Flexible Long-Term Secure Archiving

Zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)
Genehmigte Dissertation von Philipp Muth aus Heidelberg
Tag der Einreichung: 4. November 2022, Tag der Prüfung: 16. Dezember 2022

1. Gutachten: Prof. Marc Fischlin
2. Gutachten: Prof. Stefan Katzenbeisser
Darmstadt, Technische Universität Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department
Security Engineering

Flexible Long-Term Secure Archiving

Accepted doctoral thesis by Philipp Muth

Date of submission: 4. November 2022
Date of thesis defense: 16. Dezember 2022

Darmstadt, Technische Universität Darmstadt

Erklärungen laut Promotionsordnung

§ 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 4. November 2022

Philipp Muth

Preface

My first contact with academic research was during my master thesis at Julius-Maximilians-Universität Würzburg. While my interest in solving hard technical problems was sparked already during my bachelor studies, my supervisor, Prof. Jörn Steuding, managed to fully ignite my passion for research in the area of computer science and mathematics. Whether I am overall grateful for this, cannot yet be determined with absolut certainty.

First, I would like to thank Stefan Katzenbeisser for his supervision of this thesis and my Ph.D. studies at TU Darmstadt. His input was most instructive in my research and shed light on many viewpoints regarding research questions previously unnoticed.

Marc Fischlin is owed my gratitude for introducing me to the topic of security games and reduction proofs and in that highlighting the mathematical aspects of cryptographic research. And most of all for devoting his time to be the second examiner of my thesis and giving inspiring remarks on several technical aspects.

To my coauthors Matthias Geihs, Tolga Arul, Johannes Buchmann and Felix Günther I am grateful for the instructive and productive discussions we had in our joint work and the experience and knowledge I gathered from those. Most of all I would like to thank Fabio Campos for the plethora of meetings and conversations we had in developing our actively secure key exchange mechanism. This provided my with some insight into hard homogeneous spaces that I would otherwise not have gotten.

I furthermore am forever grateful to Ursula Paeckel and Heike Schmitt-Spall who supported me immeasurably throughout my studies at TU Darmstadt in taking a lot of paperwork and administratory hastle off my hands, that I would otherwise had to have invested weeks to understand and correctly submit.

Next, Julius Hardt and Anna-Katharina Wickert must be thanked for the time they invested in our collaboration of integrating ELSA into CogniCrypt. Julius deserves a special thanks for providing the implementation for this undertaking, while Anna-Katharina contributed her invaluable experience with CogniCrypt. Without them, this integration could not have succeeded.

I thank my former office mates Dominik Püllen and later on Rune Fiedler for sharing my passion in oxygen and well-ventilated offices, much to the dismay of other colleagues. More than that, I am grateful for the daily discussions on their and my research, thereby giving me some highly valued inspiration. Perhaps some day you will find as much joy in coffee as I do.

And finally and most importantly, I would like to ensure Lena Ries of my eternal gratitude for putting up with my exhaustion, moods swings, working at nightly hours and preoccupation with my research when other priorities would have deserved more attention. Also for brewing the many first daily cups of coffee that got me through my Ph.D. studies and many late and inspiring discussions about unsolved research questions.

Contents

1 Introduction

1.1 Motivation

Privacy and data protection have always been basic human needs in any society that makes use of written language. From simple personal correspondence over military communication to trade secrets or medical information, confidentiality has been of utmost importance. The implications of a leak of such sensitive information may prove devastating, as the previous examples illustrate perfectly. Furthermore reliability, that is, integrity and authenticitiy of information, is critical with risks reaching from annoying to lethal as can again be seen in the previous examples.

This need for data protection has carried over from the analogue to the digital age seamlessly with the amount of data being generated, transmitted and stored increasing steadily and containing more and more personal details. And in regard of the developments in computational technology that recent years have seen, such as the ongoing improvements with respect to quantum computing [4] as well as cryptoanalytical advances, the capabilities of attackers on the security of private information have never been more distinct. Thus the need for privacy and data protection has rarely been more dire. Data protection measures typically fall into at least one of the following categories:

Confidentiality implies that a piece of information is only accessible to parties that are authorised. In case of message delivery, this means that only the sender and receiver can read the sent message. For data storage, this means that only those parties can retrieve a stored document that were intended by the owner of the data. It goes without saying that confidentiality represents an integral part of data security.

History holds striking occasions, in which a lack of confidentiality made all the difference. One of the most prominent examples is the battle of Midway in 1942, prior to which the US Navy was able to partially decrypt messages exchanged between the Japanese armed forces. They subsequently were able to run a chosen-plaintext attack against the Japanese crypto system to confirm that the target of a planned attack was indeed Midway and hence could successfully defend their base [69].

Integrity of data says that the data cannot be altered or falsified without the tampering being detected. This includes data in transit, i.e., messages arrive at the receiver as they were sent by the sender as well as data at rest, that is, a piece of information can be retrieved in the same state as it was previously stored. The example of a patient's medical record being handed to a surgeon that is about to perform an operation said patient perfectly demonstrates the importance of data integrity. Incorrect information may well lead to a disastrous outcome in this case.

Authenticity verifies that a message or information originates from the source that it claims to stem from. Both confidentiality and integrity become meaningless without authenticity if a nefarious source can inject information, that is confidential as well as untampered yet maliciously designed. The battle of Midway once more serves as a perfect example of this.

The areas in which data protection has to be considered and implemented are various, as we coarsely sketched above. They can in a general sense be separated into the following categories.

1.1.1 Data in Transit

Data in transit is arguably the state in which it is most vulnerable. Messages can be eavesdropped upon, altered or simply intersected on their way from a sender to a receiver. While in early societies data in transit could be protected by having the messenger escorted by armed guards, the problem has been exasterbated in the modern, i.e., the digital age.

Not only has the amount of information and data being transferred steadily increased with the introduction of computers and later on the internet, but the opportunities to interfere with communications have thereby also seen a steep rise. Today's communication is handled by a plethora of services spanning from traditional email over SMS to instant messaging services such as WhatsApp, Telegram and Threema. This has made the task of protecting data in transit all the more challenging.

1.1.2 Data at Rest

The second area, in which data protection is essential, is data at rest. Data at rest takes many forms. This reaches from an analogue file cabinet over a USB stick or a hard drive at someone's home to large scale cloud storage. In any of these instances, the data contained must be protected from unauthorised access.

Protection for data at rest concerns two aspects: First, physical access must be restricted. That is, no unauthorised person shall be able to unlock the file cabinet or enter a data center's server room in which he or she has no business being in. Second, metaphysical, i.e., digital access must be considered. This is easier said than done, since in the modern work environment almost any document to which more than one person needs access to is stored in a remote location or cloud storage solution, that in many cases the data owner does not even have physical control over. This introduces a great variety of attack vectors for an attacker aiming to access data that is not intended for his eyes. The task of protecting data at rest has therefore become more pressing than ever in recent years.

1.1.3 Computation on Data

Protection for data in transit and data at rest can in some regards be considered "old-school" scenarios, as those existed even before virtually any piece of data, that is gathered, was stored in a digital database. With modern technology, a third challenge has emerged: computation on data. Guaranteeing each individual protection requirement in either of the former scenarios is a challenging task by itself. Yet combinations thereof prove all the more difficult.

For computation, however, the challenge is an altogether different one. When computing on sensitive data locally, protecting that data from external attackers is relatively simple. For larger and massive datasets this task becomes more complicated. The data need not be stored on a single machine or by a single party. Aggregating all necessary inputs to a computation and subsequently executing it in a manner that is confidential, preserves integrity and is authenticated poses an interesting challenge. This becomes even harder when two or more parties come together to execute a computation on their respective private inputs. In this case the computation should be executed in a way so that, for each party, other parties taking part in the computation do not learn its private inputs.

In conclusion, the protection of data in a modern digital environment is a multi-faceted and not at all simple challenge.

1.2 Research Questions

From the protection needs and scenarios in which data protection has to be deployed, we derive the following research questions:

1. How can data at rest be protected in all three aspects (confidentiality, integrity and authenticity) so that these three goals can simultaneously be guaranteed over extensive periods of time? This includes granting authorised parties access to the data and only those parties. In short: how must a long-term storage solution that serves multiple clients simultaneously be designed so that it maintains all three protection requirements with respect to data stored in it?

   Long-term storage implies several challenges. First, there is confidentiality. Traditional encryption schemes are a tried and tested method to provide confidentiality to whatever message was encrypted. Yet virtually any viable encryption scheme is based on a cryptographic hardness assumption, i.e., that a specific computational problem cannot be solved efficiently with the hardware and algorithms available at the time of usage. In a long-term storage setting, however, it cannnot be guaranteed that these assumptions hold true perpetually. Thus other methods to ensure confidentiality have to be found and appplied.

   A similar problem arises with respect to long-term integrity. An integrity measure cannot at the same time be unconditionally binding and unconditionally hiding, as was proven by Brassard et al. [23]. In a long-term storage setting we must not risk leaking any information on a stored document. Thus an integrity measure that is unconditionally binding is out of the question, leaving integrity measures that are binding under a cryptographic hardness assumption. This implies however that the binding property weakens over time.

   An additional challenge here is therefore: How does one achieve long-term confidentiality and how does one achieve long-term integrity in a storage architecture?

2. Braun et al. [26] introduced LINCOS, the first long-term storage architecture that provides confidentiality as well as integrity to a stored document. Geihs et al. [62] improved upon LINCOS with ELSA in that they enabled storing large datasets rather than single documents while maintaining the security guarantees of LINCOS with respect to the stored data. Both their schemes consider the long-term storage setting, that is, they are designed with storage periods of several decades in mind. The parties engaged in an instance of these architectures have to communicate via channels that hold secure against adversaries, the cryptoanalytical capabilities and computational power of which increase throughout the run-time of the storage architecture.

   The question hence is: How can one establish a secure private channel with respect to an attacker that becomes more powerful over time?

3. In the majority of cases, data collection and storage is done with the intent of analysing and processing it. What is more, in many application scenarios, the data of not just one but several parties has to be combined for successful processing. A party may, however, want to keep its inputs to a joint computation private so that no other party learns information regarding its inputs.

   An intuitive approach for this dilemma is to agree on an external trusted party and have that party execute the computation, distribute the results among those who provided input and delete its knowledge. It is however very likely that such a party does not exist or the input providers cannot agree on an external party. It is for this reason that so-called "multi-party computation protocols" were developed. They enable a set of parties to evaluate a function on their respective private inputs without revealing them to the other parties. While this approach is not as efficient as employing a trusted party for the computation, it is more secure.

   In many examples, the preprocessing that is executed prior to the actual computation consumes a significant part of the total time elapsed for the execution of the protocol. We therefore arrive at the question: How can the preprocessing phase for secret sharing based multi-party computation protocols be sped up at without disproportionate effort and without compromising on the security guarantees of the protocol?

4. Multi-party computation protocols enable parties to evaluate any function, that can be represented as an appropriate circuit, on their private inputs. While this approach provides a wide variety of use cases, a protocol tailored to the specific application achieves higher efficiency in many scenarios. In their recent work, De Feo and Meyer [45] discussed on such scenario, that is, a key exchange mechanism using isogeny based cryptography with a secret key which is shared in a Shamir sharing instance. Their scheme employs a round-robin approach to have an authorised set of shareholders execute the decapsulation protocol in an efficient manner without reconstructing the shared secret key. They furthermore derive a signature scheme by applying the Fiat-Shamir-transform to their key exchange mechanism, that again enables an authorised set of shareholders to sign a message using the shared secret key without reconstructing it. While their scheme strongly indicates at providing the most efficient solution for the specific application they consider, it only provides passive security. That is, a misbehaving shareholders cannot be detected and a falsified decapsulation result cannot, either.

   Hence the question emerges: How can the decapsulation protocol be made actively secure, i.e., how do we detect misbehaving shareholders while maintaining the original security guarantees? And can the resulting key exchange mechanism be transferred into a signature scheme?

5. With ELSA, Geihs et al. [62] presented a long-term secure storage architecture that enables securely storing large datasets in an efficient manner. The subsequent question is how to safely deploy this in a real world application scenario. It is for that aim that CogniCrypt [75] was introduced in 2017. It is an extension to the integrated development environment Eclipse, that is designed to assist a developer in correctly integrating cryptographic code into his or her projects. Implementing cryptographic primitives oneself has many pitfalls. CogniCrypt aims to eliminate those by providing a developer with the option to have cryptographic code tailored and generated to the needs of a specific project in a safe and correct manner. It is mostly focused on integrating primitives rather than complex combinations of those. ELSA itself cannot be considered a cryptographic primitive, yet it combines several classes of primitives such as signature, secret sharing and commitment schemes in each instance. CogniCrypt furthermore provides a static code analyser that can investigate whether a piece of generated cryptographic code is used in a project in a safe manner, so that no security risks arise from deploying it.

   The question thus arises: How can a complex solution like ELSA be integrated into CogniCrypt, so that thereby developers are enabled to integrate it in their projects with an appropriate and secure

choice of primitives and parameters? The challenge is furthermore to ensure that the interaction of the schemes with each other does not inflict unforeseen vulnerabilities.

1.3 Structure

We propose a solution to research question 1 in Section 3. It extends the preexisting ELSA and accommodates several clients simultaneously without compromising on the security guarantees given by ELSA. Research question 2 is discussed in Section 4. We present a channel protocol that holds secure against adversaries that grow in computational power and cryptoanalytical capabilities over time and become unbounded eventually. We address research question 3 in Section 5 and provide a novel approach to speed up the preprocessing phase of any secret sharing based multi-party computation protocol. We improve upon the approach proposed by De Feo and Meyer in Section 6 to address research question 4. And lastly, we discuss research question 5 in Section 7 and elaborate on how to implement ELSA in the context of CogniCrypt to provide developers with the ability to safely deploy an instance of ELSA in a software project.

2 Preliminaries

2.1 Notation

Throughout this work we will use the following notations. Let $n \in \mathbb{Z}$ be an integer number. For another integer d, we write $d|n$ if d is a divisor of n. In several instances we will utilise the integer torus of $\mathbb{Z} \bmod p$, which we will denote by $\mathbb{Z}_p$. The ring of polynomials over $\mathbb{Z}_p$ will be denoted by $\mathbb{Z}_p[X]$ and the set of polynomials $f \in \mathbb{Z}_p[X]$ with degree $\deg(f) = k$ is abbreviated by $\mathbb{Z}_p[X]_k$, whereas the polynomials with degree at most k we denote by $\mathbb{Z}_p[X]_{\leq k}$.

With respect to sets we will use the following notations: for a non-negative integer n, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. For an indexed set $X = \{x_i\}_{i \in I}$, we denote the projection onto a subset $I' \subset I$ of the indexset by $X_{I'} := \{x_i \in X : i \in I'\}$. We use the same notational convention for indexed tuples $(x_i)_{i \in I}$. The cardinality of a set X, i.e., the number of elements it contains, is denoted by $\#X$. The length of a list L is denoted by $|L|$. For a list L, we denote appending an entry m by $L+ = m$.

We fix the following computational notations. For an algorithm $\mathcal{A}$, we denote that $\mathcal{A}$ outputs y upon input x by $y \leftarrow \mathcal{A}(x)$ or $\mathcal{A}(x) \rightarrow y$. If $\mathcal{A}$ is a probabilistic algorithm we instead write $y \leftarrow\$ \mathcal{A}(x)$, $\mathcal{A}(x) \$\rightarrow y$ or $\mathcal{A}(x) \rightarrow_r y$, where r denotes the randomness used in the execution of $\mathcal{A}$. A scheme A consists of one or more protocols. For a protocol Prot provided by A, denote the result of that protocol's execution by $\mathsf{A.Prot}(\cdot)$ for the respective inputs. If the scheme A is clear from the context we may omit it. Take for example a commitment scheme CS (we formally introduce commitment schemes in Section 2.2.1). We denote a call to its commit protocol Commit by $\mathsf{CS.Commit}(m)$, where m is the message committed to.

Let X and Y be two distributions over the same domain D. We denote by $\Delta(X, Y)$ the statistical distance between X and Y, that is,

$$\Delta(X, Y) := \frac{1}{2} \sum_{d \in D} |\Pr[X = d] - \Pr[Y = d]| .$$

Throughout this work we use a security parameter $\lambda \in \mathbb{N}$. It is implicitly handed to a protocol whenever needed, that is, protocols with computational security obtain it as an argument even if we do not explicitly denote it. Information theoretic schemes and protocols such as secret sharing schemes used in this work do not require a security parameter.

We will in several instances make use of a function $\mathsf{time}()$, which takes no argument and returns the time that it was called upon.

2.2 Cryptographic Primitives

We now discuss the cryptographic primitives and schemes that we will use throughout this work, the functionalities they provide and the security guarantees and requirements they entail. The following definitions and notions are in accordance to the works [89, 58, 90] and [31].

2.2.1 Commitment Schemes

A commitment scheme enables a party to publicly commit to a message that it may later on reveal. For that it publishes a commitment value that can be opened with a decommitment value and the message commited to. The aim of a commitment scheme is for the message to not be altered and for the commitment value to leak as little information regarding the message as possible. Commitment schemes are hence often compared to locked boxes, where the message is the content, the decommitment value the key and the filled and locked box represents the commitment value. A commitment scheme $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Vf})$ is thus defined as follows:

$\mathsf{Setup}() \$\rightarrow \mathsf{pk}$ generates a public commitment key pk.

$\mathsf{Commit}(\mathsf{pk}, m) \$\rightarrow (c, d)$ is a (possibly) probabilistic algorithm that takes a commitment key pk and a message $m \in \mathcal{M}$ as input, where $\mathcal{M}$ is a message space, and outputs a commitment value c and a decommitment value d.

$\mathsf{Vf}(\mathsf{pk}, m, c, d) \to b$ takes a commitment key, a message, a commitment value and a decommitment value as input and outputs a bit b that indicates the validity of the commitment c to the message m.

Definition 1 (Correctness)
Let $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Vf})$ *be a commitment scheme. We call* CS correct *if we have*

$$\Pr[\mathsf{Vf}(\mathsf{pk}, m, c, d) = 1] = 1$$

for any $\mathsf{pk} \leftarrow\$ \mathsf{Setup}()$*, any message* m *and any* $(c, d) \leftarrow\$ \mathsf{Commit}(\mathsf{pk}, m)$.

Two security properties are to be considered with respect to a commitment scheme: hiding and binding. The binding property means that the committing party cannot open the commitment to a different message than it was generated to.

Definition 2 (ε-Extractable Binding)
We call a commitment scheme ε-extractable binding *if, for any* t_1*-bounded algorithm* $\mathcal{A}_1$*, there exists a* t_{Ext}*-bounded algorithm* Ext *such that, for any* t_2*-bounded algorithm* $\mathcal{A}_2$*, we have*

$$\Pr\begin{bmatrix} \mathsf{Vf}(\mathsf{pk}, m, c, d) = 1 \wedge m^* \neq m \\ \mathsf{Setup}() \,\$\!\!\to \mathsf{pk}, \mathcal{A}_1(\mathsf{pk}) \to_r c, \\ \mathsf{Ext}(\mathsf{pk}, r) \to m^*, \mathcal{A}_2(\mathsf{pk}, r) \to (m, d) \end{bmatrix} \leq \varepsilon(t_1, t_{\mathsf{Ext}}, t_2).$$

A stronger notion is that of an information-theoretically binding commitment scheme, that is, a commitment scheme $\mathsf{CS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Vf})$, for which for any $\mathsf{pk} \leftarrow\$ \mathsf{Setup}()$ and any commitment value c, there exists just and only one message m and decommitment value d, for which

$$\mathsf{Vf}(\mathsf{pk}, m, c, d) = 1$$

holds.

The hiding property on the other hand implies that little to no information with respect to the message committed to can be derived from the commitment value. For a public commitment key pk, let $C_{\mathsf{pk}}(m)$ denote the distribution of the value c in evaluating $\mathsf{Commit}(\mathsf{pk}, m) \,\$\!\!\to (c, d)$.

Definition 3 (ε-Statistical Hiding)
We call a commitment scheme ε-statistical hiding *if, for any* $\mathsf{pk} \leftarrow \mathsf{Setup}()$ *and for any pair of messages* m_1, m_2*, we have*

$$\Delta(C_{\mathsf{pk}}(m_1), C_{\mathsf{pk}}(m_2)) \leq \varepsilon.$$

An even stronger notion is that of information-theoretical hidingness for a commitment scheme.

Definition 4 (Information-Theoretical Hiding)
A commitment scheme CS *is information-theoretically hiding if, for any* $\mathsf{pk} \leftarrow\$ \mathsf{CS.Setup}()$ *and any two messages* $m_1, m_2 \in \mathcal{M}$*, we have*

$$C_{\mathsf{pk}}(m_1) = C_{\mathsf{pk}}(m_2).$$

That is, no information with respect to a message committed to can be derived from its commitment value.

It was proven by Brassard et al. [23] that a commitment scheme cannot be information-theoretically hiding and information-theoretically binding at the same time. The commitment scheme by Pedersen [93] is information-theoretically hiding. It can thus only be computationally binding.

2.2.2 Keyed Hash Functions

A keyed hash function is defined by a tuple of algorithms (K, H). The probabilistic algorithm K is executed to generate a key k. This key is then handed to the deterministic algorithm H along with a message $m \in \{0,1\}^*$, which outputs a hash value $y \in \{0,1\}^l$ for some fixed $l \in \mathbb{N}$.

Definition 5 (ε-Extractable Binding)
We call a keyed hash function is ε-extractable-binding *if, for any* t_1*-bounded algorithm* $\mathcal{A}_1$*, there exists a* t_{Ext}*-bounded algorithm* Ext*, so that for any* t_2*-bounded algorithm* $\mathcal{A}_2$*,*

$$\Pr_{K \,\$\!\to k}\begin{bmatrix} y = \mathsf{H}(k, x) = \mathsf{H}(k, x^*) \wedge x \neq x^* \\ \mathcal{A}_1(k) \to_r y, \mathsf{Ext}(k, r) \to x^*, \mathcal{A}_2(k, r) \to x \end{bmatrix} \leq \varepsilon(t_1, t_{\mathsf{Ext}}, t_2)$$

holds.

2.2.3 Public Key Infrastructure

Several schemes in this work take a public key as a parameter for their protocols (see for example Section 2.2.5, Section 2.2.6 or Section 2.2.2).

For the ease of notation and understanding, we assume throughout this work the existence of a public key infrastructure (PKI) that maintains the public keys necessary for the employed primitives. This PKI is considered as an implicit parameter for our protocols. The public key will hence be omitted as a parameter where it is appropriate.

2.2.4 Secret Sharing Schemes

A secret sharing scheme allows a dealer $\mathcal{D}$ to distribute a secret s from a secret space G among a set of shareholders $S = \{P_1, \ldots, P_n\}$. It is defined by the protocols $(\mathsf{Setup}, \mathsf{Share}, \mathsf{Reconstruct})$.

$\mathsf{Setup}(S, \Gamma, G)$ takes a set of shareholders $S = \{P_1, \ldots, P_n\}$, an access structure Γ and a secret space G as input and fixes the sharing parameters according to its inputs. The access structure $\Gamma \subset 2^S$ contains all authorised sets of shareholders, i.e., sets that can reconstruct a shared secret from their combined shares. Any set $S' \subset S$ that is not authorised ($S' \notin \Gamma$) cannot reconstruct a shared secret. Γ is called monotonous if, for any $A \in \Gamma$ and $B \supset A$ in S, we have $B \in \Gamma$.

$\mathsf{Share}(s) \$ \rightarrow \{s_1, \ldots, s_k\}$ is executed by a dealer $\mathcal{D}$. It takes a secret $s \in G$ and outputs a set of shares $\{s_1, \ldots, s_k\}$ that are assigned to the shareholders via a surjective mapping $\psi : \{1, \ldots, k\} \rightarrow \{1, \ldots, n\}$, i.e., each shareholder P_i, $i = 1, \ldots, n$, receives all shares s_j with $\psi(j) = i$. The mapping ψ is induced by the access structure Γ.

$\mathsf{Reconstruct}\left(\{s_i\}_{P_{\phi(i)} \in S'}\right) \rightarrow s$ is executed by an authorised set of shareholders $S' \in \Gamma$. It takes their shares as input and outputs the secret that the shares were produced from. If the set S' was not authorised, $\mathsf{Reconstruct}$ fails and outputs $\perp$, that is, no further information on the shared secret is obtained.

If $(\mathsf{Setup}, \mathsf{Share}, \mathsf{Reconstruct})$ is a *proactive* secret sharing scheme, it also provides the protocol $\mathsf{Reshare}(s)$, that renews the shares that each shareholder holds of a secret s. The old shares are thereby rendered obsolete.

Definition 6 (Superauthorised Sets)
For a secret sharing instance $\mathcal{S}$ with shareholders S and access structure Γ, we call a set $S' \subset S$ superauthorised *if, for any $P \in S'$, we have $S' \setminus \{P\} \in \Gamma$. We denote the set of superauthorised sets of shareholders by $\Gamma_{\mathcal{S}}^{+}$.*

Remark
In a monotone access structure, any superauthorised set is also authorised.

To an unauthorised set of shareholders the shares of a shared secret should reveal little to no information on the secret itself.

Definition 7 (ε-Statistical Hiding)
*Let $\mathcal{S}$ be a secret sharing instance and S^* an unauthorised set of shareholders. Let $D_{S^*}(s)$ denote the distribution of the shares received by S^*, when a secret s is shared. We call S ε-*statistical hiding *if, for any two secrets $s, s' \in G$, we have*

$$\Delta\big(D_{S^*}(s), D_{S^*}(s')\big) \leq \varepsilon.$$

If a secret sharing scheme is information theoretically hiding, then the distributions $D_{S^*}(s)$ and $D_{S^*}(s')$ are identical for any unauthorised $S^* \notin \Gamma$.

Example 8 (Shamir's secret sharing)
An instance of Shamir's famous secret sharing scheme consists of a set of $n > 0$ shareholders, a secret space $\mathbb{Z}_p$, where $p \in \mathbb{N}$ is a prime larger than n, and an access structure $\Gamma = \{S' \subset S : \#S' \geq t\}$ for a threshold $t \leq n$. A secret $s \in \mathbb{Z}_p$ is shared by sampling a polynomial $f \in \mathbb{Z}_p[X]_{\leq t-1}$ with $f(0) = s$ and handing each shareholder P_i the interpolation point $f(i)$. Reconstruction is achieved via Lagrange interpolation, that is,

$$s = \sum_{P_i \in S'} L_{i,S'} s_i = \sum_{P_i \in S'} \prod_{\substack{P_j \in S' \\ j \neq i}} \frac{j}{j-i} f(i)$$

$$
\begin{array}{l}
\mathsf{Exp}_{\mathsf{Sig}}^{\text{EUF-CMA}}(\mathcal{A}) \\
\hline
(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}() \\
Q = \{\} \\
(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}(\cdot)}(\mathsf{pk}) \\
\textbf{if } (m^* \in Q) \\
\quad \textbf{return } 0 \\
\textbf{else} \\
\quad \textbf{return } \mathsf{Vf}(m^*, \sigma^*) \\
\textbf{fi}
\end{array}
\qquad
\begin{array}{l}
\mathcal{O}_{\mathsf{Sign}}(m) \\
\hline
Q += m \\
\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \\
\textbf{return } \sigma
\end{array}
$$

Figure 2.1: Experiment $\mathsf{Exp}_{\mathsf{Sig}}^{\text{EUF-CMA}}(\mathcal{A})$

for some authorised $S' \in \Gamma$ and Lagrange interpolation coefficients

$$L_{i,S'} = \prod_{\substack{P_j \in S' \\ j \neq i}} \frac{j}{j-i}.$$

The superauthorised sets of shareholders are

$$\Gamma^+ = \{S' \subset S : \#S' > t\}.$$

2.2.5 Signature Schemes

A signature scheme is a cryptographic public key primitive defined by a triple $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Vf})$. It enables an owner of a secret key to generate a signature σ on a message $m \in \mathcal{M}$, where $\mathcal{M}$ denotes a message space, so that any party that knows the signer's public key can verify the correctness of the signature with respect to the message. More concretely, a signature scheme Sig is defined by the protocols:

$\mathsf{Setup}(1^\lambda) \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$ takes the security parameter as input and outputs and appropriately sampled public and secret key pair from a key space $\mathcal{K}$.

$\mathsf{Sign}(\mathsf{sk}, m) \xrightarrow{\$} \sigma$ is a probabilistic algorithm that takes a secret key sk and a message m from a message space $\mathcal{M}$ and outputs a signature σ.

$\mathsf{Vf}(\mathsf{pk}, m, \sigma) \rightarrow b$ takes the public key pk, a message m and a signature σ as inputs and outputs a decision bit b, indicating the validity of the signature σ with respect to m.

Definition 9 (Correctness)
We call a signature scheme correct *if*

$$\Pr[\mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1] = 1$$

holds for all $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{Setup}()$ *and any* $m \in \mathcal{M}$*, where* $\sigma \leftarrow\$ \mathsf{Sign}(\mathsf{sk}, m)$.

A party without knowledge of the secret key should not be able to provide a signature on a message that was not originally signed with the secret key. We capture this in Experiment $\mathsf{Exp}_{\mathsf{Sig}}^{\text{EUF-CMA}}(\mathcal{A})$ (Figure 2.1).

Definition 10 (ε-Existential Unforgeability)
We say that a signature scheme Sig *is* ε-existentially unforgeable under chosen message attack *(ε − EUF-CMA) or* ε-secure *if, for any t-bounded algorithm $\mathcal{A}$, we have*

$$\mathsf{Adv}_{\mathsf{Sig}}^{\text{EUF-CMA}}(\mathcal{A}) := \Pr\left[\mathsf{Exp}_{\mathsf{Sig}}^{\text{EUF-CMA}}(\mathcal{A}) = 1\right] \leq \varepsilon(t).$$

2.2.6 Timestamp Schemes

A timestamp scheme is a cryptographic primitive defined by a triple $(\mathsf{Setup}, \mathsf{Stamp}, \mathsf{Vf})$ that provides a party with the means of generating a timestamp on a message that can later on be utilised to prove the existence of said message at the point in time the timestamp was generated.

$\mathsf{Exp}_{\mathsf{TS}}^{\text{EUF-CMA}}(\mathcal{A})$

$\mathsf{Setup}()$
$Q = \{\}$
$(m^*, s^*, t^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Stamp}}(\cdot)}$
if $((m^*, t^*) \in Q)$
  return 0
else
  return $\mathsf{TS.Vf}(m^*, (s^*, t^*))$
fi

$\mathcal{O}_{\mathsf{Stamp}}(m)$

$(s, t) \leftarrow \mathsf{TS.Stamp}(m)$
$Q += (m, t)$
return (s, t)

Figure 2.2: Experiment $\mathsf{Exp}_{\mathsf{TS}}^{\text{EUF-CMA}}(\mathcal{A})$

$\mathsf{Setup}()$ is executed by the timestamp service to initialise itself. Depending on the concrete instantiation this may return a key or key pair to be utilised by Stamp and Vf.

$\mathsf{Stamp}(m) \xrightarrow{\$} (s, t)$ is executed to produce a timestamp on a message m. It outputs (s, t), where s is a timestamp and t the time that Stamp was called upon.

$\mathsf{Vf}(m, (s, t)) \rightarrow b$ takes a message m, a timestamp s and a time t. It outputs a decision bit b indicating whether the timestamp was correct.

Definition 11 (Correctness)
We call a timestamp scheme TS correct *if, for any message* m *and for any* $(s, t) \leftarrow\$ \mathsf{Stamp}(m)$*, we have*

$$\Pr[\mathsf{Vf}(m, (s, t))] = 1$$

after $\mathsf{TS.Setup}()$ *has been executed.*

An attacker should not be able to provide a timestamp (s, t) for a message m that was not previously stamped by the timestamp service at time t. We capture this in Experiment $\mathsf{Exp}_{\mathsf{TS}}^{\text{EUF-CMA}}(\mathcal{A})$ (Figure 2.2). In this security game, an adversary $\mathcal{A}$ is given access to a stamping oracle $\mathcal{O}_{\mathsf{Stamp}}$, that upon being queried with a message m returns a timestamp (s, t). $\mathcal{A}$ eventually returns a message m^* and a timestamp (s^*, t^*). If the timestamp is successfully validated with respect to m^* and $\mathcal{O}_{\mathsf{Stamp}}$ was not previously queried with m^*, $\mathcal{A}$ wins Experiment $\mathsf{Exp}_{\mathsf{TS}}^{\text{EUF-CMA}}(\mathcal{A})$.

Definition 12 (ε-Existential Unforgeability)
We call a timestamp scheme ε-existentially unforgeable under chosen message attack *if, for any* t*-bounded adversary* $\mathcal{A}$*, we have*

$$\mathsf{Adv}_{\mathsf{TS}}^{\text{EUF-CMA}}(\mathcal{A}) := \Pr\left[\mathsf{Exp}_{\mathsf{TS}}^{\text{EUF-CMA}}(\mathcal{A}) = 1\right] \leq \varepsilon(t).$$

Several methods of instantiation for a timestamp scheme have been developed over the years, one of which is via a signature scheme such as the signature scheme derived from Schnorr's identification scheme [99] via the Fiat-Shamir transform [56]. We give a sketch of a timestamp scheme instantiated using a signature scheme Sig in Figure 2.3. For that the timestamp service executes $\mathsf{Sig.Setup}() \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$ to establish a key pair. To timestamp a message m, the message along with the current time $t \leftarrow \mathsf{time}()$ is signed via $s = \sigma \leftarrow\$ \mathsf{Sig.Sign}(\mathsf{sk}, m||t)$. The correctness of a timestamp is trivially checked by executing $\mathsf{Sig.Vf}(\mathsf{pk}, m||t, s)$.

$\mathsf{TS.Setup}()$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{Sig.Setup}()$

$\mathsf{TS.Stamp}(m)$

$t \leftarrow \mathsf{time}()$
$s \leftarrow\$ \mathsf{Sig.Sign}(\mathsf{sk}, m||t)$
return (s, t)

$\mathsf{TS.Vf}(m, (s, t))$

$b \leftarrow \mathsf{Sig.Vf}(\mathsf{pk}, m, (s, t))$
return b

Figure 2.3: A timestamp service instantiation using a signature scheme

It is obvious that a timestamp scheme initialised with a signature scheme in this manner is $\varepsilon - \text{EUF-CMA}$ if the signature scheme is $\varepsilon - \text{EUF-CMA}$.

2.2.7 Vector Commitment Schemes

A vector commitment scheme enables a prover to commit to a vector of messages $(m_1, \ldots, m_n) \in \mathcal{M}^n$ from a message space $\mathcal{M}$ and $n \leq L$ for some fixed $L \in \mathbb{N}$ rather than to a single message in a traditional commitment scheme (see Section 2.2.1). A vector commitment scheme $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Vf})$ is defined by four protocols:

$\mathsf{Setup}() \xrightarrow{\$} \mathsf{pk}$ generates a public commitment key pk similar to a traditional commitment scheme.

$\mathsf{Commit}(\mathsf{pk}, (m_1, \ldots, m_n)) \xrightarrow{\$} (c, D)$ is a probabilistic algorithm that takes a commitment key pk and a list of messages $(m_1, \ldots, m_n)$ as input and outputs a pair of commitment and vector decommitment (c, D).

$\mathsf{Open}(\mathsf{pk}, D, i) \rightarrow d$ takes a commitment key pk, a vector decommitment D and an index i. It outputs a decommitment value d for the i-th message that was used in generating D.

$\mathsf{Vf}(\mathsf{pk}, m, c, d, i) \rightarrow b$ takes a commitment key pk, message m, commitment and decommitment value c and d along with an index i and outputs a decision bit b to indicate whether d is valid decommitment value with regards to m and c.

Definition 13 (Correctness)
We call a vector commitment scheme VC correct *if, for any* $\mathsf{pk} \leftarrow\!\!\$\ \mathsf{VC.Setup}()$ *and for any* $(m_1, \ldots, m_n) \in \mathcal{M}^n$ *with* $n \leq L$*, we have*

$$\Pr[\mathsf{VC.Vf}(\mathsf{pk}, m_i, c, d, i) = 1] = 1,$$

where $(c, D) \leftarrow\!\!\$\ \mathsf{VC.Commit}(\mathsf{pk}, (m_1, \ldots, m_n))$ *and* $d \leftarrow \mathsf{VC.Open}(\mathsf{pk}, D, i)$ *for all* $i = 1, \ldots, n$.

An adversary that has knowledge of a commitment value c and a subset of decommitment values D_I should not be able to derive information with regards to those messages, for which he did not obtain the decommitment values.

Definition 14 (ε-Statistically Hiding)
Let VC *be a vector commitment scheme. We call* VC ε-statistically hiding (under selective opening) *if, for any* $\mathsf{pk} \leftarrow\!\!\$\ \mathsf{Setup}()$*, any* $n \in [L]$*, any* $I \subset [n]$ *and any* $M_1, M_2 \in \mathcal{M}^n$ *with* $(M_1)_I = (M_2)_I$*, we have*

$$\Delta(\mathsf{CD}_k(M_1, I), \mathsf{CD}_k(M_2, I)) \leq \varepsilon,$$

where, for a message vector $M \in \mathcal{M}^n$*, a commitment* $(c, D) \leftarrow\!\!\$\ \mathsf{VC.Commit}(\mathsf{pk}, M)$ *and* $D' = \bigcup_{i \in [n]} \mathsf{VC.Open}(\mathsf{pk}, D, i)$*,* $\mathsf{CD}_k(M, I)$ *denotes the random variable* (c, D'_I).

The definition of ε-extractable bindingness carries over from standard commitment schemes.

