $m, w, m', w'$ not in $\mathbb{G}_T$
  return 1 if $\log_m(w) = \log_{m'}(w')$ and else 0
```

To win, $\mathcal{A}$ needs to find pairs $(x, y, e(x, y)^k)$ without querying $e(x, y)$ to $\mathcal{O}_{\mathsf{C\text{-}help}}$ and where $\mathcal{A}$ could not rerandomize previous such pairs as it does not know the discrete logarithm of any $x, y$ (enforced by sampling them at random using $\mathcal{O}_{\mathbb{G}\text{-r}}$). $\mathcal{A}$ is equipped with a DDH oracle $\mathcal{O}_{\mathsf{D\text{-}help}}$ in the group $\mathbb{G}_T$. The game Gapom-BDH follows the definition in [14].

## 3 Extendable Distributed Partially-Oblivious PRF

Our DPaSE construction relies on a new type of oblivious PRF (OPRF) that allows for extension queries and which we believe to be of interest for password-based protocols in general. In this section, we define this new type of OPRF and present a provably secure construction.

An OPRF is an interactive protocol between at least one user and one server. The server holds the key $K$ of a pseudorandom function PRF, the user contributes the input $x$ to the function. After the protocol runs, the user holds the PRF evaluation at $x$, $\mathsf{PRF}_K(x)$. The obliviousness property demands that, while the server actively participated in the protocol, he did not learn anything about the value $x$ he helped in evaluating the function for. On the other side, the user requires participation of the server to evaluate $\mathsf{PRF}_K()$ at any input. In a distributed OPRF, the key $K$ is split among $n$ servers.

Recently, there has been a flurry of OPRF constructions in the literature all featuring different (combinations of) properties on top of the above mentioned [21, 16, 17, 14, 18, 9, 20, 7]. For constructing DPaSE, we require a new combination of properties that we detail now. Our OPRF is called a *extendable distributed partially-oblivious PRF* (edpOPRF).

The functionality is parametrized by a security parameter $\lambda$. It interacts with servers $\mathcal{S} := \{S_1, ..., S_n\}$ (specified in the $sid$), arbitrary other parties and an adversary $\mathcal{A}$. $\mathcal{F}_{\mathsf{edpOPRF}}$ maintains a table $T(x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}})$ initially undefined everywhere, counters $\mathsf{ctr}[x_{\mathsf{pub}}]$ initially set to 0. $\mathcal{F}_{\mathsf{edpOPRF}}$ sends all inputs to $\mathcal{A}$ except for $x_{\mathsf{priv1}}, x_{\mathsf{priv2}}$.

Key Generation

**On input** $(\mathsf{KeyGen}, sid)$ from $S_i$, ignore this query if the $sid$ is marked `ready`. Otherwise, if $(\mathsf{KeyGen}, sid)$ was received from all $S_i$, mark $sid$ as `ready`, and output $(\texttt{KeyConf}, sid)$ to all $S_i$.

Evaluation

**On input** $(\texttt{EvalInit}, sid, qid, x_{\mathsf{pub}}, x_{\mathsf{priv1}})$ from any party $U$ (including $\mathcal{A}$: record $(\texttt{eval}, sid, qid, U, x_{\mathsf{pub}}, x_{\mathsf{priv1}}, \perp)$, and output $(\texttt{EvalInit}, sid, qid, x_{\mathsf{pub}})$ to all $S_i$.

**On input** $(\texttt{EvalProceed}, R, sid, qid)$ from $S_i$ where $R \in \{1, 2\}$:
- Retrieve record $(\texttt{eval}, sid, qid, U, x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}})$, where $x_{\mathsf{priv2}} = \perp$ if $R = 1$, and $x_{\mathsf{priv2}} \neq \perp$ if $R = 2$.
- If $(\texttt{EvalProceed}, R, sid, qid)$ has been received from all $S_i$, set $\mathsf{ctr}[x_{\mathsf{pub}}] \leftarrow \mathsf{ctr}[x_{\mathsf{pub}}] + 1$.

**On input** $(\texttt{EvalFollow}, sid, qid, x_{\mathsf{priv2}})$ from any party $U$ (including $\mathcal{A}$):
- Retrieve record $(\texttt{eval}, sid, qid, U, x_{\mathsf{pub}}, x_{\mathsf{priv1}}, \perp)$ for $(sid, qid, U)$.
- Update record to $(\texttt{eval}, sid, qid, U, x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}})$, and send output $(\texttt{EvalFollow}, sid, qid)$ to every $S_i$.

**On input** $(\texttt{EvalComplete}, sid, qid)$ from $\mathcal{A}$:
- Retrieve record $(\texttt{eval}, sid, qid, U, x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}})$, only proceed if $\mathsf{ctr}[x_{\mathsf{pub}}] > 0$, set $\mathsf{ctr}[x_{\mathsf{pub}}] \leftarrow \mathsf{ctr}[x_{\mathsf{pub}}] - 1$.
- If $T(x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}})$ is undefined, then pick $\rho \xleftarrow{\$} \{0,1\}^{\lambda}$ and set $T(x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}}) \leftarrow \rho$.
- Output $(\texttt{EvalComplete}, sid, qid, x_{\mathsf{priv2}}, T(x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}}))$ to $U$.

**Fig. 2.** Ideal functionality $\mathcal{F}_{\mathsf{edpOPRF}}$

**Partial Obliviousness:** The *obliviousness* property guarantees that the servers do not learn on which input ($x_{\mathsf{priv1}}$ and $x_{\mathsf{priv2}}$) the user wants to evaluate the function. *Partial* obliviousness allows for an additional public part ($x_{\mathsf{pub}}$) of the input.

**Distribution:** Obtaining a PRF value requires the active participation of all $n$ servers. No subset of $n - 1$ servers can evaluate the function themselves.

**Extendability:** After the user has provided an input $(x_{\mathsf{pub}}, x_{\mathsf{priv1}})$ and learned the corresponding output $\mathsf{PRF}_K(x_{\mathsf{pub}}, x_{\mathsf{priv1}})$, he can extend the query with a second blind input $x_{\mathsf{priv2}}$ upon which he receives $\mathsf{PRF}_K(x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}})$ (in both cases the output is conditioned on the participation of all servers of course).

While the first two properties exist (individually) already (partial obliviousness [14], and distribution [18], the concept of extendability of an OPRF is new. What is so special about this property that could not be achieved by simply evaluating the OPRF twice? The crucial difference is that an extendable OPRF guarantees that certain *blinded inputs are reused* in the second evaluation. With separate evaluation requests this cannot be guaranteed since blindings information-theoretically hide inputs and thus users can easily cheat. For DPaSE, we require such an OPRF to allow for dedicated password verification and ensuring that actual encryption/decryption happens with the *same* password. We envision extendable OPRFs to be generally useful in protocols requiring more than one OPRF evaluation and where secret inputs of these single evaluations need to be correlated.

## 3.1 Ideal functionality for edpOPRF

We define a extendable distributed partially-oblivious PRF in the Universal Composability framework [12] in terms of an ideal functionality $\mathcal{F}_{\mathsf{edpOPRF}}$ in Figure 2. For brevity, we assume the following writing conventions.

- The functionality considers a specific session $sid = (S_1, \ldots, S_n, sid')$ and only accepts inputs from servers $S_i$ that are contained in the $sid$.
- When the functionality is supposed to retrieve an internal record, but no such record can be found, then the query is ignored.
- We assume private delayed outputs, meaning that the adversary can schedule their delivery but not read their contents beyond session and sub-session identifiers.

The functionality $\mathcal{F}_{\mathsf{edpOPRF}}$ is inspired by functionalities from the literature [16–18, 20, 7, 9] and introduces extendability as a new OPRF feature. $\mathcal{F}_{\mathsf{edpOPRF}}$ talks to arbitrary users and a fixed set of servers $S_1, \ldots, S_n$. Initially, all servers are required to call the KeyGen interface, to activate the functionality. Modeling an ideal PRF, $\mathcal{F}_{\mathsf{edpOPRF}}$ chooses outputs at random, maintaining a function table $T()$ to ensure consistency. Implementing a partially-oblivious function, $\mathcal{F}_{\mathsf{edpOPRF}}$ tells the servers public input $x_{\mathsf{pub}}$ before they have to decide about their participation in the request. Participation is signaled by calling (or not calling) `EvalProceed`. The adversary may also evaluate the function, but crucially requires participation of all servers as well. If all servers are corrupted, the adversary can freely evaluate the function by sending `EvalProceed` on behalf of all the corrupted servers. To allow for efficient protocols, we employ an "evaluation ticket" counter $\mathsf{ctr}[]$ allowing mixing-and-matching evaluations w.r.t the public input, as common for OPRF notions (see, e.g., [7]).