Definition 15 (ε-Extractable Bindingness)
We call a vector commitment scheme VC ε-extractable binding *if, for all* t_1*-bounded algorithms* $\mathcal{A}_1$*, there exists* t_{Ext}*-bounded algorithm* Ext*, so that, for all* t_2*-bounded algorithms* $\mathcal{A}_2$*,*

$$\Pr\left[\begin{array}{r} \mathsf{Vf}(\mathsf{pk}, m, c, d, i) = 1 \wedge m_i \neq m: \\ \mathsf{pk} \leftarrow\!\!\$\ \mathsf{Setup}(), \mathcal{A}_1(\mathsf{pk}) \rightarrow_r c, \\ \mathsf{Ext}(\mathsf{pk}, r) \rightarrow (m_1, \ldots, m_n) \in \mathcal{M}^n, \mathcal{A}_2(\mathsf{pk}, r) \rightarrow (m, c, d, i) \end{array}\right] \leq \varepsilon(t_1, t_{\mathsf{Ext}}, t_2)$$

holds, where $n \in [L]$.

We now present a vector commitment scheme, that can be instantiated in a ε-statistically hiding and ε-extractable binding fashion for any fixed $\varepsilon > 0$. We construct the vector commitment scheme in two steps: first, we give a set of protocols based on a keyed hash function, which is ε-extractable binding if the hash function is. Second, we combine this extractable binding vector commitment scheme with an ε-statistically hiding commitment scheme to obtain a vector commitment scheme that is both extractable binding as well as statistically hiding.

Let (K, H) denote a keyed hash function. We employ H in our ε-extractable binding vector commitment scheme to construct a Merkle tree [85]. We give the vector commitment scheme in Figure 2.4.

We combine the ε-extractable binding vector commitment scheme VC given in Figure 2.4 with an ε-statistically hiding commitment CS to arrive at a vector commitment scheme VC' that is both, ε-extractable binding and ε-statistically hiding. For that we first commit to each message via the ε-statistically hiding commitment scheme and then commit to the collection of the commitment values via the ε-extractable binding vector commitment scheme. The resulting scheme is given in Figure 2.5.

We refer to Geihs et al.'s ELSA [62] for the precise security anaylsis of the resulting vector commitment scheme and give their results below.

$\mathsf{Setup}()$

$\mathsf{k} \leftarrow\$ K()$

return k

$\mathsf{VC.Open}(\mathsf{k}, D, i^*)$

$(h_{i,j})_{i=0,\ldots,l;j=0,\ldots,2^i-1} \leftarrow D$

$a_l \leftarrow i^*$

for $i' = l, \ldots, 1$

  $g_{i'} \leftarrow h_{i',a_{i'}+2(a_{i'}+1 \bmod 2)-1}$

  $a_{i'-1} \leftarrow \left\lfloor \frac{a_{i'}}{2} \right\rfloor$

endfor

$d \leftarrow (g_1, \ldots, g_l)$

return d

$\mathsf{VC.Commit}(\mathsf{k}, (m_1, \ldots, m_n))$

$l \leftarrow \lceil \log_2 n \rceil$

for $j = 0, \ldots, n-1$

  $h_l, j \leftarrow \mathsf{H}(\mathsf{k}, m_{j+1})$

endfor

for $j = n, \ldots, 2^l - 1$

  $h_l, j \leftarrow \perp$

endfor

for $i = l-1, \ldots, 0$

  for $j = 0, \ldots, 2^i - 1$

    $h_{i,j} \leftarrow \mathsf{H}(\mathsf{k}, (h_{i+1,2j}, h_{i+1,2j+1}))$

  endfor

endfor

$c \leftarrow \mathsf{H}(\mathsf{k}, (l, h_{0,0}))$

$D \leftarrow (h_{i,j})_{i=0,\ldots,l;j=0,\ldots,2^i-1}$

return (c, D)

$\mathsf{VC.Vf}(\mathsf{k}, m, c, d, i^*)$

$(g_1, \ldots, g_l) \leftarrow d$

$a_l \leftarrow i^*$

$h_l \leftarrow \mathsf{H}(\mathsf{k}, m)$

for $i = l, \ldots, 1$

  if $a_i \bmod 2 = 0$

    $b_i \leftarrow (h_i, g_i)$

  else

    $b_i \leftarrow (g_i, h_i)$

  fi

  $h_{i-1} \leftarrow \mathsf{H}(\mathsf{k}, b_i)$

  $a_{i-1} \leftarrow \left\lfloor \frac{a_i}{2} \right\rfloor$

endfor

$c' \leftarrow \mathsf{H}(\mathsf{k}, (l, h_0))$

return $(c == c')$

Figure 2.4: An ε-extractable binding vector commitment scheme

$\mathsf{VC'.Setup}()$

$\mathsf{k}_1 \leftarrow\$ \mathsf{CS.Setup}()$

$\mathsf{k}_2 \leftarrow\$ \mathsf{VC.Setup}()$

$\mathsf{k} \leftarrow (\mathsf{k}_1, \mathsf{k}_2)$

return k

$\mathsf{VC'.Open}(\mathsf{k}, D, i)$

$(\mathsf{k}_1, \mathsf{k}_2) \leftarrow \mathsf{k}$

$\left(\left((c_1, d_1), \ldots, (c_n, d_n)\right), D'\right) \leftarrow D$

$d' \leftarrow \mathsf{VC.Open}\left(\mathsf{k}_2, D', i\right)$

$d \leftarrow \left(c_i, d_i, d'\right)$

return d

$\mathsf{VC'.Commit}(\mathsf{k}, (m_1, \ldots, m_n))$

$(\mathsf{k}_1, \mathsf{k}_2) \leftarrow \mathsf{k}$

for $i = 1, \ldots, n$

  $(c_i, d_i) \leftarrow\$ \mathsf{CS.Commit}(\mathsf{k}_1, m_i)$

endfor

$\left(c, D'\right) \leftarrow\$ \mathsf{VC.Commit}(\mathsf{k}_2, (c_1, \ldots, c_n))$

$D \leftarrow \left(\left((c_1, d_1), \ldots, (c_n, d_n)\right), D'\right)$

return (c, D)

$\mathsf{VC'.Vf}(\mathsf{k}, m, c, d, i)$

$(\mathsf{k}_1, \mathsf{k}_2) \leftarrow \mathsf{k}$

$\left(c_i, d_i, d'\right) \leftarrow d$

$b_1 \leftarrow \mathsf{CS.Vf}(\mathsf{k}_1, m, c_i, d_i)$

$b_2 \leftarrow \mathsf{VC.Vf}\left(\mathsf{k}_2, c_i, c, d', i\right)$

return $(b_1 \wedge b_2)$

Figure 2.5: An ε-extractable binding and ε-statistically hiding vector commitment scheme

Lemma 16
The vector commitment scheme detailed in Figure 2.4 is correct.

Theorem 17
If the commitment scheme CS *is correct and the vector commitment scheme* VC *in Figure 2.5 is correct, then* VC′ *is correct.*

Theorem 18
If the commitment scheme CS *in Figure 2.5 is ε-statistically hiding, then the vector commitment scheme* VC′ *is $L \cdot \varepsilon'$-statistically hiding, where L is the the maximum number of messages that can be committed to.*

Lemma 19
If the keyed hash function (K, H) in Figure 2.4 is ε-extractable binding, then VC *is ε'-extractable binding, where*

$$\varepsilon'(t_1, t_{\mathsf{Ext}}, t_2) = 2L \cdot \varepsilon\left(t_1 + \frac{t_{\mathsf{Ext}}}{L}, \frac{t_{\mathsf{Ext}}}{L}, t_2\right).$$

Theorem 20
If CS *and* VC *in Figure 2.5 are ε-extractable binding, then* VC′ *is ε'-extractable binding, where*

$$\varepsilon'(t_1, t_{\mathsf{Ext}}, t_2) = L \cdot \varepsilon\left(t_1 + \frac{t_{\mathsf{Ext}}}{L}, \frac{t_{\mathsf{Ext}}}{L}, t_2\right).$$

2.2.8 Channels

A channel is defined by four protocols Init, OTKey, Send and Recv. Init is executed as an intialisation step in which some shared key material K_I is generated, usually for authentication purposes, and the sender's and receiver's states are initialised. The OTKey protocol lets the sender and receiver generate fresh key material, for example through authenticated quantum key distribution (QKD), to be used only once and in a pre-determined sequence (e.g., the order they are established in QKD). We do not specify in our abstract model how the generation of fresh key material is accomplished. Finally, the Send and Recv protocols allow to process data for the communication.

More formally, a channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{OTKey}, \mathsf{Send}, \mathsf{Recv})$ with associated sending and receiving state space $\mathcal{S}_S$, respectively $\mathcal{S}_R$, message space $\mathcal{M} \subseteq \{0,1\}^{\leq M}$ for some maximum message length $M \in \mathbb{N}$, initialisation key space $\mathcal{K}_{init} = \{0,1\}^{N_{\text{init}}}$ and per-message key space $\mathcal{K}_{msg} = \{0,1\}^N$ for some key lengths $N_{\text{init}}, N \in \mathbb{N}$, error space $\mathcal{E}$ with $\mathcal{E} \cap \{0,1\}^* = \emptyset$, consists of four efficient protocols defined as follows.

$\mathsf{Init}() \xrightarrow{\$} (K_I, \mathsf{st}_S, \mathsf{st}_R)$ outputs an initial key $K_I \in \mathcal{K}_{init}$ and initial sending and receiving states $\mathsf{st}_S \in \mathcal{S}_S$, respectively $\mathsf{st}_R \in \mathcal{S}_R$.

$\mathsf{OTKey}() \xrightarrow{\$} K \in \{0,1\}^N$ generates the next per-message key K for both parties, to be used only once.

$\mathsf{Send}(\mathsf{st}_S, K_I, K, m) \xrightarrow{\$} (\mathsf{st}_S, c)$ takes a sending state $\mathsf{st}_S \in \mathcal{S}_S$, an initial key $K_I \in \mathcal{K}_{init}$, a per-message key $K \in \mathcal{K}_{msg}$ and a message $m \in \mathcal{M}$ as input and probabilisticly outputs an updated state $\mathsf{st}_S \in \mathcal{S}_S$ and a ciphertext (or error symbol) $c \in \{0,1\}^* \cup \mathcal{E}$.

$\mathsf{Recv}(\mathsf{st}_R, K_I, K, c) \rightarrow (\mathsf{st}_R, m)$ takes a receiving state $\mathsf{st}_R \in \mathcal{S}_R$, an initial key $K_I \in \mathcal{K}_{init}$, a per-message key $K \in \mathcal{K}_{msg}$ and a ciphertext $c \in \{0,1\}^*$ and outputs an updated state $\mathsf{st}_R \in \mathcal{S}_R$ and a message (or error symbol) $m \in \mathcal{M} \cup \mathcal{E}$.

Definition 21 (Correctness)
We call a channel $\mathsf{Ch} = (\mathsf{Init}, \mathsf{OTKey}, \mathsf{Send}, \mathsf{Recv})$ correct *if, for any $i \in \mathbb{N}$, any $(K_I, \mathsf{st}_{S,0}, \mathsf{st}_{R,0}) \leftarrow\$ \mathsf{Init}()$, any $(K_1, \ldots, K_i) \in (\mathcal{K}_{msg})^i$ with $K_j \leftarrow\$ \mathsf{OTKey}()$ in sequence for $j = 1$ to $j = i$, any $(m_1, \ldots, m_i) \in \mathcal{M}^i$, any sequence $(\mathsf{st}_{S,1}, c_1) \leftarrow\$ \mathsf{Send}(\mathsf{st}_{S,0}, K_I, K_1, m_1)$, …, $(\mathsf{st}_{S,i}, c_i) \leftarrow\$ \mathsf{Send}(\mathsf{st}_{S,i-1}, K_I, K_i, m_i)$, and $(\mathsf{st}_{R,1}, m'_1) \leftarrow \mathsf{Recv}(\mathsf{st}_{R,0}, K_I, K_1, c_1)$, …, $(\mathsf{st}_{R,i}, m'_i) \leftarrow \mathsf{Recv}(\mathsf{st}_{R,i-1}, K_I, K_i, c_i)$, it holds that*

$$(m_1, \ldots, m_i) = (m'_1, \ldots, m'_i).$$

The messages obtained by the receiver should coincide with those sent by the sender, that is, they should not be altered in transmission without detection. We capture this first security notion in Experiment $\mathsf{Exp}^{\mathsf{INT\text{-}SFCTXT}}_{\mathsf{Ch}}(\mathcal{I})$, which can be found in Figure 2.6.

$\mathsf{Exp}_{\mathsf{Ch}}^{\text{INT-SFCTXT}}(\mathcal{I})$

$(K_I, \mathsf{st}_S, \mathsf{st}_R) \leftarrow\$ \mathsf{Init}()$
$K_1, K_2, K_3, \ldots \leftarrow\$ \mathsf{OTKey}()$
$\textsc{out-of-sync} \leftarrow \mathsf{false}$
$\textsc{int-broken} \leftarrow \mathsf{false}$
$i, j \leftarrow 0$
$\mathcal{I}^{\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, \cdot), \mathcal{O}_{\mathsf{Recv}}(\mathsf{st}_R, K_I, \cdot)}$
return $\textsc{int-broken}$

$\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, m)$

$i \leftarrow i + 1$
$(C_i, \mathsf{st}_S) \leftarrow \mathsf{Send}(\mathsf{st}_S, K_I, K_i, m)$
return C_i

$\mathcal{O}_{\mathsf{Recv}}(\mathsf{st}_R, K_I, C)$

$j \leftarrow j + 1$
$(m, \mathsf{st}_R) \leftarrow \mathsf{Recv}(\mathsf{st}_R, K_I, K_j, C)$
if $(j > i$ **or** $C \neq C_j)$ **then**
  $\textsc{out-of-sync} \leftarrow \mathsf{true}$
endif
if $(m \neq \perp$ **and** $\textsc{out-of-sync})$
  $\textsc{int-broken} \leftarrow \mathsf{true}$
endif
return $\perp$

Figure 2.6: Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\text{INT-SFCTXT}}(\mathcal{I})$

$\mathsf{Exp}_{\mathsf{Ch}}^{\text{IND-CPA}}(\mathcal{A})$

$c \leftarrow\$ \{0, 1\}$
$(K_I, \mathsf{st}_S, \mathsf{st}_R) \leftarrow\$ \mathsf{Init}()$
$K_1, K_2, K_3, \ldots \leftarrow\$ \mathsf{OTKey}()$
$i \leftarrow 0$
$c' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, \cdot, \cdot)}$
return $c == c'$

$\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, K_i, m_0, m_1)$

assert $|m_0| = |m_1|$
$i \leftarrow i + 1$
$(C_i, \mathsf{st}_S) \leftarrow \mathsf{Send}(\mathsf{st}_S, K_I, K_i, m_c)$
return C_i

Figure 2.7: Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\text{IND-CPA}}(\mathcal{B})$

In this game, a channel is initialised and a series of per-message keys is generated. The adversary is then given access to a sending and a receiving oracle. The sending oracle $\mathcal{O}_{\mathsf{Send}}$ takes a sending state st_S, an initialisation key K_I and message m as input, increments the number of received queries i, updates the sending state st_S and outputs a ciphertext C_i. The receiving oracle $\mathcal{O}_{\mathsf{Recv}}$ on the other hand takes a receiving state st_R, an initialisation key K_I and a ciphertext c as input. If the adversary $\mathcal{I}$ queries $\mathcal{O}_{\mathsf{Recv}}$ so that c successfully decrypts and the query was put out of the order of queries to the $\mathcal{O}_{\mathsf{Send}}$, then $\mathcal{I}$ wins the game.

Definition 22 (Ciphertext Integrity)
For an adversary $\mathcal{I}$, we define the advantage in Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\text{INT-SFCTXT}}(\mathcal{I})$ (Figure 2.6) as:

$$\mathsf{Adv}_{\mathsf{Ch}}^{\text{int-sfctxt}}(\mathcal{I}) = \Pr\left[\mathsf{Exp}_{\mathsf{Ch}}^{\text{INT-SFCTXT}}(\mathcal{I}) == \mathsf{true}\right].$$

The second central security notion for channels in this work is that of indistinguishability under chosen-plaintext attacks. We capture this notion in the security game Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\text{IND-CPA}}(\mathcal{B})$ in Figure 2.7. In this game, a challenge bit b is sampled and the channel is initiated via Init. Key material is then generated and the adversary is given access to a sending oracle $\mathcal{O}_{\mathsf{Send}}$. This oracle takes a sending state st_S, a key K_I and messages m_1 and m_2 of identical length and returns a ciphertext C_i, where i denotes the number of queries that have been handed to $\mathcal{O}_{\mathsf{Send}}$. The adversary eventually outputs a bit b'; he wins the game if the challenge bit b and b' coincide.

Definition 23 (Chosen-Plaintext Security)
For a channel protocol $\mathsf{Ch} = (\mathsf{Init}, \mathsf{OTKey}, \mathsf{Send}, \mathsf{Recv})$ and for an adversary $\mathcal{A}$, we define the advantage

$$\mathsf{Adv}_{\mathsf{Ch}}^{\text{IND-CPA}}(\mathcal{A}) = \left|\Pr\left[\mathsf{Exp}_{\mathsf{Ch}}^{\text{IND-CPA}}(\mathcal{B}) == \mathsf{true}\right] - \frac{1}{2}\right|$$

in Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\text{IND-CPA}}(\mathcal{B})$ (Figure 2.7).

A channel protocol Ch) is called secure under chosen-plaintext attack *(IND-CPA-secure) if, for any adversary $\mathcal{A}$,*

$$\mathsf{Adv}_{\mathsf{Ch}}^{\text{IND-CPA}}(\mathcal{A}) = \mathsf{negl}(\lambda),$$

i.e., the advantage of $\mathcal{A}$ is negligible in the implicit security parameter λ.

$\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$

$b \leftarrow\$ \{0,1\}$
$(K_I, \mathsf{st}_S, \mathsf{st}_R) \leftarrow\$ \mathsf{Init}()$
$K_1, K_2, K_3, \ldots \leftarrow\$ \mathsf{OTKey}()$
$\textsc{out-of-sync} \leftarrow \mathsf{false}$
$i, j \leftarrow 0$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\text{Send}}(\mathsf{st}_S, K_I, \cdot, \cdot), \mathcal{O}_{\text{Recv}}(\mathsf{st}_S, K_I, \cdot)}()$
return $b == b'$

$\mathcal{O}_{\text{Send}}(\mathsf{st}_S, K_I, m_0, m_1)$

assert $|m_0| = |m_1|$
$i \leftarrow i + 1$
$(C_i, \mathsf{st}_S) \leftarrow \mathsf{Send}(\mathsf{st}_S, K_I, K_i, m_b)$
return C_i

$\mathcal{O}_{\text{Recv}}(\mathsf{st}_R, K_I, C)$

$j \leftarrow j + 1$
$(m, \mathsf{st}_R) \leftarrow \mathsf{Recv}(\mathsf{st}_R, K_I, K_j, C)$
if $(j > i$ **or** $C \neq C_j)$ **then**
  $\textsc{out-of-sync} \leftarrow \mathsf{true}$
endif
if $(\textsc{out-of-sync}$ **and** $b == 0)$ **then**
  return m
endif
return $\perp$

Figure 2.8: Experiment $\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$

The CPA indistinguishability game is identical to the CCA game but does not give the adversary access to the receiver oracle $\mathcal{O}_{\text{Recv}}$, merging the two-stage adversary into a single one. The integrity experiment allows the adversary to see ciphertexts of chosen messages via oracle $\mathcal{O}_{\text{Send}}$, and merely checks if the adversary manages to send a new or out-of-order ciphertext which decrypts correctly. We thus arrive at the following result.

Proposition 24
Let $\mathsf{Ch} = (\mathsf{Init}, \mathsf{OTKey}, \mathsf{Send}, \mathsf{Recv})$ *be a channel protocol. If* Ch *is secure under chosen-ciphertext attack, then it is also secure under chosen-plaintext attack.*

The third security notion for channels, that we will discuss in this work, follows the common ones for channels (or stateful authenticated encryption) by Bellare, Kohno and Namprempre [11]. It combines confidentiality and integrity in a single game, following what is sometimes referred to as CCA3 security [104]. The adversary $\mathcal{A}$ can repeatedly ask the sender (oracle) to encrypt one of two messages. The choice of which message to encrypt is based on a secret bit b which the adversary tries to predict eventually. On the receiver's side the adversary may submit arbitrary ciphertexts C in order to learn something about the bit b. Indeed, if the adversary manages to forge a ciphertext (decrypting to a non-error) on the receiver's side, either by creating a fresh valid ciphertext or by changing the order of the sender's ciphertexts, then we give the adversary enough information to predict b. The latter is achieved for a ciphertext forgery by returning the encapsulated message m if $b = 0$, and $\perp$ otherwise.

In more detail, the corresponding security game Experiment $\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$ works as follows: The adversary can call the sending oracle $\mathcal{O}_{\text{Send}}$ about two equal-length messages m_0, m_1, then the sender encapsulates m_b (and updates its state st_S) and returns the ciphertext. We keep track of the order of ciphertexts by a counter i. The receiver's oracle $\mathcal{O}_{\text{Recv}}$ is more involved. When called with a ciphertext C it first increments its counter j and then decapsulates the message and updates its state st_R. There are now various cases to distinguish, relating to the question whether the ciphertext C is a forgery or not:

- If $j > i$ or $C \neq C_j$, i.e., if this is a new ciphertext or one which has not been produced by the sender as the i-th ciphertext before, then we say that the ciphertext sequences are not in-sync anymore. This is captured by a flag $\textsc{out-of-sync}$.

- If we have reached an $\textsc{out-of-sync}$ situation, either in this call to $\mathcal{O}_{\text{Recv}}$ or an earlier one, then we provide the adversary with the received message in case $b = 0$. This enforces that, for a scheme to be secure, whenever the received ciphertext sequences goes out of sync, the output of Recv must be $\perp$, as otherwise it would be easily distinguishable from the case $b = 1$ always outputting $\perp$.

The overall goal of the adversary is to predict b, either by distinguishing the messages encapsulated by the sender, or by breaking integrity and learning about b through a receiver's reply.

Definition 25
For a channel protocol $\mathsf{Ch} = (\mathsf{Init}, \mathsf{OTKey}, \mathsf{Send}, \mathsf{Recv})$ *and for an adversary* $\mathcal{A}$*, define its advantage in*

$\mathsf{Exp}_{\mathsf{M}}^{\text{SUF-CMA}}(\mathcal{F})$

$K_{\mathsf{MAC}} \leftarrow\$ \mathsf{MKGen}()$
$Q \leftarrow \emptyset$
$(m^*, t^*) \leftarrow\$ \mathcal{F}^{\mathcal{O}_{\mathsf{MAC}}(K_{\mathsf{MAC}},\cdot)}$
if $\mathsf{Verify}(K_{\mathsf{MAC}}, m^*, t^*) == \mathsf{true}$
  and $(m^*, t^*) \notin Q$ **then**
  return 1
else
  return 0

$\mathcal{O}_{\mathsf{MAC}}(K_{\mathsf{MAC}}, m)$

$t \leftarrow\$ \mathsf{MAC}(K_{\mathsf{MAC}}, m)$
$Q \leftarrow Q \cup \{(m, t)\}$
return t

Figure 2.9: Experiment $\mathsf{Exp}_{\mathsf{M}}^{\text{SUF-CMA}}(\mathcal{F})$

Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\text{IND-SFCCA}}(\mathcal{A})$ *(Figure 2.8) as*

$$\mathsf{Adv}_{\mathsf{Ch}}^{\text{ind-sfcca}}(\mathcal{A}) = \left|\Pr\left[\mathsf{Exp}_{\mathsf{Ch}}^{\text{IND-SFCCA}}(\mathcal{A}) == \mathsf{true}\right] - \frac{1}{2}\right|.$$

As was the case for a channel that is secure under chosen-plaintext attack, we expect the advantage for an adversary to be negligible in the implied security parameter λ.

Definition 26 (Stateful Chosen-Ciphertext Security)
We call the channel Ch secure under chosen-ciphertext attack *(*IND-CCA*-secure) if, for any adversary $\mathcal{A}$, the following holds.*

$$\mathsf{Adv}_{\mathsf{Ch}}^{\text{ind-sfcca}}(\mathcal{A}) = \mathsf{negl}(\lambda)$$

That is, the advantage of $\mathcal{A}$ is negligible in the security parameter λ.

2.2.9 Message Authentication Codes

We define message authentication codes (MAC) and their security. A MAC $\mathsf{M} = (\mathsf{MKGen}, \mathsf{MAC}, \mathsf{Verify})$ with associated message space $\mathcal{M}$ consists of three algorithms.

$\mathsf{MKGen}() \xrightarrow{\$} K_{\mathsf{MAC}}$ outputs a key K_{MAC}.

$\mathsf{MAC}(K_{\mathsf{MAC}}, m) \xrightarrow{\$} t$ probabilisticly maps a key K_{MAC} and a message $m \in \mathcal{M}$ to a tag t.

$\mathsf{Verify}(K_{\mathsf{MAC}}, m, t) \rightarrow b$ takes a key K_{MAC}, a message m, and a tag t as input, and outputs a decision bit b indicating the validity of the tag t with respect to the message m.

Definition 27 (Correctness)
We call a message authentication code correct *if, for any $K_{\mathsf{MAC}} \leftarrow\$ \mathsf{MKGen}()$, any $m \in \mathcal{M}$ and any $t \leftarrow\$ \mathsf{MAC}(K_{\mathsf{MAC}}, m)$, we have*

$$\Pr[\mathsf{Verify}(K_{\mathsf{MAC}}, m, t) \rightarrow 1] = 1.$$

A MAC is typically used to verify that a tagged message has not been altered. It should hence on the one hand be infeasible to find a valid tag for a previously untagged message, and on the other hand difficult to find a different valid tag for a previously tagged message. We model this as strong unforgeability, which follows for example for unforgeable MACs where authentication is deterministic and verification is done by recomputing the tag and checking the result against the given tag [10].

Definition 28 (Strong Unforgeability)
For an adversary $\mathcal{F}$ define the advantage in Experiment $\mathsf{Exp}_{\mathsf{M}}^{\text{SUF-CMA}}(\mathcal{F})$ *(Figure 2.9) as:*

$$\mathsf{Adv}_{\mathsf{M}}^{\text{SUF-CMA}}(\mathcal{F}) = \Pr\left[\mathsf{Exp}_{\mathsf{M}}^{\text{SUF-CMA}}(\mathcal{F}) == 1\right]. \tag{2.2.1}$$

We say that $\mathcal{F}$ is q-query bounded if $|Q| \leq q$ in the experiment.

Note that here the adversary $\mathcal{F}$ may be bounded or unbounded in computation time. For unbounded $\mathcal{F}$ we usually assume that the adversary can only make a single query to oracle during the attack $\mathcal{O}_{\text{MAC}}$ and is thus 1-query bounded.

Two possible instantiations which are relevant for us in this work are the HMAC algorithm which provides strong unforgeability under reasonable assumptions about the compression function in the underlying hash function [9, 8], and Carter-Wegman MACs which are unconditionally secure for 1-bounded adversaries [113] and also follow the verification-through-recomputation paradigm.

$$\mathsf{Exp}^{\mathsf{dist}}_{\mathsf{Sim}\left(\{x_i,y_i\}_{P_i\in C}\right),\{\mathsf{view}_i\}_{P_i\in C}}(\mathcal{A})$$

$b \leftarrow\$ \{0,1\}$
$t_0 \leftarrow \{\mathsf{view}i\}_{P_i\in C}$
$t_1 \leftarrow \mathsf{Sim}\left(\{x_i,y_i\}_{P_i\in C}\right)$
$b' \leftarrow \mathcal{A}(t_b)$
$\mathbf{return}\ b == b'$

Figure 2.10: Experiment $\mathsf{Exp}^{\mathsf{dist}}_{\mathsf{Sim}\left(\{x_i,y_i\}_{P_i\in C}\right),\{\mathsf{view}_i\}_{P_i\in C}}(\mathcal{A})$

2.2.10 Multi-party Computation Protocols

A multi-party computation (MPC) protocol is an algorithm, which enables two or more parties $P_1,\ldots,P_n$ to jointly evaluate a prescribed function on their respective inputs. In this work, we focus on secret sharing based MPC protocols. That is, we consider protocols that evaluate arithmetic circuits C_f representing a function

$$f:\mathbb{F}^n \to \mathbb{F}^n; (x_1,\ldots,x_n) \mapsto (y_1,\ldots,y_n)$$

for a field $\mathbb{F}$, where x_i is a party P_i's input and y_i its output for $P_i \in \{P_1,\ldots,P_n\}$.

The execution of a secret sharing based MPC protocol is handled in two phases: the offline or preprocessing phase and the online phase. In the offline phase the parties generate and share the auxiliary data necessary for the evaluation of the circuit C_f. This includes but is not limited to the randomness used throughout the online phase and additional shared data for the evaluation of individual gates such as Beaver triples. The data generated in the offline phase is independent of the inputs that the parties provide in the online phase. Therefore significant time can elapse between the execution of the offline phase and the online phase. In the online phase, the circuit C_f is evaluated with the parties' inputs utilising the data generated during the offline phase. Each player receives his output according to the prespecified output gates.

A multi-party computation protocol has two main aims: correctness and privacy. Correctness implies that, for any input $(x_1,\ldots,x_n)$, the MPC protocol outputs $(y_1,\ldots,y_n) = f(x_1,\ldots,x_n)$ to the engaged parties in the online phase. Privacy means that the protocol must avoid divulging more information to a party P_j with respect to the other parties' inputs than P_j could otherwise derive from its input x_j and the output y_j. For that, we define the view of a party P_i as its input, the randomness used in the online phase and the messages it received, i.e.,

$$\mathsf{view}_i := \left\{x_i, (r_i)_j, (m_i)_j\right\}.$$

Since the view of a party P_i contains the randomness $(r_i)_j$ that P_i uses in the online phase and the messages that it receives from the other parties, view_i is a random variable.

If a view of an unauthorised set of parties, that is indistinguishable from a real view, can be produced from that parties' inputs and outputs, we have perfect privacy, since no information can be derived from the knowledge gained during the online phase.

Definition 29 (Simulatability)
We call an MPC protocol simulatable *if there exists an efficient algorithm* Sim *that, for any unauthorised set* $C \subset \{P_1,\ldots,P_n\}$*, upon input* $\{x_i,y_i\}_{P_i\in C}$ *produces an output that is perfectly indistinguishable from the real view of* C*, i.e., for any adversary* $\mathcal{A}$*, we have*

$$\mathsf{Adv}^{dist}_{\mathsf{Sim}\left(\{x_i,y_i\}_{P_i\in C}\right),\{\mathsf{view}i\}_{P_i\in C}}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{dist}_{\mathsf{Sim}\left(\{x_i,y_i\}_{P_i\in C}\right),\{\mathsf{view}i\}_{P_i\in C}}(\mathcal{A}) = 1\right] - \frac{1}{2}\right| = 0.$$

2.2.11 Hard Homogeneous Spaces

A hard homogeneous space (HHS) is a tuple $(\mathcal{E},\mathcal{G})$ that consists of a set $\mathcal{E}$ and a group $(\mathcal{G},\odot)$ equipped with a transitive action $*:\mathcal{G}\times\mathcal{E}\to\mathcal{E}$. It was first discussed by Couveignes [37] in 2006. The action has the following properties:

- Compatibility: For any $g,g' \in \mathcal{G}$ and any $E \in \mathcal{E}$, we have $g*(g'*E) = (g\odot g')*E$.
- Identity: For any $E\in\mathcal{E}$, $i*E = E$ holds if and only if $i\in\mathcal{G}$ is the identity element.

$\mathsf{Exp}^{\mathsf{GAIP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$

$E \leftarrow\$ \mathcal{E}$
$g \leftarrow\$ \mathcal{G}$
$E' \leftarrow g * E$
$g' \leftarrow \mathcal{A}(E, E')$
if $g == g'$
  return 1
else
  return 0
fi

(a) Experiment $\mathsf{Exp}^{\mathsf{GAIP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$

$\mathsf{Exp}^{\mathsf{PP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$

$E \leftarrow\$ \mathcal{E}$
$F \leftarrow\$ \mathcal{E}$
$g \leftarrow\$ \mathcal{G}$
$E' \leftarrow g * E$
$F' \leftarrow \mathcal{A}(E, E', F)$
if $F' == g * F$
  return 1
else
  return 0
fi

(b) Experiment $\mathsf{Exp}^{\mathsf{PP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$

Figure 2.11: Experiments $\mathsf{Exp}^{\mathsf{GAIP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$ and $\mathsf{Exp}^{\mathsf{PP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$

- Transitivity: For any $E, E' \in \mathcal{E}$, there exists exactly one $g \in \mathcal{G}$ such that $g * E = E'$.

Remark (Notation)
For a HHS $(\mathcal{E}, \mathcal{G})$ with a fixed $g \in \mathcal{G}$, let $p|\#\mathcal{G}$ be a fixed prime.We denote

$$[s]\, E := g^s * E$$

for all $s \in \mathbb{Z}_p$ and all $E \in \mathcal{E}$.

The following operations are assumed to be efficiently computable in a HHS $(\mathcal{E}, \mathcal{G})$, i.e., there exist polynomial-time algorithms to solve them:

- Group operations on $\mathcal{G}$ (membership, inverting elements, evaluating $\odot$).
- Sampling elements of $\mathcal{E}$ and $\mathcal{G}$.
- Testing the membership of $\mathcal{E}$.
- Computing the transitive action $*$. That is, given $g \in \mathcal{G}$ and $E \in \mathcal{E}$ as input, compute $g * E$.

Whereas the subsequent problems are assumed to be hard in a HHS $(\mathcal{E}, \mathcal{G})$.

Problem 30 (Group Action Inverse Problem (GAIP))
Given two elements $E, E' \in \mathcal{E}$ as input, the challenge is to provide $g \in \mathcal{G}$ with $E' = g * E$. We model this in the security game Experiment $\mathsf{Exp}^{GAIP}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$ in Figure 2.11a. In a hard homogeneous space, the probability of any adversary $\mathcal{A}$ winning Experiment $\mathsf{Exp}^{GAIP}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$ is assumed to be negligible in the security parameter, that is,

$$\mathsf{Adv}^{gaip}_{(\mathcal{E},\mathcal{G})}(\mathcal{A}) := \Pr\left[\mathsf{Exp}^{GAIP}_{(\mathcal{E},\mathcal{G})}(\mathcal{A}) = 1\right] = \mathsf{negl}(\lambda).$$

Problem 31 (Parallelisation Problem)
An instance of the Parallelisation Problem *is defined by a triple $(E, E', F) \in \mathcal{E}^3$ with $E' = g * E$. The challenge is to provide F' with $F' = g * F$. We model this in the security game Experiment $\mathsf{Exp}^{PP}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$ in Figure 2.11b. The advantage of an adversary $\mathcal{A}$ winning this game is assumed to be negligible in the implicit security parameter λ, i.e.,*

$$\mathsf{Adv}^{pp}_{(\mathcal{E},\mathcal{G})}(\mathcal{A}) := \Pr\left[\mathsf{Exp}^{PP}_{(\mathcal{E},\mathcal{G})}(\mathcal{A}) = 1\right] = \mathsf{negl}(\lambda).$$

The parallelisation problem has an intuitive decisional continuation.

Problem 32 (Decisional Parallelisation Problem)
An instance of the Decisional Parallelisation Problem *is defined by a base element $E \in \mathcal{E}$ and a triple $(E_{a^*}, E_{b^*}, E_{c^*})$ with $E_{a^*} = [a^*]E$, $E_{b^*} = [b^*]E$ and $E_{c^*} = [c^*]E$. The challenge is to distinguish whether $c^* = a^* + b^* \bmod p$ or $c^* \leftarrow\$ \mathbb{Z}_p$ was randomly sampled.*

The advantage of an adversary $\mathcal{A}$ in Experiment $\mathsf{Exp}^{DPP}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$ is assumed to be negligible in the implicit security parameter λ, i.e.,

$$\mathsf{Adv}^{\mathrm{dpp}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{DPP}_{(\mathcal{E},\mathcal{G})}(\mathcal{A}) = 1\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda).$$

$\mathsf{Exp}^{\mathsf{DPP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$

$b \leftarrow\$ \{0,1\}$
$E \leftarrow\$ \mathcal{E}$
$a^* \leftarrow\$ \mathbb{Z}_p; b^* \leftarrow\$ \mathbb{Z}_p$
$E_{a^*} \leftarrow [a^*]E; E_{b^*} \leftarrow [b^*]E$
if $b == 0$
  $c^* \leftarrow a^* + b^* \bmod p$
else
  $c^* \leftarrow\$ \mathbb{Z}_p$
fi
$E_{c^*} \leftarrow [c^*]E$
$b' \leftarrow \mathcal{A}(E, E_{a^*}, E_{b^*}, E_{c^*})$
return $b == b'$

Figure 2.12: Experiment $\mathsf{Exp}^{\mathsf{DPP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$

Remark
It is obvious that the decisional parallelisation problem reduces to the parallelisation problem, which in turn reduces to the group action inverse problem.

2.2.12 Key Exchange Mechanisms

A key exchange mechanism is a cryptographic public key primitive defined by a triple of algorithms $(\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$. Its purpose is to enable two parties to establish a shared symmetric key between them.

$\mathsf{KGen}() \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$ outputs a secret/ public key pair.

$\mathsf{Encaps}(\mathsf{pk}) \xrightarrow{\$} (\mathsf{k}, c)$ is a probabilistic algorithm that takes a public key as input and outputs a key k from a key space K and a ciphertext c.

$\mathsf{Decaps}(\mathsf{sk}, c) \rightarrow \mathsf{k}'$ takes a secret key sk and a ciphertext c as input. It outputs a key k' or an error symbol $\perp \notin K$.