Our $\mathcal{F}_{\mathsf{edpOPRF}}$ provides a new feature: it can be extended to output a second PRF value which is related to the first evaluation. This works as follows. A user obtains an evaluation on $x_{\mathsf{pub}}, x_{\mathsf{priv1}}$ by calling `EvalInit` with session identifier $qid$. (The output is only generated if all servers participate and the adversary allows the output by calling `EvalComplete`, which is standard procedure for distributed OPRFs and we thus not elaborate here.) Afterwards, the user can provide a third input $x_{\mathsf{priv2}}$ via interface `EvalFollow`, using the still active session $qid$. $\mathcal{F}_{\mathsf{edpOPRF}}$ outputs the function value at $x_{\mathsf{pub}}, x_{\mathsf{priv1}}, x_{\mathsf{priv2}}$, ensuring that inputs $x_{\mathsf{pub}}, x_{\mathsf{priv1}}$ from the first evaluation are reused by looking them up using $qid$.

## 3.2 Our edpOPRF Construction

We now present our construction of a extendable distributed partially-oblivious PRF. $\Pi_{\mathsf{edpOPRF}}$ computes the function

$$\begin{aligned}\mathsf{PRF}(K(x_{\mathsf{pub}}), x_{\mathsf{priv1}}, 1) &= \mathsf{H}_T(x_{\mathsf{priv1}}, e(\mathsf{H}_1(x_{\mathsf{priv1}}), \mathsf{H}_2(1))^{K(x_{\mathsf{pub}})})\\ \mathsf{PRF}(K(x_{\mathsf{pub}}), x_{\mathsf{priv1}}, x_{\mathsf{priv2}}) &= \mathsf{H}_T(x_{\mathsf{priv1}}, x_{\mathsf{priv2}}, e(\mathsf{H}_1(x_{\mathsf{priv1}}), \mathsf{H}_2(x_{\mathsf{priv2}}))^{K(x_{\mathsf{pub}})})\end{aligned}$$

with $K(x_{\mathsf{pub}}) \leftarrow \sum_{i=1}^{n} \mathsf{F}(k_i, x_{\mathsf{pub}})$ for (standard) PRF $\mathsf{F} : \{0,1\}^* \to \mathbb{Z}_q$, and $k_i$ from F's key space is held by server $S_i$. $x_{\mathsf{pub}}$ denotes the public input and $x_{\mathsf{priv1}}, x_{\mathsf{priv2}}$ the private inputs. The function $e()$ denotes a pairing, and $\mathsf{H}_i$ denote hash functions.

**Setup and Key Generation:** We require an asymmetric bilinear group $(g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ and hash functions $\mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{G}_1$, $\mathsf{H}_2 \colon \{0,1\}^* \to \mathbb{G}_2$, $\mathsf{H}_T \colon \{0,1\}^* \to \mathbb{G}_T$. We assume servers to choose keys $k_i \leftarrow \mathbb{Z}_q, i \in [n]$ at the beginning of the protocol.

**Evaluation:** Our PRF is essentially the "2Hash Diffie-Hellman" function [16, 17] $\mathsf{PRF}(k, x_{\mathsf{priv1}}) = \mathsf{H}(x_{\mathsf{priv1}}, \mathsf{H}'(x_{\mathsf{priv1}})^k)$. Let us briefly explain how evaluating this function would work. A user *blinds* his input $x_{\mathsf{priv1}}$ with randomness $r$ as $\mathsf{H}'(x_{\mathsf{priv1}})^r$ and sends this value to the server. The server sends back $\mathsf{H}'(x_{\mathsf{priv1}})^{rk}$, from which the user can compute $\mathsf{H}'(x_{\mathsf{priv1}})^k$ by exponentiation with $1/r$. This is enough for the user to compute $\mathsf{H}(x_{\mathsf{priv1}}, \mathsf{H}'(x_{\mathsf{priv1}})^k)$.

Partial obliviousness is now achieved as in Everspaugh et al. [14] by combining blinded private inputs as $e(\mathsf{H}_1(x_{\mathsf{priv1}})^r, \mathsf{H}_2(x_{\mathsf{priv2}})^k)$ using the pairing $e()$. Due to the bilinear property of $e()$ this is equal to $e(\mathsf{H}_1(x_{\mathsf{priv1}}), \mathsf{H}_2(x_{\mathsf{priv2}}))^{rk}$, which again allows the user to remove the blinding factor $r$. One can additively share $k$ among all servers and let the client combine evaluation shares using the group operation in $\mathbb{G}_T$. The function is computed as $\mathsf{PRF}(k, x_{\mathsf{priv1}}, x_{\mathsf{priv2}}) = \mathsf{H}_T(x_{\mathsf{priv1}}, x_{\mathsf{priv2}}, e(\mathsf{H}_1(x_{\mathsf{priv1}}), \mathsf{H}_2(x_{\mathsf{priv2}}))^k)$.

For our new extendability property we require a PRF evaluated on three inputs. Fortunately, we can efficiently and securely augment the function given above with another input by "squeezing" it into the function's key. This technique is inspired by the work of Jarecki et al. [19]. We set $K(x_{\mathsf{pub}}) := k \leftarrow \sum_{i \in [n]} \mathsf{F}(k_i, x_{\mathsf{pub}})$ for a (standard) PRF $\mathsf{F}$ and $k_i$ being the servers' secret keys. One subtlety here occurs in the first evaluation on $x_{\mathsf{pub}}$ and $x_{\mathsf{priv1}}$ only. We cannot save the pairing evaluation and simply use $H_1(x_{\mathsf{priv1}})^k$ as $k$-dependent value instead: with this value, a user could compute arbitrary function evaluations on input $x_{\mathsf{priv1}}$ by himself by applying the pairing. We therefore let servers use a "dummy" value $H_2(1)$ and pair it with the user's blinded input $x_{\mathsf{priv1}}$.

A full formal description of our OPRF construction $\Pi_{\mathsf{edpOPRF}}$ can be found in Figure 3. Its security is stated in the following theorem, for which a proof sketch can be found in full version, Appendix B.

| User $U$ | | Server $S_i$, holding $k_i$ |
|---|---|---|
| On input $(\texttt{EvalInit}, x_{\mathsf{pub}}, x_{\mathsf{priv1}})$ | | |
| $r_1 \xleftarrow{\$} \mathbb{Z}_p$, $x_1 \leftarrow \mathsf{H}_1(x_{\mathsf{priv1}})^{r_1}$ | $\xrightarrow{x_{\mathsf{pub}}, x_1}$ | Output $(\texttt{EvalInit}, x_{\mathsf{pub}})$ |
| | | On input $(\texttt{EvalProceed}, 1)$ |
| | | $osk_i \leftarrow \mathsf{F}(k_i, x_{\mathsf{pub}})$ |
| If all $S_j \in \mathcal{S}$ sent $y_j$: | $\xleftarrow{y_i}$ | $y_i \leftarrow e(x_1, H_2(1))^{osk_i}$ |
| $y \leftarrow \prod_{j \in [n]} y_j^{r_1^{-1}}$, $Y_1 \leftarrow \mathsf{H}_T(x_{\mathsf{priv1}}, y)$ | | |
| Output $(\texttt{EvalComplete}, \perp, Y_1)$ | | |
| On input $(\texttt{EvalFollow}, x_{\mathsf{priv2}})$ | | |
| $r_2 \xleftarrow{\$} \mathbb{Z}_p$, $x_2 \leftarrow \mathsf{H}_2(x_{\mathsf{priv2}})^{r_2}$ | $\xrightarrow{x_2}$ | Output $(\texttt{EvalFollow})$ |
| | | On input $(\texttt{EvalProceed}, 2)$ |
| If all $S_j \in \mathcal{S}$ sent $y_j$: | $\xleftarrow{y'_i}$ | $y'_i \leftarrow e(x_1, x_2)^{osk_i}$ |
| $y' \leftarrow \prod_{j \in [n]} y_j'^{r_1^{-1} r_2^{-1}}$, | | |
| $Y_2 \leftarrow \mathsf{H}_T(x_{\mathsf{priv1}}, x_{\mathsf{priv2}}, y')$ | | |
| Output $(\texttt{EvalComplete}, x_{\mathsf{priv2}}, Y_2)$ | | |

**Fig. 3.** Protocol $\Pi_{\mathsf{edpOPRF}}$. We assume all messages, inputs and outputs to include *sid*, *qid*.

**Theorem 1.** *Let $\mathbb{G} = (p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be a bilinear group. If the Gap One-More BDH (Gapom-BDH) assumption (c.f. Definition 3) holds for $\mathbb{G}$ then the protocol $\Pi_{\mathsf{edpOPRF}}$ given in Figure 3, with $\mathsf{H}_1, \mathsf{H}_2$ and $\mathsf{H}_T$ modeled as random oracles and $\mathsf{F}$ being a (standard) PRF, UC-emulates $\mathcal{F}_{\mathsf{edpOPRF}}$ in the random oracle model assuming secure and server-side authenticated channels, static honest-but-curious corruption of servers and static malicious corruption of clients.*

***On malicious security.*** As detailed above, our $\mathcal{F}_{\mathsf{edpOPRF}}$ ensures that the PRF is always evaluated w.r.t the same key. This rules out protocols where servers can freely decide what key material to use in an evaluation. Let us note that it is quite common in the literature [18, 20, 7] to relax this property by letting the OPRF functionality maintain different lists representing different PRF keys. The adversary can then determine which list is going to be used (in case of server corruption). And indeed, it turns out that our $\mathcal{F}_{\mathsf{edpOPRF}}$ enforcing such consistency in keys is challenging to realize in the presence of malicious servers. The reason is that we cannot use standard techniques such as NIZK proofs of honest behavior with respect to some public server key (e.g., [14]), since our server keys are user-specific. Reliable distribution of such keys would involve frequent interaction with a trusted authority. Another inefficient way to obtain malicious security is to use a 3-linear map instead of a 2-linear map (pairing), together with a NIZK. The map would allow us to have (NIZK-compatible) *uid*-independent key shares simply by putting *uid* as third input parameter to the map. We choose not to give a maliciously secure protocol with such inefficient techniques, and leave the construction of an efficient maliciously secure (i.e., *verifiable*) extendable distributed partially-oblivious PRF as an open problem.

## 4 DPaSE

In this section we introduce distributed password-authenticated symmetric encryption (DPaSE). DPaSE is an interactive protocol between many users and a fixed set of $n$ servers, where the servers assist users in conveniently and securely encrypting their data under a single password. DPaSE operates account-based: First, users register with a username and a single password at all servers. After account creation, the servers (blindly) assist users in encryption and decryption provided that they are using the correct password.

We recall the key security properties of DPaSE as already explained in more detail in Section 1: correct and authenticated encryption, no reuse of encryption keys, security against offline and online attacks, and confidentiality of password and access patterns. We will detail in the upcoming subsection how our definition ensures these properties.

Our concrete scheme will leverage the servers mainly to (re)construct object-specific encryption keys, whereas encryption and decryption happens locally at the user side. This might pose the question why we are modelling DPaSE as an encryption and not key management protocol. Capturing the full encryption/decryption process is necessary to avoid similar misconceptions as with PASS, which was believed to be a suitable out-of-the-box tool for password-based encryption. Only with modelling and considering the full encryption process this can be ensured.

### 4.1 An ideal functionality for DPaSE

We define DPaSE in terms of an ideal functionality $\mathcal{F}_{\mathsf{DPaSE}}$, which takes inputs of parties and hands them their securely computed outputs. $\mathcal{F}_{\mathsf{DPaSE}}$ abstracts away any protocol details and states only the required functionality and leakage and influence (i.e., attacks) allowed by an adversary.

We assume the same writing conventions as for $\mathcal{F}_{\mathsf{edpOPRF}}$. In addition, we assume the adversary gets to acknowledge all inputs, but not learn their private content. For example, if the functionality receives input "$(\texttt{Encrypt}, sid, qid, x)$ from a party $P$" and "keeps $x$

private", we assume that the functionality sends ($\texttt{Encrypt}, sid, qid, P$) to the adversary and only processes the original input after input an acknowledgement from the adversary.

Our ideal functionality $\mathcal{F}_{\mathsf{DPaSE}}$ is depicted in Figure 4, with labeled instructions to enable easy matching to the explanations in this section. On a high level, $\mathcal{F}_{\mathsf{DPaSE}}$ is a password-protected lookup table for (username,message,ciphertext) tuples. Users can create new such tupes by first logging in to their account stored by $\mathcal{F}_{\mathsf{DPaSE}}$ with a username and password, and then encrypt a message of their choice, obtaining back the ciphertext. Decryption works in a similar fashion. $\mathcal{F}_{\mathsf{DPaSE}}$ stores a password for every registered user, and refuses service if a user does not remember his password correctly when he wants to encrypt or decrypt. In order to perform registration, encryption or decryption, $\mathcal{F}_{\mathsf{DPaSE}}$ requires participation of $n$ distributed servers.

Let us first give some intuition on how $\mathcal{F}_{\mathsf{DPaSE}}$ ensures the required security and privacy properties of DPaSE. $\mathcal{F}_{\mathsf{DPaSE}}$ associates a password with each username $uid$ and creates an "encryption" entry $(uid, m, c)$ upon request of user $uid$ if and only if the user provided the password that is associated with $uid$. This ensures **correct** and **authenticated encryption**. By verifying password matches internally and *not* revealing passwords nor message/ciphertext from encryption/decryption requests to the servers, $\mathcal{F}_{\mathsf{DPaSE}}$ ensures **obliviousness**. By requiring all servers to assist in proceeding any request (register, encryption, or decryption), $\mathcal{F}_{\mathsf{DPaSE}}$ ensures security against **offline attacks**. $\mathcal{F}_{\mathsf{DPaSE}}$ however leaks the username $uid$ to the servers, which allows servers to refuse participation if they are suspicious of an **online password guessing attack** against user $uid$.

We now describe the interfaces of $\mathcal{F}_{\mathsf{DPaSE}}$ in more detail. The functionality talks to arbitrary users and a fixed set of servers $S_1, \ldots, S_n$.

**Account Creation:** Any user can register with $\mathcal{F}_{\mathsf{DPaSE}}$ by calling its `Register` interface with a username $uid$ and a password $pw$. If no account $uid$ exists yet ( R.3 ), $\mathcal{F}_{\mathsf{DPaSE}}$ informs all servers about the new registration request and the $uid$, but keeps the password private ( R.4 ). Servers can now decide to participate in the registration by sending `ProceedRegister` to $\mathcal{F}_{\mathsf{DPaSE}}$. Only if all servers do so ( PR.2 ), $\mathcal{F}_{\mathsf{DPaSE}}$ stores the account $(uid, pw)$ and confirms the registration to the user. `ProceedRegister` inputs of servers are matched with their corresponding registration requests via subsession identifiers $qid$. Since those identifiers are unique, collecting servers' participation among different requests is prevented by $\mathcal{F}_{\mathsf{DPaSE}}$.