Definition 33 (Correctness)
We call a key exchange mechanism $\mathcal{K}$ correct *if, for any $(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{KGen}()$ and any $(\mathsf{k}, c) \leftarrow \mathsf{Encaps}(\mathsf{pk})$, we have*

$$\Pr[\mathsf{Decaps}(\mathsf{sk}, c) = \mathsf{k}] = 1.$$

An attacker should not be able to distinguish, which key has been encapsulated, if he obtains a previously generated ciphertext. This is captured in Experiment $\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A})$.

Definition 34 (Indistinguishability under chosen-plaintext attack)
Let $\mathcal{K} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a key exchange mechanism with key space K. We call $\mathcal{K}$ IND-CPA-secure *if, for any adversary $\mathcal{A}$, the advantage in Experiment $\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A})$ is negligible in the security parameter λ, i.e., we have*

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A}) := \left| \Pr\left[\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A}) = 1\right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda).$$

2.2.13 Piecewise Verifiable Proofs

A piecewise verifiable proof (PVP) is a cryptographic primitive in the context of hard homogeneous spaces and was first introduced by Beullens et al. [14]. It is defined by a tuple $(\mathsf{Pv}, \mathsf{Vf})$ and forms a compact non-interactive zero-knowledge proof of knowledge of a witness $f \in \mathbb{Z}_q[X]$ for a statement

$$x = ((E_0, E_1), s_1, \ldots, s_n) \tag{2.2.2}$$

with statement pieces $s_i = f(i)$ for $i = 0, \ldots, n$ and $E_1 = [s_0]E_0 \in \mathcal{E}$. A piecewise verifiable proof thus proves knowledge of a sharing polynomial f for a secret s_0 that connects E_0 and E_1.

$$
\begin{array}{l}
\mathsf{Exp}_{\mathcal{K}}^{\text{IND-CPA}}(\mathcal{A}) \\
\hline
(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathsf{KGen}() \\
(\mathsf{k}_0^*, c^*) \leftarrow\$ \mathsf{Encaps}(\mathsf{pk}) \\
\mathsf{k}_1^* \leftarrow\$ K \\
b \leftarrow\$ \{0,1\} \\
b' \leftarrow \mathcal{A}(\mathsf{pk}, c^*, \mathsf{k}_b^*) \\
\textbf{return } b == b'
\end{array}
$$

Figure 2.13: Experiment $\mathsf{Exp}_{\mathcal{K}}^{\text{IND-CPA}}(\mathcal{A})$

$\mathsf{Pv}(x, f) \$\rightarrow \left(\pi, \{\pi_i\}_{i=1,\ldots,n}\right)$ is a probabilistic algorithm. It takes a statement x of the form (2.2.2) and a witness f for x and outputs a proof $\left(\pi, \{\pi_i\}_{i=0,\ldots,n}\right)$, where (π, π_i) is a proof piece for the statement piece s_i for $0 \leq i \leq n$.

$\mathsf{Vf}(i, s_i, (\pi, \pi_i)) \rightarrow b$ takes an index i, a statement piece s_i and a proof piece (π, π_i) as input and outputs a decision bit b indicating the validity of the proof piece with respect to the statement piece.

Let $\mathcal{R} = \{(x, f)\}$ denote the set of all pairs (x, f), where f is a witness for the statement x. The projection R_I for some $I \subset \{0, \ldots, n\}$ denotes (x_I, f).

We give the proving and verifying protocols presented by Beullens et al. [14] in Figure 2.14 and Figure 2.15, where H denotes a hash function mapping to bits string of length λ and CS is a commitment scheme. The verifying protocol follows the verification-through-recomputation paradigm.

Definition 35 (Completeness)
We call a PVP complete *if, for any $(x, f) \in \mathcal{R}$ and*

$$\left(\pi, \{\pi_i\}_{i=0,\ldots,n}\right) \leftarrow \mathsf{PVP.Pv}\,(x, f)\,,$$

the verification succeeds, that is,

$$\forall j \in \{0, \ldots, n\}\colon\ \Pr[\mathsf{PVP.Vf}\,(j, x_j, (\pi, \pi_j)) = \mathsf{true}] = 1.$$

Definition 36 (Soundness)
A PVP is called sound *if, for any adversary $\mathcal{A}$, any $I \subset \{0, \ldots, n\}$ and any x, for which there exists no f with $(x_I, f) \in \mathcal{R}_I$,*

$$\Pr[\mathsf{PVP.Vf}(j, x_j, (\pi, \pi_j)) = \mathsf{true}]$$

is negligible in the security parameter λ for all $j \in I$, where $\left(\pi, \{\pi_i\}_{i \in I}\right) \leftarrow \mathcal{A}(1^\lambda)$.

Definition 37 (Zero-Knowledge)
A piecewise verifiable proof PVP *is* zero-knowledge *if, for any $I \subset \{1, \ldots, n\}$ and any $(x, f) \in \mathcal{R}$, there exists a simulator* Sim *such that for any polynomial-time distinguisher $\mathcal{A}$ the advantage*

$$\left|\Pr\left[\mathcal{A}^{\mathsf{Sim}(x_I)}\left(1^\lambda\right) = 1\right] - \Pr\left[\mathcal{A}^{P(x,f)}\left(1^\lambda\right) = 1\right]\right|$$

is negligible in the security parameter λ, where P is an oracle that upon input (x, f) returns $\left(\pi, \{\pi_j\}_{j \in I}\right)$ with $\left(\pi, \{\pi_j\}_{j=0,\ldots,n}\right) \leftarrow \mathsf{PVP.Pv}(x, f)$.

We refer to [14] for the security analysis of the proving and verifying protocols given in Figure 2.14 and Figure 2.15, respectively. In combination they state a complete, sound and zero-knowledge non-interactive PVP. A prover can hence show knowledge of a sharing polynomial f to a secret $s_0 = f(0)$ with shares $s_i = f(i)$.

2.2.14 Threshold Group Action

Let $(\mathcal{E}, \mathcal{G})$ be a hard homogeneous space with a fixed $g \in \mathcal{G}$ of order p. Consider furthermore a Shamir sharing instance with secret space $\mathbb{Z}_p$ for a prime p with shareholders $P_1, \ldots, P_n$. Let $s \in \mathbb{Z}_p$ be a shared secret, that is, each P_i holds a share $s_i = f(i)$ of a sharing polynomial f with $f(0) = s$, $i = 1, \ldots, n$.

$\mathsf{PVP.Pv}(((E_0, E_1), s_1, \ldots, s_n), f)$

for $i = 1, \ldots, \lambda$

$\quad b_i \leftarrow\$ \mathbb{Z}_p[X]_{\leq k-1}$

$\quad \hat{E}_j \leftarrow [b_i(0)]\, E_0$

endfor

$y_0, y_0' \leftarrow\$ \{0,1\}^\lambda$

$C_0 \leftarrow \mathsf{CS.Commit}\left(\hat{E}_1 || \ldots || \hat{E}_\lambda, y_0\right)$

$C_0' \leftarrow \mathsf{CS.Commit}\left(E_0 || E_1, y_0'\right)$

for $i = 1, \ldots, n$

$\quad y_i, y_i' \leftarrow\$ \{0,1\}^\lambda$

$\quad C_i \leftarrow \mathsf{CS.Commit}(b_1(i) || \ldots || b_\lambda(i), y_i)$

$\quad C_i' \leftarrow \mathsf{CS.Commit}\left(s_i, y_i'\right)$

endfor

$C \leftarrow (C_0, \ldots, C_n)$

$C' \leftarrow \left(C_0', \ldots, C_n'\right)$

$\mathbf{c} \leftarrow (c_1, \ldots, c_\lambda) \ = \mathsf{H}\left(C, C'\right)$

for $i = 1, \ldots, \lambda$

$\quad r_j \leftarrow b_j - c_j f \in \mathbb{Z}_p[X]_{\leq k-1}$

endfor

$\mathbf{r} = (r_1, \ldots, r_\lambda)$

for $i = 0, \ldots, n$

$\quad \pi_i \leftarrow y_i$

endfor

$\pi \leftarrow \left(C, C', \mathbf{r}\right)$

return $\left(\pi, \{\pi_i\}_{i=0,\ldots,n}\right)$

Figure 2.14: The protocol PVP.Pv

$\mathsf{PVP.Vf}(i, s_i, (\pi, \pi_i))$

$(C, C', \mathbf{r}) \leftarrow \pi$
$(r_1, \ldots, r_\lambda) \leftarrow \mathbf{r}$
$(y_i, y_i') \leftarrow \pi_i$
if $C_i' \neq \mathsf{CS.Commit}(s_i, y_i')$
  return 0
fi
$(c_1, \ldots, c_\lambda) \leftarrow \mathsf{H}(C, C')$
if $i == 0$
  $(E_0, E_1) \leftarrow s_i$
else
  $x_i \leftarrow s_i \in \mathbb{Z}_N$
fi
if $i == 0$
  for $i = 1, \ldots, \lambda$
    $\tilde{E}_j \leftarrow [r_j(0)]\, E_{c_j}$
  endfor
  return $C_0 == \mathsf{CS.Commit}\left(\tilde{E}_1 || \ldots || \tilde{E}_\lambda, y_0\right)$
else
  return $C_i == \mathsf{CS.Commit}(r_1(i) + c_1 x_i || \ldots || r_\lambda(i) + c_\lambda x_i, y_i)$
fi

Figure 2.15: The protocol PVP.Pv

$\mathsf{TGA}(E, S')$

$E^0 \leftarrow E$
$k \leftarrow 0$
for $P_i \in S'$
  if $E^k \notin \mathcal{E}$
    P_i outputs $\perp$ and aborts.
  else
    $k \leftarrow k + 1$
    $E^k \leftarrow \left[L_{i,S'} \cdot s_i\right] E^k$ **fi**
endfor
return E^k

Figure 2.16: The protocol TGA

$\mathsf{ZK.Pv}\Big(s, (E_i, E_i')_{i=1,\ldots,m}\Big)$

for $j = 1, \ldots, \lambda$
  $b_j \leftarrow\$ \mathbb{Z}_p$
  for $i = 1, \ldots, m$
    $\hat{E}_{ij} \leftarrow [b_j]\, E_i$
  endfor
endfor
$(c_1, \ldots, c_\lambda) \leftarrow \mathsf{H}\Big(E_1, E_1', \ldots, E_m, E_m', \hat{E}_{1,1}, \ldots, \hat{E}_{m,\lambda}\Big)$
for $j = 1, \ldots, m$
  $r_j \leftarrow b_j - c_j s$
endfor
$\pi \leftarrow (c_1, \ldots, c_\lambda, r_1, \ldots, r_\lambda)$
return π

Figure 2.17: The protocol ZK.Pv

The threshold group action presented in Figure 2.16 enables an authorised set to compute $E' = [s]\, E = g^s * E$ for an arbitrary but fixed $E \in \mathcal{E}$ without reconstructing s. If it is executed successfully, we have by the compatibility property of $*$ in $(\mathcal{E}, \mathcal{G})$ and the repeated application of $E^k \leftarrow [L_{i,S'} s_i]\, E^{k-1}$ the result

$$E^{\#S'} = \left[\sum_{P_i \in S'} L_{i,S'} s_i\right] E = [s]\, E.$$

2.2.15 Zero-Knowledge Proofs for the GAIP

In Section 2.2.11, we gave the definition of the group action inverse problem in a hard homogeneous space $(\mathcal{E}, \mathcal{G})$. We now present a non-interactive zero-knowledge proof protocol for an element $s \in \mathbb{Z}_p$ with respect to the group action inverse problem. For that, let $g \in \mathcal{G}$ with $\#g = p$. A prover shows the knowledge of s so that

$$E_i' = [s]\, E_i$$

for $E_i, E_i' \in \mathcal{E}$ and $i = 1, \ldots, m$ simultaneously without revealing s. To that end, the prover samples $b_j \in \mathbb{Z}_p$ and computes

$$\hat{E}_{i,j} \leftarrow [b_j]\, E_i$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, \lambda$. He then derives challenge bits

$$(c_1, \ldots, c_\lambda) \leftarrow \mathsf{H}\Big(E_1, E_1', \ldots, E_m, E_m', \hat{E}_{1,1} \ldots, \hat{E}_{m,\lambda}\Big)$$

via a hash function $\mathsf{H} : \mathcal{E}^{(2+\lambda)m} \to \{0,1\}^\lambda$ and prepares the answers $r_j \leftarrow b_j - c_j s$, $j = 1, \ldots, \lambda$, to the challenge $(c_1, \ldots, c_\lambda)$. The proof $\pi = (c_1, \ldots, c_\lambda, r_1, \ldots, r_\lambda)$ is eventually published. The proving protocol is given in a succinct manner in Figure 2.17.

The verification protocol is straight forward: given a statement $(E_i, E_i')_{i=1,\ldots,m}$ and a proof $\pi = (c_1, \ldots, c_\lambda, r_1, \ldots, r_\lambda)$, the verifier computes $\tilde{E}_{i,j} \leftarrow [r_j]\, E_i$ if $c_j = 0$ and $\tilde{E}_{i,j} \leftarrow [r_j]\, E_i'$ otherwise, for $i = 1, \ldots, m$ and $j = 1, \ldots, \lambda$. He then generates verification bits

$$(\tilde{c}_1, \ldots \tilde{c}_\lambda) \leftarrow \mathsf{H}\Big(E_1, E_1', \ldots, E_m, E_m', \tilde{E}_{1,1} \ldots, \tilde{E}_{m,\lambda}\Big).$$

He accepts the proof iff $(c_1, \ldots, c_\lambda) = (\tilde{c}_1, \ldots, \tilde{c}_\lambda)$. The precise verifying protocol is given in Figure 2.18. We do not restate the proofs of completeness, soundness and zero-knowledge with respect to the security parameter λ here, but refer to [15].

$\mathsf{ZK.Vf}\Big(\pi, (E_i, E'_i)_{i=1,\ldots,m}\Big)$

$(c_1, \ldots, c_\lambda, r_1, \ldots, r_\lambda) \leftarrow \pi$

for $i = 1, \ldots, m$

  for $j = 1, \ldots, \lambda$

    if $c_j == 0$

      $\tilde{E}_{i,j} \leftarrow [r_j]\, E_i$

    else

      $\tilde{E}_{i,j} \leftarrow [r_j]\, E'_i$

    fi

  endfor

endfor

$(c'_1, \ldots, c'_\lambda) \leftarrow \mathsf{H}\Big(E_1, E'_1, \ldots, E_m, E'_m, \tilde{E}_{1,1}, \ldots, \tilde{E}_{m,\lambda}\Big)$

return $(c_1, \ldots, c_\lambda) == (c'_1, \ldots, c'_\lambda)$

Figure 2.18: The protocol ZK.Vf

3 MCELSA

This chapter is based on the work [89].

3.1 Motivation

In Section 1.2, we put forward research question 1 on how to achieve long-term archiving in an efficient and secure manner so that multiple clients can be served simultaneously.

It is self-evident that sensitive data, that is stored over extended periods of time, has a particular need for protection. The challenge of data protection is threefold: most importantly, confidentiality for stored data has to be guaranteed. A long-term storage solution is of little use if the stored data is not correctly stored, thus integrity has to be ensured. And lastly, in the case of multiple clients, authenticity is essential to verify the identity of a document's author.

Confidentiality of a stored date means that it cannot be accessed by an unauthorised party. Traditionally this is any party beside the data owner. Integrity with respect to stored data fundamentally means that a stored data item cannot be altered without it being detected. Authenticity of a stored document guarantees that the document was generated and stored by the party that it claims to be. The three aspects of data protection must hold with respect to every individual data item, that is stored, independently of each other. That is, if one data item becomes corrupted, it must not affect other data stored in the storage solution.

Confidentiality is most commonly achieved via encryption with schemes like RSA [98] or AES [1]. Most encryption schemes rely on certain cryptographic assumptions to hold true, many of which will not hold once quantum computers are available [103]. From the setting of long-term storage thus arises a new challenge: if an attacker on the confidentiality of a long-term storage solution obtains the ciphertexts of data items stored in the storage solution, he can store them and wait for weaknesses in the encryption scheme to emerge or for his computing power to increase sufficiently so that the encryption can be broken. Thus an approach that does not rely on cryptographic assumptions has to be chosen in a long-term storage setting to ensure confidentiality throughout the time of storage, i.e., a scheme that provides information-theoretic rather than computational confidentiality. The most prevalent method to achieve this are secret sharing schemes. Famous examples of this are Shamir's work of 1979 [101] and Tassa's elaborate extension [107] of Shamir's scheme. There exist several long-term storage solutions that provide confidentiality protection, an overview was given by Braun et al. [25].

Integrity on the other hand is commonly ensured using commitment schemes like Pedersen's approach [93] or digital signatures like [85]. For MCELSA, we choose a commitment scheme to guarantee the integrity of a stored document. The security of these schemes holds under certain cryptographic hardness assumptions, e.g., Pedersen's scheme relies on the hardness of computing a discrete logarithm in a finite group. Most integrity measures therefore must be considered insecure once quantum computers become available or in the face of cryptoanalytical breakthroughs. An almost folkloristic result by Brassard et al. [23] provides another challenge: a commitment scheme cannot be information-theoretically hiding as well as information-theoretically binding. In a long-term storage setting we must prevent any integrity measure to divulge any knowledge about the data item, the integrity of which it ensures. Thus an information-theoretically hiding but only computationally binding commitment scheme has to be employed. This results in the binding property of the commitment weakening over time. Any commitment to stored data items hence has to be renewed periodically. Approaches that provide long-term integrity protection also exist in many shapes, Vigil et al. [112] gave an overview of existing solutions.

Authenticity is typically provided by having a party sign a document upon storing it. Since virtually all existing signature schemes are based on cryptographic assumptions, the authenticity of a document can only be guaranteed at the time of storing. We shall, counterintuitively, apply this method in our long-term, multi-client storage architecture MCELSA. Not only will we store documents in a fashion that maintains long-term integrity, but also the accompanying signatures. We hence ensure that a signature cannot be altered throughout the lifetime of a stored document. It can thus be assumed that, at any point in time, an accompanying signature coincides with the state that it was in, when the document was stored. Thus it ascertains the authenticity of the document, it is attached to, even after it would otherwise have to be considered expired. This approach has previously been taken by Bayer et al. [5], who used timestamps to effectively prolong the lifespan of digital signatures.

Yet long-term storage solutions that combine all three aspects of security, i.e., confidentiality, integrity and authenticity, are scarce. Braun et al. [26] proposed LINCOS, the first long-term storage architecture that provides long-term confidentiality and long-term integrity protection. Their main caveat is that by design it can only protect a single data item. LINCOS can be extended so that multiple data items may be stored, but the resource demand then equals that of an individual instance of LINCOS per document. Furthermore, the storage demand in LINCOS increases linear in time.

Geihs et al. [62] proposed ELSA. It improves upon LINCOS in that it is capable of storing large sets of individual data items, whose integrity is protected independently of each other. It further more optimises the storage demand so that it is linear in time but constant in the number of stored data items. ELSA also can only accommodate a single client, thus it cannot be used as a shared data storage for multiple clients.

3.1.1 Related Work

We base our work on Geihs et al.'s ELSA [62]. They propose a long-term secure storage architecture for large datasets. Yet an instance of ELSA can only serve a single client. It therefore does not provide access management for the stored documents. ELSA in turn was based on Braun et al.'s LINCOS [26], which could only store one document at a time. Yet it was the first storage solution to simultaneously provide long-term confidentiality and integrity. Geihs et al. further improved upon LINCOS when proposing PROPYLA [63], a storage architecture that presents the same functionality as LINCOS to a client, yet it hides the document access pattern from the system iself. The access pattern hiding property introduces additional overhead of computation and communication.

LINCOS and all architectures based on it have a proactive secret sharing scheme at their core. Since we are constructing a long-term storage solution, it is vital to employ an information-theoretic secret sharing scheme, that is, the confidentiality of shared secret does not depend on any computational hardness assumption. The most prevalent examples of those are Shamir's and Blakley's works [101] and [18], respectively. Any secret sharing scheme employed in MCELSA has to be proactive, i.e., enable the shareholders to update their shares of the secrets without interacting with the dealer. Both, Shamir's and Blakleys' approach, can be extended to form proactive secret sharing schemes. Depending on the instantiation, it may be suitable to employ a secret sharing scheme with a more elaborate access structure. Several schemes have been proposed that are proactive and feature a hierarchical access structure [111, 54, 107].

We continue to apply the vector commitment scheme of [62], which is similar to that of Catalano et al. [33], yet implements the extractable binding property established by Buldas et al. [29].

3.1.2 Our Contribution

We present MCELSA, an evolution of ELSA. We improve upon previous works [62] in the following points. We enhance ELSA, so that a single instance becomes capable of serving multiple clients simultaneously. These clients can share their documents and data stored in the instance of MCELSA with each other and dynamically grant and withdraw several levels of access privileges regarding their documents. We also rework the protocols of ELSA to optimise the performance in terms of storage demand, so that we achieve significantly lower demand in comparison to the corresponding number of instances of ELSA that would be necessary to serve the same number of clients. Where many protocols of ELSA were error-prone to the detriment of many functionalities, we correct those protocols to ensure that a client in MCELSA has the full intended functionality at his or her disposal. We provide a refreshed benchmark implementation of MCELSA, where each party engaged in an instance of MCELSA runs on a separate machine. This way we ensure a more realistic testing scenario compared to ELSA's performance measurements. As expected, MCELSA improves on the performance of ELSA considerably.

3.2 MCELSA: Efficient Long-Term Secure Storage Architecture for Multiple Clients

We present MCELSA, our extension to ELSA, that serves multiple clients simultaneously. We illustrate MCELSA in light of the parties engaged in an instance of it and the protocols it provides.

Notation

Let us first clarify some notational conventions that we will use throughout this chapter. We denote a document that is to be stored in MCELSA by file. It comprises itself of a handle file.name and its content

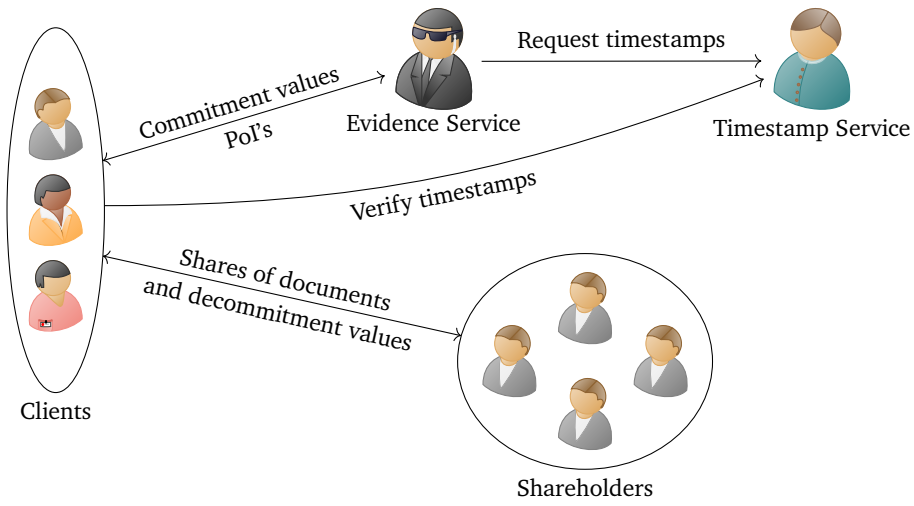

Figure 3.1: Interaction of parties in MCELSA

file.dat. MCELSA makes use of several cryptographic schemes. We will denote signature schemes by Sig, vector commitment schemes by VC and timestamp schemes by TS.

MCELSA is defined by a set of protocols that are to be executed by the parties engaged in an instance of it, that is clients, shareholders, evidence service and timestamp service. We denote each protocol by P.Prot, where P is the party executing it and Prot is the protocols name. P falls into either of these categories: $P =$ MCELSA, if the protocol is executed by a client or more than one kind of party is involved in its execution. $P =$ ES, if the evidence service runs the protocol. And $\mathsf{P} = \mathcal{S}$ if the shareholders execute it. For an execution of a protocol Prot, we denote the set of messages sent and received throughout the execution by view(Prot). If the protocol is evident from the context, we simply write view. Let S be a set of parties involved in the execution of Prot. We denote their view by $\mathsf{view}_S(\mathsf{Prot})$ or view_S. This is in line with the notation of the view of parties in a multi-party computation protocol as detailed in Section 2.2.10.

3.2.1 The Parties

There are three types of party that are engaged in an instance of MCELSA.

The Clients

An instance of MCELSA includes a set of one or more clients, that are served simultaneously. A client's role entails several tasks: storing documents, retrieving stored documents, updating the access permissions with respect to the documents said client has stored and partaking in the maintenance tasks that MCELSA necessitates. A client can also verify the integrity of a retrieved document utilising the integrity measures that are applied to each document stored in MCELSA.

The Evidence Service

The task of the evidence service ES is to maintain a proof of integrity for each document, that is stored within an instance of MCELSA. It contains the publicly available parts of the integrity measures, i.e., the vector commitment and timestamps, that are used in MCELSA. We denote the set of all partial proofs of integrity maintained by ES by `evidence`. The proof for a specific document with handle name is accessed by `evidence` [name].

All three integrity measures employed by MCELSA, that is signature, timestamp and vector commitment schemes, are public key protocols. The security of such a scheme deteriorates over time, it hence has to be renewed in regular intervals. It is the responsibility of the evidence service to update the timestamps that are put on the proofs of integrity and to alert the other parties engaged in an instance of MCELSA to the expiry of a vector commitment or a signature, so that they can renew those. For that, the evidence service maintains an initially empty list of proofs of integrity, that over time have to be renewed. We denote this list by `renewLists`.

Should a client wish to verify the integrity of a document, then ES is to transmit the proof of integrity to that client as part of the process.

The Shareholders

Storing the documents in a confidential manner is the main task fulfilled by the shareholders. They furthermore keep the individual decommitment values and the signature attributed to each document, that a client deposits in MCELSA.

MCELSA implements distributed access management. That is, access to a document is not determined by one central entity, but several. More concretely, we have the shareholders attach an access permission description to each share they hold. If a client requests access to a document, a shareholder only transmits his share to the client, if the access permission is successfully verified.

The third task, that we endow the shareholders with, is the coordination of the commitment renewal procedure. For a client to renew a commitment to a stored document, he has to retrieve it. Thus a combination of clients has to determined, that is permitted to retrieve each document, to which a commitment has to be renewed. For that, the shareholders maintain a list l_u for each client u, that contains all documents, that the client shall provide a renewed commitment for.

Communication

The communication between the parties in an instance of MCELSA falls into either of two categories: confidential and non-confidential. The messages that a client exchanges with the evidence service do not contain any knowledge of the stored document, this communication can therefore be realised with a public channel. The evidence service and the shareholders only communicate in one direction, that is when the evidence service informs the shareholders about commitments that are to be renewed. This communication does not contain any confidential information, either. It can therefore also be executed over a public channel. The communication between a client and the shareholders on the other hand must be confidential, since it contains the shares of the documents that are being stored in MCELSA. We discuss how such a secure private channel can be achieved in Section 4.

3.2.2 Amendments to the Secret Sharing Scheme

For MCELSA, we adapt the protocols provided by the secret sharing scheme, so that they accommodate multiple dealers or clients simultaneously and the respective access permissions attributed to the stored secrets and documents. We also introduce a new protocol UpdatePerm, that allows clients to amend the access permissions with respect to individual stored documents.

A secret sharing scheme as detailed in Section 2.2.4 is defined by the protocols Setup, Share and Reconstruct. If it is proactive, it also provides the protocol Reshare. For an instance $\mathcal{S}$, we have a set of shareholders S, an access structure Γ and a secret space G. In MCELSA, we task the shareholders with access control to the stored documents. That is, we attach a list of access permissions to each document, or each share respectively, that indicates whether a client is authorised to access a document according to the query he issued. Queries in MCELSA typically entail storing a document, retrieving a stored document or changing the access permissions with respect to one or more stored documents. We thus introduce the following new protocols for interacting with the secret sharing scheme in MCELSA:

$\mathsf{Setup}^*(S, \Gamma, G)$ establishes the secret sharing parameters as $\mathsf{Setup}(S, \Gamma, G)$ does. Yet it additionally fixes the layout, in which the permissions for each document are to be denoted and initialises an empty list l_u for each shareholder. Subsequent data access queries must adhere to these parameters.

$\mathsf{Store}(\text{'spec'}, h, \mathsf{dat}, p)$ takes an enumerator 'spec', that is either 'data' or 'decom', a handle h, a piece of data dat to be shared and permission list p adhering to the layout specified by Setup^* as input. If 'spec' = 'data', the handle h is a simple document identifier name. In the case of 'spec' = 'decom', the handle is a tuple (name, i) identifying the document and the index of the decommitment to be stored. The protocol shares dat via $\mathcal{S}.\mathsf{Share}(\mathsf{dat})$ and sends the resulting shares to the shareholders along with 'spec', h and p.

$\mathsf{Retrieve}(\text{'spec'}, h)$ is issued by a client and takes an enumerator 'spec' and a handle h as input. If 'spec' = 'data', the handle is a document identifier name. Should otherwise 'spec' = 'decom', then the handle is a tuple (name, i) identifying a document name and the index i of the decommitment value to be retrieved. Each shareholder is requested to send his share of the document name to the

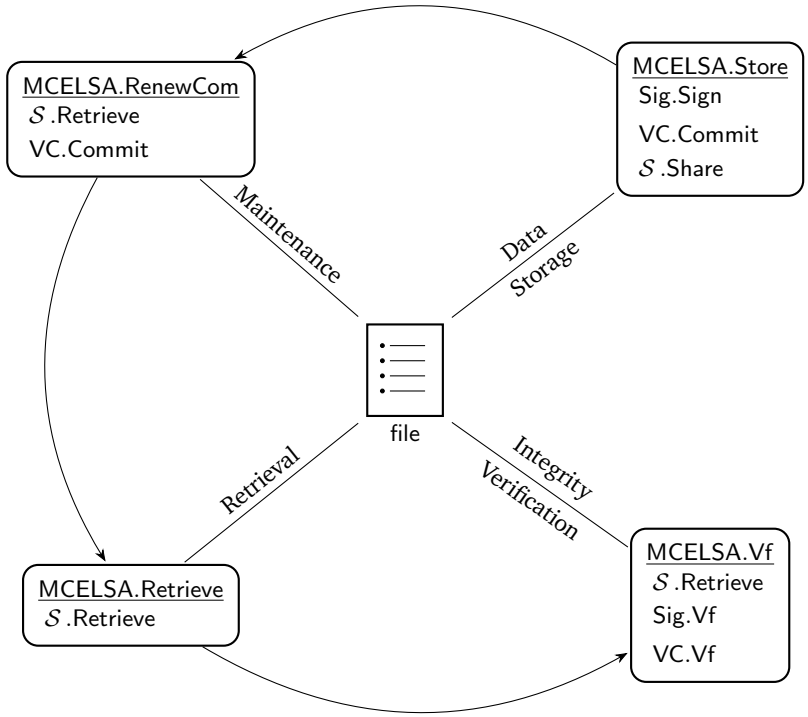

Figure 3.2: The life cycle of a document file in MCELSA

issuing client. If the client is indeed authorised to access name according to the access permissions attached to the share, the shareholder transmits his share. Otherwise the request is ignored. If the client obtains a sufficient set of shares, he can reconstruct name via $\mathcal{S}$.Reconstruct.

Reshare$^*()$ is similar to the protocol $\mathcal{S}$.Reshare in that it has the shareholders derive new, uncorrelated shares of the stored documents. Yet we point out, that the attached permissions for each share must remain unchanged. For MCELSA we assume the utilised secret sharing scheme to be proactive in order to maintain the long-term confidentiality of the stored documents.

UpdatePerm$\Big((\mathsf{name}_i, p_i)_{i\in[n]}\Big)$ is a novel protocol by which we extend the secret sharing scheme to enable the clients to change the permissions with respect to a document without having to retrieve and share it anew. When a client issues UpdatePerm with a list of document identifiers and permissions as input, the shareholders verify whether that client is sufficiently authorised for his query. If he is, they replace the existing permissions attached to their shares of name_i for p_i, $i = 1, \ldots, n$, and the respective data stored with 'spec' = 'decom' under the handle (name_i, j). The permissions are thereby updated without retrieving and storing the documents again.

We also let the shareholders maintain an initially empty list of document identifiers for each client. This list will be denoted by l_u, where u goes over all clients. The purpose of the list will be detailed in Section 3.2.3.

3.2.3 General Setup and Protocols

We previously elaborated on the general role each party plays in an instantiation of MCELSA. We will now discuss the concrete protocols that MCELSA is defined by.

Initialisation

To initialise an instance of MCELSA, the clients agree on a secret sharing scheme and its sharing parameters (S, Γ, G), i.e., the set of shareholders, the access structure and the secret space. Furthermore a neutral external party is determined for the role of the evidence service ES. The protocol

$$\frac{\mathsf{MCELSA.Setup}((S,\Gamma,G)\,,\mathsf{esURL})}{\begin{array}{l}\mathcal{S}\,.\mathsf{Setup}((S,\Gamma,G))\\ \mathsf{ES.Setup}(\mathsf{esURL})\end{array}}$$

Figure 3.3: The protocol MCELSA.Setup

$$\frac{\mathsf{ES.Setup}(\mathsf{esURL})}{\begin{array}{l}\texttt{renewLists} \leftarrow []\\ \texttt{evidence} \leftarrow []\end{array}}$$

Figure 3.4: The protocol ES.Setup

MCELSA.Setup$((S,\Gamma,G)\,,\mathsf{esURL})$ (Figure 3.3) initialises the secret sharing scheme and sets up the evidence service, that is ES.`renewLists` and ES.`evidence` are set as empty lists.

We assume, that all signature, vector commitment and timestamp schemes utilised in an instance of MCELSA, have been initialised upfront. That is, the Setup protocol for each cryptographic primitive has been executed before usage in MCELSA and the resulting public keys have been made available in a public key infrastructure.

Storing Documents

A client executes MCELSA.Store$\Big((\mathsf{file}_i)_{i\in[n]}\,,(p_i)_{i\in[n]}\,,\mathsf{Sig},\mathsf{VC},\mathsf{TS}\Big)$ (Figure 3.5) to store a list of documents $(\mathsf{file}_i)_{i\in[n]}$. To do so, he first initiates an empty list `filenames`. He then signs a document's content file_i.dat with his secret key via the signature scheme Sig, thereby obtaining a signature $\sigma_i \leftarrow$ Sig.Sign(file_i.dat). The data, handle of the signature scheme and the signature are then shared among the shareholders via $\mathcal{S}$ .Store('data', file_i.name, (file_i.dat, Sig, σ_i) , p_i). The handle is added to the list of stored documents `filenames` $+=$ file_i.name. This process is repeated for all documents file_i, $i = 1,\ldots,n$.

The client then commits to the data, signature scheme handle and signatures via $(c,D) \leftarrow$ VC.Commit((file_i.dat, Sig,
He then opens $d_i \leftarrow$ VC.Open(D,i). This initial decommitment value is again shared via $\mathcal{S}$ .Store('decom', (file_i.name, 0
Again, this process is repeated for all documents file_i, $i = 1,\ldots,n$.

As a final step, the file handles, description of the vector commitment, commitment value and handle of the timestamp scheme are appended to `evidence` and the maintenance list `renewLists` at the evidence service's site via ES.AddCom(`filenames`, VC, c, TS) (Figure 3.6).

Upon being queried with ES.AddCom(`filenames`, VC, c, TS), the evidence service generates a timestamp $(s,t) \leftarrow$ TS.Stamp(VC, c) on the description of the vector commitment scheme VC and the commitment value c. It defines an evidence entry $l = (\mathsf{VC}, c, \perp, \mathsf{TS}, (s,t))$ and appends to the proof of integrity of each

$$\frac{\mathsf{MCELSA.Store}\Big((\mathsf{file}_i)_{i\in[n]}\,,(p_i)_{i\in[n]}\,,\mathsf{Sig},\mathsf{VC},\mathsf{TS}\Big)}{\begin{array}{l}\texttt{filenames} \leftarrow \{\}\\ \textbf{for } i = 1,\ldots,n\\ \quad \sigma_i \leftarrow \mathsf{Sig.Sign}(\mathsf{file}_i.\mathsf{dat})\\ \quad \mathcal{S}\,.\mathsf{Store}(\text{'data'},\mathsf{file}_i.\mathsf{name},(\mathsf{file}_i.\mathsf{dat},\mathsf{Sig},\sigma_i)\,,p_i)\\ \quad \texttt{filenames} += \mathsf{file}_i.\mathsf{name}\\ \textbf{endfor}\\ (c,D) \leftarrow \mathsf{VC.Commit}\Big((\mathsf{file}_i.\mathsf{dat},\mathsf{Sig},\sigma_i)_{i\in[n]}\Big)\\ \textbf{for } i = 1,\ldots,n\\ \quad d_i \leftarrow \mathsf{VC.Open}(D,i)\\ \quad \mathcal{S}\,.\mathsf{Store}(\text{'decom'},(\mathsf{file}_i.\mathsf{name},0)\,,(d_i,i)\,,p_i)\\ \textbf{endfor}\\ \mathsf{ES.AddCom}(\texttt{filenames},\mathsf{VC},c,\mathsf{TS})\end{array}}$$

Figure 3.5: The protocol MCELSA.Store

$\mathsf{ES.AddCom}(\texttt{filenames}, \mathsf{VC}, c, \mathsf{TS})$

$(s,t) \leftarrow \mathsf{TS.Stamp}(\mathsf{VC}, c)$
$l \leftarrow (\mathsf{VC}, c, \perp, \mathsf{TS}, (s,t))$
for $\mathsf{name} \in \texttt{filenames}$
  $\texttt{evidence}\,[\mathsf{name}] += l$
endfor
$\texttt{renewLists} += l$

Figure 3.6: The protocol ES.AddCom

$\mathsf{ES.RenewTS}(\texttt{renewLists}, \mathsf{VC}, \mathsf{TS})$

$(c, D) \leftarrow \mathsf{VC.Commit}(\texttt{renewLists})$
$(s,t) \leftarrow \mathsf{TS.Stamp}((\mathsf{VC}, c))$
for $i \in [|\texttt{renewLists}|]$
  $d_i \leftarrow \mathsf{VC.Open}(D, i)$
  for $\mathsf{name} \in \texttt{evidence}$
    if $\texttt{evidence}\,[\mathsf{name}]_{|\texttt{evidence}[\mathsf{name}]|}.\mathsf{TS} = \texttt{renewLists}_i.\mathsf{TS}$
      $\texttt{evidence}\,[\mathsf{name}] += (\mathsf{VC}, c, (d_i, i), \mathsf{TS}, (s,t))$
    fi
  endfor
endfor
$\texttt{renewLists} \leftarrow []$
$\texttt{renewLists} += (\mathsf{VC}, c', \perp, \mathsf{TS}, (s,t))$

Figure 3.7: The protocol ES.RenewTS

document handle name contained in `filenames`. And finally l is added to the list of evidence entries to be maintained.