**Encryption:** A message encryption is initiated by a user sending an `Encrypt` request to $\mathcal{F}_{\mathsf{DPaSE}}$ which includes $uid, pw$ and a message $m$. If account $(uid, pw')$ exists ( E.3 ), $\mathcal{F}_{\mathsf{DPaSE}}$ first informs all servers about an incoming request for $uid$, keeping the password as well as the message private ( E.4 ). Only if all server agree to participate in this request for $uid$ by using the `Proceed` interface for the corresponding subsession ( P.2 ), $\mathcal{F}_{\mathsf{DPaSE}}$ continues the encryption request. By not giving away any information before, $\mathcal{F}_{\mathsf{DPaSE}}$ prevents offline attacks. $\mathcal{F}_{\mathsf{DPaSE}}$ now verifies that the provided password $pw$ is equal to $pw'$ stored with $uid$ ( P.2.3 ). All servers and the user are informed about the outcome of password verification by input either `PwdOK` or `PwdFail` ( P.2.5 ). Being informed about failed password attempts and requests of $uid$ in general allows servers to protect accounts against online guessing attacks: based on this information, they can decide to throttle requests by refusing to send `Proceed`, e.g., after 5 failed attempts within 1 minute. Such throttling is however decided by the application using $\mathcal{F}_{\mathsf{DPaSE}}$.

Finally, if verification was successful, the user obtains a ciphertext $c$ from $\mathcal{F}_{\mathsf{DPaSE}}$. While $c$ is adversarially chosen, we stress that, in an honest encryption procedure, the adversary only learns the length of the message from $\mathcal{F}_{\mathsf{DPaSE}}$ ( P.2.2 ). Thus, $\mathcal{F}_{\mathsf{DPaSE}}$ ensures that the ciphertext does not contain any information about $m$ beyond its length. Further, $\mathcal{F}_{\mathsf{DPaSE}}$ ensures that no two encryption requests yield the same ciphertext by rejecting repeated ciphertexts sent by the adversary ( P.2.2 ). $\mathcal{F}_{\mathsf{DPaSE}}$ stores the pair $(m, c)$ together with $uid$, and sends the ciphertext to the user ( P.2.7 ). Handing out the (fresh) ciphertext only if the correct password was provided ensures correct encryption. For this, note that for honestly registered accounts (which would not have $pw = \perp$) $\mathcal{F}_{\mathsf{DPaSE}}$ prevents the adversary from influencing the "verification bit" $b$ in encryption procedures in any way.

**Decryption:** A user initiates a decryption procedure by calling the `Decrypt`interface of $\mathcal{F}_{\mathsf{DPaSE}}$ with $uid$, $pw$ and a ciphertext $c$. It is instructive to note that $\mathcal{F}_{\mathsf{DPaSE}}$ does not give out any information about ciphertexts it is not explicitly queried for, and thus cannot be used as a storage for ciphertexts. $\mathcal{F}_{\mathsf{DPaSE}}$ informs servers about a request for $uid$, keeping password and ciphertext private ( D.2 ). Similar to an encryption procedure, all servers are required to call `Proceed` in order for $\mathcal{F}_{\mathsf{DPaSE}}$ to continue with password verification ( P.2 ). However, $\mathcal{F}_{\mathsf{DPaSE}}$ does not inform servers about which ciphertext should be decrypted, in order to hide access patterns on user data. We choose to not even inform servers about the type of request - encrypt or decrypt - in order to allow also for protocols where password verification requests do not yet reveal what a user wants to do. Coming back to the decryption procedure, $\mathcal{F}_{\mathsf{DPaSE}}$ now verifies the password ( P.2.3 ). In case of success and if $\mathcal{F}_{\mathsf{DPaSE}}$ finds a record $(uid, m, c)$, the message $m$ is given to the requesting user ( P.2.7 ). By storing $uid$ along with message-ciphertext pairs and revealing $m$ only if $uid$'s password was provided in the decryption query, $\mathcal{F}_{\mathsf{DPaSE}}$ enforces authenticated encryption.

**Adversarial Interfaces:** As explained before, we let $\mathcal{A}$ determine ciphertexts as common for functionalities modeling symmetric encryption. However, ciphertexts cannot depend on the message beyond its length, since $\mathcal{F}_{\mathsf{DPaSE}}$ ensures that the adversary is oblivious of messages to be encrypted ( E.2 and P.2.2 ). $\mathcal{A}$ may also influence password verification in the following ways. First, modeling DoS attacks, we allow $\mathcal{A}$ to make individual servers believe that password verification failed even though the password might have been correct. This attack is carried out by setting $b_{S_i} \leftarrow 0$ for the corresponding server $S_i$ ( P.2.2 and P.2.4 , "otherwise" case). Second, $\mathcal{A}$ may make servers believe that password verification suceeded even when a wrong password was used, but only for accounts belonging to the adversary. $\mathcal{F}_{\mathsf{DPaSE}}$ marks a $uid$ `corrupted` if this is the case, i.e., if a corrupted user performed a successful password verification with respect to username $uid$ ( P.2.1 ). The adversary then can fake successful password verification towards $S_i$ by setting $b_{S_i} \leftarrow 1$ ( P.2.4 , "Else, if" case). The motivation is that for such corrupted accounts we have to assume that the adversary knows all secrets. It is then plausible that he can compute whatever proof a protocol requires to convince servers of knowledge of the correct password.

We further weaken $\mathcal{F}_{\mathsf{DPaSE}}$ by allowing the adversary to start, e.g., an encryption request without yet knowing what message to decrypt, and under which password. Technically, this is enabled by $\mathcal{F}_{\mathsf{DPaSE}}$ accepting overwrite requests in adversarial records ( R.1 , E.1 and D.1 ). While this does not constitute a meaningful attack for real-world applications (we stress that the adversary is only allowed to change inputs for his own requests, not for the ones of honest users), it allows for efficient realizations of $\mathcal{F}_{\mathsf{DPaSE}}$ such as ours based on oblivious pseudo-random functions and random oracles.

**A special case - all servers corrupted:** We want to highlight which guarantees $\mathcal{F}_{\mathsf{DPaSE}}$ gives in this worst case scenario. In any DPaSE protocol with $n$ servers storing a somehow shared information about the user's password, these servers can always throw their data together, guess a password and run the password verification procedure of the DPaSE protocol to learn whether the guess was correct. This is unavoidable unless we involve more parties (such as an external password hardening service), which is not the scope of this work. However, we require that this is the best possible attack on the user's password when all servers are corrupted: they have to invest some computation to test each of their password guesses. This way, users with strong passwords will remain safe even in this worst case scenario. Since $\mathcal{F}_{\mathsf{DPaSE}}$ enforces authenticated encryption by revealing messages only if the correct password was supplied ( P.2.3 and P.2.7 ) - regardless of how many servers are corrupted - not only passwords but also encrypted data of users with strong passwords remain secure.

### 4.2 A DPaSE protocol $\Pi_{\mathsf{DPaSE}}$

We now present our DPaSE protocol $\Pi_{\mathsf{DPaSE}}$. The detailed formal description can be found in Figure 5. $\Pi_{\mathsf{DPaSE}}$ uses hash functions $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H\text{-}PRG}\colon \{0,1\}^\lambda \to \{0,1\}^*$, a signature scheme SIG and $\mathcal{F}_{\mathsf{edpOPRF}}$ as ideal building block. The main principle of $\Pi_{\mathsf{DPaSE}}$ is that the servers assist the user in turning his (low-entropy, but unique) authentication data, i.e., username and password, into various (high-entropy) cryptographic keys. Those keys are subsequently used for proving knowledge of the password to servers, and to encrypt or decrypt the data. We describe the three phases of $\Pi_{\mathsf{DPaSE}}$, account creation, encryption and decryption, in more detail in the below.

**Account Creation:** To create an account, a user derives a signing key pair $(usk, upk)$ from its username $uid$ and password $pw$. For this, $\mathcal{F}_{\mathsf{edpOPRF}}$ is queried with inputs $uid, pw$ by the user, yielding $Y \leftarrow \mathsf{PRF}(k, (pw, 1, uid))$ if all servers participate in the evaluation. Partial blindness ensures that servers learn $uid$ but not $pw$. The user then computes $(usk, upk) \leftarrow \mathsf{SIG.Gen}(Y)$, sends $upk$ to all servers and can afterwards delete the key pair. Servers are required to store $(uid, upk)$.

**Encryption:** Users are required to provide their correct password whenever they want to encrypt (or decrypt) any data. This is now straightforward: as in account creation, the user calls $\mathcal{F}_{\mathsf{edpOPRF}}$ with inputs $(uid, pw')$, receiving PRF value $Y_1$ if again all servers participate in the PRF evaluation. The user now computes a signing key pair from $Y_1$, signs part of the transcript (we mention that the identifier of this encryption session, $qid'$, is globally unique) and sends the resulting signature $\sigma_U$ to each server. Servers will accept (i.e., output `PwdOK`) only if $\sigma_U$ is a verifying signature under $upk$ stored with $uid$, which happens if and only if $pw = pw'$. Of course, this verification technique only works if servers reliably learn $uid$ used in the PRF compuation, which is ensured by the partial obliviousness of $\mathcal{F}_{\mathsf{edpOPRF}}$.

Symmetric encryption in $\Pi_{\mathsf{DPaSE}}$ is simply a one-time pad, with an object-specific encryption key which is computed again from a PRF value and with the help of all servers.

The functionality is parametrized by a security parameter $\lambda$. It interacts with servers $\mathcal{S} := \{S_1, ..., S_n\}$ (specified in the *sid*), as well as arbitrary users and an adversary $\mathcal{A}$.

<u>Account Creation</u>

**On input** $(\texttt{Register}, qid, uid, pw)$ from $U$ or $\mathcal{A}$:

R.1 If $\exists$ record $(\texttt{register}, qid, U, uid, pw')$, if $U = \mathcal{A}$ then overwrite $pw'$ with $pw$ and if $U \neq \mathcal{A}$ then ignore the query.

R.2 Keep $pw$ private

R.3 If $\nexists$ record $(\texttt{account}, uid, *)$: create record $(\texttt{register}, qid, U, uid, pw)$ and R.4 send delayed output $(\texttt{Register}, qid, uid)$ to all $S_i \in \mathcal{S}$.

**On input** $(\texttt{ProceedRegister}, qid, uid)$ from server $S_i$:

PR.1 Retrieve record $(\texttt{register}, qid, U, uid, pw)$.

PR.2 If $(\texttt{ProceedRegister}, qid, uid)$ has been received from all $S_i$:

- Record $(\texttt{account}, uid, pw)$; if $U$ is corrupted mark $uid$ $\texttt{corrupted}$. Send delayed output $(\texttt{Registered}, qid, uid)$ to $U$.

<u>Encryption and Decryption</u>

**On input** $(\texttt{Encrypt}, qid', uid, m, pw')$ from party $U$ or $\mathcal{A}$:

E.1 If $\exists$ record $(*, qid', U, uid, m', pw)$, if $U = \mathcal{A}$ then overwrite it with $(\texttt{enc}, qid', U, uid, m, pw')$ and if $U \neq \mathcal{A}$ then ignore the query.
// $\mathcal{A}$ can change mode, password and message in his own requests

E.2 Keep $\texttt{Encrypt}$, $m$ and $pw'$ private, leak $\texttt{Request}$ tag and $\ell(m)$ to $\mathcal{A}$

E.3 If $\exists$ record $(\texttt{account}, uid, pw)$: create record $(\texttt{enc}, qid', U, uid, m, pw')$ and E.4 send delayed output $(\texttt{Request}, qid', uid)$ to all $S_i \in \mathcal{S}$.

**On input** $(\texttt{Decrypt}, qid', uid, c, pw')$ from party $U$ or $\mathcal{A}$:

D.1 If $\exists$ record $(*, qid', U, uid, pw)$, if $U = \mathcal{A}$ then overwrite it with $(\texttt{dec}, qid', U, uid, c, pw')$ and if $U \neq \mathcal{A}$ then ignore the query.

D.2 Keep $\texttt{Decrypt}$, $c$ and $pw'$ private, leak $\texttt{Request}$ tag and $\ell(c)$ to $\mathcal{A}$

D.3 If $\exists$ $(\texttt{account}, uid, pw)$: record $(\texttt{dec}, qid', U, uid, c, pw')$ and

D.4 send a delayed output $(\texttt{Request}, qid', uid)$ to all $S_i \in \mathcal{S}$ .

**On input** $(\texttt{Proceed}, qid')$ from server $S_i$:

P.1 Retrieve records $(\texttt{mode}, qid', U, uid, \mathsf{obj}, pw')$ and $(\texttt{account}, uid, pw)$ with $\texttt{mode} \in \{\texttt{enc}, \texttt{dec}\}$.

P.2 If $(\texttt{Proceed}, qid')$ has been received from all $S_i$:

P.2.1 If $U$ corrupted and $pw == pw'$ then mark $uid$ $\texttt{corrupted}$.
// DoS attacks: $\mathcal{A}$ can prevent or sometimes even fake password confirmation (send $b_{S_i} = 0$ or $b_{S_i} = 1$)

P.2.2 Send $(\texttt{Complete}, qid', pw == pw')$ to $\mathcal{A}$ and receive back $(\texttt{Complete}, qid', b_{S_1}, \ldots, b_{S_n}, c)$. Abort if $\texttt{mode} = \texttt{enc}$ and $c$ has been sent before.

P.2.3 If $pw = \perp$ then set $b \leftarrow \perp$; otherwise set $b \leftarrow (pw = pw')$.

P.2.4 For $i \in [n]$, if $pw = \perp$ set $b'_i \leftarrow 0$. Else, if $U$ and $uid$ corrupted then set $b'_i \leftarrow b_{S_i}$, otherwise set $b'_i \leftarrow b \wedge b_{S_i}$.

P.2.5 For $i \in [n]$, if $b'_i = 0$ output $(\texttt{PwdFail}, qid')$ and otherwise output $(\texttt{PwdOK}, qid')$ to all $S_i$.

P.2.6 If $b = 0$ output $(\texttt{PwdFail}, qid')$ to $U$.

P.2.7 If $b = 1$ then

* If $\texttt{mode} = \texttt{dec}$ and $\exists$ record $(uid, m, \mathsf{obj})$, output $(\texttt{Plaintext}, qid', m)$ to $U$.
* If $\texttt{mode} = \texttt{enc}$: store $(uid, \mathsf{obj}, c)$, output $(\texttt{Ciphertext}, qid', c)$ to $U$.

**Fig. 4.** Ideal functionality $\mathcal{F}_{\mathsf{DPaSE}}$ for distributed password-authenticated symmetric encryption. For easy access to explanations we use highlighted numbering in both figure and text.