Timestamp Renewal

Any timestamp generated in MCELSA has to be renewed periodically. This process is carried out by the evidence service self-sufficiently, so it does not involve any interaction between the parties. To renew the timestamps, the evidence service decides on a vector commitment scheme VC and a timestamp scheme TS and executes $\mathsf{ES.RenewTS}(\texttt{renewLists}, \mathsf{VC}, \mathsf{TS})$. For that, ES commits to `renewLists` via $(c, D) \leftarrow \mathsf{VC.Commit}(\texttt{renewLists})$. The resulting commitment is then timestamped by $(s,t) \leftarrow \mathsf{TS.Stamp}((\mathsf{VC}, c))$. The decommitment value D is opened for each entry of `renewLists`, that is the evidence service computes $d_i \leftarrow \mathsf{VC.Open}(D, i)$ and appends $(\mathsf{VC}, c, (d_i, i), \mathsf{TS}, (s,t))$ to each proof of integrity that is affected by the expiring timestamp. To finalise the timestamp renewal procedure, `renewLists` is emptied and $(\mathsf{VC}, c, \perp, \mathsf{TS}, (s,t))$ is appended.

Commitment Renewal

The commitment scheme used in the construction of the vector commitment scheme in Section 2.2.7 must – in a long-term storage setting – be chosen to be statistically hiding, else information on the stored documents may be leaked over time. The vector commitment scheme is only computationally binding. The binding property thus weakens over time, so that the commitments placed on the stored documents need to be renewed periodically as the timestamps do, too. This task cannot be carried out by the evidence service, since it does not have access to the stored documents nor to the decommitment values, that are stored in the secret sharing instance. Since all documents have to be retrieved and recommitted to, this task can in a majority of scenarios not be executed by one client alone. If necessary, we shall thus employ a set of clients to renew the commitments on the stored documents. This we achieve in three steps:

1. Assume that a vector commitment (c, D) generated by a client is expiring. The evidence service ES, having knowledge of all commitment values in MCELSA, executes $\mathsf{MCELSA.detRecom}(c)$ (Fig-

```
ES.detRecom(c)
L ← []
for name ∈ evidence
  e ← evidence [name]
  i ← |e|
  repeat
    if e_i.c = c ∧ e_i.d = ⊥
      L+ = name
    fi
    i ← i − 1
  until i = 0 ∨ e_i.d = ⊥
endfor
return L
```

Figure 3.8: The protocol ES.detRecom

ure 3.8) to compile a list L of documents, that are affected by (c, D)'s expiry. ES then hands L to the shareholders.

2. The shareholders have knowledge of the access permissions with respect to the stored documents. They assign the documents in L to clients in such a manner, that the smallest set of clients is assigned the documents, that are to be committed to, while respecting the access permissions. This also minimises the number of commitments to maintain.

   To do so, the shareholders execute MCELSA.distRecom(L) (Figure 3.9). For that, they iteratively select the client u, that is permitted to retrieve the largest number of documents in L. They assign $\texttt{perm}_u \cap L$ to the client u by merging it with l_u and remove $\texttt{perm}_u \cap L$ from L, where $\texttt{perm}_u$ denotes the set of all documents that u may retrieve. This process is repeated until we have $L = []$, that is all documents have been assigned.

   Thus, for each client u, the list l_u contains the identifiers of the documents, that have been assigned to u for commitment renewal. We point out, that since MCELSA is a long-term storage solution, a client u may exist, for which l_u is not empty when MCELSA.distRecom is initiated. In Figure 3.9, we denote the set of all clients by U.

3. Whenever a client u issues a query to the shareholders, he is first handed the list of documents l_u to which he is to recommit to and executes MCELSA.ClientRenewCom$(l_u, \mathsf{VC}, \mathsf{TS})$ (Figure 3.10), where VC and TS denote a vector commitment scheme and a timestamp scheme, respectively. For that, he retrieves all documents in l_u (by design he is permitted access to each name $\in l_u$) and their respective complete proof of integrity. These consists of the partial proofs stored at ES and the decommitments values deposited in the secret sharing scheme. Afterwards, the client u generates a vector commitment (c, D) on the list of gathered documents and their respective proofs of integrity. The resulting decommitment values are then stored in $\mathcal{S}$ appropriately, whereas the commitment value c is handed to ES via ES.AddCom $(l_u, \mathsf{VC}, c, \mathsf{TS})$ to be appended to the proofs of integrity associated with the documents in l_u. After the client u has successfully renewed the commitments on the documents in l_u, the shareholders set $l_u \leftarrow []$.

For the protocol in Figure 3.8 and Figure 3.10 to be executed successfully, we make the following assumptions:

- The access permissions, that are attached to the files in L at the shareholders site, coincide. If this were not the case, constellations can be constructed quite easily in which two shareholders arrive at two distinct document distributions after the execution of MCELSA.distRecom.

- The clients connect to MCELSA within sufficiently short time frames to recommit to the documents in l_u before the expiry of the affected vector commitments.

- For each document stored in MCELSA, there exists a client with the permission to retrieve it.

MCELSA.distRecom(L)

Determine $u_0 \in U$:

$\#\left(\mathtt{perm}_{u_0} \cap L\right) = \max_{u \in U} \#\left(\mathtt{perm}_u \cap L\right)$

$l_{u_0} += \left(\mathtt{perm}_{u_0} \cap L\right)$

$L \leftarrow L \setminus \mathtt{perm}_{u_0}$

$n \leftarrow 1$

while $L \neq [] \wedge \{u_0, \ldots, u_{n-1}\} \neq U$

  Determine $u_n \in U \setminus \{u_i\}_{i=0}^{n-1}$:

    $\#\left(\mathtt{perm}_{u_n} \cap L\right) = \max_{u \in U \setminus \{u_i\}_{i=0}^{n-1}} \#\left(\mathtt{perm}_u \cap L\right)$

  $l_{u_n} += \mathtt{perm}_{u_n} \cap L$

  $L \leftarrow L \setminus \mathtt{perm}_{u_n}$

  $n \leftarrow n + 1$

endwhile

Figure 3.9: The protocol MCELSA.distRecom

MCELSA.ClientRenewCom$(l_u, \mathsf{VC}, \mathsf{TS})$

```
DocIndices ← {}
comCount ← {}
L ← []
for name ∈ l_u
  (dat, Sig, σ) ← S.Retrieve('data', name)
  e ← ES.evidence [name]
  for i ∈ [|e|]
    if e_i.d = ⊥
      e_i.d ← S.Retrieve('decom', (name, i))
    fi
  endfor
  L+ = (dat, Sig, σ, e)
  DocIndices [name] ← |L|
  comCount [name] ← |e|
endfor
(c, D) ← VC.Commit(L)
for name ∈ l_u
  d_name ← VC.Open(D, DocIndices [name])
  index ← DocIndices [name]
  S.Store('decom', (name, index), (d_name, index), L_index.p)
endfor
ES.AddCom(l_u, VC, c, TS)
```

Figure 3.10: The protocol MCELSA.ClientRenewCom

```
MCELSA.RenewShares()
S.Reshare*()
```

Figure 3.11: The protocol MCELSA.RenewShares

```
MCELSA.Retrieve(name)
e ← ES.evidence [name]
(dat, Sig, σ) ← S.Retrieve('data', name)
for i = 1, ..., |e|
  if e_i.d = ⊥
    e_i.d ← S.Retrieve('decom', (name, i))
  fi
endfor
E ← (Sig, σ, e)
return (dat, E)
```

Figure 3.12: The protocol MCELSA.Retrieve

Secret Share Renewal

In order to maintain the long-term confidentiality of the documents stored in MCELSA, the protocol MCELSA.RenewShares (Figure 3.11) is executed periodically by the shareholders. The shares of each stored document as well as the decommitment values attached to them are replaced with new, independent shares. Old shares are hence rendered useless. An attacker that manages to obtain a number of shares over time thus has to start over after the execution of MCELSA.RenewShares. This protocol does not necessitate the interaction with the clients nor the evidence service.

Data Retrieval

A client issues MCELSA.Retrieve(name) (Figure 3.12) to the shareholders and evidence service to retrieve a document name and the respective proof of integrity, that has been stored in MCELSA.

First, the proof of integrity is retrieved from the evidence service. If the client is authorised, the document along with the decommitment values stored in the secret sharing instance is also retrieved. The proof of integrity is then completed with the decommitment values that are stored in the secret sharing instance. The client thereby obtains the requested document and the completed proof of integrity if he is authorised to do so.

Verification of Retrieved Documents

The long-term multi-client storage solution MCELSA not only provides confidentiality for the stored documents but also integrity. By executing MCELSA.Vf(name, t_{store}) (Figure 3.13), a client that is authorised to retrieve a document name and its attached proof of integrity can verify that name in its current state coincides with that at the time of storage, where t_{store} denotes the point in time when name has been stored in MCELSA.

For that, the client first retrieves the document and its completed proof of integrity. The initial entry of the proof of integrity is then verified against the retrieved document, that is, the client executes Sig.Vf, VC.Vf and TS.Vf with the appropriate parameters to validate the first entry. He then goes over the proof of integrity recursively and ensures that the vector commitment and timestamp of each entry are correct. Should any of the checks not pass, then the document has been altered and MCELSA.Vf returns 0. Otherwise 1 is returned, since all checks passed successfully.

Lemma 38
The integrity verification protocol MCELSA.Vf *returns* 1*, i.e., indicates successful verification iff the subprotocol* MCELSA.Vf′ *returns* 1.

We do not prove Lemma 38 here, since the proof is obvious and does not provide further insight.

$\mathsf{MCELSA.Vf}(\mathsf{name}, t_{\text{store}})$

$e \leftarrow \mathsf{ES.}\texttt{evidence}\,[\mathsf{name}]$

$e' \leftarrow e$

for $i = 0, \ldots, \left|e'\right| - 1$

  if $e'_i.d = \perp$

    $e'_i.d \leftarrow (d, j) \leftarrow \mathcal{S}\,.\mathsf{Retrieve}(\text{'decom'}, (\mathsf{name}, i))$

  fi

endfor

$(\mathsf{dat}, \mathsf{Sig}, \sigma) \leftarrow \mathcal{S}\,.\mathsf{Retrieve}(\text{'data'}, \mathsf{name})$

return $\mathsf{MCELSA.Vf'}\big((\mathsf{dat}, \mathsf{Sig}, \sigma,)\,, t_{\text{store}}, \big(e, e'\big)\big)$

$\mathsf{MCELSA.Vf'}((\mathsf{dat}, \mathsf{Sig}, \sigma)\,, t_{\text{store}}, (e, e'))$

$(\mathsf{VC}, c, (d, j)\,, \mathsf{TS}, (s, t)) \leftarrow e'_0$

$t' \leftarrow t_{\text{store}}$

$b \leftarrow \mathsf{Sig.Vf}(\mathsf{dat}, \sigma) \wedge \mathsf{VC.Vf}((\mathsf{dat}, \mathsf{Sig}, \sigma)\,, c, d, j) \wedge \mathsf{TS.Vf}\big((\mathsf{VC}, c)\,, \big(s, t'\big)\big)$

for $i = 1, \ldots, |e| - 1$

  $(\mathsf{VC}, c, (d, j)\,, \mathsf{TS}, (s, t)) \leftarrow e'_i$

  if $e_i.d = \perp$

    $b = b \wedge \mathsf{VC.Vf}(\Big(\mathsf{dat}, \mathsf{Sig}, \sigma, e'_{[i-1]}\Big)\,, c, d, j) \wedge \mathsf{TS.Vf}((\mathsf{VC}, c)\,, (s, t))$

  else

    $b = b \wedge \mathsf{VC.Vf}(e_{[i-1]}, c, d, j) \wedge \mathsf{TS.Vf}((\mathsf{VC}, c)\,, (s, t))$

  fi

endfor

return b

Figure 3.13: The protocol MCELSA.Vf

$\mathsf{Exp}_{\mathcal{S}}^{\mathsf{Forge}}(\mathcal{A})$

$\mathsf{SetupExperiment}()$
$\big((\mathsf{dat}, \mathsf{Sig}, \sigma), t_{\mathsf{store}}, (e, e')\big) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Clock}}, \mathcal{O}_{\mathsf{Sig}}, \mathcal{O}_{\mathsf{TS}}, \mathcal{O}_{\mathsf{Break}}}$
$b \leftarrow \mathsf{MCELSA.Vf}'\big((\mathsf{dat}, \mathsf{Sig}, \sigma), t_{\mathsf{store}}, (e, e')\big)$
$\quad \wedge\ \mathsf{dat} \notin Q_{t_{\mathsf{store}}}$
return b

$\mathsf{SetupExperiment}()$
$\mathsf{time} \leftarrow 0$
for $i = 0, \ldots, |S|$
$\quad S_i.\mathsf{Setup}()$
endfor

$\mathcal{O}_{\mathsf{Clock}}(t)$
if $t > \mathsf{time}$
$\quad \mathsf{time} \leftarrow t$
fi

$\mathcal{O}_{\mathsf{TS}}(i, m)$
assert $S_i.\mathsf{type} = \mathsf{Timestamp}$
$(s, t) \leftarrow S_i.\mathsf{Stamp}(m)$
return (s, t)

$\mathcal{O}_{\mathsf{Sig}}(i, m)$
assert $S_i.\mathsf{type} = \mathsf{Signature}$
$Q\,[\mathsf{time}] += m;$
$\sigma \leftarrow S_i.sign(\mathsf{sk}_i, m)$
return σ

$\mathcal{O}_{\mathsf{Break}}(i)$
if $\mathsf{time} > t^i_{\mathsf{Break}}$
$\quad$ **return** sk_i
else
$\quad$ **return** $\perp$
fi

Figure 3.14: Experiment $\mathsf{Exp}_{\mathcal{S}}^{\mathsf{Forge}}(\mathcal{A})$

3.2.4 Security

The main aim for ELSA [62] was to provide prolonged computational integrity and long-term statistical confidentiality for the documents that are stored within an instance of it. We maintain this goal for MCELSA. To that end, we restate the attacker model that was applied in ELSA. The attacker model as well as the security proofs given in [62] transfer directly to the setting of MCELSA servicing multiple clients simultaneously. We restate them here, too, for the sake of completeness.

Attacker Model

In the long-term storage setting, we have to consider an attacker that increases his computational capabilities throughout the running time of an instance of MCELSA. As was the case in ELSA [62], we apply the attacker model of Buldas et al. [29] to capture this notion. To this end, let $\mathcal{M}_t$ denote the set of computing machines available at a time $t \in \mathbb{N}_0$. This set monotonically increases over time, that is, $\mathcal{M}_t \subset \mathcal{M}_{t'}$ for all $t \leq t'$. We collect all sets $\mathcal{M}_t$ in $\mathcal{M} := (\mathcal{M}_t)_{t \in \mathbb{N}_0}$. We model an adversary against MCELSA as a series of computing machines $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots)$, where $\mathcal{A}_t \in \mathcal{M}_t$ for all t. The attacker $\mathcal{A}$ – and thereby each $\mathcal{A}_t$ – is given access to a global clock Clock. When $\mathcal{A}^{\mathsf{Clock}}$ is started, then $\mathcal{A}_0^{\mathsf{Clock}}$ is started. Any $\mathcal{A}_t^{\mathsf{Clock}}$ may advance Clock to a time $t' > t$, but cannot revert the clock to an earlier point in time. When Clock is advanced to t', the current attacker $\mathcal{A}_t^{\mathsf{Clock}}$ stops and $\mathcal{A}_{t'}^{\mathsf{Clock}}$ is started with the internal state of $\mathcal{A}_t^{\mathsf{Clock}}$ as input.

We say the attacker $\mathcal{A}^{\mathsf{Clock}}$ is ρ-bounded if $\mathcal{A}_t^{\mathsf{Clock}}$ makes at most $\rho(t)$ computing steps for a fixed function $\rho : \mathbb{N} \to \mathbb{N}$.

Integrity

For an attacker to successfully break the integrity measures, that MCELSA puts in place for the documents stored in it, he has to provide a document dat, a signature σ and a proof of integrity (e, e') for said document as well as a time of storage t_{store}, so that these successfully verify. That is, we have

$$\mathsf{MCELSA.Vf}'\big((\mathsf{dat}, \mathsf{Sig}, \sigma), t_{\mathsf{store}}, (e, e')\big) = 1$$

yet dat has not been stored in MCELSA at the time t_{store}. We capture this notion in Experiment $\mathsf{Exp}_{\mathcal{S}}^{\mathsf{Forge}}(\mathcal{A})$ (Figure 3.14). MCELSA uses several cryptographic schemes to achieve its functionality. We represent this in Experiment $\mathsf{Exp}_{\mathcal{S}}^{\mathsf{Forge}}(\mathcal{A})$ by denoting a list S of schemes that are used in MCELSA.

Definition 39 (Integrity)
We say MCELSA *is* $(\mathcal{M}, \varepsilon)$*-unforgeable for schemes* S *if, for any* p*-bounded machine* $\mathcal{A} \in \mathcal{M}$*,*

$$\Pr\Big[\mathsf{Exp}_{S}^{\mathit{Forge}}(\mathcal{A}) = 1\Big] \leq \varepsilon(p).$$

The adversary $\mathcal{A}$ can access all signature and timestamp schemes via the respective oracles $\mathcal{O}_{\mathsf{Sig}}$ and $\mathcal{O}_{\mathsf{TS}}$. We attribute each scheme S_i with a break time t_{Break}, after which it is considered insecure. If a scheme is indeed assumed insecure, that adversary can obtain the respective secret keys via the oracle $\mathcal{O}_{\mathsf{Break}}$.

$\mathsf{Exp}^{\mathsf{Dist}}_{S,L}(\mathcal{A}, F_0, F_1, I)$

$b \leftarrow\$ \{0,1\}$
$\mathsf{SetupExperiment}()$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Clock}}, \mathcal{O}_{\mathsf{MCELSA}}}$
$\mathbf{return}\ b == b'$

$\mathsf{SetupExperiment}()$

$\mathsf{time} \leftarrow 0$
$N \leftarrow 0$
$\mathsf{MCELSA.Setup}((S, \Gamma, G), \mathsf{esURL})\}$

$\mathcal{O}_{\mathsf{Clock}}(t)$

$\mathbf{if}\ t > \mathsf{time}$
  $\mathsf{time} \leftarrow t$
$\mathbf{fi}$

$\mathcal{O}_{\mathsf{MCELSA}}(\mathsf{op}, \mathsf{param})$

$\mathbf{if}\ N < L$
  $N \leftarrow N + 1$
  $\mathbf{if}\ \mathsf{op} == \mathsf{Store}$
    $\left((p_i)_{i \in [|F_b|]}, \mathsf{Sig}, \mathsf{VC}, \mathsf{TS}, \mathcal{T}\right)$
    $\mathbf{if}\ \mathsf{param} \notin \Gamma_{\mathcal{S}}$
      $\mathsf{view} \leftarrow \mathsf{view}\,(\mathsf{MCELSA.Store}$
        $\left(F_b, (p_i)_{i \in [|F|_b]}, \mathsf{Sig}, \mathsf{VC}, \mathsf{TS}\right))$
    $\mathbf{fi}$
  $\mathbf{elseif}\ \mathsf{op} == \mathsf{Retrieve}$
    $(\mathsf{name}, \mathcal{T}) \leftarrow \mathsf{param}$
    $\mathbf{if}\ \mathsf{name} \in I \wedge \mathcal{T} \notin \Gamma_{\mathcal{S}}$
      $\mathsf{view} \leftarrow \mathsf{view}(\mathsf{MCELSA.Retrieve}(\mathsf{name}))$
    $\mathbf{fi}$
  $\mathbf{elseif}\ \mathsf{op} == \mathsf{RenewTS}$
    $(\mathsf{VC}, \mathsf{TS}, \mathcal{T}) \leftarrow \mathsf{param}$
    $\mathsf{view} \leftarrow \mathsf{view}(\mathsf{ES.RenewTS}(\mathsf{VC}, \mathsf{TS}))$
  $\mathbf{elseif}\ \mathsf{op} == \mathsf{RenewCom}$
    $(\mathsf{VC}, \mathsf{TS}, l, \mathcal{T}) \leftarrow \mathsf{param}$
    $\mathsf{view} \leftarrow \mathsf{view}(\mathsf{MCELSA.ClientRenewCom}(l, \mathsf{VC}, \mathsf{TS}))$
  $\mathbf{elseif}\ \mathsf{op} == \mathsf{RenewShares}$
    $\mathcal{T} \leftarrow \mathsf{param}$
    $\mathsf{view} \leftarrow \mathsf{view}(\mathsf{MCELSA.RenewShares}())$
  $\mathbf{fi}$
  $\mathbf{return}\ \mathsf{view}_{\{\mathsf{ES}, \mathcal{T}\}}$
$\mathbf{fi}$

Figure 3.15: Experiment $\mathsf{Exp}^{\mathsf{Dist}}_{S,L}(\mathcal{A}, F_0, F_1, I)$

To advance the clock, an individual adversary $\mathcal{A}_t$ queries the oracle $\mathcal{O}_{\mathsf{Clock}}$. The adversary $\mathcal{A}$ wins the security game if he can produce a triple $((\mathsf{dat}, \mathsf{Sig}, \sigma), t_{\mathsf{store}}, (e, e'))$, so that $\mathsf{MCELSA.Vf}'$ returns 1 upon being handed the triple as input. We restate the security result given in [62], adapted to the notational conventions we use in MCELSA.

Theorem 40
Let $\mathcal{M} = (\mathcal{M}_t)_t$ specify the computational technology available at time $t \in \mathbb{N}_0$ and let S be a set of cryptographic schemes, where each scheme $S_i \in S$ is associated with a breakage time t^i_{Break} and is ε_i-secure against adversaries using computational technology $\mathcal{M}_{t^i_{\mathsf{Break}}}$. In particular, we require unforgeability-security for signature schemes and extractable-binding-security for commitment schemes. Let p_E be any computational bound and L be an upper bound on the maximum vector length of the commitment schemes in S. Then, MCELSA *is $(\mathcal{M}, \varepsilon)$-unforgeable for S with*

$$\varepsilon(p) = \left(\sum_{i \in \mathsf{Sig}} \varepsilon_i \left(p\left(t^i_{\mathsf{Break}}\right) p_E\left(t^i_{\mathsf{Break}}\right) L^2\right)\right) + \left(\sum_{i \in \mathsf{CS}} \varepsilon_i \left(p\left(t^i_{\mathsf{Break}}\right), p_E\left(t^i_{\mathsf{Break}}\right), p\left(t^i_{\mathsf{Break}}\right)\right)\right).$$

Confidentiality

The documents in MCELSA are stored in an ε-statistically hiding secret sharing scheme. Yet for each document, additional data with relation to its contents like commitments and signatures is generated and also stored in MCELSA. An attacker with access to one document should not be able to derive information with respect to another document to which he does not have access. We model this in Experiment $\mathsf{Exp}^{\mathsf{Dist}}_{S,L}(\mathcal{A}, F_0, F_1, I)$ (Figure 3.15).

The task of an adversary $\mathcal{A}$ in Experiment $\mathsf{Exp}^{\mathsf{Dist}}_{S,L}(\mathcal{A}, F_0, F_1, I)$ is to distinguish between two instances of MCELSA that store either of the data sets F_0 or F_1, which coincide on an index set I. The adversary

$\mathcal{A}$ has access to the oracles $\mathcal{O}_{\mathsf{Clock}}$ and $\mathcal{O}_{\mathsf{MCELSA}}$. The former enables $\mathcal{A}$ to advance the global clock to a later point in time. The latter provides the adversary $\mathcal{A}$ with the means to interact with the instance of MCELSA. That is, he can have MCELSA store a predefined set of documents F_0 or F_1 (depending on the choice bit), retrieve documents, that are contained in I, and have MCELSA perform the maintenance tasks RenewTS, RenewCom and RenewShares. He will obtain the view of the evidence service ES and a fixed unauthorised set of shareholders $\mathcal{T}$ after the execution of each of MCELSA's protocols.

We thus define the advantage of an adversary $\mathcal{A}$ in Experiment $\mathsf{Exp}^{\mathsf{Dist}}_{S,L}(\mathcal{A}, F_0, F_1, I)$ as follows.

Definition 41 (Confidentiality)
We say MCELSA *is ε-statistically-hiding for a set of schemes S if, for all machines $\mathcal{A}$, index sets I, sets of files F_0, F_1 with $(F_0)_I = (F_1)_I$ and for all $L \in \mathbb{N}$, we have*

$$\left|\Pr\left[\mathsf{Exp}^{\mathsf{Dist}}_{S,L}(\mathcal{A}, F_0, F_1, I) = 1\right] - \frac{1}{2}\right| \leq \varepsilon(L).$$

Theorem 42
Let S be a set of schemes, where $S.\mathcal{S}$ is an ε-statistically-hiding secret sharing scheme and every commitment scheme in S is ε-statistically-hiding. Then, MCELSA *is ε'-statistically-hiding for S with $\varepsilon'(L) = 2L\varepsilon$.*

For the proofs of Theorem 40 and Theorem 42 we refer to [62]. Their proofs in the context of ELSA transfer directly to MCELSA, which can be regarded as an improved and updated multi-client extension of ELSA.

3.3 Performance Evaluation

To evaluate the performance of our secure long-term storage solution MCELSA, we test its performance in terms of resource demand for the evidence service and the secret sharing instance. We will also measure the time elapsed for integrity verification of individual stored documents. The results that our long-term storage solution achieves are then compared to the results of ELSA, the predecessor of MCELSA.

3.3.1 Testing Parameters

We simulate a continued runtime of 80 years for MCELSA, i.e., approximately one human lifetime, since storage of medical data is an intuitive application of a solution such as MCELSA. During that period we let each client store one document each month for the duration of the experiment.

We assume the following renewal schedule for protecting the evidence against the weakening of cryptographic primitives. For the timestamps, we use a signature scheme based approach as discussed in Section 2.2.6. The typical lifetime of a public key certificate is two years. Hence we choose this as the timespan after which the timestamps are renewed. While signature scheme instances can only be secure as long as the corresponding private signing key is not leaked to an adversary, commitment scheme instances do not involve the usage of any secret parameters. Therefore, their security is not threatened by key leakage and we assume that they only need to be renewed every 10 years in order to adjust the cryptographic parameter sizes or to choose a new and more secure scheme.

As the vector commitment scheme we use the construction given in Figure 2.5 with the statistically hiding commitment scheme by Halevi and Micali [66], the security of which is based on the security of the used hash function which we instantiate with members of the SHA-2 hash function family [91]. If we model hash functions as random oracles, the extractable binding property required by Theorem 40 is provided. This vector commitment scheme construction also provides the statistical hiding property as required by Theorem 42. We adjust the signature and commitment scheme parameters over time as proposed by Lenstra and Verheul [79, 78].

Concerning the document storage, i.e., the secret sharing scheme, we instantiate it as a standard Shamir secret sharing [101] with four shareholders and a reconstruction threshold of three, that is, any set of three or more shareholders is considered authorised.

The layout of the access permissions strongly influences the resulting distribution of the maintenance tasks among the clients, we hence distinguish the three following cases.

Single client: We form a baseline for our performance evaluation by testing MCELSA with a single client. We compare these results with those of ELSA, the predecessor of our solution and up until now the best performing storage architecture that provides long-term integrity and confidentiality.

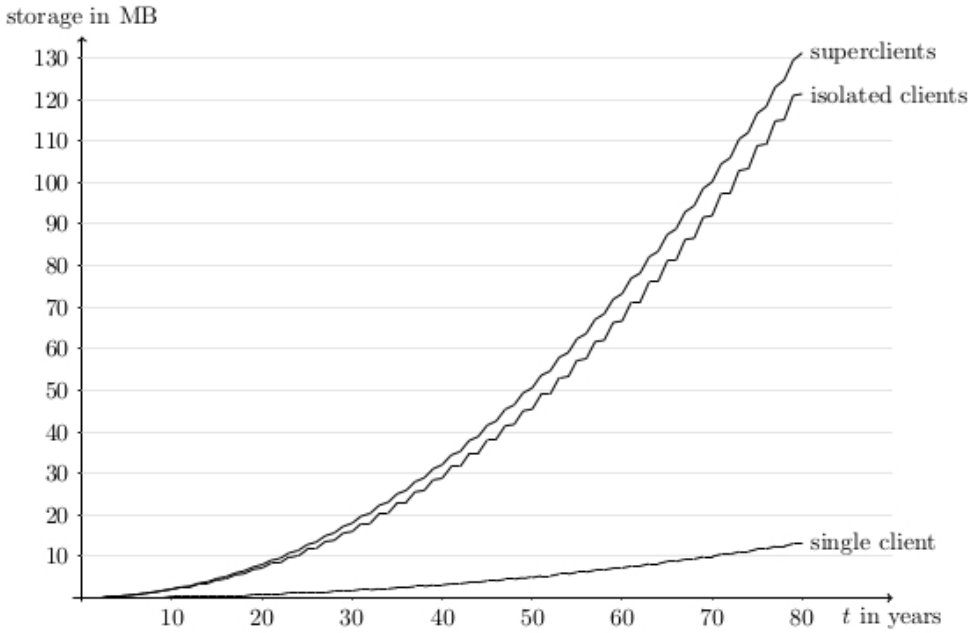

Figure 3.16: Storage need at the evidence service

Scenario	Storage $\mathcal{S}$	Storage ES	Verificaton time per file
Single Client	53.26 MiB	13.115 MiB	1.41s
Isolated Clients	670.44 MiB	121.305 MiB	1.18s
Super Clients	439.30 MiB	131.198 MiB	0.94s

Figure 3.17: Evaluation of the three tested scenarios

Isolated clients: Ten clients are being simulated for this test, where each client is permitted to access only his own documents. The provided functionality is equal to running ten parallel instances of ELSA, yet in using MCELSA, only one instance $\mathcal{S}$ of the secret sharing scheme and one instance ES of the evidence service is needed.

Privileged clients: In this case, we simulate a small doctor's office. Again, we consider ten clients, eight of which represent patients and two of which doctors. Each patient may of course only access his own health record, whereas the doctors can access those of all their patients.

After the 80th year has passed, the client(s) verify the integrity of the stored documents and we measure the time that is needed for the integrity verification.

Our implementation was done using the programming language Java. In order to have consistent environment parameters, the experiments were performed in a virtual Linux machine with a two-core processor running at 2.8 GHz, 16 GB of storage space and 8 GB of RAM. Each party was run in an independent context of each other.

3.3.2 Results

Our performance test yields the following results. Figure 3.16 shows the storage demand at the evidence service's site over the course of eighty years. The storage consumption at the shareholders' site is plotted in Figure 3.18. In Figure 3.17, we summarize our evaluation results. We point out that the length of a document's proof of integrity increases linearly with the storage duration. It is hence to be expected that the time needed for integrity verification does so, too.

We observe that increasing the number of client scales reasonably well in MCELSA. More than that, MCELSA performs about 5% better than ELSA; this can be seen by comparing with ELSA's performance results in [62]. We attribute this increased performance to the optimisations, that we applied to the maintenance protocols, and to some improvements in the simulation implementation.

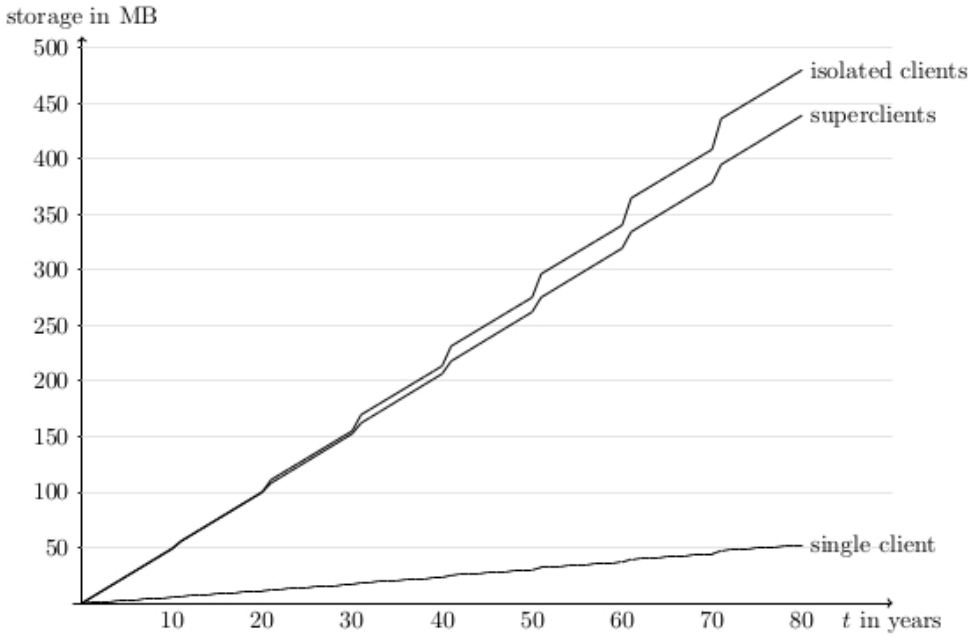

Figure 3.18: Storage need at the secret sharing service

In conclusion, we see that the storage demand in the scenario of several isolated clients is approximately the same as that of a single client multiplied with the number of engaged isolated clients. Yet it must be pointed out that the overhead of maintaining a secret sharing instance has to be mustered but once in MCELSA compared to several parallel instances in ELSA.

We furthermore observe that the used resources have been considerably decreased by employing two clients, which were able to retrieve all documents in the super client scenario.

4 Information-Theoretic Security of Cryptographic Channels

This chapter is based on the work [58].

4.1 Motivation

In the previous chapter, we discussed MCELSA, a long-term storage solution that provides prolonged computational integrity and unconditional confidentiality to the documents stored within an instance of it. It is set up to serve multiple clients simultaneously and enables distributed and dynamic management for the access permissions with respect to the stored data. For the communication between the clients and the shareholders engaged in a instance of MCELSA, the existence of secure private channels is assumed. We raised the question, how such a channel can be instantiated facing an adversary the cryptoanalytical capabilities and compuational power of which increase over time, in research question 2. In this chapter, we will elaborate on how to establish such channels in the context of a shared source for symmetric key material available to a pair of parties communicating with each other.

The need for secure private channels does of course not only apply to a long-term storage architecture such as MCELSA, but is prevalent in all modern communication, digital or analogue. Striking examples are digital tax data or electronic medical records which need to be kept for years or even decades according to legal stipulations, requiring also to uphold the involved individuals' right to privacy for extended periods of time. In some cases the protection time span is quasi indefinite if one considers for example genetic data which descendants (partially) inherit from their ancestors.

The cryptographic challenge here is that the long-term protection schemes must be able to withstand unexpected cryptanalytic advances, but also predictable advances in computational power. An adversary may store digital data and aim to break the underlying cryptographic scheme later with new methods or by pure advances in technology. Remarkably, this does not only hold for data at rest but also for data in transmission: An adversary may record encrypted communication today and try to break confidentiality tomorrow. If we talk about transmissions over unreliable networks then the adversary may also use additional means to attack schemes, such as omission, injection or modification of transmitted ciphertexts.

The above challenge is the starting point of our work. We consider the security of cryptographic channels against potentially unbounded adversaries, denoted as information-theoretically secure channels.[1] A subsequent question arises: What security guarantee for such a channel can be provided in a setting of unbounded adversaries, and how can we accomplish this?

4.1.1 Modeling Information-Theoretically Secure Channels

For a channel protocol to achieve guaranteed security against adversaries while the channel is open and against future cryptoanalytic advances, only unconditionally security is admissible. Thus by Shannon's famous result [102], the length of the key material used in sending messages over said channel has to agree with the total length of the messages sent. Several models have been developed, that enable two parties to securely obtain a sufficient amount of keying material, e.g., the bounded storage model [65, 49] and quantum key distribution (QKD) [61] to name the most prolific examples. These methods are rather costly compared to traditional methods of key exchange, that are in use in our everyday life. Yet we argue that in transmitting highly sensitive data, the security of which must be ensured under all circumstances, the high pricetag can be considered acceptable. For our work we hence assume that the sender and the receiver have access to a shared source of keying material, the amount required will be quantified by our channel notion.

As we consider an adversary that is active while the channel is being used and continues to compute on the knowledge gained after the channel has been closed, we use a two-stage adversary. The first stage is active while the channel is, too. The second takes over from the first once the channel is closed, obtaining all knowledge that the first stage was able to gather. This notion has been introduced by Bindel et al. [17] and continued in subsequent works like [16].

[1] Our notion of (cryptographic) channels should not be confused with other concepts like Wyner's wire-tap channels [114] or other measures to generate information-theoretically secure keys from physical assumptions. We are interested in how to transmit data securely once the sender and the receiver already share a key.

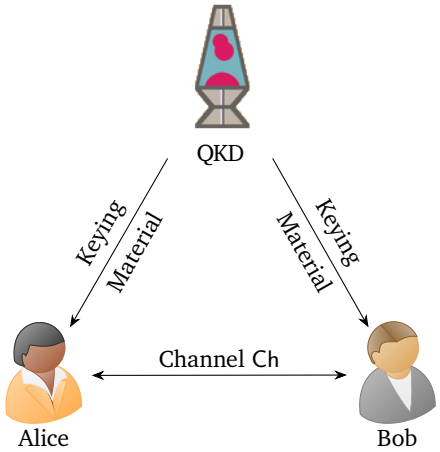

Figure 4.1: Two parties sharing a source of symmetric keying material

We distinguish between two types of adversaries, depending on whether the adversary is bounded or unbounded in its first stage, i.e., during the execution of the channel protocol. The second stage is always considered unbounded, independent of the the first stage.

- For *future-secure* channels, the first-stage adversary is bounded in computational resources when the channel protocol is running, but may store the data gathered from the communcation over the channel and later try to decrypt when having more computational power or more time.
- For *unconditionally-secure* channels, the first-stage adversary already has extreme computational power when the channel protocol is executed, such that we need to protect against unbounded adversaries immediately.

The adversary's capabilities are assumed as follows: it can see and tamper with all network communication. Our channel hence has to be secure against replay and re-ordering attacks – among others. This in particular distinguishes our setting from prior works concerned with the unconditional security of *individual* messages (but without ordering requirements), e.g., aiming at everlasting privacy in e-voting [86].

4.1.2 Achieving Information-Theoretically Secure Channels

We next show how one can build future-secure and unconditionally-secure channel protocols. We follow the common paradigm to encrypt and authenticate the data in transmission. For encryption we need unconditional security for both channel types, because any break of confidentiality, during the protocol execution or afterwards, violates long-term secrecy of the data. This suggests to use the one-time pad encryption.

Authenticity, on the other hand, is a property which has to hold only during the channel's life time, in order to decide if a transmission comes from the expected sender. This is also remarked in [87] where the authors combine quantum key distribution with short-term authentication methods. In our channel instantiation aiming at future security we can thus use computationally-secure authentication methods like HMAC [9]. For unconditionally-secure channels we need information-theoretically secure authentication schemes like Carter-Wegman MACs [113].

Before diving into the construction we first carefully adapt the classic composition theorem of Bellare and Namprempre [12] to the setting of information-theoretically secure channels: we show that an IND-CPA secure protocol which additionally provides INT-CTXT integrity of ciphertexts is also IND-CCA secure. As we will see, in our setting IND-CPA (even against unbounded adversaries) holds based on using one-time pad encryption; the composition result hence elegantly allows us to focus on establishing INT-CTXT (computationally or unconditionally) via appropriate authentication methods. This way, we obtain IND-CCA future-secure channels if we use computational authentication, and even IND-CCA unconditionally-secure channels if we use information-theoretically secure authentication.

We then give two concrete channel protocols, combining one-time pad encryption with computationally-secure MACs like HMAC, resp. with information-theoretically secure schemes like Carter-Wegman MACs. For the future-secure channel we use a counter to prevent repetition and out-of-order attacks, and show

that the channel is IND-CPA secure and (computationally) INT-CTXT secure. Our general composition theorem therefore shows that the channel is IND-CCA future-secure. For the unconditional case it turns out that we do not need counters since we use a one-time key in each authentication step. We show, applying once more the composition theorem, that we achieve unconditional security of the channel if we apply Carter-Wegman MACs to the (plain) one-time pad encryption. Due to unforgeability of Carter-Wegman MACs linearly degrading with the number of transmitted messages, our results exhibit a noteworthy trade-off between the future- and unconditionally-secure constructions.

4.1.3 Further Related Work

Alternative approaches to unconditionally-secure encryption include limiting the adversary's memory capacity in the bounded-storage model [84, 30]. As such restrictions may regularly not apply in practice for small-bandwidth, but highly-critical communication data, we in contrast consider fully-unbounded adversaries (and hence have to resort to the one-time pad for confidentiality).

Künzler et al. [77] consider which functions are securely computable in the long-term scenario when one assumes short-term authenticated channels, i.e., channels which are only computationally secure during the computation. In a similar vein, Müller-Quade and Unruh [88] define a statistical version of the universal composition framework, enabling long-term security considerations. The work shows how to build commitments and zero-knowledge protocols in this setting, again assuming that secure channels are available.

4.2 Security of Information-Theoretically Secure Channels

The traditional security notions as presented in Section 2.2.8 do not capture the setting of a two-stage adversary that is bounded or unbounded while the channel is active and becomes unbounded after the channel was finalised. We therefore rephrase the notions of stateful indistinguishability under chosen-ciphertext attack and the respective security game Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\mathsf{IND\text{-}SFCCA}}(\mathcal{A})$.

To capture unconditionally-secure and future-secure channels, respectively, in a single game we represent the two-stage adversary $\mathcal{A}$ in two phases, $\mathcal{A}_1$ and $\mathcal{A}_2$. In the first phase the adversary has access to both the sender and receiver oracle. In this first stage the adversary may still be bounded in running time (for future-secure channels), respectively already be unbounded (for unconditionally-secure channels). In the second stage the adversary is in both cases unbounded but can no longer access the receiver oracle. This allows us to model future-secure channels where $\mathcal{A}_1$ is restricted and the authentication only needs to be temporarily secure, and in the second phase of the unbounded $\mathcal{A}_2$ past and future sender messages remain confidential (but computational authentication may now be broken). For unconditionally-secure channels we allow already $\mathcal{A}_1$ to be unbounded such that $\mathcal{A}_2$ acts merely as a dummy. We give the resulting security game Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\mathsf{IND\text{-}SFCCA}}(\mathcal{A})$ in Figure 4.2. The definition of an adversary's advantage in this setting carries over from those given in Section 2.2.8 with respect to Experiment $\mathsf{Exp}_{\mathsf{Ch}}^{\mathsf{IND\text{-}SFCCA}}(\mathcal{A})$.

We stress, however, that we do not formalise the notion of being bounded or unbounded in our concrete security analysis. Instead, we give reductions to underlying problems, e.g., if $\mathcal{A}_1$ breaks integrity of the channel then we break some underlying primitive with (roughly) the same running time. By this we get a reasonable security guarantee from computationally secure authentication schemes such as HMAC, as well as from unconditionally secure ones such as Carter-Wegman MACs.

4.3 Composition Theorem

We now show that for any channel protocol Ch chosen-ciphertext security follows from chosen-plaintext security and integrity, similar to the composition result for classical channels [11]. The security reduction shows that the derived attackers $\mathcal{B}$ against $\mathrm{IND\text{-}CPA}$ and $\mathcal{I}$ against $\mathsf{INT\text{-}SFCTXT}$ have roughly the same running time characteristics as the adversary against $\mathsf{IND\text{-}SFCCA}$. In particular, if the first-stage adversary $\mathcal{A}_1$ against $\mathsf{IND\text{-}SFCCA}$ is bounded (or unbounded), then so is the adversary $\mathcal{B}$ against $\mathrm{IND\text{-}CPA}$ and also $\mathcal{I}$.

Theorem 43 ($\mathrm{IND\text{-}CPA} \wedge \mathsf{INT\text{-}SFCTXT} \Rightarrow \mathsf{IND\text{-}SFCCA}$)
For any channel protocol Ch *and any* $\mathsf{IND\text{-}SFCCA}$ *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, we can construct an* $\mathsf{INT\text{-}SFCTXT}$ *adversary* $\mathcal{I}$ *and an* $\mathrm{IND\text{-}CPA}$ *adversary adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}_{\mathsf{Ch}}^{\mathsf{ind\text{-}sfcca}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathsf{Ch}}^{\mathsf{int\text{-}sfctxt}}(\mathcal{I}) + \mathsf{Adv}_{\mathsf{Ch}}^{\mathrm{IND\text{-}CPA}}(\mathcal{B}) . \tag{4.3.1}$$

Here, $\mathcal{B}$ *and* $\mathcal{I}$ *use approximately the same resources as* $\mathcal{A}_1$.

$\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$

$b \leftarrow\$ \{0,1\}$
$(K_I, \mathsf{st}_S, \mathsf{st}_R) \leftarrow\$ \mathsf{Init}()$
$K_1, K_2, K_3, \ldots \leftarrow\$ \mathsf{OTKey}()$
$\textsc{out-of-sync} \leftarrow \mathsf{false}$
$i, j \leftarrow 0$
$\mathsf{st}_{\mathcal{A}} \leftarrow\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, \cdot, \cdot), \mathcal{O}_{\mathsf{Recv}}(\mathsf{st}_S, K_I, \cdot)}()$
$b' \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, \cdot, \cdot)}(\mathsf{st}_{\mathcal{A}})$
$\mathbf{return}\ b == b'$

$\mathcal{O}_{\mathsf{Recv}}(\mathsf{st}_R, K_I, C)$

$j \leftarrow j + 1$
$(m, \mathsf{st}_R) \leftarrow \mathsf{Recv}(\mathsf{st}_R, K_I, K_j, C)$
$\mathbf{if}\ (j > i\ \mathbf{or}\ C \neq C_j)\ \mathbf{then}$
  $\textsc{out-of-sync} \leftarrow \mathsf{true}$
$\mathbf{endif}$
$\mathbf{if}\ (\textsc{out-of-sync}\ \mathbf{and}\ b == 0)\ \mathbf{then}$
  $\mathbf{return}\ m$
$\mathbf{endif}$
$\mathbf{return}\ \perp$

$\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, m_0, m_1)$

$\mathbf{assert}\ |m_0| = |m_1|$
$i \leftarrow i + 1$
$(C_i, \mathsf{st}_S) \leftarrow \mathsf{Send}(\mathsf{st}_S, K_I, K_i, m_b)$
$\mathbf{return}\ C_i$

Figure 4.2: Experiment $\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$

$\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$

$b \leftarrow\$ \{0,1\}$
$(K_I, \mathsf{st}_S, \mathsf{st}_R) \leftarrow\$ \mathsf{Init}()$
$K_1, K_2, K_3, \ldots \leftarrow\$ \mathsf{OTKey}()$
$\textsc{out-of-sync} \leftarrow \mathsf{false}$
$i, j \leftarrow 0$
$\mathsf{st}_{\mathcal{A}} \leftarrow\$ \mathcal{A}_1^{\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, \cdot, \cdot), \mathcal{O}_{\mathsf{Recv}}(\mathsf{st}_S, K_I, \cdot)}()$
$b' \leftarrow\$ \mathcal{A}_2^{\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, \cdot, \cdot)}(\mathsf{st}_{\mathcal{A}})$
$\mathbf{return}\ b == b'$

$\mathcal{O}_{\mathsf{Recv}}(\mathsf{st}_R, K_I, C)$

$j \leftarrow j + 1$
$(m, \mathsf{st}_R) \leftarrow \mathsf{Recv}(\mathsf{st}_R, K_I, K_j, C)$
$\mathbf{if}\ (j > i\ \mathbf{or}\ C \neq C_j)\ \mathbf{then}$
  $\textsc{out-of-sync} \leftarrow \mathsf{true}$
$\mathbf{endif}$
$\mathbf{if}\ (\textsc{out-of-sync}\ \mathbf{and}\ b == 0)\ \mathbf{then}$
  $\mathbf{return}\ \perp$ // instead of m
$\mathbf{endif}$
$\mathbf{return}\ \perp$

$\mathcal{O}_{\mathsf{Send}}(\mathsf{st}_S, K_I, m_0, m_1)$

$\mathbf{assert}\ |m_0| = |m_1|$
$i \leftarrow i + 1$
$(C_i, \mathsf{st}_S) \leftarrow \mathsf{Send}(\mathsf{st}_S, K_I, K_i, m_b)$
$\mathbf{return}\ C_i$

Figure 4.3: Modified receiver oracle experiment $\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$ for $\textsc{Game}_1$ in the proof of Theorem 43.

Proof. The proof follows the common game-hopping technique, where $\textsc{Game}_0$ denotes $\mathcal{A}$'s attack in Experiment $\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$. In $\textsc{Game}_1$ we modify the receiver oracle $\mathcal{O}_{\mathsf{Recv}}$ by letting it return $\perp$ instead of m for an out-of-sync query (for which in addition $b == 0$). This is depicted in Figure 4.3. The other steps of the experiment remain unchanged.

We argue that the difference of $\mathcal{A}$'s advantage between the two games lies in a potential first-stage query of $\mathcal{A}_1$ to the receiver oracle which returns a message $m \neq \perp$ in $\textsc{Game}_0$ but not in $\textsc{Game}_1$. We show that the probability of this happening is bounded by the integrity guarantees of the channel. To this end we build a reduction $\mathcal{I}$ mounting an attack according to Experiment $\mathsf{Exp}^{\text{INT-SFCTXT}}_{\mathsf{Ch}}(\mathcal{I})$. This algorithm $\mathcal{I}$ runs a black-box simulation of $\mathcal{A}_1$ (in $\textsc{Game}_0$). Any oracle call $\mathcal{O}_{\mathsf{Recv}}$ of $\mathcal{A}_1$ is forwarded directly to the corresponding oracles of $\mathcal{I}$. Algorithm $\mathcal{I}$ initially also picks a random bit $b \leftarrow\$ \{0,1\}$ and whenever $\mathcal{A}_1$ makes an oracle call m_0, m_1 to $\mathcal{O}_{\mathsf{Send}}$, then $\mathcal{I}$ first checks that $|m_0| = |m_1|$ and returns $\perp$ if not; else it forwards m_b to its own oracle $\mathcal{O}_{\mathsf{Send}}$ to receive a ciphertext C_i. Algorithm $\mathcal{I}$ returns C_i in the simulation of $\mathcal{A}_1$. Algorithm $\mathcal{I}$ stops if $\mathcal{A}_1$ stops.

Note that the only difference between the two games from $\mathcal{A}$'s perspective is that $\textsc{Game}_0$, in case $b = 0$, returns an actual message m in a call to $\mathcal{O}_{\mathsf{Recv}}$ if (a) $m \neq \perp$, and (b) $\textsc{out-of-sync}$ has been set to true (in this call or a previous call). This, however, means that all prerequisites in the $\mathcal{O}_{\mathsf{Recv}}$ oracle of the integrity experiment are satisfied, causing $\textsc{int-broken}$ to become true and to make $\mathcal{I}$ win the game. Hence, any

difference between the games can be bounded by the advantage against integrity.

A careful inspection of the modified $\mathcal{O}_{\text{Recv}}$ oracle now shows that this oracle always returns $\perp$ and only changes the state of the OUT-OF-SYNC variable. The latter only affects the $\mathcal{O}_{\text{Recv}}$ oracle itself. It follows that we can simulate this oracle by returning $\perp$ immediately for any query to $\mathcal{O}_{\text{Recv}}$. Formally, this is a black-box simulation $\mathcal{B}$ of $\mathcal{A}$, where $\mathcal{B}$ relays all communication of $\mathcal{A}_1$ and $\mathcal{A}_2$ with oracle $\mathcal{O}_{\text{Send}}$ without modification, but returns $\perp$ to $\mathcal{A}_1$ for any call of $\mathcal{A}_1$ to $\mathcal{O}_{\text{Recv}}$. Hence, in the next game hop we can eliminate the $\mathcal{O}_{\text{Recv}}$ oracle altogether, obtaining the CPA-game GAME_2. For this game we can bound the advantage by the CPA-security of the channel. □

4.4 Instantiations

In this section we discuss that instantiations combining the one-time pad encryption scheme with a computationally-secure MAC like HMAC, and with an unconditionally-secure one like Carter-Wegman MACs, provide future security, respectively unconditional security for the channel protocol. This of course requires additional steps to prevent replay attacks or protection against omission of ciphertexts. For the computational case we choose here for the sake of concreteness a sequence number on the sender's and receiver's side. For the unconditional MAC we can omit the sequence number because we use a fresh key portion with each message anyway.

In both cases we use our composition result from Theorem 43 to argue security. IND-CPA security of the encryption scheme follows by the perfect secrecy of the one-time pad encryption and the fact that we use a fresh key for each ciphertext. This holds even against unbounded adversaries. It hence suffices to argue INT-SFCTXT security to conclude IND-SFCCA security of the channel protocol. For this we need the strong unforgeability of the authentication algorithm as layed out in Section 2.2.9.

4.4.1 Future-Secure Channels

For a future-secure channel we define the sender and receiver algorithms as follows. We initialise counters for the sender and the receiver, respectively, both as zero. Algorithm Send first generates a ciphertext c via one-time pad encryption $\mathsf{OTP.Enc}(K, m) = m \oplus K$ under the fresh per-message key K. It then authenticates the ciphertext c, prepended with a fixed-length encoding of the counter value in st_S, under a computationally-secure MAC, using the steady key K_I.[2] The sender then increments its counter to be stored in the updated state st_S, and outputs the full ciphertext consisting of the OTP ciphertext and MAC tag.

The receiver algorithm Recv, when receiving a ciphertext $C = (c, t)$, first checks if the state st_R indicates a previous failed decryption or if the MAC is invalid. If so, Recv returns the error symbol $\perp$ and keeps this information in its state. Otherwise Recv decrypts the ciphertext part c with the per-message key, $\mathsf{OTP.Dec}(K, c) = c \oplus K$, increments the counter, and stores the updated value in its state st_R.

Construction 44 (Future-Secure Channel)
Define the channel protocol $\mathsf{FSCh} = (\mathsf{Init}, \mathsf{OTKey}, \mathsf{Send}, \mathsf{Recv})$ *for message space* $\mathcal{M} = \{0,1\}^{\leq M}$ *and key space* $\mathcal{K} = \{0,1\}^M$ *by the algorithms in Figure 4.4.*

We next argue INT-SFCTXT security of the channel protocol, assuming that the underlying MAC scheme M is strongly unforgeable:

Lemma 45
For any INT-SFCTXT *adversary* $\mathcal{I}$ *there exists an adversary* $\mathcal{F}$ *such that*

$$\mathsf{Adv}^{\text{int-sfctxt}}_{\mathsf{FSCh}}(\mathcal{I}) \leq \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{M}}(\mathcal{F}). \tag{4.4.1}$$

Furthermore, $\mathcal{F}$ *uses approximately the same resources as* $\mathcal{I}$.

Proof. We show that if $\mathcal{I}$ at some point during the integrity experiment sets INT-BROKEN to true, then we can break (strong) unforgeability of the MAC scheme. To this end we let a forger $\mathcal{F}$ run a black-box simulation of $\mathcal{I}$, simulating the other steps of the channel protocol FSCh like encryption locally, and only using the oracle access to $\mathcal{O}_{\text{MAC}}(K_I, \cdot)$ to compute MACs when required. For the simulated receiver oracle $\mathcal{F}$ always answers $\perp$. Algorithm $\mathcal{F}$ also keeps track of sent and received ciphertexts in the simulation, including the values i and j. When $\mathcal{I}$ sends the first ciphertext $C^* = (c^*, t^*)$ to the receiver oracle such

[2] Technically, the encoded counter restricts the number of messages that can be sent. If there are n bits reserved for the counter value then one can transmit at most 2^n messages. In practice this is not an issue and deployed channel protocols today commonly have such restrictions as well (e.g., TLS 1.3 [96] uses an $n = 64$ bit sequence number).

```
Init()
K_I ←$ MKGen()
st_S ← 0
st_R ← (⊤, 0)
return (K_I, st_S, st_R)

OTKey()
K ←$ 𝒦
return K

Send (st_S, K_I, K, m)
c ← OTP.Enc (K, m)
t ← MAC (K_I, st_S||c)
C ← (c, t)
st_S ← st_S + 1
return (C, st_S)

Recv (st_R, K_I, K, C)
parse st_R = (b, j) and C = (c, t)
d ← Verify(K_I, c, t)
if b == ⊥ or d == false then
    m ← ⊥
    st_R ← (⊥, 0)
else
    m ← OTP.Dec (K, c)
    st_R ← (⊤, j + 1)
fi
return (m, st_R)
```

Figure 4.4: Future-secure channel protocol FSCh

that C^* has not been the next ciphertext prepared by the sender (i.e., C^* is entirely new or a modification of the j-th sent ciphertext $C_j = (c_j, t_j)$), then $\mathcal{F}$ outputs $(j||c^*, t^*)$ as its forgery attempt.

Note that the simulation is perfect, as the receiver oracle always returns $\perp$. Furthermore, $\mathcal{F}$ outputs a forgery as soon as INT-BROKEN is set to true. This can only happen if OUT-OF-SYNC has become true (according to the model) but the MAC verification has returned true (according to the protocol). The former implies that the ciphertext C^* must have been new or reordered ($j > i$ or $C^* \neq C_j$). And since the channel starts returning error symbols $\perp$ whenever it has encountered an invalid MAC, it must be the first such out-of-sync ciphertext C^* which, too, carries a valid MAC, to get some output $m \neq \perp$ from the receiver oracle.

Assume that $j > i$ for the first valid out-of-sync ciphertext $C^* = (c^*, t^*)$. In this case, since the receiver in the protocol holds the same counter value j in st_R up to this point, the receiver verifies t^* with regard to $j||c^*$. Since $j > i$, the sender oracle (and thus the MAC oracle in the simulation) has not issued any MAC for this counter value yet, such that the "message" $j||c^*$ for valid tag t^* in $\mathcal{F}$'s output constitutes a fresh forgery. Analogously, if $j \leq i$ and $C^* = (c^*, t^*)$ is different from $C_j = (c_j, t_j)$, then the pair $(j||c_j, t_j)$ is a successful strong forgery for $\mathcal{F}$ because the sender oracle (and thus MAC oracle) has only issued one tag for value j, with a different result $(j||c_j, t_j) \neq (j||c^*, t^*)$.

It follows that whenever $\mathcal{I}$ breaks integrity of the channel protocol we have a forgery for the underlying MAC scheme. For efficient $\mathcal{I}$ the resulting forger $\mathcal{F}$ is also efficient. □

We can now apply the composition theorem (Theorem 43), noting that the one-time-pad encryption ensures perfect IND-CPA security (such that, independently of the adversarial resources, the advantage is always 0), and using that integrity is bounded by the security of the strong unforgeability of the MAC scheme:

Theorem 46 (Future-Secure Channel)
For the channel protocol FSCh *in Construction 44 and any* IND-SFCCA *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, we can construct and* SUF-CMA *adversary* $\mathcal{F}$ *such that*

$$\mathsf{Adv}^{\text{ind-sfcca}}_{\mathsf{FSCh}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{M}}(\mathcal{F}). \tag{4.4.2}$$

Here, $\mathcal{F}$ *uses approximately the same resources as* $\mathcal{A}_1$.

For an unbounded $\mathcal{A}_1$ – and hence an unbounded $\mathcal{I}$ in the proof – however, equation (4.4.2) may become void, since $\mathcal{I}$ may win Experiment $\mathsf{Exp}^{\text{SUF-CMA}}_{\mathsf{M}}(\mathcal{F})$ with significant probability.

4.4.2 Unconditionally-Secure Channels

For an unconditionally-secure channel we assume that both adversarial stages $\mathcal{A}_1$ and $\mathcal{A}_2$ in Experiment $\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$ are unbounded, that is, we consider an unbounded adversary throughout the entire Experiment $\mathsf{Exp}^{\text{IND-SFCCA}}_{\mathsf{Ch}}(\mathcal{A})$. Our construction therefore asks for a fresh authentication key (part) with each send operation: we first split the per-message key K into two parts, K_1 and K_2. The former, K_1, is used for encryption via OTP. The latter, K_2, is used for authentication via an unconditionally-secure Carter-Wegman-MAC. For messages of length M bits we typically need M bits for the one-time pad and $2M$ bits for the Carter-Wegman MAC. More abstractly we consider a 1-query bounded MAC M in the construction below:

```
Init()
K_I ← ⊥
st_S ← ⊤
st_R ← ⊤
return (K_I, st_S, st_R)

OTKey()
K_1 ←$ {0,1}^M
K_2 ←$ MKGen()
return K_1||K_2

Send (st_S, K_I, K, m)
// let K = K_1||K_2
c ← OTP.Enc (K_1, m)
t ← MAC (K_2, c)
C ← (c, t)
return (C, st_S)

Recv (st_R, K_I, K, C)
// let K = K_1||K_2
d ← Verify(K_2, c, t)
if st_R == ⊥ or d == false then
    m ← ⊥
    st_R ← ⊥
else
    m ← OTP.Dec (K_1, c)
fi
return (m, st_R)
```

Figure 4.5: Unconditionally-secure channel protocol USCh

Construction 47 (Unconditionally-Secure Channel)
Define the channel protocol USCh $= (\mathsf{Init}, \mathsf{OTKey}, \mathsf{Send}, \mathsf{Recv})$ *for message space* $\mathcal{M} = \{0,1\}^{\leq M}$ *by the algorithms in Figure 4.5.*

Once more we first argue INT-SFCTXT security of the channel protocol, assuming that the underlying MAC scheme M is strongly unforgeable against unbounded adversaries. The noteworthy fact here is that we lose a factor of $q_{\mathsf{Send}} + 1$ of sender queries in the security bound:

Lemma 48
For any INT-SFCTXT *adversary* $\mathcal{I}$ *making at most* q_{Send} *sender oracle queries there exists a* 1*-query bounded adversary* $\mathcal{F}$ *such that*

$$\mathsf{Adv}^{\text{int-sfctxt}}_{\mathsf{USCh}}(\mathcal{I}) \leq (q_{\mathsf{Send}} + 1) \cdot \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{M}}(\mathcal{F}). \tag{4.4.3}$$

Furthermore, $\mathcal{F}$ *uses the same resources as* $\mathcal{I}$*.*

Note that a Carter-Wegman MAC satisfies $\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{M}}(\mathcal{F}) \leq 2^{-M}$ if we authenticate messages of at most M bits with $2M$ key bits [113]. This means that, as long as the number q_{Send} of sent ciphertexts is limited, the bound in the lemma is still reasonably small. Interestingly, for small message sizes M though and with a focus on "only" future-secure protection, an HMAC-based instantiation of Construction 44 can provide better concrete security.

Proof. The proof follows the one for the computational case closely. Only this time $\mathcal{F}$ guesses in advance, with probability $\frac{1}{q_{\mathsf{Send}}+1}$, the number i of the sender query for which $\mathcal{I}$ sends the first modified ciphertext $C^* \neq C_i$ to the receiver oracle, where we account for the possibility that $j > i$ with the additional choice $i = q_{\mathsf{Send}} + 1$. Algorithm $\mathcal{F}$ simulates an execution of $\mathcal{I}$ by doing all steps locally, and answering each receiver request with $\perp$. Only in the i-th sender oracle query, $\mathcal{F}$ uses the external MAC oracle to compute the tag (still using a self-chosen, independent key part K_1 to encrypt the message before). When the integrity adversary $\mathcal{I}$ outputs the first modified ciphertext $C^* = (c^*, t^*)$ to the receiver oracle, then $\mathcal{F}$ returns the pair (c^*, t^*) as its forgery attempt.

Given that the guess i is correct it follows as in the computational case that $\mathcal{F}$ wins the 1-query bounded unforgeability game if $\mathcal{I}$ wins the integrity game. Here we use that $\mathcal{F}$ at most makes a single external MAC query – or none if $i = q_{\mathsf{Send}} + 1$ – and creates a (strong) forgery against the MAC scheme, because the pair (c^*, t^*) must be distinct from the MAC query (for $i \leq q_{\mathsf{Send}}$) or even new (for $i = q_{\mathsf{Send}} + 1$). □

It follows – as in the computational case – that Theorem 43 yields overall security of the unconditionally-secure channel protocol.

Theorem 49 (Unconditionally-Secure Channel)
For the channel protocol USCh *in Construction 47 and any* IND-SFCCA *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *where* $\mathcal{A}_1$ *makes at most* q_{Send} *sender oracle queries, we can construct an* INT-SFCTXT *adversary* $\mathcal{F}$ *such that*

$$\mathsf{Adv}^{\text{ind-sfcca}}_{\mathsf{FSCh}}(\mathcal{A}) \leq (q_{\mathsf{Send}} + 1) \cdot \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{M}}(\mathcal{F}). \tag{4.4.4}$$

Here, $\mathcal{F}$ *uses the same resources as* $\mathcal{A}_1$ *and is* 1*-query bounded.*

4.4.3 Application in a Long-Term Storage Solution

In Section 3, we discussed that to instantiate a long-term storage solution such as ELSA or MCELSA, secure communication between the parties has to be guaranteed. As we showed in this Section 4, this can be achieved by providing any pair of parties in need of a secure, private channel, e.g., a client and a shareholder in an instance of MCELSA, with a source of symmetric, confidential keying material such as quantum key distribution. The parties can thereby deploy a future-secure or unconditionally-secure channel, respectively, so that throughout the run-time of the storage solution their communication cannot be intercepted or manipulated. We concede that current QKD implementations come with a significant price tag, yet in a scenario in which information security is indispensable as for example storage of personal medical data, such an investment is justifiable.

5 Assisted Multi-Party Computation

This chapter is based on the work [90].

5.1 Motivation

In Section 3, we have introduced MCELSA, a storage architecture that provides long-term confidentiality and indefinitely prolongable integrity as well as authenticity for stored documents. The existence of private channels between the shareholders, that are engaged in an instance of it, and the clients, that send the shares of the documents and decommitment values attached to them to the shareholders, is vital for the security of an instance of MCELSA.

We discussed how to establish such a channel between two parties, that have access to a shared source of secret symmetric keying material, in a manner that is future-secure or unconditionally-secure in Section 4. That is, the communication exchanged over such a channel is confidential, authentic and has integrity with regards to an attacker who has control over the channel and grows in computational power and cryptoanalytic capabilities over time.

In this chapter, we go one step further and discuss, how to perform computations on data that has been stored in a secret sharing instance, as is the case in LINCOS, ELSA or MCELSA. More specifically, we address research question 3 and investigate how to improve the efficiency of the preprocessing step of multi-party computation (MPC) protocols, that are based on secret sharing schemes.

The most intuitive approach for computing on secret shared data is to reconstruct the shared secret(s) and execute the computation on the results of the reconstruction. While this approach is a rather efficient one, it harbours the danger of reconstructing the secret in the clear at the machine executing the computation, thereby posing a chance of exposing it, as discussed in [28]. A less intuitive yet all the more secure approach is to employ an MPC protocol, a concept that was introduced in the 1980s with works like [64]. An MPC protocol enables a set of parties to jointly evaluate an agreed upon function represented as a circuit so that each party provides a private input, the parties jointly execute the computation and a result is obtained while each party learns only the result and what can be derived from it concerning the other parties' inputs.

find Thomas' paper

Garbled circuits and secret sharing based protocols are two of the most important categories of MPC. The most prominent example of the former category is Yao's garbled circuit [115]. The latter category contains protocols in which inputs to a circuit are distributed among the parties via a secret sharing scheme. The circuit is evaluated layer by layer in communication rounds and the result is jointly opened by publishing the shares of the result. In this Section 5 we focus on secret sharing based MPC protocols.

Secret sharing based MPC protocols comprise themselves of an offline or preprocessing phase and an online phase. In the offline phase, the engaged parties jointly generate and share the auxiliary data necessary for the evaluation of the circuit. For example, many secret sharing based MPC protocols utilise random "Beaver triples" [7] for performing multiplications, which are pre-computed in the offline phase. The offline phase is executed independently of the parties' inputs, it solely depends on the structure of the circuit. In the online phase, the actual evaluation of the circuit takes place. The parties provide their respective private inputs and jointly compute the output of each gate to arrive at the final result.

In recent years, the focus of most research aiming to improve the efficiency of MPC protocols has been on optimising the online phase, and remarkable achievements have been made. Yet in most cases, these improvements to the online phase entail performance penalties in the offline phase. While preprocessing can be executed far in advance to the online phase and is independent of the parties' inputs, it can however not be skipped. An example, that illustrates the performance disparity between offline and online phase, is the minimum euclidean distance computation for 1000 values of 128 bits in the framework ABY [48] between two parties with a network delay of 50ms. The time for the online phase amounts to 4442ms, whereas the offline phase takes 16506ms, i.e., close to 80 percent of the protocol's duration was spent on preprocessing. While this disparity was addressed in [97], we think that further performance improvements may be possible.

5.1.1 Our Contribution

We propose an approach to significantly improve the performance of the offline phase for secret sharing based MPC protocols. We achieve this by introducing an independent helper party that assists in executing the offline phase and sharing the resulting data among the original parties. The helper party will not take part in the online phase, it can hence not obtain any knowledge with respect to the parties' inputs or the execution of the online phase. Indeed, we show that the introduction of the helper party does not affect the security guarantees provided by the original MPC protocols. We furthermore demonstrate the practical applicability of our approach by giving an implementation of the helper party for the SPDZ offline phase. We empirically test the efficiency of our implementation in two scenarios. The first is the rate of Beaver triples generated and shared per second among two to five parties. The second is the preprocessing for a Vickrey auction with one hundred bids. Our tests show a performance improvement of up to a factor of 69 in comparison to previous works like SPDZ [43] and evolutions thereof. We discuss three real world instantiations of our approach. We elaborate on their respective limitations and advantages regarding performance. We thereby demonstrate that our approach is not of a purely theoretical nature, but can in fact be applied in real executions of secret sharing based MPC protocols. Our approach is applicable to any MPC protocol based on secret sharing.

5.1.2 Structure

First, we present our approach to improve the efficiency of the offline phase in Section 5.2, that is, an additional helper party P_h that takes on executing the offline phase and distributing the resulting shares to the original parties of the MPC protocol. Second, we prove that if the original protocol was simulatable, the resulting protocol with P_h executing the offline phase is simulatable as well. Third, we discuss how our approach can be applied to SPDZ [43] in Section 5.4.1. Fourth, we give benchmark results for our implementation of the helper party in the context of SPDZ in Section 5.4.2, that show a significant performance benefit from our approach in comparison to the unmodified protocols. Finally we discuss feasible real world implementations of our model in Section 5.3.

5.1.3 Related Work

Secret sharing schemes were first introduced independently in the 1970s by Shamir [101] and Blakley [18]. Both proposed threshold secret sharing schemes with perfect secrecy over the ring $\mathbb{Z}_p$ for a prime p larger than the number of shareholders. Verifiable secret sharing was developed as an additional feature in works like [93] and [60]. Fitzi et al. [59] presented a different approach to verifiable secret sharing, that provided a storage efficient scheme in three rounds to share a secret. Tassa [108] introduced an extension of Shamir's scheme, that enables a multi-level threshold access structures, so that for reconstruction the threshold of each hierarchy level has to be fulfilled. Traverso et al. [111] further improved the capabilities of hierarchical secret sharing by proposing a scheme that was also dynamic and verifiable. Our approach is applicable to any secret sharing based MPC protocol, independent of the underlying secret sharing scheme.

A scheme enabling integer secrets outside $\mathbb{Z}_p$ was later presented by Damgard and Thorbek [44] along with a protocol for distributed exponentiation on the shared secret. Its caveat is that the security was only computational. Rabin and Ben-Or [95] combined verifiable secret sharing with general computation on shared secrets, thus a more general approach, yet less performant.

The field of secure multi-party computation holds a wide choice of schemes that have been established over the years, most prominent of which are Goldreich et al.'s GMW [64] and Damgard et al.'s SPDZ [43]. Both schemes rely on computational assumptions with respect to security, whereas Bogdanov et al. [19] provide an entirely information-theoretically secure approach with Sharemind at the cost of performance. Cramer et al. [40] transposed SPDZ to a setting where computation is not over finite fields $\mathbb{F}_{p^k}$ but over the more practical binary field $\mathbb{F}_{2^k}$, providing the same security guarantees. We will show that our proposition maintains the security guarantees of the MPC protocol it is being applied to while giving a significant performance boost to the offline phase.

A secure two-party computation protocol employing a third party for the circuit evaluation was discussed by Feige et al. [55]. We continue this direction of research by employing the additional party exclusively for preprocessing, thereby strengthening the security properties of their approach, since we neither introduce nor rely on further computational hardness assumptions.

The model of secure outsourced computation was addressed in Loftus and Smart's work [82], in which they propose a protocol, that has a set of input parties I outsource the computation to a set P and a set R obtain the results, where P is disjoint from I and R. Outsourcing the offline phase of the SPDZ protocol to a smaller set of parties – either disjoint or a subset of the original set of parties – was discussed

previously by Scholl et al. [100]. TaaS by Smart and Tanguy [105] extended the previous work in that they enable a flexible set of servers to be used in the outsourcing of the preprocessing of SPDZ-style MPC protocols. They only require an honest majority in the set of servers, yet in their approach the servers are not allowed to communicate after the preprocessing is triggered. Damgard et al. [42] further investigated the approach of outsourced preprocessing in their protocol for two-party computation, where a set of servers provides correlated randomness to the two parties of the online phase. They allow for at most t corrupted servers and one corrupted party in the online phase. Chaudhari et al. [34] proposed ASTRA, a protocol for three parties, one of which is to execute the offline phase and supply the other two with the necessary preprocessing, who then execute the online phase without the first party. In contrast to our approach, all three parties may contribute an input to the online phase, whereas we strictly exclude the helper party from the online phase. With Tetrad, Koti et al. [74] proposed a four-party computation protocol in which one party is designated to provide the preprocessing for the other three parties. Koti et al. [73] focused on improving the efficiency of the online phase their MPC protocol. For that they split the parties in their protocol into two groups, i.e., a set of helpers $\mathcal{D}$ and a set of evaluators $\mathcal{E}$, one of which takes the distinguished role of P_{king}. This party takes a central position in the reconstruction of shared secrets in the protocol.