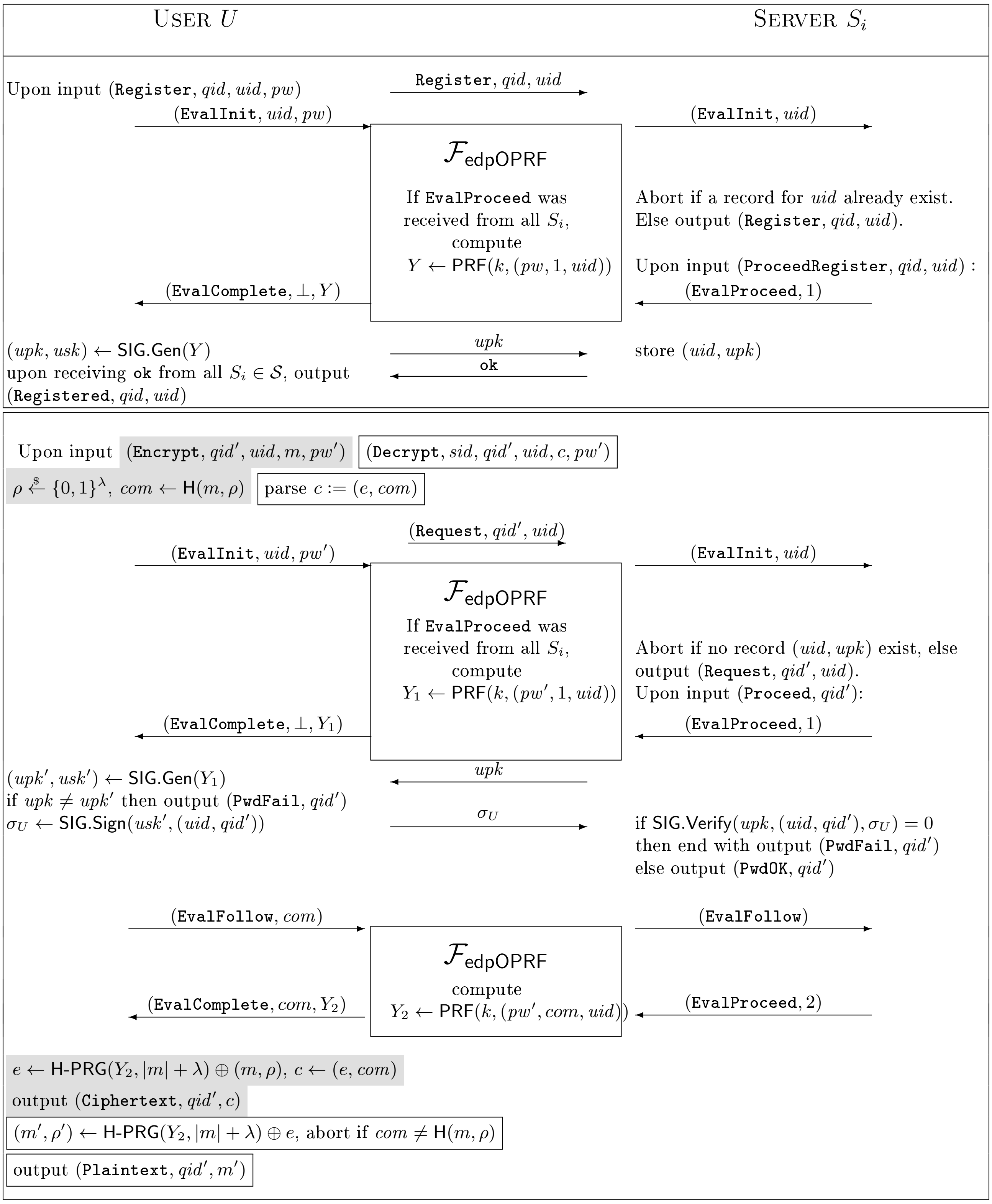
USER $U$
SERVER $S_i$
Upon input $(\texttt{Register}, qid, uid, pw)$
$\texttt{Register}, qid, uid$
$(\texttt{EvalInit}, uid, pw)$
$(\texttt{EvalInit}, uid)$
$\mathcal{F}_{\mathsf{edpOPRF}}$
If $\texttt{EvalProceed}$ was received from all $S_i$, compute $Y \leftarrow \mathsf{PRF}(k, (pw, 1, uid))$
Abort if a record for $uid$ already exist. Else output $(\texttt{Register}, qid, uid)$.
Upon input $(\texttt{ProceedRegister}, qid, uid)$ :
$(\texttt{EvalComplete}, \perp, Y)$
$(\texttt{EvalProceed}, 1)$
$(upk, usk) \leftarrow \mathsf{SIG.Gen}(Y)$
upon receiving $\texttt{ok}$ from all $S_i \in \mathcal{S}$, output $(\texttt{Registered}, qid, uid)$
$upk$
$\texttt{ok}$
store $(uid, upk)$
Upon input $(\texttt{Encrypt}, qid', uid, m, pw')$ $\boxed{(\texttt{Decrypt}, sid, qid', uid, c, pw')}$
$\rho \overset{\$}{\leftarrow} \{0,1\}^\lambda$, $com \leftarrow \mathsf{H}(m, \rho)$
$\boxed{\text{parse } c := (e, com)}$
$(\texttt{Request}, qid', uid)$
$(\texttt{EvalInit}, uid, pw')$
$(\texttt{EvalInit}, uid)$
$\mathcal{F}_{\mathsf{edpOPRF}}$
If $\texttt{EvalProceed}$ was received from all $S_i$, compute $Y_1 \leftarrow \mathsf{PRF}(k, (pw', 1, uid))$
Abort if no record $(uid, upk)$ exist, else output $(\texttt{Request}, qid', uid)$.
Upon input $(\texttt{Proceed}, qid')$:
$(\texttt{EvalComplete}, \perp, Y_1)$
$(\texttt{EvalProceed}, 1)$
$(upk', usk') \leftarrow \mathsf{SIG.Gen}(Y_1)$
if $upk \neq upk'$ then output $(\texttt{PwdFail}, qid')$
$\sigma_U \leftarrow \mathsf{SIG.Sign}(usk', (uid, qid'))$
$upk$
$\sigma_U$
if $\mathsf{SIG.Verify}(upk, (uid, qid'), \sigma_U) = 0$ then end with output $(\texttt{PwdFail}, qid')$ else output $(\texttt{PwdOK}, qid')$
$(\texttt{EvalFollow}, com)$
$(\texttt{EvalFollow})$
$\mathcal{F}_{\mathsf{edpOPRF}}$
compute $Y_2 \leftarrow \mathsf{PRF}(k, (pw', com, uid))$
$(\texttt{EvalComplete}, com, Y_2)$
$(\texttt{EvalProceed}, 2)$
$e \leftarrow \mathsf{H\text{-}PRG}(Y_2, |m| + \lambda) \oplus (m, \rho)$, $c \leftarrow (e, com)$
output $(\texttt{Ciphertext}, qid', c)$
$\boxed{(m', \rho') \leftarrow \mathsf{H\text{-}PRG}(Y_2, |m| + \lambda) \oplus e, \text{ abort if } com \neq \mathsf{H}(m, \rho)}$
$\boxed{\text{output } (\texttt{Plaintext}, qid', m')}$

**Fig. 5.** Our protocol $\Pi_{\mathsf{DPaSE}}$ using a signature scheme $\mathsf{SIG}$, $\Pi_{\mathsf{edpOPRF}}$ protocol and hash functions $\mathsf{H}, \mathsf{H\text{-}PRG}$. Top box shows registration, bottom box shows encryption and decryption. Gray instructions are only executed in encryption, $\boxed{\text{framed}}$ ones only in decryption. Each encryption and decryption query has to use a fresh subsession identifier $qid'$.

But now, $\Pi_{\mathsf{DPaSE}}$ crucially relies on the extendability of the PRF to ensure correct and authenticated encryption of message $m$ under encryption randomness $\rho$. Namely, the key is computed from $\mathsf{H}(m, \rho)$ and $uid, pw$ that successfully verified before. Note that this requires to evaluate the PRF on three inputs, while the two latter are reused from the password verification procedure detailed above. The extendability property of $\mathcal{F}_{\mathsf{edpOPRF}}$ allows this by calling `EvalFollow` with input $\mathsf{H}(m, \rho)$, still using the identifier $qid'$ of the ongoing encryption session. The user obtains $Y_2 \leftarrow \mathsf{PRF}(k, (pw, \mathsf{H}(m, \rho), uid))$ from $\mathcal{F}_{\mathsf{edpOPRF}}$.

To encrypt, $Y_2$ is XORed with $(m, \rho)$ (applying $\mathsf{H\text{-}PRG}$ first to account for differences in lengths). The resulting ciphertext is augmented with $\mathsf{H}(m, \rho)$. The reason for appending this auxiliary information will become apparent below.

**Decryption:** In order to compute a decryption key, a user first has to successfully pass password verification. This is done in the exact same way as for an encryption request (in fact, in our protocol, servers cannot distinguish an encryption request from a decryption request). Computation of the decryption key is also done the exact same way as in encryption – now it becomes apparent why $com \leftarrow \mathsf{H}(m, \rho)$ is required to be part of the ciphertext. The user decrypts $m, \rho$ by XORing the decryption key with the first part of the ciphertext. Finally, the user verifies correct decryption by recomputing $com$ from $m, \rho$. While $\mathcal{F}_{\mathsf{edpOPRF}}$ already provides correct results, the latter check is still required since otherwise faulty ciphertexts (where the $com$ contains another message) would decrypt faithfully and users would recover data that they never encrypted.

## 4.3 Security of $\Pi_{\mathsf{DPaSE}}$

For analyzing the security of $\Pi_{\mathsf{DPaSE}}$ we assume that honest users delete all protocol values such as $Y_1, Y_2, usk$ after performing an encryption or decryption, i.e., upon closing a subsession. Further, we assume that within ongoing subsessions (the identifier $qid$ indicates one such session) an honest user does not get corrupted. This seems reasonable given the fact that, in reality, the time between password verification and encryption (or decryption) will be only very few seconds.