Our approach distinguishes itself from those above in that they propose concrete MPC protocols that utilise one or more parties to execute the preprocessing, whereas our approach for concentrates on speeding up the preprocessing in any secret sharing based MPC protocol.

A different direction was taken in Keller et al.'s MASCOT [70], where the offline phase was improved by employing oblivious transfer protocols (OT). In Overdrive, Keller et al. [71] proposed that SPDZ with improvements to its original design provides an offline phase that is similar to the improvements achieved by MASCOT. With Overdrive2k, Orsini et al. [92] transferred the improvements of Overdrive to the setting of computation over F_{2^k}. Their works engage all parties of the online phase in the offline phase, whereas we have the helper party P_h execute the offline phase. We test the performance of our approach on the offline phase of SPDZ.

5.2 Model

Let us first specify the adversary that we consider with regards to our approach.

5.2.1 Adversary

We consider a static and active attacker. That is, the attacker corrupts an unauthorised set of shareholders $C \subset \{P_1, \ldots, P_n\}$ upon the initialisation of the protocol. Throughout the execution of the MPC protocol, this set cannot be changed. The adversary obtains all knowledge that the corrupted shareholders have. This includes their inputs $\{x_i\}_{P_i \in C}$, the randomness used by the shareholders $\{(r_i)_j\}_{P_i \in C}$ and the messages they receive from the other parties in the protocol $\{(m_i)_j\}_{P_i \in C}$. The adversary controls all outputs of the corrupted parties and the messages, that they send to other parties.

5.2.2 The Helper Party P_h

In the offline phase of an MPC protocol, the parties $P_1, \ldots, P_n$ jointly generate data, that is used in the online phase to evaluate C_f. This data is independent of the inputs that $P_1, \ldots, P_n$ provide in the online phase. The structure of the data is determined by the gates contained in C_f. For an arbitrary but fixed circuit C_f, let $\{t_1, \ldots, t_m\}$ therefore be the set of all types of gate contained in C_f. In an arithmetic circuit, t_1 may for example denote input gates, t_2 multiplication gates, t_3 addition gates and t_4 output gates. We denote the set of all gates in C_f by $\mathcal{G} = \left\{g_1, \ldots, g_{|C_f|}\right\}$. Furthermore, we define a function $\phi : \mathcal{G} \to \{t_1, \ldots, t_m\}$, that maps a gate to its type. In the offline phase, the parties $P_1, \ldots, P_n$ jointly sample a data set $D = \left\{d_1, \ldots, d_{|C_f|}\right\}$, where d_i contains the data necessary for the evaluation of the gate g_i for $i = 1, \ldots, |C_f|$. Let $D_1, \ldots, D_m$ be a partition of D faithful to the gates' type, that is,

$$\bigcup_{i=1}^{m} D_i = D \tag{5.2.1}$$

holds, while $D_i \cap D_j = \emptyset$ for all $1 \leq i < j \leq m$, as well as

$$\forall i = 1, \ldots, m : \forall d, d' \in D_i : \phi(d) = \phi\big(d'\big).$$

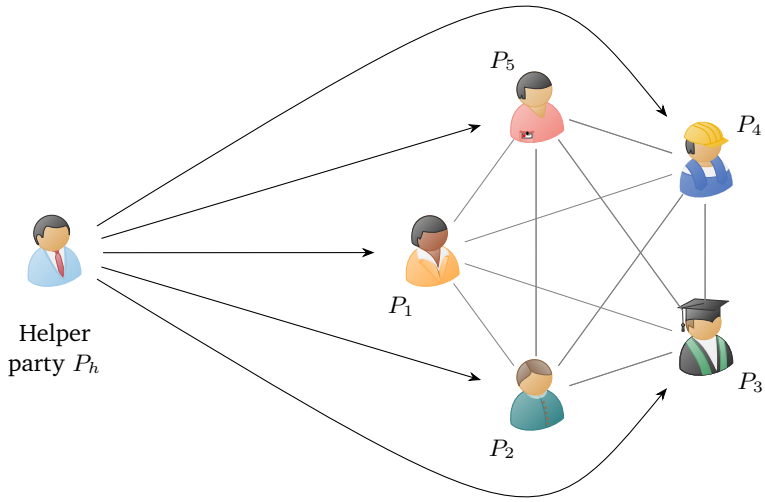

Figure 5.1: The Helper Party P_h's Interaction with the Original Parties $P_1, \ldots, P_5$

For each D_i, $1 \leq i \leq m$, there exists an underlying distribution X_i from which D_i is sampled.

We introduce a new party to the MPC protocol. This party we call the "helper party" P_h. Its purpose is to execute the offline phase in place of the parties $P_1, \ldots, P_n$ in order to give a significant speed up compared to the traditional execution. The parties $P_1, \ldots, P_n$ give a description of the circuit C_f and a probability $p_f > 0$, that indicates the trust they have in P_h, as input to P_h. We further elaborate on p_f later on. From the descrciption of C_f, P_h derives the set of data D according to (5.2.1), that is to be produced in the offline phase. P_h generates and shares a data set corresponding to the distribution of D, thereby executing the offline phase in place of $P_1, \ldots, P_n$. After providing $P_1, \ldots, P_n$ with the auxiliary data necessary for the evaluation of C_f, P_h resets itself if it will be employed in future executions of the MPC protocol. Otherwise it shuts down. P_h does not take part in the online phase, it especially neither provides input nor receives output with respect to the evaluation of C_f.

Our approach implies that the helper party P_h sends the generated shares to each party P_i, $i = 1, \ldots, n$, privately. Communication between pairs P_i and P_j, $1 \leq i < j \leq n$, is not intended. This contrasts traditional approaches, in which the offline phase requires communication among all parties. We therefore reduce the number of necessary secure private channels from $n(n-1)$ between each pair of parties to n channels between P_h and each P_i, $i = 1, \ldots, n$. This further improves the efficiency of the offline phase. We point out that this does not have any impact on the online phase, which is explicitly left unmodified in any MPC protocol our approach is applied to.

Depending on the concrete instantiation of P_h, the parties $P_1, \ldots, P_n$ must assume that P_h does not collude with any other party. We give examples for the non-collusion of P_h being a necessary assumption as well as is not being necessary in Section 5.3.

While P_h has neither input nor output in the online phase, malformed preprocessing data can falsify the output or even reveal a party's input. We thus present P_h in two flavours, depending on whether P_h is trusted by $P_1, \ldots, P_n$, which is indicated by the probability p_f.

- If P_h is regarded as a trusted party by $P_1, \ldots, P_n$, we have $p_f = 1$. P_h produces a data set D' corresponding to the distribution of the set D. That is, for each D_i, $i = 1, \ldots, m$, P_h samples a set D'_i of size $\#D_i$ from X_i. P_h shares each $d \in D'_i$ among $P_1, \ldots, P_n$ via $\mathcal{S}.\mathsf{Share}(d)$, where $\mathcal{S}$ is the secret sharing scheme underlying the MPC protocol. The parties $P_1, \ldots, P_n$ utilise the shares received from P_h in place of the output of the offline phase and evaluate C_f accordingly. The online phase hence remains unchanged.

- An untrusted helper party is indicated by $p_f < 1$. The parties $P_1, \ldots, P_n$ agreed on a probability p_f, that they accept as a chance of undetected dishonest behaviour on P_h's side. For each D_i in (5.2.1),

$\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{A})$

$c \leftarrow\$ \{0,1\}$
$S' \leftarrow\$ 2^S \setminus \Gamma$
$p \leftarrow\$ [0,1]$
$\mathsf{prep}_0 \leftarrow \mathcal{O}_{\text{real prep}}(C_f, p, S')$
$\mathsf{prep}_1^* \leftarrow P_h(C_f, p)$
$\mathsf{prep}_1 \leftarrow \mathsf{prep}_1^*|_{S'}$
$c' \leftarrow \mathcal{A}(\mathsf{prep}_c)$
return $c == c'$

(a) Experiment $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{A})$

$\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\mathcal{A})$

$c \leftarrow\$ \{0,1\}$
$S' \leftarrow\$ 2^S \setminus \Gamma$
for $P_i \in S'$
  $x_i \leftarrow\$ X$
  $y_i \leftarrow\$ Y$
endfor
$t_0 \leftarrow \mathsf{Sim}_{\mathcal{P}}\left((x_i, y_i)_{P_i \in S'}\right)$
$t_1 \leftarrow \mathsf{Sim}_{\mathcal{P}'}\left((x_i, y_i)_{P_i \in S'}\right)$
$c' \leftarrow \mathcal{A}(t_c)$
return $c == c'$

(b) Experiment $\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\mathcal{A})$

Figure 5.2: Experiments $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{A})$ and $\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\mathcal{A})$

P_h generates

$$k_i = \left\lceil \frac{(1-p_f)}{p_f} \#D_i \right\rceil$$

additional items according to the distribution of D_i for each $i \in \{1, \ldots, m\}$. We denote the resulting set by D'_i, where $\#D'_i = \#D_i + k_i$.

Prior to evaluating C_f in the online phase, $P_1, \ldots, P_n$ apply a cut-and-choose approach to the shares they received from P_h in that they randomly choose a portion of k_i elements of each set D'_i, $i = 1, \ldots, m$, and publicly reconstruct them. If a malformed data item is opened this way, P_h will be considered dishonest and the received data is discarded. Otherwise the data received from P_h is deemed correct and the evaluation of C_f is continued with the remaining unopened data being used as the result of the preprocessing step. Since we have $\#D'_i = \#D_i + k_i$ items for each $i = 1, \ldots, m$, a sufficient number of unopened data items remains. We show in Theorem 50 that p_f bounds the probability with which P_h can successfully provide malformed data items without being detected.

Remark
The scenario of a trusted helper party can be regarded as a special case of the untrusted helper party, where we have $p_f = 1$.

Theorem 50 (Cheating probability of P_h)
The probability that P_h generates a malformed data set without detection is upper bounded by p_f.

Proof. Fix an arbitrary subset D'_i, where $1 \leq i \leq m$. Let a denote the number of malformed items in D'_i. If $a > \#D_i$, then the computing parties necessarily selects a malformed data item for opening, since $\#D'_i = \#D_i + k$. The probability of successful cheating for P_h is thus 0. We hence assume $a \leq \#D_i$. Let us first consider the case of single malformed data item in D_i, i.e., $a = 1$. The probability of $P_1, \ldots, P_n$ not selecting the malformed item is therefore

$$\frac{\binom{\#D'_i - 1}{k}\binom{1}{0}}{\binom{\#D'_i}{k}} = \frac{\binom{\#D_i + k - 1}{k}\binom{1}{0}}{\binom{\#D_i + k}{k}} = \frac{\frac{(\#D_i + k - 1)!}{k!(\#D_i - 1)!}}{\frac{(\#D_i + k)!}{k!\#D_i!}}$$
$$= \frac{\#D_i}{\#D_i + k} \leq \frac{\#D_i}{\#D_i + \frac{1-p_f}{p_f}\#D_i} = p_f.$$

The statement therefore holds for $a = 1$. For $a > 1$, that is, more than one malformed data item, the probability of malformed items not being selected and opened in the cut-and-choose paradigm is clearly upper bounded by the probability in the case of $a = 1$. Therefore, P_h can only successfully cheat with a probability at most p_f. □

5.2.3 Security

We now prove that the introduction of the helper party does not weaken the security of the MPC protocol. To this end, we show that a simulatable MPC protocol remains simulatable after the introduction of P_h.

We model the indistinguishability of the output of a real preprocessing from the output of P_h in Experiment $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{A})$ given in Figure 5.2a. In this game, we denote by $\mathcal{O}_{\text{real prep}}(\cdot)$ an oracle, that upon being handed the circuit description C_f, the failure probability p and an unauthorised set S' internally executes a preprocessing phase according to C_f and p and outputs the shares of the parties in S'.

Definition 51
For a circuit C_f, let $\mathcal{A}$ be an arbitrary algorithm. The advantage of $\mathcal{A}$ in Experiment $\mathsf{Exp}^{\textit{ind-prep}}_{C_f}(\mathcal{A})$ is defined as

$$\mathsf{Adv}^{\textit{ind-prep}}_{C_f}(\mathcal{A}) = \left|\frac{1}{2} - \Pr\left[\mathsf{Exp}^{\textit{ind-prep}}_{C_f}(\mathcal{A}) = \mathsf{true}\right]\right|.$$

Lemma 52
For any unauthorised subset of shareholders $S' \subset \{P_1, \ldots, P_n\}$, the data produced in the preprocessing phase is perfectly indistinguishable from the data provided by P_h. That is, for any adversary $\mathcal{A}$ and any circuit C_f and any failure probability p_f, we have

$$\mathsf{Adv}^{\textit{ind-prep}}_{C_f}(\mathcal{A}) = 0.$$

Proof. Each data item produced in a real preprocessing phase is information-theoretically hidden from S'. This holds for each data item that was generated and shared among the computing parties by P_h. Since each data item is sampled independently from each other, the data sets received are perfectly indistinguishable. □

We now prove that the MPC protocol, that arises from introducing the helper party to a simulatable MPC protocol, is simulatable itself. We capture this notion in $\mathsf{Exp}^{\text{ind-trans}}(\cdot)$.

Definition 53
The advantage of an adversary $\mathcal{A}$ in Experiment $\mathsf{Exp}^{\textit{ind-trans}}_{C_f}(\mathcal{A})$ is defined as

$$\mathsf{Adv}^{\textit{ind-trans}}_{C_f}(\mathcal{A}) = 2\left|\frac{1}{2} - \Pr\left[\mathsf{Exp}^{\textit{ind-trans}}_{C_f}(\mathcal{A}) = \mathsf{true}\right]\right|.$$

An MPC protocol is simulatable *if, for any circuit C_f and any adversary $\mathcal{A}$, we have*

$$\mathsf{Adv}^{\textit{ind-trans}}_{C_f}(\mathcal{A}) = 0.$$

Theorem 54
Let $\mathcal{P}$ be a simulatable, secret sharing based MPC protocol. And let $\mathcal{P}'$ be an identical protocol, yet with the modification detailed above applied to it, that is, the offline phase is executed by a helper party P_h. Then $\mathcal{P}'$ is simulatable.

Proof. We prove Theorem 54 in three steps: first, we give a simulator for any unauthorised subset of $\{P_1, \ldots, P_n\}$ in the protocol $\mathcal{P}'$. Second, we prove the indistinguishability of said simulator from that of the original protocol $\mathcal{P}$, which gives us the simulatibility of $\mathcal{P}'$ in a third step.

We denote by C_f the circuit, that $\mathcal{P}$ and hence $\mathcal{P}'$ are to evaluate. Let $S' \notin \Gamma$ be an arbitrary unauthorised subset of $\{P_1, \ldots, P_n\}$ and let $\mathsf{Sim}_{\mathcal{P}}$ denote the simulator of $\mathcal{P}$. Thus upon receiving

$$\left\{(x_i, y_i)_{P_i \in S'}\right\}$$

as input, $\mathsf{Sim}_{\mathcal{P}}$ outputs a transcript that is indistinguishable from $\{\mathsf{view}_i\}_{P_i \in S'}$.

We give a simulator $\mathsf{Sim}_{\mathcal{P}'}$ for $\mathcal{P}'$. The simulator $\mathsf{Sim}_{\mathcal{P}'}$ uses $\mathsf{Sim}_{\mathcal{P}}$ in the following manner. Let $\left\{(x'_i, y'_i)_{P_i \in S'}\right\}$ denote the input of $\mathsf{Sim}_{\mathcal{P}'}$. $\mathsf{Sim}_{\mathcal{P}'}$ runs $\mathsf{Sim}_{\mathcal{P}}\left((x'_i, y'_i)_{P_i \in S'}\right)$ and returns whatever $\mathsf{Sim}_{\mathcal{P}}$ outputs.

It remains to prove that the output of $\mathsf{Sim}_{\mathcal{P}'}$ indistinguishable from that of $\mathsf{Sim}_{\mathcal{P}}$. We capture this in Experiment $\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\mathcal{A})$ in Figure 5.2b. We give a reduction of the preprocessing distinguishing problem to the simulator distinguishing problem to show the hardness of the former. Hence let $\mathcal{D}$ be an adversary in $\mathsf{Exp}^{\text{ind-trans}}_{C_f}(\cdot)$ with positive advantage. We construct a polynomial-time adversary $\mathcal{D}'$ against $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\cdot)$ that uses $\mathcal{D}$ to gain the same advantage. The input to an adversary $\mathcal{D}'$ in $\mathsf{Exp}^{\text{ind-prep}}_{C_f}(\mathcal{D}')$ is a

set of preprocessing shares $\{\mathsf{prep}_i^*\}_{P_i \in S'}$ for an unauthorised set S'. To simulate $\mathsf{Exp}_{C_f}^{ind-trans}(\cdot)$ to $\mathcal{D}$, $\mathcal{D}'$ samples $(x_i)_{P_i \in S'}$ and $(y_i)_{P_i \in S'}$ from their respective distributions. $\mathcal{D}'$ then hands

$$t_c = \mathsf{Sim}_{\mathcal{P}}\Big((x_i, y_i)_{P_i \in S'}\Big)$$

to $\mathcal{D}$. $\mathcal{D}'$ then outputs whatever decision bit $\mathcal{D}$ outputs. It remains to argue that $\mathcal{D}'$ has the same advantage in $\mathsf{Exp}_{C_f}^{\text{ind-prep}}(\mathcal{D}')$ as $\mathcal{D}$ has in $\mathsf{Exp}_{C_f}^{\text{ind-trans}}(\mathcal{D})$. The output of a simulator $\mathsf{Sim}\Big((x_i, y_i)_{P_i \in S'}\Big) = (x_i, (r_i)_j, (m_i)_j)_{P_i \in S'}$ looks identically distributed to an unauthorised set of parties $S' \notin \Gamma$. The randomness $(r_i)_j$ is either shared among all parties or locally sampled for each party P_i in a secret sharing based MPC protocol. Thus, for any $\{(r_i)_j\}_{P_i \in S'}$, we have

$$\begin{aligned}\mathsf{Sim}_{\mathcal{P}}\Big((x_i, y_i)_{P_i \in S'}\Big) &= (x_i, (r_i)_j, (m_i)_j)_{P_i \in S'} \overset{\text{perf}}{=} \\ &(x_i, (r_i')_j, (m_i)_j)_{P_i \in S'} = \mathsf{Sim}'_{\mathcal{P}}\Big(\big((x_i, y_i), (r_i')_j\big)_{P_i \in S'}\Big),\end{aligned}$$

where $\mathsf{Sim}'_{\mathcal{P}}$ outputs the same as $\mathsf{Sim}_{\mathcal{P}}$, but replaces the randomness $(r_i)_j$ with $(r_i')_j$. This gives us

$$\mathsf{Sim}_{\mathcal{P}}\Big((x_i, y_i)_{P_i \in S'}\Big) \overset{\text{perf}}{=} \mathsf{Sim}_{\mathcal{P}'}\Big((x_i, y_i)_{P_i \in S'}\Big)$$

and thus

$$\mathsf{Adv}_{\mathcal{D}'}^{\text{ind-prep}}(C_f) = \mathsf{Adv}_{\mathcal{D}}^{\text{ind-trans}}(C_f).$$

With Lemma 52, this gives us $\mathsf{Adv}_{\mathcal{D}'}^{\text{ind-trans}}(C_f) = 0$, thus the output of $\mathsf{Sim}'_{S'}$ is thus indistinguishable from that of $\mathsf{Sim}_{S'}$, which in turn is indistinguishable from the real view $\mathsf{view}_{S'}$. The MPC protocol $\mathcal{P}'$ is therefore simulatable.

□

5.3 Instantiations for the Helper Party

In Section 5.2, we introduced the helper party P_h and proved that the security of an MPC protocol it is applied to is not impacted. We now discuss feasible real world instantiations for P_h and their advantages and shortcomings as well as the necessary assumptions. Since the data shared by the helper party is known in plaintext to P_h, the role of P_h cannot be assumed by a computing party P_i, $1 \leq i \leq n$, itself.

5.3.1 Trusted Execution Environment

The computing parties $P_1, \ldots, P_n$ delegate the offline phase to an agreed upon trusted execution environment (TEE) such as ARM's TrustZone [80] or AMD's Secure Processor [83]; these are just a few examples, other implementations are available. A TEE provides the capability to have an (almost) arbitrary computation executed by an external party in a secure and trusted way. This is achieved by having the TEE prove that the program executed coincides with the program that was given as input via a remote attestation protocol. Thus the parties ascertain that the TEE executes the protocol for the helper party and only that. A TEE is to be found in virtually any main stream CPU sold today, thus it is widely available.

To ensure safe communication with the TEE, each party P_i, $i = 1, \ldots, n$, establishes a secure and private channel with the TEE by appropriate means. This channel is then used to transmit the shares, that the TEE generates in executing the helper party's task.

Utilising a TEE to implement P_h allows for a entirely counter-intuitive approach: a party $P_j \in \{P_1, \ldots, P_n\}$ that has a TEE at its disposal may provide the other parties access to it and have the helper party's protocol executed in it. With the parties P_i, $i \neq j$, establishing private channels to the TEE, P_j cannot obtain knowledge on the shares received by P_i, less it breaks the TEE or the channel protocol.

It is reasonable to propose that the parties forgo the MPC protocol by sending the TEE their private inputs and having it evaluate the circuit C_f. Yet modern applications making use of library components such the GNU C library (glibc) or the C mathematical library (libm) require substantial amounts of memory for their execution due to the size of said libraries. In many cases, these exceed the hardware limitations inherent to the TEE implementation. The direct evaluation of large circuits such as privacy preserving machine learning [35, 41] in a typical TEE hence can therefore only be achieved with significant engineering effort [24, 6]. Executing the offline phase is nevertheless entirely actionable with the amount of data to be persistently held in memory being almost negligible.

Having a TEE implement the helper party allows $P_1, \ldots, P_n$ to assume P_h as trusted. The protocol of P_h can therefore be instantiated in its most efficient configuration.

Overall we claim that a TEE represents a feasible instantiation of P_h, since its widespread availability and the helper party protocol being in its most efficient configuration outweigh the limitations inherent to this approach.

5.3.2 Unrelated External Party

The parties $P_1, \ldots, P_n$ may alternatively employ an unrelated external party for the task of P_h. This approach distinguishes itself from the former in that P_h is considered untrusted outright, that is, the shares received from it are not assumed correct. The parties hence agree on a probability $p_f < 1$ as detailed in Section 5.2 and apply the cut-and-choose paradigm to verify the correctness of the received shares. The external party executing the task of P_h is incentivised to behave honestly by monetary reward. This means that if the shares are considered honestly generated after $P_1, \ldots, P_n$ verified them, the helper party receives a previously agreed upon payment. With the computational effort of the protocol for the helper party being comparatively low even for large circuits, the monetary reward for the external party is little. As a concrete instantiation the parties may employ a minimalistic cloud instance as can readily be hired at a wide variety of commercial providers.

A major advantage of this approach in comparison to a TEE is that the external party is not limited with respect to the implementation of P_h. This results in an efficiently computed preprocessing phase by the external party.

A caveat of this approach is that the parties $P_1, \ldots, P_n$ must assume that P_h is not colluding with either of the parties, as we illustrated in Section 5.2. Also, the protocol for P_h cannot be instantiated in its most efficient fashion, i.e., a trusted helper party, since $p_f < 1$ must hold.

In fact, the performance analysis, that we will present in Section 5.4.2, was obtained in this setting, that is, on a commodity PC without specialised hardware. Further improved performance can be achieved with the use of hardware specialised for parallel computations.

5.3.3 Minimal Special Purpose Hardware

The parties $P_1, \ldots, P_n$ may deploy a piece of purpose-built computing hardware to execute the task of the helper party P_h. The design for said hardware is made public in a format like VHDL so that any party can verify that the computation carried out agrees with the protocol for P_h. As we already discussed in Section 5.3.1, even a minimalistic hardware design is sufficient for a successful execution of the preprocessing phase. Furthermore, purpose built hardware achieves significantly higher efficiency compared to general computing hardware such as a TEE or an external party. Since the design of the purpose-built hardware is public, no trust assumptions have to be placed in the helper party, which enables the protocol of P_h in its most efficient configuration, i.e., $p_f = 1$. Possible modifications to the design during manufacturing can be excluded by the parties inspecting the hardware prior to deployment.

Obtaining a piece of purpose-built hardware is however rather costly compared to the previously proposed instantiations. With the preprocessing data for an MPC protocol being identical with respect to structure and at most distinct in proportion, a piece of dedicated hardware is highly reusable in future executions of the MPC protocol.

The advantages of employing a minimalistic piece of hardware to implement P_h are efficient computation of the preprocessing data and an independence from a third party hosting a TEE or executing the protocol of P_h. In our oppinion these advantages outweigh the caveat of the initial cost of deployment by far.

The real world instantiations of the helper party discussed above are of course not exhaustive. We give three examples to illustrate that it is feasible to instantiate P_h without substantial monetary or organisational overhead while obtaining reasonable guarantees with respect to correctness and confidentiality of the generated data.

5.4 SPDZ Application and Performance

5.4.1 Application to SPDZ

We demonstrate the practicality of our approach by applying it to the offline phase in the SPDZ multi-party computation protocol established by Damgard et al. [43]. SPDZ enables a set of at least two parties to evaluate arithmetic circuits in $\mathbb{Z}_p$ for a prime p, where all gate inputs and outputs are certified with a global key. SPDZ thus naturally integrates a correctness measure for the computation.

# of parties	Average Beaver triples/s	Interquartile range	Maximum	Minimum
This work				
2	134324.81	3252.30	140252.45	125786.16
3	108193.12	1521.08	111982.08	103092.78
4	85519.79	1096.75	87412.59	82101.81
5	69115.70	741.08	70721.36	65876.15
SPDZ				
2	4,200			
MASCOT				
2	4800			
5	1000			
Overdrive				
2 (Low Gear)	15,000			
2 (High Gear)	2,300			

Figure 5.3: Benchmark results for 10,000 Beaver triples in $\mathbb{F}_p$, $\lceil \log_2 p \rceil = 128$

The offline phase in SPDZ can be considered as a process in two steps. First, a public key pk is established with an according secret key α. Each party P_i, $1 \leq i \leq n$, obtains a share of α. Second, the data necessary for the evaluation of each gate in the circuit to be evaluated in the online phase is generated. In SPDZ, we distinguish two methods of sharing. For a value x, we denote

$$[x] = \left((x_1, \ldots, x_n), \left(\beta_i, \gamma(x)^i_1, \ldots, \gamma(x)^i_n \right)_{i=1,\ldots,n} \right),$$

where $\sum_{i=1}^n x_i = x$ and $\sum_{i=1}^n \gamma(x)^i_j = x\beta_j$ for all $j = 1, \ldots, n$. Each party P_i holds shares x_i, β_i, $\gamma(x)^i_1, \ldots, \gamma(x)^i_n$. The secret key α is shared as $[\alpha]$. The sharing $\langle x \rangle$ denotes

$$\langle x \rangle = \left(\delta, (x_1, \ldots, x_n), \left(\gamma(x)_1, \ldots, \gamma(x)_n \right) \right),$$

where δ is publicly known and each party P_i, $1 \leq i \leq n$, obtains x_i and $\gamma(x)_i$ so that $\sum_{i=1}^n x_i = x$ and $\sum_{i=1}^n \gamma(x)_i = \alpha\,(x + \delta)$ hold.

A circuit suitable for SPDZ contains four types of gate: input gate, addition gate, multiplication gate and output gate. The preprocessing for an input gate consists of a random value r, that is shared as $\langle r \rangle$ as well as $[r]$. To input a value x, a party P_i has the other parties open $[r]$ to him and computes and publishes $\epsilon \leftarrow x - r$. The parties then derive their local shares of $x = \epsilon + \langle r \rangle$. For an addition of two shared values $\langle x \rangle$ and $\langle y \rangle$, the parties locally add their respective shares according to the sharing described above. Thus no preprocessing is required for addition gates. A multiplication gate requires two Beaver triples $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$ and $\langle x \rangle$, $\langle y \rangle$, $\langle z \rangle$, where $ab = c$ and $xy = z$, and a random $[t]$. The sharings $\langle x \rangle$, $\langle y \rangle$, $\langle z \rangle$ and $[t]$ are opened to verify the correctness of the triple $\langle a \rangle$, $\langle b \rangle$ and $\langle c \rangle$, which is then used for the multiplication itself. The preprocessing for an output gate entails a simple shared random value $[r]$.

We instantiate the helper party as an untrusted external party as detailed in Section 5.3.2. That is, we run the implementation on off-the-shelf hardware and have it distribute the shares resulting from the SPDZ offline phase among the shareholders, i.e., the parties of the online phase.

In assisting the offline phase of SPDZ, the helper party analyses the circuit to be evaluated for the gates types it contains and determines the data to be generated. The helper party P_h then samples the respective data items and shares them among $P_1, \ldots, P_n$ in the appropriate format of $\langle \cdot \rangle$ or $[\cdot]$. We assume the existence of secure private channels between P_h and each P_i, $i = 1, \ldots, n$. For methods to instantiate such a channel, we refer to Section 4.

5.4.2 Performance

We demonstrate the performance gains of our approach in the offline phase over the original SPDZ protocol as presented in [43] and the improvements proposed in Overdrive [71] and MASCOT [70]. For that, we implement the protocol to be executed by P_h in C++. We use the boost library in its version 1.74. This natively enables us to generate and share data items of 128 bits so that our results are comparable to those of [71, 70], who also used 128 bits of randomness in their implementation.

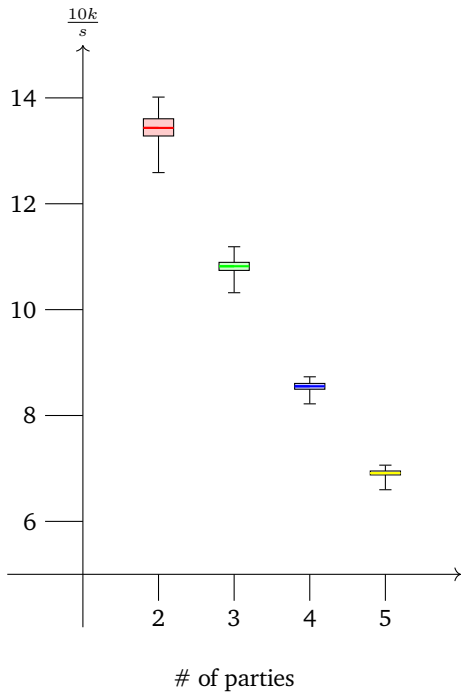

Figure 5.4: Generating Beaver triples for up to five parties

We evaluate the performance of our implementation on commodity PCs in a local network, where the helper party is run on a machine with eight cores and thirty-two GB of RAM. This setup is almost identical to that of the performance test of MASCOT [70].

Our test is executed in two scenarios: First, we measure the maximum possible rate of generating and sharing Beaver triples $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$ and $\langle x \rangle$, $\langle y \rangle$, $\langle z \rangle$ along with the complimenting randomness $[t]$. Second, we execute the offline phase for a Vickrey auction with one hundred bids.

Output of Beaver triples for SPDZ.

We measure the time elapsed for generating the preprocessing for 10,000 multiplication gates, deriving the throughput per second. Our test is executed for two to five parties. We give the resulting performance numbers in Figure 5.3 with a visual representation in Figure 5.4. It can be seen, that in the setting of two parties in the online phase, our implementation improves the results of SPDZ and MASCOT at a factor of 30, and those of Overdrive more than nine-fold. In the setting of five parties, our implementation outperforms MASCOT by a factor of 69.

Vickrey Auction.

In the second scenario, we simulate the preprocessing for a Vickrey auction with one hundred bids. The winner of the auction is the highest bidder, yet the price to be paid by the winning party is the second highest bid. The layout of this auction incentivises the parties to provide realistic bids. A modified version of this is in fact utilised by online auction house eBay.

We simulate the preprocessing for an online phase that is carried out between two up to one hundred parties. We test each setting one hundred times. The circuit is taken from the performance test of [71] and uses 44571 Beaver triples. We give the performance numbers in Figure 5.6.

As can be seen in Figure 5.5, the time elapsed for the preprocessing increases linearly with the number of parties in the online phase, which agrees with the amount of data to be generated. In MASCOT a setting of two parties was evaluated, which we outperform by a factor of over 60. And in Overdrive the online phase was carried out between 100 parties, which we improved on by a factor of over 20.

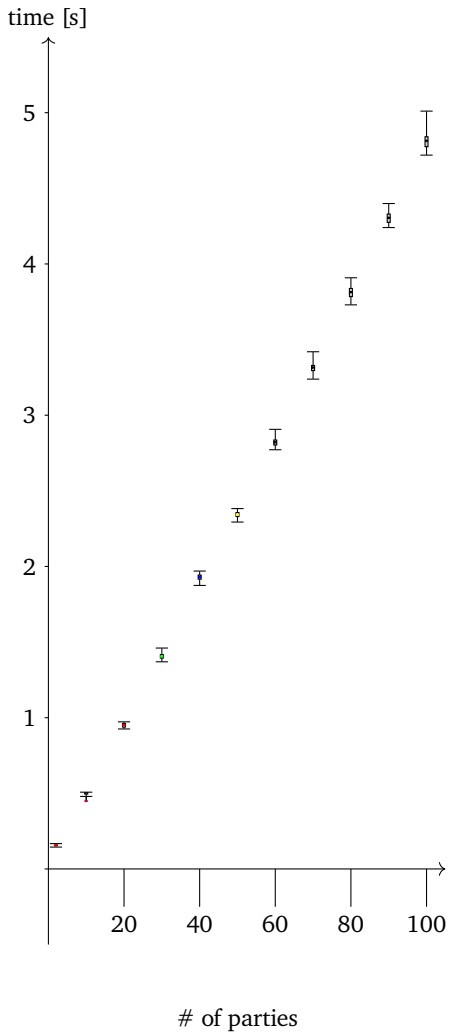

Figure 5.5: Vickrey auction preprocessing for up to 100 parties

# of parties	Average time [s]	Interquartile range	Maximum	Minimum
This work				
2	0.157	0.171	0.141	0.004
10	0.449	0.511	0.483	0.009
20	0.951	0.976	0.922	0.023
30	1.405	1.464	1.366	0.023
40	1.929	1.973	1.871	0.027
50	2.342	2.386	2.29	0.026
60	2.821	2.91	2.768	0.033
70	3.315	3.423	3.235	0.034
80	3.814	3.912	3.726	0.054
90	4.305	4.403	4.237	0.055
100	4.814	5.014	4.716	0.066
MASCOT				
100	1,300			
Overdrive				
100 (High Gear)	98			

Figure 5.6: The Vickrey auction preprocessing for up to 50 parties in $\mathbb{F}_p$, $\lceil \log_2 p \rceil = 128$

6 General Access Structures for Isogeny based Cryptography

This chapter is based on the work [31].

6.1 Motivation

The documents deposited in an instance of MCELSA are stored within a secret sharing scheme. Thus to compute a function with one or more stored documents as input such as statistical evaluation thereof, secret sharing based MPC protocols can be applied in a straightforward manner. Especially since reconstructing a shared secret can be avoided, they maintain the confidentiality of the data computed on.

More general, MPC protocols enable a set of parties to evaluate any function representable as an appropriate circuit on their respective private inputs without revealing them to the other parties. But in this capacity, MPC protocols are oftentimes surpassed in efficiency by protocols that are specifically tailored to the concrete function, that is to be computed.

In this Section 6, we hence address research question 4 and discuss a specific multi-party public key computation with secret shared secret key set in a hard homogeneous space. More concretely, a key exchange mechanism and signature scheme using isogeny based cryptography. The setting of hard homogeneous spaces raises a further problem: most MPC protocols are not natively equipped for computation in a HHS, which introduces further efficiency deficits. This makes the use of protocols tailored to the specific application all the more appealing, when considering computations in hard homogeneous spaces.

This Section 6 was inspired by De Feo and Meyer [45], who previously introduced a key exchange mechanism using isogeny based public key cryptography in which the secret key was shared in a Shamir threshold scheme. While their protocols present an efficient solution for the problem at hand, it suffers from two caveats: on the one hand, their appraoch is only passively secure, in that while even an active attacker cannot obtain information on the secret key shares of other shareholders participating in an execution of the decapsulation or signing protocols, his deviation from the protocol cannot be detected. And on the other hand, by utilising Shamir's threshold secret sharing scheme they restrict themselves to rudimentary threshold access structures.

Our task therefore is two-fold. First, we need to find a suitable method to ensure active security in a key exchange mechanism, that has its secret key shared among a set of shareholders and is based in a hard homogeneous space. That active security measure should furthermore enable participants in the decapsulation protocol to identify a misbehaving party so that it can be excluded from future executions. We furthermore discuss whether the key exchange mechanism arising from the applied security measures can still be transferred into a signature scheme by the Fiat-Shamir-transformation without complications, as De Feo and Meyer did with their's. Second, we characterise the necessary properties for a secret sharing scheme to be compatible with our key exchange mechanism. This way, we can open our key exchange mechanism to a wider field of access structures and application scenarios. This continues the recent trend of developing new applications for secret sharing schemes [81, 51, 50, 22].

6.1.1 Our Contribution

Our contribution hence is manifold. First, we transfer the active security measures outlined in Section 2.2.13 and Section 2.2.15 from their setting of full engagement protocols to a setting of threshold secret sharing. We thereby enable a significantly wider range of applications for those security measures and also improve upon their efficiency. Second, we apply the adapted active security measures to propose an actively secure key exchange mechanism with secret shared secret key. Third, we present an actively secure signature scheme by applying a Fiat-Shamir transform to our key exchange mechanism. And fourth, we expand our key encapsulation mechanism and our signature scheme to a wider field of secret sharing schemes. For that, we characterise the necessary properties for a secret sharing scheme and give examples of compatible and incompatible schemes to show the potential and limits of our approach.

We point out that Section 6.3 was contributed by Fabio Campos, whereas the remaining Sections 6.1, 6.2 and 6.4 were contributed by the author of this work.

6.1.2 Related Work

Threshold secret sharing schemes were first introduced by Blakley [18] and Shamir [101] in 1979. In both their approaches, secrets from the secret space $\mathbb{Z}_p$ for prime p are shared by distributing interpolation points of randomly sampled polynomials. Damgård and Thorbek [44] presented a secret sharing scheme with secret space $\mathbb{Z}$. Thorbek [109] later improved their scheme, yet their scheme is only computationally confidential, compared to the information-theoretical confidentiality of Shamir and Blakley's schemes. Tassa [107] opened Shamir's scheme to more general application scenarios by utilising the derivatives of the sharing polynomial to construct a hierarchical access structure. These basic secret sharing schemes rely on the dealer providing honestly generated shares to the shareholders. Verifiable secret sharing schemes eliminate this drawback by providing the shareholders with the means to verify the correctness of the received shares with varying overhead. Examples of these are [13, 93, 106]. With minor efficiency losses, Herranz and Sáez [68] were able to achieve verifiable secret sharing for generalised access structures. Traverso et al. [110] proposed an approach for evaluating arithmetic circuits on secrets shared in Tassa's scheme, that also enabled auditing the results. Cozzo and Smart [38] investigated the possibility of constructing schemes with a shared secret based on the Round 2 candidate signature schemes in the NIST standardization process[1]. Based on CSI-FiSh [15], De Feo and Meyer [45] introduced threshold variants of passively secure encryption and signature schemes in the Hard Homogeneous Spaces (HHS) setting. Cozzo and Smart [39] presented the first actively secure but not robust distributed signature scheme based on isogeny assumptions. In [14], the authors presented CSI-RAShi, a robust and actively secure distributed key generation protocol based on Shamir's secret sharing in the setting of HHS, which necessitates all shareholders to participate.

6.1.3 Outline

We present an actively secure threshold key exchange mechanism and prove our scheme's active security and simulatability in Section 6.2. The actively secure signature scheme resulting from applying the Fiat-Shamir-transform to our key exchange mechanism is discussed in Section 6.3. Finally, the necessary properties for a secret sharing scheme to be compatible with our key exchange mechanism and signature scheme are characterised in Section 6.4 in order to enable applying a more general class of secret sharing schemes.

6.2 Key Exchange Mechanism

We introduced the cryptographic primitive of a key exchange mechanism (KEMs) in Section 2.2.12. A KEM is defined by three protocols: KGen, Encaps and Decaps. We present our actively secure key exchange mechanism with a secret key that is shared among a set of shareholders. An authorised subset can execute the protocol Decaps without reconstructing the secret key. We assume that the secret sharing scheme, in which the secret key is shared, is information-theoretically hiding. Traditional KEMs assume that a single party holds the secret key; this is not the case in this scenario. We therefore have to adjust the protocols, by which a KEM is defined, to fit the secret shared setting.

6.2.1 Public Parameters

We give our key exchange mechanism in the context of Shamir's secret sharing scheme and elaborate possible extensions to other, more general secret sharing schemes in Section 6.4. We therefore fix the following publicly known parameters.

- A secret sharing instance $\mathcal{S}$ with shareholders $S = \{P_1, \ldots, P_n\}$, secret space $\mathbb{Z}_p$ and access structure Γ.
- A hard homogeneous space $(\mathcal{E}, \mathcal{G})$ with a fixed starting point $E_0 \in \mathcal{E}$.
- A fixed element $g \in \mathcal{G}$ with $\mathsf{ord} g = p$ for the mapping $[\cdot]\cdot \colon \mathbb{Z}_p \times \mathcal{E} \to \mathcal{E}; s \mapsto g^s E$.

[1] https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization

$\mathsf{KGen}(S)$

$\mathsf{sk} \leftarrow\$ \mathbb{Z}_p$
$\mathsf{pk} \leftarrow [\mathsf{sk}]\, E_0$
$\{s_1, \ldots, s_n\} \leftarrow \mathcal{S}.\mathsf{Share}(s)$
for $i = 1, \ldots, n$
  $f_i \leftarrow\$ \mathbb{Z}_p\, [X]_{\leq k-1} : f_i(0) = s_i$
endfor
publish pk
for $i = 1, \ldots, n$
  send $\left\{s_i, f_i, \{f_j(i)\}_{j=1,\ldots,n}\right\}$ to P_i
endfor

Figure 6.1: The protocol KGen

6.2.2 The Adversary

We consider a static and active adversary. At the beginning of a protocol execution, the adversary corrupts a set of shareholders. The adversary is able to see their inputs and control their outputs. The set of corrupted shareholders cannot be changed throughout the execution of the protocol.

The adversary's aim is two-fold. On the one hand it wants to obtain information on the uncorrupted parties' inputs, on the other hand it wants to falsify the output of the execution of our protocol without being detected.

6.2.3 Communication channels

Both our schemes, that is, the key exchange mechanism and the derived signature scheme, assume the existence of a trusted dealer in the secret sharing instance. The shareholders' communication occurs in the execution of the decapsulation protocol of our key exchange mechanism and the signing protocol of our signature scheme.

The communication from the dealer to a shareholder must not be eavesdropped upon or tampered with, we hence assume secure private channels between the dealer and each shareholder. We discussed methods how establish such a channel in Section 4. However, the communication between shareholders need not be kept private, thus we assume a simple broadcast channel between the shareholders.

6.2.4 Key Generation

A public and secret key pair is established by a trusted dealer (even an untrusted dealer is feasible by employing verifiable secret sharing schemes) executing KGen (see Figure 6.1) with the set of shareholders S as input. For that, he samples a secret key $\mathsf{sk} \in \mathbb{Z}_p$ and publishes the public key $\mathsf{pk} \leftarrow [\mathsf{sk}]E_0$. The secret key sk is then shared among the shareholders $\{P_1, \ldots, P_n\}$ via $\mathcal{S}.\mathsf{Share}(\mathsf{sk})$. The dealer shares each share s_i of sk, $i = 1, \ldots, n$, once more with a sharing polynomial f_i. Each shareholder $P_i \in S$ eventually receives s_i, f_i and $f_j(i)$, that is, his share s_i of sk, the polynomial f_i and a share $f_j(i)$ of each other s_j, $j \neq i$.

This key generation protocol can be regarded as a "two-level sharing", where each share of the secret key is itself shared again among the shareholders. While this is not necessary for De Feo and Meyer's passively secure protocol [45], we require the two-level sharing in ensuring the active security of our key encapsulation mechanism.

6.2.5 Encapsulation

With a public key $\mathsf{pk} \in \mathcal{E}$ as input, the encapsulation protocol Encaps (Figure 6.2) samples an element $b \in \mathcal{G}$, computes the ephemeral key $\mathsf{k} \leftarrow b * \mathsf{pk} \in \mathcal{E}$ and a ciphertext $c \leftarrow b * E_0 \in \mathcal{E}$. It then returns (k, c).

Our encapsulation protocol is identical to that of De Feo and Meyer's work [45], thus we just give a short sketch and refer to [45] for the respective proofs of security.

$\underline{\mathsf{Encaps}(\mathsf{pk})}$

$b \leftarrow\$ \mathcal{G}$

$\mathsf{k} \leftarrow b * \mathsf{pk}$

$c \leftarrow b * E_0$

$\textbf{return } (\mathsf{k}, c)$

Figure 6.2: The protocol Encaps

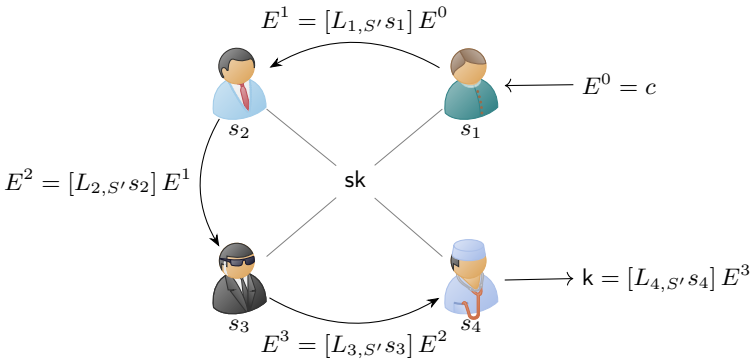

Figure 6.3: Shareholders $P_1, \ldots, P_4$ decapsulating a ciphertext c

6.2.6 Decapsulation

A traditional decapsulation protocol takes a ciphertext c and a secret key sk as input and outputs a key k. De Feo and Meyer [45] applied the threshold group action (see Figure 2.16) so that an authorised set $S' \in \Gamma$ decapsulates a ciphertext c and produces a key $[s]\, c = [s]\, (b * E_0) = b * ([s]\, E_0)$. For that, the shareholders agree on an arbitrary order of turns. With $E^0 := c$, the kth shareholder P_i outputs $E^k = [L_{i,S'} s_i]\, E^{k-1}$ for $k = 1, \ldots, \#S'$. The last shareholder outputs the decapsulated ciphertext $E^{\#S'} = [s]\, c$. We sketch this protocol for a setting of four shareholders $P_1, \ldots, P_4$ holding shares $s_1, \ldots, s_4$ of the secret key sk in Figure 6.3. Their approach is simulatable. It hence does not leak any information on the shares s_i, yet it is only passively secure. We extend their approach to enable identifying misbehaving shareholders in an execution of the decapsulation protocol. For that, we maintain the threshold group action and apply the PVP and zero-knowledge proof for the group action inverse problem as layed out in Section 2.2.13 and Section 2.2.15.

Amending the PVP

In the PVP protocol sketched in Section 2.2.13, a prover produces a proof of knowledge for a witness polynomial f of the statement $((E_0, E_1), s_1, \ldots, s_n)$, where $E_0 \leftarrow\$ \mathcal{E}$, $E_1 = [s_0]\, E_0$ and $s_i = f(i)$ for $i = 0, \ldots, n$. He thereby proves knowledge of the sharing polynomial f of $s_0 = f(0)$.

This approach does not agree with the threshold group action detailed in Section 2.2.14, for which a shareholder P_i's output in the round-robin approach is $E^k \leftarrow [L_{i,S'} s_i]\, E^{k-1}$ rather than $E^k \leftarrow [s_i]\, E^{k-1}$, where E^{k-1} denotes the previous shareholder's output. Futhermore, authorised sets need not contain all shareholders. Example 55 illustrates a further conflict with of the PVP with the threshold group action.

Example 55
Let sk *be a secret key generated and shared by* KGen. *That is, each shareholder P_i holds*

$$\left\{ s_i, f_i, \{f_j(i)\}_{P_j \in S} \right\}.$$

Also, let $S' \in \Gamma$ be a minimally authorised set executing the threshold group action protocol given in Figure 2.16, i.e., for any $P_i \in S'$, $S' \setminus \{P_i\}$ is unauthorised. Thus, for any arbitrary but fixed $s_i' \in \mathbb{Z}_p$, there exists a polynomial $f_i' \in \mathbb{Z}_p[X]_{k-1}$ so that $f_i'(j) = L_{i,S'} f_i(j)$ and $R' = [f_i'(0)]\, R$ for any $R, R' \in \mathcal{E}$. Therefore, P_i can publish $\left(\pi, \{\pi_j\}_{P_j \in S'}\right)$ with

$$\left(\pi, \{\pi_j\}_{P_j \in S}\right) \leftarrow \mathsf{PVP.Pv}\left(\left((R, R'), (L_{i,S'} s_{ij})_{P_j \in S}\right), f_i'\right),$$

$\mathsf{TPVP.Pv}\Big(i, f, S^*, \Big((E_0, E_1), (s_{ij})_{P_j \in S^*}\Big)\Big)$

for $l = 1, \ldots, \lambda$

$\quad b_l \leftarrow\$ \mathbb{Z}[X]_{\leq k-1}$

$\quad \hat{E}_l \leftarrow [b_l(0)]\, E_0$

endfor

$y_0, y_0' \leftarrow\$ \{0,1\}^\lambda$

$C_0 \leftarrow \mathsf{CS}\Big(\hat{E}_1 || \ldots || \hat{E}_\lambda, y_0\Big)$

$C_0' \leftarrow \mathsf{CS}(E_0 || E_1, y_0')$

for $P_j \in S^*$

$\quad y_j, y_j' \leftarrow\$ \{0,1\}^\lambda$

$\quad C_j \leftarrow \mathsf{CS}(b_1(j) || \ldots || b_\lambda(j), y_j)$

$\quad C_j' \leftarrow \mathsf{CS}(L_{i,S^*} \cdot s_{ij}, y_j')$

endfor

$C \leftarrow (C_j)_{P_j \in S^*}$

$C' \leftarrow (C_j')_{P_j \in S^*}$

$(c_1, \ldots, c_\lambda) \leftarrow \mathsf{H}(C, C')$

for $l = 1, \ldots, \lambda$

$\quad r_l \leftarrow b_l - c_l \cdot L_{i,S^*} \cdot f$

endfor

$\mathbf{r} \leftarrow (r_1, \ldots, r_\lambda)$

$\Big(\pi, \{\pi_j\}_{P_j \in S^*}\Big) \leftarrow \Big((C, C'\mathbf{r}), \{(y_j, y_j')\}_{P_j \in S^*}\Big)$

Figure 6.4: The protocol TPVP.Pv

which is indistinguishable from

$$\mathsf{PVP.Pv}\Big(\Big((E_0, E_1), (L_{i,S'} s_{ij})_{P_j \in S}\Big), L_{i,S'} f_i\Big)$$

to $S' \setminus \{P_i\}$ *with* $E_0 \leftarrow\$ \mathcal{E}$ *and* $E_1 = [L_{i,S'} s_i]\, E_0$*. Thus, for a minimally authorised set* S'*, the soundness of the PVP does not hold with respect to* $P_i \in S'$ *and* f_i*.*

We resolve the conflicts by amending [14]'s PVP protocol so that a shareholder $P_i \in S^*$ proves knowledge of a witness polynomial $L_{i,S^*} f_i$ for a statement

$$\Big((R, R'), (f_i(j))_{P_j \in S^*}\Big),$$

to a superauthorised set S^*, where $R \leftarrow\$ \mathcal{E}$, $R' = [L_{i,S^*} f_i(0)]\, R = [L_{i,S^*} s_i]\, R$. The inputs of our amended proving protocol are the proving shareholder's index i, the witness polynomial f_i, the superauthorised set $S^* \in \Gamma^+$ and the statement $\Big((R, R'), (f_i(j))_{P_j \in S^*}\Big)$. The resulting threshold piecewise verifiable proving protocol can be found in Figure 6.4, in which CS denotes a commitment scheme. The verifying protocol in turn has the prover's and the verifier's indices i and j, respectively, a set $S^* \in \Gamma^+$, a statement piece x_j and a proof piece (π, π_j) as input, where $x_j = (R, R') \in \mathcal{E}^2$ if $j = 0$ and $x_j \in \mathbb{Z}_p$ otherwise. The threshold verifying protocol is given in Figure 6.5.

It is here, that the two-level sharing we introduced in Section 6.2.4 comes into play. We will have each shareholder P_i, that is engaged in an execution of Decaps, provide a PVP with respect to its share s_i of the secret key sk, that is then verified by each other participating shareholder with its respective share of s_i.

The definitions of soundness and zero-knowledge for our threshold PVP scheme carry over from the non-threshold setting they were introduced in in Section 2.2.13 intuitively, yet we restate the completeness definition for the threshold setting.

Definition 56 (Completeness in the threshold setting)
We call a threshold PVP scheme complete *if, for any* $S' \in \Gamma$*, any* $(x, f) \in \mathcal{R}$*, any* $P_i \in S'$ *and* $\Big(\pi, \{\pi_j\}_{P_j \in S'}\Big) \leftarrow$

$\mathsf{TPVP.Vf}(i, j, S^*, x_j, (\pi, \pi_j))$

$(C, C', \mathbf{r}) \leftarrow \pi$
$(r_1, \ldots, r_l) \leftarrow \mathbf{r}$
$(y_j, y'_j) \leftarrow \pi_j$
$(c_1, \ldots, c_\lambda)\, \mathsf{H}(C, C')$
if $j == 0$
  if $C'_j \neq \mathsf{CS}(x_j, y'_j)$
    return false
  fi
  for $l = 1, \ldots, \lambda$
    $\tilde{E}_l \leftarrow [r_l(0)]\, E_{c_l}$
  endfor
  return $C_0 == \mathsf{CS}\left(\tilde{E}_1 || \ldots || \tilde{E}_\lambda, y_0\right)$
else
  if $C'_j \neq \mathsf{CS}(L_{i,S^*} x_j, y'_j)$
    return false
  fi
  return $C_j == \mathsf{CS}(r_1(j) + c_1 \cdot L_{i,S^*} \cdot x_j || \ldots || r_\lambda(j) + c_\lambda \cdot L_{i,S^*} \cdot x_j, y_j)$

Figure 6.5: The protocol TPVP.Vf

$\mathsf{TPVP.Pv}(i, f, S', x_{S'})$, *we have*

$$\Pr\left[\mathsf{TPVP.Vf}\left(i, j, S', x_j, (\pi, \pi_j)\right) = \mathsf{true}\right] = 1 \textit{ for all } P_j \in S'.$$

The proofs for soundness, correctness and zero-knowledge for Beullens et al.'s [14] approach are easily transferred to our amended protocols, thus we do not restate them here.

An Actively Secure Decapsulation Protocol

We arrive at our decapsulation protocol Decaps, that takes a ciphertext c and a superauthorised set of shareholders S^* as parameters. The shareholders in S^* fix a turn order. A shareholder P_i's turn consists of the following steps.

1. If the previous shareholder's output E^{k-1} is not in $\mathcal{E}$, P_i outputs $\perp$ and aborts. The first shareholder's input E^0 is the protocol's input ciphertext c.
2. Otherwise P_i samples $R_k \leftarrow\$ \mathcal{E}$ and computes $R'_k \leftarrow [L_{i,S^*} s_i]\, R_k$.
3. P_i computes and publishes
$$\left(\pi^k, \left\{\pi^k_j\right\}_{P_j \in S^*}\right) \leftarrow \mathsf{TPVP.Pv}\left(i, f_i, S^*, \left(\left(R_k, R'_k\right), (f_i(j))_{P_j \in S^*}\right)\right).$$
4. P_i computes $E^k \leftarrow [L_{i,S^*} s_i]\, E^{k-1}$ and the zero-knowledge proof
$$zk^k \leftarrow \mathsf{ZK.Pv}\left(\left(R_k, R'_k\right), \left(E^{k-1}, E^k\right), L_{i,S^*} s_i\right).$$
He publishes both.
5. Each shareholder $P_j \in S^* \setminus \{P_i\}$ verifies
$$\mathsf{TPVP.Vf}\left(i, j, S^*, f_i(j), \left(\pi^k, \pi^k_j\right)\right) \wedge \mathsf{TPVP.Vf}\left(i, 0, S^*, \left(R_k, R'_k\right), \left(\pi^k, \pi^k_0\right)\right) \tag{6.2.1}$$
and
$$\mathsf{ZK.Vf}\left(\left(R_k, R'_k\right), \left(E^{k-1}, E^k\right), zk^k\right). \tag{6.2.2}$$
If (6.2.1) fails, P_j issues a complaint against P_i. If P_i is convicted of cheating by more than $\#S^*/2$ shareholders, decapsulation is restarted with an $S^{*\prime} \in \Gamma^+$ so that $P_i \notin S^{*\prime}$. If (6.2.2) fails, the decapsulation is restarted outright with $S^{*\prime} \in \Gamma^+$ so that $P_i \notin S^{*\prime}$.

$\mathsf{Decaps}(c, S^*)$

$E^0 \leftarrow c$
$k \leftarrow 0$
for $P_i \in S^*$
  if $E^k \notin \mathcal{E}$
    P_i outputs $\perp$ and aborts.
  fi
  $k \leftarrow k + 1$
  $R_k \leftarrow\$ \mathcal{E}$
  $R'_k \leftarrow [L_{i,S^*} \cdot s_i]\, R_k$
  $\left(\pi^k, \left\{\pi^k_j\right\}_{P_j \in S^*}\right) \leftarrow \mathsf{TPVP.Pv}\Big(i, f_i, S^*, \Big(\left(R_k, R'_k\right), (f_i(j))_{P_j \in S^*}\Big)\Big)$
  P_i publishes $\left(R_k, R'_k\right)$ and $\left(\pi^k, \left(\pi^k_j\right)_{P_j \in S^*}\right)$
  $E^k \leftarrow [L_{i,S^*} s_i]\, E^{k-1}$
  $zk^k \leftarrow \mathsf{ZK.Pv}\Big(\left(R_k, R'_k\right), \left(E^{k-1}, E^k\right), L_{i,S^*} s_i\Big)$
  P_i publishes $\left(E^k, zk^k\right)$
  for $P_j \in S^* \setminus \{P_i\}$
    if $\mathsf{ZK.Vf}\Big(\left(R_k, R'_k\right), \left(E^{k-1}, E^k\right), zk^k\Big) = \mathsf{false}$
      return $\mathsf{Decaps}(c, S^{*\prime})$ with $S^{*\prime} \in \Gamma^+ \wedge P_i \notin S^{*\prime}$
    fi
    if $\mathsf{TPVP.Vf}\Big(i, j, S^*, f_i(j), \left(\pi^k, \pi^k_j\right)\Big) = \mathsf{false} \vee \mathsf{TPVP.Vf}\Big(i, 0, S^*, \left(R_k, R'_k\right), \left(\pi^k, \pi^k_0\right)\Big) = \mathsf{false}$
      P_j publishes $f_i(j)$
      if P_i is convicted of cheating
        return $\mathsf{Decaps}(c, S^{*\prime})$ with $S^{*\prime} \in \Gamma^+ \wedge P_i \notin S^{*\prime}$
      fi
    fi
  endfor
endfor
return $\mathsf{k} \leftarrow E^k$

Figure 6.6: The protocol Decaps

6. Otherwise, P_i outputs E^k and finalises its turn.
7. The protocol terminates with the last shareholder's $E^{\#S^*}$ as output.

The combination of the threshold PVP and the zero-knowledge proof in steps 3 and 4 ensure that P_i has knowledge of the sharing polynomial $L_{i,S^*} f_i$ and also inputs $L_{i,S^*} f_i(0)$ to compute E^k. We give the precise protocol in Figure 6.6.

Definition 57 (Correctness)
A key exchange mechanism with secret shared private key is correct *if, for any authorised set S', any public key* pk *and any* $(\mathsf{k}, c) \leftarrow \mathsf{Encaps}(\mathsf{pk})$*, we have* $\mathsf{k} = \mathsf{k}' \leftarrow \mathsf{Decaps}(c, S')$.

The correctness of our key exchange mechanism presented in Figure 6.1, Figure 6.2 and Figure 6.6 follows from the correctness of the threshold group action (see Figure 2.16). Let sk be a secret key and $\mathsf{pk} = [\mathsf{sk}]\, E_0$ be the respective public key, that have been generated by executing $\mathsf{KeyGen}(S)$. Thus each shareholder P_i holds a share s_i of sk, $i = 1, \ldots, n$. For an authorised set S', we therefore have

$$\mathsf{sk} = \sum_{P_i \in S'} L_{i,S'} s_i.$$

Furthermore, let $(\mathsf{k}, c) \leftarrow \mathsf{Encaps}(\mathsf{pk})$. To show correctness, $\mathsf{k}' = \mathsf{k}$ has to hold, where $\mathsf{k}' \leftarrow \mathsf{Decaps}(c, S')$. Now, after executing $\mathsf{Decaps}(c, S')$, we have $\mathsf{k}' = E^{\#S'}$ emerging as the result of the threshold group

action applied to c. This gives us

$$\mathsf{k}' = \left[\sum_{P_i \in S'} L_{i,S'} s_i\right] c = [\mathsf{sk}]\,(b * E_0) = b * \mathsf{pk} = \mathsf{k}.$$

The decapsulation is executed by superauthorised sets $S^* \in \Gamma^+ \subset \Gamma$. This shows that our key exchange mechanism is correct.

6.2.7 Security

There are three aspects of security to consider for a key exchange mechanism with secret shared secret key:

- Active security: A malicious shareholder cannot generate his contribution to the decapsulation protocol dishonestly without being detected. We prove this by showing that an adversary that can provide malformed inputs without detection can break the threshold PVP or the zero-knowledge proof of knowledge.
- Simulatability: An adversary that corrupts an unauthorised set of shareholders cannot learn any information about the uncorrupted shareholders' inputs from an execution of the decapsulation protocol. We show this by proving the simulatability of Decaps.
- Indistinguishability of encapsulated keys: An adversary, that obtains a ciphertext must not learn anything about the key, that it was generated alongside with. We will prove that the key exchange mechanism we presented in Figure 6.1, Figure 6.2 and Figure 6.6 is IND-CPA-secure.

Active Security

Theorem 58
Let $S^* \in \Gamma^+$ and let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(S)$ be a public/secret key pair, where sk has been shared. Also let $(\mathsf{k}, c) \leftarrow \mathsf{Encaps}(\mathsf{pk})$. Denote the transcript of $\mathsf{Decaps}(c, S^*)$ by

$$\left(E^k, \left(R_k, R'_k\right), \left(\pi^k, \left\{\pi_j^k\right\}_{P_j \in S^*}\right), zk_k\right)_{k=1,\ldots,\#S^*}.$$

Let $P_i \in S^*$ be an arbitrary but fixed shareholder. If $\mathsf{Decaps}(c, S^*)$ terminated successfully and $P_{i'}$'s output was generated dishonestly, then there exists an algorithm that breaks the soundness property of TPVP or ZK.

Proof. Let $P_{i'}$ be the malicious shareholder and let k' be the index of $P_{i'}$'s output in the transcript. Since $\mathsf{Decaps}(c, S^*)$ terminated successfully, we have

$$\mathsf{TPVP.Vf}\left(i', j, S^*, f_{i'}(j), \left(\pi^{k'}, \pi_j^{k'}\right)\right) = \mathsf{true} \tag{6.2.3}$$

$$\mathsf{TPVP.Vf}\left(i', 0, S^*, \left(R_{k'}, R'_{k'}\right), \left(\pi^{k'}, \pi_0^{k'}\right)\right) = \mathsf{true} \tag{6.2.4}$$

$$\mathsf{ZK.Vf}\left(\left(E^{k'-1}, E^{k'}\right), \left(R_{k'}, R'_{k'}\right), zk^{k'}\right) = \mathsf{true} \tag{6.2.5}$$

for all $P_j \in S^* \setminus \{P_{i'}\}$. $E^{k'}$ was generated dishonestly, thus we have

$$E^{k'} = [\alpha]\, E^{k'-1}, \text{ for some } \alpha \neq L_{i',S^*} s_{i'}.$$

We distinguish two cases: $R'_{k'} \neq [\alpha]\, R_{k'}$ and $R'_{k'} = [\alpha]\, R_{k'}$.

In the first case, $P_{i'}$ published a zero-knowledge proof $zk^{k'}$ so that (6.2.5) holds, where $E^{k'} = [\alpha]\, E^{k'-1}$ yet $R'_{k'} \neq [\alpha]\, R_{k'}$. $P_{i'}$ thus broke the soundness property of the zero-knowledge proof.

In the second case, $P_{i'}$ published $\left(\pi^{k'}, \left\{\pi_j^{k'}\right\}_{P_j \in S^*}\right)$ so that (6.2.3) and (6.2.4) hold for all $P_j \in S^* \setminus \{P_{i'}\}$ and for $j = 0$. Thus $P_{i'}$ proved knowledge of a witness polynomial f' with

$$f'(j) = L_{i',S^*} f_{i'}(j) \tag{6.2.6}$$

for all $P_j \in S^* \setminus \{P_{i'}\}$ and $R'_{k'} = [f'(0)]\, R_{k'}$, that is, $f'(0) = \alpha$. Since f' has degree at most $k-1$, it is well-defined from (6.2.6). Thus, we have $f' \equiv L_{i',S^*} f_{i'}$, where $f_{i'}$ is the polynomial with which $s_{i'}$ was shared, i.e., $f_{i'}(0) = s_{i'}$. This gives us $\alpha = f'(0) = L_{i',S^*} f_{i'}(0) = L_{i',S^*} s_{i'}$. We arrive at a contradiction, assuming the soundness of the threshold PVP. □

$\mathsf{Exp}^{\text{dist-transcript}}_{\mathcal{S},\mathsf{Sim}}(\mathcal{A})$

$b \leftarrow\$ \{0,1\}$

$S^* \leftarrow\$ \Gamma^+$

$\left(\{s_i, \{s_{ij}\}, \{s_{ji}\}\}_{P_i, P_j \in S}, \mathsf{pk},\right) \leftarrow \mathsf{KGen}(S)$

$(\mathsf{k}, c) \leftarrow \mathsf{Encaps}(\mathsf{pk})$

$t_0 \leftarrow \mathsf{Sim}\left(\mathsf{k}, c, \{s_i, \{s_{ij}\}, \{s_{ji}\}\}_{P_i \in S^*, P_j \in S}\right)$

$E^0 \leftarrow E_0$

$k \leftarrow 0$

for $P_i \in S^*$

  $k \leftarrow k+1$

  $E^k \leftarrow \left[L_{i,S^*} s_i\right] E^{k-1}$

  $R_k \leftarrow\$ \mathcal{E}$

  $R'_k \leftarrow \left[L_{i,S^*} s_i\right] R_k$

  $\left(\pi^k, \left\{\pi^k_j\right\}_{P_j \in S^*}\right) \leftarrow \mathsf{PVP.Pv}\left(i, f_i, S^*, ((R_k, R'_k), (L_{i,S^*} s_{ij})_{P_j \in S^*})\right)$

  $zk^k \leftarrow \mathsf{ZK.Pv}\left(\left(R_k, R'_k\right), \left(E^{k-1}, E^k\right), L_{i,S^*} s_i\right)$

endfor

$t_1 \leftarrow \left(E^k, \left(\pi^k, \left\{\pi^k_j\right\}_{P_j \in S^*}\right), zk^k\right)_{k=1,\ldots,\#S^*}$

$b' \leftarrow \mathcal{A}(t_b)$

return $b == b'$

Figure 6.7: Experiment $\mathsf{Exp}^{\text{dist-transcript}}_{\mathcal{S},\mathsf{Sim}}(\mathcal{A})$

Simulatability

We show that an adversary who corrupts an unauthorised subset of shareholders does not learn any additional information from an execution of the decapsulation protocol.

Definition 59 (Simulatability)
We call a key exchange mechanism simulatable *if, for any HHS $(\mathcal{E}, \mathcal{G})$ with security parameter λ and any compatible secret sharing instance $\mathcal{S}$, there exists a polynomial-time algorithm* Sim *so that for any polynomial-time adversary $\mathcal{A}$ the advantage*

$$\mathsf{Adv}^{\text{dist}-\text{transcript}}_{(\mathcal{E},\mathcal{G}),\mathcal{S},\mathsf{Sim}}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{\textit{dist-transcript}}_{\mathcal{S},\mathsf{Sim}}(\mathcal{A})\right] - \frac{1}{2}\right|$$

in Figure 6.7 is negligible in λ.

Theorem 60
If the TPVP *protocol and the GAIP* ZK *protocol employed are zero-knowledge, then the decapsulation protocol (Algorithm 6.6) is simulatable.*

Proof. We give a finite series of simulators, the first of which simulates the behaviour of the uncorrupted parties faithfully and the last of which fulfills the secrecy requirements. This series is inspired by the simulators, that [14] gave for the secrecy proof of their key generation algorithm, yet differs in some significant aspects. The outputs of the respective simulators will be proven indistinguishable, hence resulting in the indistinguishability of the first and last one. As a slight misuse of the notation, we denote the set of corrupted shareholders by $\mathcal{A}$, where $\mathcal{A}$ is the adversary corrupting an unauthorised set of shareholders. This means P_i is corrupted iff $P_i \in \mathcal{A}$.

The input for each simulator is a ciphertext c, a derived key k and the adversary's knowledge after KGen was successfully executed, that is,

$$\left\{s_i, f_i, \{f_j(i)\}_{P_j \in S^* \setminus \mathcal{A}}\right\}_{P_i \in \mathcal{A}}.$$

1. The adversary corrupted an unauthorised set $\mathcal{A}$, hence each share of the secret key is uniformly distributed from his view. Sim^1 samples a polynomial $f_i' \in \mathbb{Z}_p\left[X\right]_{k-1}$ with

$$\forall P_j \in \mathcal{A} : f_i'(j) = f_i(j)$$

uniformly at random for each $P_i \in S^* \setminus \mathcal{A}$. Since $\mathcal{A}$ is unauthorised, f_i' exists.

Sim^1 then proceeds by honestly producing the output of each $P_i \in S^* \setminus \mathcal{A}$ according to the decapsulation protocol, i.e., it samples $R_k \leftarrow\$ \mathcal{E}$, computes $R_k' \leftarrow \left[L_{i,S^*} f_i'(0)\right] R_k$ and outputs

$$\mathsf{TPVP.Pv}\Big(i, f_i', S^*, \Big(\left(R_k, R_k'\right), \left(f_i'(j)\right)_{P_j \in S^*}\Big)\Big),$$

$$E^k \leftarrow \left[L_{i,S^*} f_i'(0)\right] E^{k-1}$$

and

$$\mathsf{ZK.Pv}\Big(\left(R_k, R_k'\right), \left(E^{k-1}, E^k\right), L_{i,S^*} f_i'(0)\Big),$$

where k is the index of P_i's output in the transcript. Since, for all $P_i \in S^* \setminus \mathcal{A}$, s_i is information-theoretically hidden to the adversary, the resulting transcript is identically distributed to a real transcript.

2. Let i' denote the index of the last honest party in the execution of the decapsulation protocol and k' the index of its output. Sim^2 behaves exactly as Sim^1 with the exception that it does not compute the threshold PVP itself but calls the simulator $\mathsf{Sim}^{\mathsf{TPVP}}$ for the threshold PVP to generate the proof $\left(\pi^{k'}, \left\{\pi_j^{k'}\right\}\right)$ for the statement $\left(\left(R_{k'}, R_{k'}'\right), \left(f_{i'}(j)\right)_{P_j \in S^*}\right)$. Since the threshold PVP is zero-knowledge, Sim^2's output is indistinguishable from that of Sim^1.

3. Sim^3 behaves identical to Sim^2 apart from not generating the zero-knowledge proof for $P_{i'}$ itself, but outsourcing it to the simulator for the zero-knowledge proof. That is, Sim^3 hands tuples $\left(R_{k'}, R_{k'}'\right)$ and $\left(E^{k'-1}, E^{k'}\right)$ to $\mathsf{Sim}^{\mathsf{ZK}}$ and publishes its answer as the zero-knowledge proof. With ZK being zero-knowledge, the output of Sim^3 is indistinguishable from that of Sim^2.

4. The final simulator, Sim^4, enforces the correct decapsulation output, that is, $E^{\#S^*} = \mathsf{k}$. Since, for $P_j \in \mathcal{A}$, s_j was provided as input and $P_{i'}$ is the last honest shareholder in the order of decapsulation execution, Sim^4 computes

$$\sum_{P_j \in S'} L_{j,S^*} s_j,$$

where S' contains the shareholders, whose turn is after $P_{i'}$'s. To achieve the correct output of the decapsulation E, Sim^4 thus sets

$$E^{k'} \leftarrow \left[-\sum_{P_j \in S'} L_{j,S^*} s_j\right] E$$

instead of $E^{k'} \leftarrow \left[L_{i',S^*} s_{i'}'\right] E^{k'-1}$. Assuming the soundness of the threshold PVP as well as of the zero-knowledge proof, this guarantees the result to be $E^{\#S^*} = E$, since

$$E^{\#S^*} = \left[\sum_{P_j \in S'} L_{j,S^*} s_j\right] E^{k'} = E$$

holds. It remains to show that the output of Sim^4 cannot be distinguished from that of Sim^3. The following reasoning is similar to that of [14], yet for completeness we give a reduction $\mathcal{B}'$, that uses a distinguisher $\mathcal{A}'$, that distinguishes Sim^3 from Sim^4 to break the decisional parallelisation problem. We highlight the necessary modifications.

Let (E_a, E_b, E_c) be an instance of the decisional parallelisation problem with base element c. $\mathcal{B}'$ computes

$$E^{k'} \leftarrow \left[\sum_{P_j \in S^* \setminus \left(S' \cup \left\{P_{i'}\right\}\right)} L_{j,S^*} s_j\right] E_a.$$

With $s_{i'}$ looking uniformly distributed from $\mathcal{A}$'s view, this choice of $E^{k'}$ is indistinguishable from $E^{k'} = [L_{i',S^*} s'_{i'}]\, E^{k'-1}$. $\mathcal{B}'$ furthermore does not sample $R_{k'} \leftarrow\$ \mathcal{E}$ but puts $R_{k'} \leftarrow E_b$ and $R'_{k'} \leftarrow E_c$. The resulting transcript is handed to $\mathcal{A}'$ and $\mathcal{B}'$ outputs whatever $\mathcal{A}'$ outputs.