**Theorem 2.** *The protocol $\Pi_{\mathsf{DPaSE}}$ given in Figure 5 with $\mathsf{H}, \mathsf{H\text{-}PRG}$ modeled as random oracles and $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ an EUF-CMA-secure signature scheme UC-emulates $\mathcal{F}_{\mathsf{DPaSE}}$ in the $\mathcal{F}_{\mathsf{edpOPRF}}$-hybrid random oracle model w.r.t static malicious user corruptions, semi-honest server corruptions, and assuming server-side authenticated and secure channels.*

*Proof (Proof Sketch.).* A detailed description of simulated cases can be found in Table 5 in the Appendix, full version.

**Simulation of honest servers.** Since servers do not obtain any secret input that is kept from the adversary, simulating honest servers is quite trivial: the simulator $\mathcal{S}$ just follows the server's protocol.

**Simulate honest user without password.** First note that the password influences the outputs of the (deterministic) PRF. To know whether former PRF values have to be reused as output (i.e., in case of a correct password), it is enough for the simulator to learn whether password verification was successful. Fortunately, $\mathcal{S}$ learns this information from $\mathcal{F}_{\mathsf{DPaSE}}$ on time (via $(\texttt{Complete}, \dots)$ message) before having to commit to any $\mathcal{F}_{\mathsf{edpOPRF}}$ output.

**Extraction of corrupted user's secrets.** Since any user, even a corrupted one, needs to use $\mathcal{F}_{\mathsf{edpOPRF}}$ in order to obtain a key, $\mathcal{S}$ can extract a corrupted user's password and message or ciphertext from his inputs to $\mathcal{F}_{\mathsf{edpOPRF}}$. Another way to see this is that, while usage of $\mathcal{F}_{\mathsf{edpOPRF}}$ simplifies our $\mathsf{DPaSE}$ simulator's life in this case, the burden is on the protocol realizing $\mathcal{F}_{\mathsf{edpOPRF}}$. This protocol has to ensure that secrets can be extracted from adversarial messages.

**Three different ways to encrypt or decrypt.** The simulation is complicated by the fact that $\mathcal{Z}$ can initiate, e.g., an encryption procedure either via an honest user and then recompute the symmetric decryption key via either a corrupted user or via $\mathcal{Z}$'s adversarial interface at $\mathcal{F}_{\mathsf{edpOPRF}}$. Knowing only the ciphertext so far, $\mathcal{S}$ now needs to produce the symmetric key without knowing the plaintext it should decrypt to. However, all honest servers need to agree to help $\mathcal{Z}$ in computation of the symmetric key. $\mathcal{S}$ can use the server's agreement to obtain the plaintext message from $\mathcal{F}_{\mathsf{DPaSE}}$, compute the key linking this message with the ciphertext and send it to $\mathcal{Z}$ (for this, $\mathcal{S}$ has to program the random oracle $\mathsf{H\text{-}PRG}$ to point to the key).

We obtain the following by combining Theorems 2 and 1.

**Corollary 1.** *Let* $\mathbb{G} = (p, g_1, g_2, g_T, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ *be a bilinear group and* $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_T, \mathsf{H}$, $\mathsf{H\text{-}PRG}$ *hash functions as described in* $\Pi_{\mathsf{edpOPRF}}$ *and* $\Pi_{\mathsf{DPaSE}}$*, modeled as random oracles. If the Gapom-BDH assumption holds for* $\mathbb{G}$, $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is an EUF-CMA-secure signature scheme and* $\mathsf{F}$ *a (standard) PRF, then* $\Pi_{\mathsf{DPaSE}}$ *with* $\mathcal{F}_{\mathsf{edpOPRF}}$ *instantiated by* $\Pi_{\mathsf{edpOPRF}}$ *UC-emulates* $\mathcal{F}_{\mathsf{DPaSE}}$ *in the random oracle model w.r.t static honest-but-curious server corruption and assuming server-side authenticated and secure channels.*

***Towards security against malicious servers.*** In Theorem 2 we restrict to static server corruptions, which is a limitation inherited from building block $\Pi_{\mathsf{edpOPRF}}$, which only features semi-honest security. Our proof however directly carries over to the malicious setting: any maliciously secure realization of $\mathcal{F}_{\mathsf{edpOPRF}}$ plugged into $\Pi_{\mathsf{DPaSE}}$ would yield a maliciously secure $\mathcal{F}_{\mathsf{DPaSE}}$ realization. This is witnessed by our simulation (cf. Table 5 in the Appendix, full version), which considers malicious server behavior *except* for instructions handled by $\mathcal{F}_{\mathsf{edpOPRF}}$. We mention again that considering honest-but-curious servers already captures the main threat to passwords: an adversary stealing the password database (or other offline-attackable information).

## 5 Evaluation & Comparison

In this section, we consider an instantiation of the $\Pi_{\mathsf{DPaSE}}$ protocol (from Section 4.2), where the functionality $\mathcal{F}_{\mathsf{edpOPRF}}$ is instantiated with $\Pi_{\mathsf{edpOPRF}}$ (from Section 3.2), and the signature scheme $\mathsf{SIG}$ with ECDSA. We report on the efficiency of our scheme, by counting the number of exponentiations per group and pairings, being the most expensive operations of such protocols. We compare our $\Pi_{\mathsf{DPaSE}}$ protocol with what we believe

| Scheme | #(Exponentiations + Pairings) per Encryption | |
|---|---|---|
| | client/rate limiter | server |
| PHE [25] | 7 exps (in $\mathbb{G}$) | 10 exps (in $\mathbb{G}$) |
| $\Pi_{\mathsf{DPaSE}}$ (Our Work) | 10 exps ($= 2\mathbb{G}_1 + 2\mathbb{G}_2 + 4\mathbb{G}_T + 2\mathbb{G}_{\mathsf{p\text{-}256}}$) | 4 exps ($= 2\mathbb{G}_T + 2\mathbb{G}_{\mathsf{p\text{-}256}}$) +2 pairings |

**Table 2.** Comparison of $\Pi_{\mathsf{DPaSE}}$ with closest password-based encryption scheme PHE, where the exponentiations are counted per group $\mathbb{G}$, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, the pairing is mapped as $\mathbb{G}_1 \times \mathbb{G}_2 \colon \to \mathbb{G}_T$ and $\mathbb{G}_{\mathsf{p\text{-}256}}$ represents the prime group in the ECDSA signature scheme secp256r1.

to be the closest related password-based encryption scheme, namely Password Hardened Encryption (PHE) [25] (see also Table 1 for the overlap of properties of both schemes). Considering each exchange of messages between the client and servers as one round of communication, $\Pi_{\mathsf{DPaSE}}$ requires 2 rounds for Account Creation and 3 rounds for Authentication followed by a Encryption (Decryption) request.

| Protocol | No. of Servers | Execution Time (in ms) | | Requests per server (per second) |
|---|---|---|---|---|
| | | user | server | |
| Account Creation | 2 | 18 | 13 | 76 |
| | 6 | 19 | 13 | 76 |
| | 8 | 19 | 13 | 76 |
| | 10 | 19 | 13 | 76 |
| Authenticate + Encrypt | 2 | 32 | 27 | 37 |
| | 6 | 37 | 27 | 37 |
| | 8 | 40 | 27 | 37 |
| | 10 | 43 | 27 | 37 |

**Table 3.** Timing measurements of the protocol $\Pi_{\mathsf{DPaSE}}$ run between one user and $k = \{2, 6, 8, 10\}$ number of servers.

***Benchmarks.*** We carried out a proof-of-concept implementation [1] of our $\Pi_{\mathsf{DPaSE}}$ protocol and report preliminary benchmarks on the same. We implement in Java, and use the MIRACL - AMCL library for the pairing computation and exponentiation operations. We use the Boneh-Lynn-Shacham pairing with 461 bit curves for the pairing $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ in $\Pi_{\mathsf{edpOPRF}}$, ECDSA with sec256r1, SHA-512 as the underlying hash function $\mathsf{H}$, AES-256 to construct the standard PRF function $\mathsf{F}$ and $\mathsf{H\text{-}PRG}$ and the Java's inbuilt KeyPairGenerator class for user key pair generation $\mathsf{SIG.Gen}$. The elements in groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are implemented using single exponentiation operations with the respective group generators. The hash functions $\mathsf{H}_1$, $\mathsf{H}_2$ and $\mathsf{H}_T$ are implemented by first applying SHA-512 followed by an exponentiation in the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ respectively.

We measured our implementation on a machine running a Intel Core i7-7500U series CPU with 4 virtual CPUs, 16 GiB of RAM. We focused on measuring the local computation times both on the client and the server sides, and did not consider delays due to network latency. The details of our timing measurements corresponding to a $\Pi_{\mathsf{DPaSE}}$ protocol run between one user and a number of servers can be found in the Table 3. The time taken by each server for processing an account creation is 13 milliseconds, while that required for processing an user authentication followed by an encryption request is 27 milliseconds. Consequently, each server is able to process 76 account creation requests and 37 encryption requests per second. Since the computation underlying an encryption or a decryption is almost the same, we have only detailed the encryption timings. We stress here that the timing benchmarks can be further improved by exploiting the parallelizability of the underlying algorithms as well as utilizing the capabilities of multiple cores of a computer. Since this enhancement was not the focus of our work, in our implementation, we have relied on standard cryptographic libraries as mentioned above, which create the bottleneck in our timing measures. Close to our work, Pythia achieves a throughput of 130 requests ps [ECS+15] (also pointed in [LER+18]). In theory, efficiency of our $\Pi_{\mathsf{DPaSE}}$ protocol is lower-bounded by half the throughput of Pythia, which is 65 enc/dec requests ps. This is because each enc/dec request in $\Pi_{\mathsf{DPaSE}}$ requires 2 OPRF evaluations, and each has the same computational cost as one Pythia evaluation. We note that password verification and encryption both add only little overhead.

***On Scalability.*** Our $\Pi_{\mathsf{DPaSE}}$ protocol is highly scalable as can be observed from the benchmarks in Table 3. The time required by an individual server to process an account creation request or an encryption (decryption) request is independent of the number of servers used. While on the other hand, the time taken by the client only slightly increases with increased number of servers.

***Deployment Considerations.*** Let us now address some of the key points to be considered when deploying the $\Pi_{\mathsf{DPaSE}}$ protocol in a real-world environment. Availability is an inherent challenge in any distributed protocol, naturally also in $\Pi_{\mathsf{DPaSE}}$. If one of the servers goes offline due to a Denial-of-Service attack or simply because of a network connection problem, then this would lead to an abort in $\Pi_{\mathsf{DPaSE}}$. This issue can be addressed by using multiple machines for each server and load balancing between them.

In a real-world deployment, servers would ideally be run by different organisations on different physical machines/locations. But $\Pi_{\mathsf{DPaSE}}$ could also be used by a single organization providing all servers, which uses the protocol to "internally" distribute the secret key and thereby minimize the risk of a detrimental server breach.

From the usability perspective, if end-users wish to change their passwords at a later point of time, that is possible and will require decrypting the files under the old password first, and re-encrypt them under the new password. This seems (somewhat) inherent in any scheme where ciphertexts truly depend on passwords, which is needed to protect against a full set of corrupt servers.

## 6 Conclusion and Open Questions

In this paper, we attempted to answer the following question: *Can we design a device-independent cryptographic protocol for password-based encryption with very strong security and privacy?* We formalize our interpretation of strong security and privacy by introducing the notion of *Distributed Password-Authenticated Symmetric Encryption* (DPaSE). DPaSE uses ciphertext-specific encryption keys, prevents encryption under mistyped passwords, hides users' access pattern, protects against on- and offline attacks on the user password, and maintains all these guarantees even in a concurrent setting with arbitrary other protocols.

We answer the question above in the affirmative, by providing a DPaSE protocol based on a new type of oblivious pseudo-random functions (OPRF). The OPRF is evaluated twice: first, to let the user turn her password into high-entropy authentication data, and second, to let the user compute a password- and ciphertext-dependent symmetric key. We give a construction for such an OPRF, which we believe is of independent interest as a new building block for password-based cryptographic protocols. We provide proof-of-concept implementations for our DPaSE construction (including our OPRF construction) and compare efficiency to related protocols in the literature. Our protocol provides only little overhead over existing solutions for password-based key retrieval/encryption, scales well in the number of users and servers, and features provable security under standard bilinear discrete-log based assumptions in the random oracle model.

An interesting future direction is the construction of a *threshold* version of DPaSE, where only an arbitrary subset of all servers is required to participate in each user request. This would improve usability of the protocol, since users would not have to wait for answers of busy servers. Our user-specific OPRF keys ($osk_i = \mathsf{F}(k, uid)$) hinders us to choose $osk_i$ as shares of standard threshold scheme. However constructing $\mathsf{F}$ as threshold PRF might give an interesting solution towards thresholdization.

Finally, security in the presence of malicious servers would be enabled by constructing a maliciously secure extendable distributed partially-oblivious PRF. Alternatively, for ensuring correct encryption it seems to be sufficient to have servers use the same keys in both OPRF evaluations. This flavor of verifiability in our OPRF seems to be achievable with standard techniques. Although key switching between different requests of a specific user would not significantly weaken but clutter the description of $\mathcal{F}_{\mathsf{DPaSE}}$, we decide to present the more secure and cleaner version here, and leave the slightly weaker but maliciously secure version as future work.

## References

1. Dpase poc implementation. https://gitlab.com/DPaSEcode/dpase-submission-code.
2. Internet identity: The end of usernames and passwords. https://tinyurl.com/6rrhvzr2.
3. Mega: Secure cloud storage and communication privacy by design. https://mega.nz/.
4. Tresorit: Cloud storage + end-to-end encryption. https://tresorit.com/security/encryption.
5. S. Agrawal, P. Mohassel, P. Mukherjee, and P. Rindal. DiSE: Distributed symmetric-key encryption. pages 1993–2010, 2018.
6. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. pages 433–444, 2011.
7. C. Baum, T. K. Frederiksen, J. Hesse, A. Lehmann, and A. Yanai. Pesto: Proactively secure distributed single sign-on, or how to trust a hacked server. IEEE European Symposium on Security and Privacy, 2020.
8. J. Brost, C. Egger, R. W. F. Lai, F. Schmid, D. Schröder, and M. Zoppelt. Threshold password-hardened encryption services. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.
9. J. Camenisch and A. Lehmann. Privacy-preserving user-auditable pseudonym systems. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P*. IEEE, 2017.
10. J. Camenisch, A. Lehmann, A. Lysyanskaya, and G. Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. pages 256–275, 2014.
11. J. Camenisch, A. Lehmann, and G. Neven. Optimal distributed password verification. pages 182–194, 2015.
12. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. pages 136–145, 2001.
13. M. Christodorescu, S. Gaddam, P. Mukherjee, and R. Sinha. Amortized threshold symmetric-key encryption. In Y. Kim, J. Kim, G. Vigna, and E. Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.
14. A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart. The pythia PRF service. pages 547–562, 2015.
15. M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. 2012.

16. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. pages 233–253, 2014.
17. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *IEEE European Symposium on Security and Privacy, EuroS&P*, 2016.
18. S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. pages 39–58, 2017.
19. S. Jarecki, H. Krawczyk, and J. K. Resch. Updatable oblivious key management for storage systems. pages 379–393, 2019.
20. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. pages 456–486, 2018.
21. S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. pages 577–594, 2009.
22. M. Klooß, A. Lehmann, and A. Rupp. (R)CCA secure updatable encryption with integrity protection. pages 68–99, 2019.
23. V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. pages 818–829, 2016.
24. M.-S. Lacharité, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. pages 297–314, 2018.
25. R. W. F. Lai, C. Egger, M. Reinert, S. S. M. Chow, M. Maffei, and D. Schröder. Simple password-hardened encryption services. In *27th USENIX Security Symposium, USENIX Security*, 2018.
26. A. Lehmann. Scrambledb: Oblivious (chameleon) pseudonymization-as-a-service. *Proc. Priv. Enhancing Technol.*, 2019.
27. A. Lehmann and B. Tackmann. Updatable encryption with post-compromise security. pages 685–716, 2018.
28. X. Wang and B. Huson. Robust distributed symmetric-key encryption. *IACR ePrint*, 2020.