Comparing the distributions, we see that

$$E^{k'} = [a]\, E^{k'-1} = [a] \left(\left[\sum_{P_j \in S^* \setminus (S' \cup \{P_{i'}\})} L_{j,S^*} s_j \right] c \right)$$

if and only if $E_a = [a]c$, where $s_j := s'_j$ for $P_j \notin \mathcal{A}$. Furthermore, $R'_{k'} = [a]R_{k'}$ is equivalent to $E_c = [a]E_b$. In the case of $E_a = [a]c$ and $E_c = [a]E_b$, the transcript handed to $\mathcal{A}'$ is identically distributed to Sim^3's output. If, on the other hand, (E_a, E_b, E_c) is a random triple, then the transcript follows the same distribution as Sim^4's output. $\mathcal{B}'$ thus breaks the DPP with the same advantage as $\mathcal{A}'$ distinguishes Sim^3 from Sim^4.

Sim^4 outputs a transcript of the decapsulation protocol with input c and output k that cannot be distinguished from the output of Sim^1, which is indistinguishable from a real execution protocol. □

Indistinguishability of encapsulated keys

Theorem 61
Let $\mathcal{K} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a key exchange mechanism in a HHS $(\mathcal{E}, \mathcal{G})$ with distinct element E_0 and key space K as detailed in Figure 6.1, Figure 6.2 and Figure 6.6. Then $\mathcal{K}$ is IND-CPA*-secure.*

Proof. We prove Theorem 6.2.7 by giving a reduction of the decisional parallelisation problem as detailed in Figure 2.12 to Experiment $\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A})$. For that, let $\mathcal{A}'$ be an adversary to the decisional parallelisation problem, that runs an instance of the adversary $\mathcal{A}$ against Experiment $\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A})$. $\mathcal{A}'$ receives $(E, E_{a^*}, E_{b^*}, E_{c^*})$ as input, where $E_{a^*} = [a^*]\, E$ and $E_{b^*} = [b^*]\, E$. Its task is to decide whether $E_{c^*} = [a^* + b^* \bmod p]\, E$ or $E_{c^*} = [c^*]E$ for some randomly selected $c^* \leftarrow\$ \mathbb{Z}_p$.

$\mathcal{A}'$ simulates the IND-CPA-security game as stated in Figure 2.13 to an adversary $\mathcal{A}$ in the context of our key exchange mechanism. For that, $\mathcal{A}'$ hands (E, E_{b^*}, E_{c^*}) to $\mathcal{A}$ as the challenge. $\mathcal{A}'$ returns whatever decision bit $\mathcal{A}$ outputs. It remains to argue that the advantage of $\mathcal{A}'$ in Experiment $\mathsf{Exp}^{\text{DPP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$ is at least that of $\mathcal{A}$ in Experiment $\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A})$. Two cases are to be considered:

- E_{c^*} was generated by $c^* = a^* + b^*$, that is, $b = 0$ in Experiment $\mathsf{Exp}^{\text{DPP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$, and thus $E_{c^*} = [a^*]\, E_{b^*} = [a^* + b^* \bmod p]\, E$. This corresponds directly to the case of $k_0^* = [\mathsf{sk}]\, c = \left[\mathsf{sk} + \log_g b\right] \mathsf{pk}$ in Experiment $\mathsf{Exp}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A})$.
- In the case of $b = 1$ in Experiment $\mathsf{Exp}^{\text{DPP}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A})$, we have $E_{c^*} = [c^*]\, E = g^{c^*} * E$ for a randomly chosen $c^* \leftarrow\$ \mathbb{Z}_p$. Due to the transitivity of the operation $*$ in $(\mathcal{E}, \mathcal{G})$, E_{c^*} is thus uniformly distributed in $\mathcal{E}$. This corresponds to k_1^* being randomly selected from K in Figure 2.13.

We thus have

$$\mathsf{Adv}^{\text{dpp}}_{(\mathcal{E},\mathcal{G})}(\mathcal{A}') = \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{K}}(\mathcal{A}) = \mathsf{negl}(\lambda).$$

□

6.2.8 Efficiency

Each shareholder engaged in an execution of the decapsulation protocol has one round of messages to send. The messages of the k-th shareholder consist of the tuple (R_k, R'_k), a threshold PVP proof $\left(\pi^k, \{\pi^k_j\}_{P_j \in S^*}\right)$, the output E^k and the zero-knowledge proof zk. Thus, the total size of a shareholder's messages is

$$\begin{aligned} &2x + 2c + \lambda k \log p + 2\lambda(\#S^*) + x + \lambda k \log p + \lambda \\ =&3x + 2c + \lambda\,(1 + 2(\#S^*) + 2k \log p) \end{aligned}$$

where x is the size of the bit representation of an element of $\mathcal{E}$ and c is the size of a commitment produced in $\mathsf{TPVP.Pv}$. Assuming x, c and the secret sharing parameters k and p to be constant, the message size is thus linear in the security parameter λ with moderate cofactor.

6.2.9 Verifiable Secret Sharing via Decapsulation

It is in many occasions not practical to employ a verifiable secret sharing scheme in order to ensure that the dealer did indeed generate the key pair $(\mathsf{sk}, \mathsf{pk})$ honestly and shared the secret key sk correctly. We therefore propose a method of verifying the correctness of the received shares without reconstructing the secret key. It uses the decapsulation protocol Decaps and a publicly encapsulated pair of key and ciphertext (k, c).

Assume, that the key generation protocol KGen has been executed, pk has been published and each shareholder P_i holds $\left\{s_i, f_i, \{f_j(i)\}_{j=1,\ldots,n}\right\}$, where s_i is P_i's share of sk and each s_j was shared again using the polynomial f_j. The shareholders want to verify that their shares of the secret key sk were honestly generated and that sk is indeed the secret key that corresponds to the public key pk. For that, the shareholders publicly execute $\mathsf{Encaps}(\mathsf{pk})$ and derive a tuple (k, c). The set of all shareholders S is a superauthorised set, they can hence execute $\mathsf{Decaps}(c, S)$. If sk and the shares of it were generated correctly, we have by the correctness of the KEM, that

$$\mathsf{k} = \mathsf{Decaps}(c, S).$$

If, however, $\mathsf{pk} \neq [\mathsf{sk}]\, E_0$, that is, pk was not derived correctly from sk, we have

$$\Pr[\mathsf{k} = \mathsf{Decaps}(c, S)] = \frac{1}{|\mathcal{E}|}. \tag{6.2.7}$$

This holds, because the randomly selected k and c were not known to the dealer before executing KGen, thus he cannot have generated sk and pk fittingly. The successful termination of the execution of Decaps can furthermore not be assumed. Let therefore $\varepsilon < 1$ denote the probability of Decaps terminating successfully. This gives us

$$\Pr[\mathsf{k} = \mathsf{Decaps}(c, S)] = \varepsilon \cdot \frac{1}{|\mathcal{E}|} < \frac{1}{|\mathcal{E}|}.$$

This process can be repeated by the shareholders to achieve an appropriate probability of the secret and public key being generated dishonestly without detection.

6.3 Actively Secure Secret Shared Signature Protocols

We convert the key exchange mechanism detailed in Figure 6.1, Figure 6.2 and Figure 6.6 into an actively secure signature scheme with secret shared signing key.

A signature scheme is defined by the protocols Setup, Sign and Vf. Since we are in a setting, where the secret key is not held by a single party, but can be produced by any authorised set of shareholders, we have to amend the signing protocol Sign to accommodate the secret key being contributed by a an authorised set rather than in the clear. The unmodified key generation protocol from the key exchange mechnism in Section 6.2.4 forms the protocol Setup. It produces a public key pk and shares the secret key sk among the shareholders in a two-level sharing fashion. We apply the Fiat-Shamir-transformation [56] to our decapsulation protocol Decaps (refer to Figure 6.6). The Fiat-Shamir-transformation as originally detailed considers identification schemes rather than key exchange mechanisms. We argue that a key exchange mechanism can be regarded as an identification scheme, in that a successful decapsulation clearly identifies an authorised set of shareholders. The verifying protocol follows straightforward. The protocols are given in Figure 6.8 and Figure 6.9.

We concede that applying active security measures to a signature scheme to ensure the correctness of the resulting signature is counter-intuitive, since the correctness of a signature can easily be checked through the verifying protocol. Yet verification returning false only shows that the signature is incorrect; a misbehaving shareholder cannot be identified this way. An actively secure signature scheme achieves just that. An identified cheating shareholder can hence be excluded from future runs of the signing protocol.

Similar to [15], the results from [52] on Fiat-Shamir in the QROM can be applied to our setting as follows. First, in the case without hashing, since the sigma protocol has special soundness [15] and in our case perfect unique responses, [52] shows that the protocol is a quantum proof of knowledge. Further, in the case with hashing, the collapsingness property implies that the protocol has unique responses in a quantum scenario.

$\mathsf{Sig.Sign}(m, S^*)$

$(E_1^0, \ldots, E_\lambda^0) \leftarrow (E_0, \ldots, E_0)$
$k \leftarrow 0$
for $P_i \in S^*$
  $k \leftarrow k + 1$
  for $l \in 1, \ldots, \lambda$
    $b_{il} \leftarrow\$ \mathbb{Z}_q[X]_{\leq k-1}$
    $R_{il}^k \leftarrow\$ \mathcal{E}; R_{il}^{k\,\prime} \leftarrow [b_{il}(0)]\, R_{il}^k$
    P_i publishes $\left(R_{il}^k, R_{il}^{k\,\prime}\right)$
    $\left(\pi_l^i, \left\{\pi_{l\,j}^i\right\}_{P_j \in S^*}\right) \leftarrow \mathsf{TPVP.Pv}\left(i, b_{il}, S^*, \left(\left(R_{il}^k, R_{il}^{k\,\prime}\right), (b_{il}(l))_{P_j \in S^*}\right)\right)$
    P_i publishes $\left(\pi_l^i, \left\{\pi_{l\,j}^i\right\}_{P_j \in S^*}\right)$
    P_i outputs $E_l^k \leftarrow [b_{il}(0)]\, E_l^{k-1}$
    $zk_l^k \leftarrow \mathsf{ZK.Pv}\left(\left(R_{il}^k, R_{il}^{k\,\prime}\right), \left(E_l^{k-1}, E_l^k\right), b_{il}(0)\right)$
    P_i publishes zk_l^k
    if $\mathsf{ZK.Vf}\left(\left(R_{il}^k, R_{il}^{k\,\prime}\right), \left(E_l^{k-1}, E_l^k\right), zk_l^k\right) = \mathsf{false}$
      return $\mathsf{Sign}\left(m, S^{*\prime}\right)$ with $S^{*\prime} \in \Gamma^+ \wedge P_i \notin S^{*\prime}$
    fi
  endfor
endfor
$(c_1, \ldots, c_\lambda) \leftarrow \mathsf{H}\left(E_1^{\#S^*}, \ldots, E_\lambda^{\#S^*}, m\right)$
for $P_i \in S^*$
  for $l = 1, \ldots, \lambda$
    P_i outputs $z_{il} = b_{il} - c_l L_{i,S^*} \cdot s_i$
    for $P_j \in S^*$
      $b'_{il}(j) \leftarrow z_{il}(j) + c_l L_{i,S^*} f_i(j)$
      if $\mathsf{TPVP.Vf}\left(i, j, S^*, b'_{il}(j), \pi_l^i, \pi_{l\,j}^i\right) = \mathsf{false}$
        $\vee\ \mathsf{TPVP.Vf}\left(i, 0, S^*, \left(R_{il}^k, R_{il}^{k\,\prime}\right), \pi_l^i, \pi_{l\,0}^i\right) = \mathsf{false}$
        P_j publishes $f_i(j)$
        if P_i is convicted of cheating
          return $\mathsf{Sign}\left(m, S^{*\prime}\right)$ with $S^{*\prime} \in \Gamma^+ \wedge P_i \notin S^{*\prime}$
        fi
      fi
    endfor
  endfor
endfor
for $l = 1, \ldots, \lambda$
  $z_j \leftarrow \sum_{P_i \in S^*} z_{ij}$
endfor
return $((c_1, \ldots, c_\lambda), (z_1, \ldots, z_\lambda))$

Figure 6.8: The protocol Sig.Sign

$\mathsf{Sig.Vf}(m, \sigma, \mathsf{pk})$

$(c_1, \ldots, c_\lambda, z_1, \ldots, z_\lambda) \leftarrow \sigma$

for $j = 1, \ldots, \lambda$

  if $c_j == 0$

    $E'_j \leftarrow [z_j]\, E_0 = \left[\sum_{P_i \in S^*} b_{ij} \right] E_0$

  else

    $E'_j \leftarrow [z_j]\, \mathsf{pk} = \left[\sum_{P_i \in S^*} b_{ij} - L_{i,S^*} s_i + s \right] E_0$

  fi

endfor

$(c'_1, \ldots, c'_\lambda) \leftarrow \mathsf{H}(E'_1, \ldots, E'_\lambda)$

return $(c_1, \ldots, c_\lambda) == (c'_1, \ldots, c'_\lambda)$

Figure 6.9: The protocol Sig.Vf

6.3.1 Verifiable Secret Sharing via Message Signing

In Section 6.2.9, we elaborated on how the shareholders can ensure that the secret and public key pair was generated honestly if the utilised secret sharing is not verifiable. The same process can be applied here. That is, by signing a publicly known message and later on verifying the resulting signature an appropriate number of times, the shareholders can check that their shares of the secret key were generated correctly and whether the secret key and the public key form a proper key pair.

6.3.2 Instantiations

As a practical instantiation, we propose the available parameter set for CSIDH-512 HHS from [15]. Currently no other instantiation of the presented schemes seems feasible in a practical sense. Furthermore, according to recent works [94, 21] CSIDH-512 may not reach the initially estimated security level.

6.4 Generalising the Secret Sharing Schemes

We constructed the protocols above in the context of Shamir's secret sharing protocol [101]. The key exchange mechanism in Section 6.2 as well as the signature scheme in Section 6.3 can be extended to more general secret sharing schemes. In the following, we characterise the requirements that a secret sharing scheme has to meet in order to successfully implement the key exchange mechanism and the signature scheme.

6.4.1 Compatibility Requirements

Definition 62 (Independent Reconstruction)
We say a secret sharing instance $\mathcal{S} = (S, \Gamma, G)$ is independently reconstructible *if, for any shared secret $s \in G$, any $S' \in \Gamma$ and any shareholder $P_i \in S'$, P_i's input to reconstructing s is independent of the share of each other engaged shareholder $P_j \in S'$.*

A secret sharing scheme compatible with our key exchange mechanism and signature scheme has to be independently reconstructible, since each shareholder's input into the threshold group action is hidden from every other party by virtue of the GAIP.

Definition 63 (Self-contained reconstruction)
An instance $\mathcal{S} = (S, \Gamma, G)$ of a secret sharing scheme is called self-contained *if, for any authorised set S', the input of any shareholder $P_i \in S'$ in an execution of* Reconstruct *is an element of G.*

It is necessary that $G = \mathbb{Z}_p$ for some prime p holds to enable the mapping $\cdot \mapsto [\cdot]$. This requirement may be loosened by replacing $\cdot \mapsto [\cdot]$ appropriately. To enable two-level sharing, it has to hold that for a share

$s_i \in \mathcal{S}.\mathsf{Share}(s)$ of a secret s, $s_i \in G$ holds. The secret sharing scheme also has to allow for a PVP scheme that is compatible with a zero-knowledge proof for the GAIP.

6.4.2 Examples of Secret Sharing Schemes

The following examples of compatible and incompatible secret sharing schemes illustrate the potential and limits of our key exchange mechanism and the signature scheme derived from it.

- It is evident that Shamir's approach fulfills all aforementioned requirements. In fact, the two-level sharing and the threshold PVP have been tailored to Shamir's polynomial based secret sharing approach.
- Tassa [107] extended Shamir's approach of threshold secret sharing to a hierarchical access structure. To share a secret $s \in \mathbb{Z}_p$ with prime p, a polynomial f with constant term s is sampled. Shareholders of the top level of the hierarchy are assigned interpolation points of f as in Shamir's scheme. The k-th level of the hierarchy receives interpolation points of the $k-1$st derivative of f. The shares in Tassa's scheme are elements of $\mathbb{Z}_p$ themselves. The key generation protocol (Figure 6.1) can easily be transferred to this setting, as each shareholder receives a description of the polynomial utilised in sharing his share. Hence all derivatives and their respective interpolation points can easily be computed. Reconstructing a shared secret is achieved via Birkhoff interpolation, the execution of which is independent and self-contained. The zero-knowledge proof (Figure 2.17 and Figure 2.18) as well as the piecewise verifiable proof (Figure 6.4 and Figure 6.5) thus directly transfer to Tassa's approach utilising the appropriate derivatives in the verifying protocols. The decapsulation and the signing protocols hence can be executed with adjustments only to the verifying steps.
- In 2006, Damgard and Thorbek proposed a linear integer secret sharing scheme [44] with secret space $\mathbb{Z}$. Given an access structure Γ, a matrix M is generated in which each shareholder is assigned a column so that iff $S' \in \Gamma$, the submatrix $M_{S'}$ has full rank. A secret s is shared by multiplying a random vector v with first entry s with M and sending the resulting vector entries to the respective shareholders. Reconstruction follows intuitively. Their scheme hence further generalises Tassa's with respect to secret space and feasible access structures. With the secret space $\mathbb{Z}$ their approach is not compatible with the mapping $\cdot \mapsto [\cdot]$ and our PVP scheme. Thus, neither our key exchange mechanism nor our signature scheme can in its current form be instantiated with Damgard's and Thorbek's scheme.
- Additive secret sharing is the simplest of all secret sharing schemes. For a given secret s, each shareholder P_i receives a share s_i with $s = \sum_{P_i} s_i$, $i = 1, \ldots, n$. Additive secret sharing has self-contained as well as independent reconstruction. Yet it is a full threshold secret sharing scheme, that is, $\Gamma = \{S\} = \{\{P_1, \ldots, P_n\}\}$. Thus, for any $P_i \in S$, the remaining shareholders $\{P_1, \ldots, P_n\} \setminus \{P_i\}$ form an unauthorised set. Active security therefore cannot be provided for the threshold group action. This renders additive secret sharing incompatible with our key exchange mechanism and signature scheme.

7 Integrating ELSA into CogniCrypt

In this chapter we briefly summarise the contributions made to the integration of ELSA into CogniCrypt during the supervision of the bachelor thesis of Julius Hardt [67].

7.1 Motivation

Implementing cryptographic code for a real world deployment can be considered one the hardest challenges a programmer can face. The list of potential pitfalls includes implementations errors, race conditions and side-channel attacks, to name a few examples. It is therefore generally recommended to use existing libraries and implementations instead of writing cryptographic code oneself to avoid insecurities and potential data loss or worse: leaks. The challenge hardens dramatically if the code, that is to be integrated into a project, contains not just an isolated primitive but a combination of cryptographic schemes interacting with each other. The danger of potentially catastrophic implementation errors becomes all the more imminent. If, on top of that, the implementation is not run on a single local computing node but in a distributed system such as ELSA or MCELSA, further complications are unavoidable.

It is for this reason, that we integrated ELSA into CogniCrypt [75].

7.1.1 CogniCrypt

CogniCrypt is an extension of the integrated development environment Eclipse, that is designed to assist a developer in integrating cryptographic code into projects in a manner that ensures correct and therefore safe execution (in a cryptographic sense). It does so by providing essentially two functionalities to a developer: a code generator and a static code analyser.

A developer may utilise the code generator to select a functionality from a list of already implemented cryptographic primitives and solutions such as block ciphers, signature schemes or message authentication codes. For the selected functionality, he then gives a set of parameters that more concretely define the context in which it is to be deployed. The validity of the combination of parameters, that the developer chooses, is verified via a set of rules, the so-called "CrySL rules". These CrySL rules give predefined boundaries, in which the parameters ought to be set in order to provide a secure instantiation of the selected functionality. Once the parameters are successfully verified, CogniCrypt generates a piece of code, that represents an implementation of the functionality according to the chosen parameters. This can then be utilised by the developer in the desired project. CogniCrypt furthermore will alert the developer if the code is used in an unsafe manner and provides corrections where possible.

The static code analyser, that CogniCrypt provides, is designed to ensure the correct and secure use of cryptographic APIs. That is, when a developer saves a piece of code in the IDE Eclipse, the static code analyser is triggered. It checks whether cryptographic APIs, that appear in the code, are used correctly. If not, a warning to the developer is issued along with a suggestion how correct use can be restored.

7.1.2 Integrating ELSA into CogniCrypt

ELSA itself is not a cryptographic primitive, but a storage architecture that combines several primitives to provide a client with the ability to store datasets, maintain their integrity measures, retrieve stored data and verify the correctness thereof. The primitives, that an instance of ELSA employs, fall into the following categories:

- Secret sharing schemes: The documents and decommitment values attributed to them are stored in an information-theoretically secure secret sharing instance. The secret sharing instance may be swapped during the runtime of an instance of ELSA to adapt to external circumstances.

- Signature schemes: Upon storing a set of documents, a client provides a digital signature for each document. The authenticity of a retrieved document can thereby be verified upon retrieval.

- Vector commitment schemes: Each document stored in an instance of ELSA periodically has a vector commitment generated on it. This is part of the recursive proof of integrity, that ELSA maintains on any document stored in it.

- Timestamp schemes: The evidence service computes a timestamp on each commitment value to be added to a proof of integrity so that its existence at the time of receival can be proven.

- Keyed hash functions: The vector commitment schemes employed in ELSA use keyed hash functions that are ε-extractable binding to compute a Merkle hash tree so that the vector commitments are ε'-statistically hiding under selective opening. Thereby the integrity of a single document can be verified without compromising other stored documents.

- Commitment schemes form an integral part of the vector commitment schemes to ascertain the ε'-statistical hiding property of the vector commitment schemes.

For each of those primitives, a CrySL rule can be implemented in order to ensure, that it is securely instantiated and used. Yet for an architecture like ELSA, in which these schemes interact, this task is considerably more complicated. This is especially true if a scheme, that in its protocols uses other primitives, requires those primitives to have certain security properties to be considered secure itself.

Example 64
In the protocol ELSA.Store *(similar to* MCELSA.Store *in Figure 3.5),* ELSA *uses a vector commitment scheme* VC′ *(Figure 2.5). This vector commitment scheme is assumed to be ε'-extractable binding. The vector commitment scheme* VC′ *in turn employs a vector commitment scheme* VC *and a commitment scheme* CS. *We consult Theorem 20 and see that, for* VC′ *to be extractable binding, it is a sufficient condition to have* VC *and* CS *be ε-extractable binding.*

The vector commitment scheme VC *in turn needs the commitment scheme* CS *to be ε-statistically hiding to achieve ε-statistical hidingness itself, as Theorem 18 shows.*

The relations, that we illustrate in Example 64, cannot be reflected in CogniCrypt in its current state. This shows just one of the challenges, that we face in integrating ELSA into CogniCrypt.

7.1.3 Goals of our Integration

We want to provide a developer, that uses our extension to CogniCrypt, with the following capabilities:

1. We implement the tool ComposableCrypto. It provides the following functionalities to a developer. A developer can select a security solution, that has been integrated into CogniCrypt, and configure it so that it provides the intended functionalities and security properties. The code for this configuration can then be generated and included in an existing code project so that the security solution can be utilised in said project. The correct use of the protocols, that the solution provides, is ensured to avoid improper execution and with that potential security risks. The security solution may be executed on a single computing node or represent a distributed system, in which multiple computing nodes interact with each other. Individual schemes in a security solution can dynamically be replaced to ensure that all schemes in an instantiation adhere to the most recent recommendations in terms of implementation and parametrisation.

2. Not only can a developer select existing security solutions to integrate them into his code projects, but he may also introduce novel solutions to ComposableCrypto. This enables a developer to utilise the features that CogniCrypt provides for security solutions with respect to correct execution of a solution's protocols and algorithms, automatic code generation and competent choice of parameters as well as the possibility of updating individual components if necessary.

3. New cryptographic schemes can easily be introduced to the catalogue of implementations that CogniCrypt provides. Thereby recent updates to implementations of specific schemes can easily be integrated and altogether new schemes can be added. If a security solution, that a developer integrated into ComposableCrypto, necessitates a scheme that is not yet provided by CogniCrypt a developer can hence directly provide an implementation that can be used for code generation.

7.1.4 Challenges

From the goals stated in Section 7.1.3, the following challenges emerge.

1. Configuring an instance of a security solution and subsequently generating the code for its implementation has to be executed in such a fashion, that not only a functional, but also a secure implementation is achieved. This holds especially true for distributed systems such as ELSA, in which more than one computing node has to operate concurrently. In CogniCrypt, the correct order of execution for a scheme or – more generally – a set of protocols is ensured by a set of so-called CrySL rules. These can be represented as a state machine, that indicates a wrongful application.

   For the benchmark implementation of ELSA, REST APIs were used for the communication between the individual computing nodes. This approach does not reflect any state in the query the parties make to each other, thus the interaction between the nodes is inherently stateless. Yet CogniCrypt's approach for verifying a correct execution of a set of protocols is genuinely stateful. We will have to bridge this gap in our extension ComposableCrypto.

   Moreover, for a distributed system such as ELSA, the code for each computing node has to be generated. CogniCrypt is not yet equipped for this task, we thus have to devise a method that generates code for multiple computing nodes rather than a singular one. Again, to ensure correct execution, a set of CrySL rules for each node has to be constructed.

2. Integrating a new security solution into ComposableCrypto has to be as efficient and as frictionless as possible to a developer. It is therefore vital to provide a tool that enables a developer to state the intended purpose, general layout, i.e., the computing nodes and their interaction among each other, and the cryptographic properties, that are necessary for correct and secure instantiation of a solution to be integrated. Providing an easy-to-use interface for the integration of new implementations of cryptographic schemes along with their security properties and dependencies must also be accomplished.

3. Along with the integration of new cryptographic implementations comes the task of enabling a developer to update existing configurations in order to maintain security and functionality in the face of cryptoanalytical developments and ever increasing computing power. This should be made possible without rebuilding an existing instantiation in its entirety.

We will not give a detailed discussion of the inner workings of our extension ComposableCrypto of CogniCrypt, but rather the usage by a developer, who wishes to integrate a (possibly distributed) security solution into a code project. Where it is appropriate, we detail the challenges we faced in accomplishing the tasks we set out to do.

7.2 Terminology

Let us first clarify the terminology that we will use in this chapter.

- An architecture such as ELSA or MCELSA employs several cryptographic schemes to provide its functionalities to its clients. We call a set of protocols that implement a given scheme a *cryptographic component* or *building block*. Such a cryptographic component may depend on and make use of other components to fulfill the tasks it sets out to. We call a cryptographic component that does not depend on other components a *cryptographic primitive*.

  In Example 64, we illustrated the example of the vector commitment scheme employed in ELSA and MCELSA, respectively. This component depends on a keyed hash function and a commitment scheme. It therefore does not constitute a cryptographic primitive. A hash function like SHA3-256 [53] on the other hand is a cryptographic primitive.

  The security of a cryptographic building block and its respective implementation depends on certain security properties of its subcomponents if it is not a primitive itself as can be seen in Example 64. We shall represent this relation in our extension of CogniCrypt.

- We represent the combination of a cryptographic component and the subcomponents, that it uses for its implementation, in ComposableCrypto as a tree. As can be seen at the example of ELSA, a security solution may depend on more than one initial component to achieve its functionality. This naturally leads to more than one tree of cryptographic components, that are to be configured separately. As is tradition in the literature, we call the collection of trees in a security solution a *forrest*.

- The communication between computing nodes and – more general individual pieces of code – is executed via application programming interfaces (APIs). A special subcategory of APIs are so-called *representational state transfer APIs* (REST APIs). This type of API processes the communication in distributed systems, that is, architectures, in which two or more distinct computing nodes jointly fulfill a computational task, and adhere to a set of guidelines that ensure interoperability.

  The most important characteristics, to which a REST API has to adhere, are as follows.

  - REST APIs are stateless. That is, each request drawn from a REST APIs must be interpretable and executeable without regard of previous queries, that were posed in the same session. The statelessness makes integrating REST APIs into CogniCrypt all the more challenging, since its code verifier is an inherently stateful tool.
  - Queries in a REST API are one-directional. This means that there is always a client posing a query and a server receiving it and if appropriate answering it.
  - A REST API is agile in the sense that the functionalities it provides to a client can be amended and extended, while the respective server is active. The system or parts of it hence do not have to be powered down or taken offline to update a REST API.

- The REST APIs that we use for the communication between computing nodes in a distributed system and for calling individual cryptographic components will be specified using the OpenAPI standard. OpenAPI is a language standard that is used to specify the layout of a REST API. It includes the addressees for individual requests, the necessary resources, what responses are to be expected as well as the data with which a request has to be parametrised.

  An OpenAPI specification itself is most commonly held in JSON or YAML. There exists a wide variety of tools to generate server and client stubs for APIs that are specified in an OpenAPI style, in particular for RESTful APIs.

- The correct use of a cryptographic implementation is crucial for its secure deployment in a code project. CogniCrypt ensures that a developer does not misuse the API presented to him by setting a set of so-called *CrySL rules*, that describe correct behaviour for each implementation of a cryptographic component. We give an example in Figure 7.1. It illustrates the correct use of the storage protocol Store in ELSA. A CrySL rule consists of the mandatory blocks SPECS, OBJECTS, EVENTS. Other blocks may appear in a CrySL rule such as FORBIDDEN, ORDER, ENSURES, NEGATES, CONSTRAINTS and REQUIRES. We discuss the mandatory blocks at the example of Figure 7.1[1] and omit the optional blocks. An in depth analysis of the structure of a CrySL rule can be found in [76]. A CrySL rule specifies the the class to which it applies in SPEC. In this instance, this is the class "ELSAStoreOperation". The objects, that are to be used as parameters or return values for the methods of the class specified in SPEC, are listed in OBJECTS. For ELSA's store operation, these are an ELSAClient object client, an inputstream inputStream and a universally unique identifier uuid. The methods, that contribute to the successful integration of a cryptographic implementation, and their parameters are given in EVENTS. In our example these are getInstance, addFile and store. Each method is given a label (Get, Add, Str). Multiple methods or labels thereof can be combined into method patterns with a new label for later usage. These methods should be executed in the correct order and number of occurence. This is specified in the block ORDER. It is written in the style of a regular expression. The modelling for a correct execution hence is quite expressive.

7.3 Integrating a Security Solution into a Code Project

We demonstrate the usage of ComposableCrypto, our extension of CogniCrypt in the IDE Eclipse, at the example of ELSA.

7.3.1 Configuring Cryptographic Components

The developer selects the desired security solution – in this case ELSA – from the list of solutions, that have been integrated into ComposableCrypto. He or she is presented with a set of root cryptographic components, that are necessary for the successful implementation of the security solution. In the case of ELSA, these are a secret sharing scheme, a channel protocol, a signature scheme, a timestamp scheme

[1] This example is taken from https://github.com/juliushardt/ComposableCrypto/blob/9a3b3f14f5dc1b7447d48d284b0c67b32a9b098b/CrySLRules/ELSAStoreOperation.crysl

```
SPEC de.tu_darmstadt.crossing.composable_crypto.components.custom.
    long_term_storage.ELSAStoreOperation

OBJECTS
    de.tu_darmstadt.crossing.composable_crypto.components.custom.
        long_term_storage.ELSAClient client;
    java.io.InputStream inputStream;
    java.util.UUID uuid;

EVENTS
    Get: getInstance(client);
    Add: uuid = addFile(inputStream);
    Str: store();

ORDER
    Get, Add, Str
```

Figure 7.1: CrySL rules for ELSA.Store

and a vector commitment scheme. For their respective application and use by the parties engaged in an instantiation of ELSA, we refer to Section 3.

For each of these root nodes, a concrete scheme has to be selected. If a scheme, that the developer chooses, necessitates further cryptographpic components, i.e., it is not a primitive, the respective branches are then presented to the developer. This process recursively continues, until only cryptographic primitives are selected as a last step. We illustrate the configuration of a non-interactive timestamp scheme as is used in ELSA in Figure 7.2[2]. The developer has to select a hash function and a signature scheme as children of the timestamp scheme. In this case, SHA-256 for the hash function and an RSA signature scheme are chosen. The RSA signature in turn also makes use of a hash function, for which in this case again SHA-256 is selected.

The components have to be selected in such a manner that the resulting implementation forms a correct and secure instantiation of the desired security solution. We discuss, how this is ensured for the final component selection in Section 7.3.3.

7.3.2 Automated Tree Building

A developer that uses CogniCrypt cannot necessarily be assumed to have in-depth knowledge of cryptographic protocols and their respective security properties. We therefore implemented BuildCompliantSolution, a utility that automatically generates a valid forrest of cryptographic components given that the roots have already been defined. This is the case for any security solution integrated in ComposableCrypto.

BuildCompliantSolution takes said roots and computes the set of all possible configurations – under consideration of certain appropriate heuristic boundaries – that arise from the root cryptographic components received as input. The algorithm stops, as soon as a functional configuration is found. The tool BuildCompliantSolution will also complete forrests that have been partially filled by a developer. For that, the prefilled structure is set as fixed and the remaining components are completed accordingly. If no fitting configuration can be found based on the given preset, an error message is shown.

If individual cryptographic components have to be exchanged or updated due to – for example – changing cryptograhpic hardness assumptions or improved implementations thereof, BuildCompliantSolution can adapt existing forrests so that a functional configuration can be achieved and only a minimal amount of implementation components have to be updated. In this case, BuildCompliantSolution deconstructs the existing forrest so that it does not contain any components that are to be exchanged and rebuilds it as in the case of the partially configured forrest to achieve a secure implementation of the security solution.

[2]This graphic was generated utilising the ComposableCrypto main application to be found at `https://github.com/juliushardt/composablecrypto`.

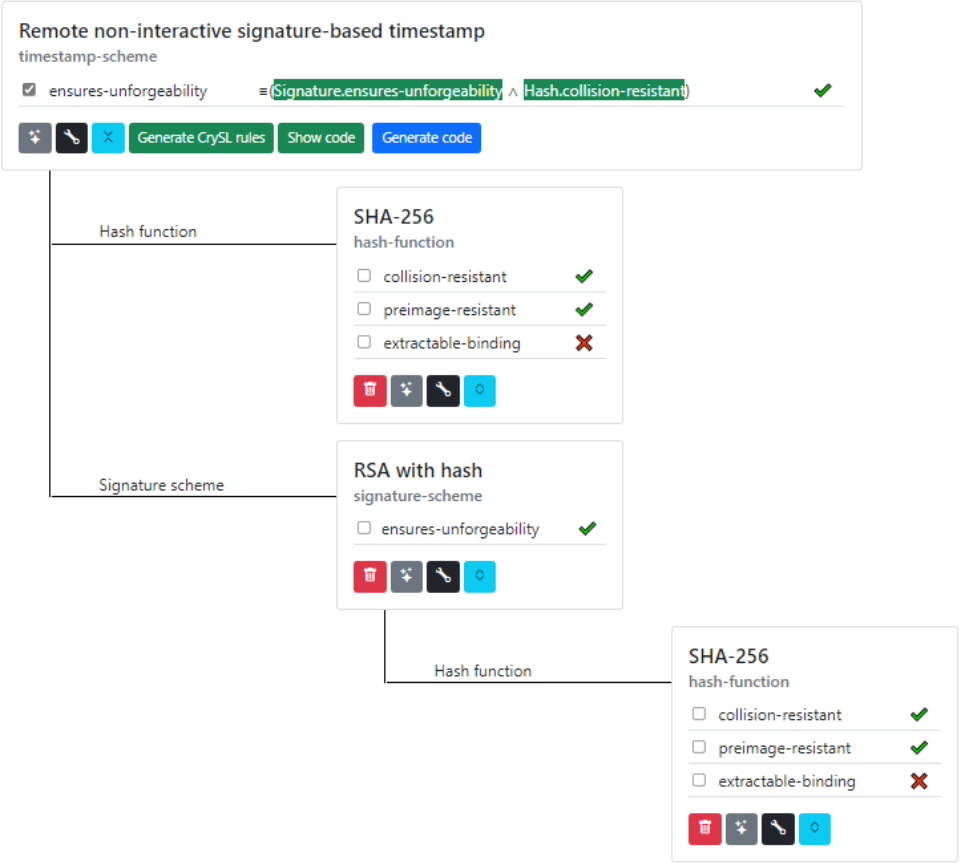

Figure 7.2: A tree of cryptographic components

7.3.3 Verifying the Correctness of the Deployment

The procedures as described in Section 7.3.1 and Section 7.3.2, respectively, let a developer combine cryptographic components or respectively help the developer in configuring a combination of components so that the resulting configuration provides the functionalities, that are necessary for the security solution to be integrated into a code project.

This process does, however, not ensure that the security properties of the selected components form a secure implementation. The security of the selection therefore has to be verified explicitly. Let us first consider a singular cryptographic component.

Example 65
Consider the signature scheme we presented in Section 6.3. This scheme is secure, i.e., simulatable and existentially unforgeable under the following assumptions:

- *In the hard homogeneous space $(\mathcal{E}, \mathcal{G})$, the group action inverse problem (Figure 2.11a) is hard.*
- *The instance of the secret sharing scheme, that holds the secret key, is information-theoretically hiding.*
- *The hash function, that is utilised in computing the challenge bits, is preimage resistant.*
- *The Fiat-Shamir-transform produces an existentially unforgeable signature scheme from a secure identification scheme.*

We categorise the security properties of a scheme into two types: (1) dependent on exterior factors, that is, the properties of subschemes employed by the scheme or cryptographic hardness assumptions determine whether a given property holds or (2) fixed, that is, a property holds independent of exterior influences and thereby holds true irrespectively of properties of subschemes or hardness assumptions. For cryptographic primitives only hardness assumptions can influence whether a property holds as a primitive does not make use of other schemes.

The approach for verifying that a secure selection has been made is therefore a reversal of the initial tree building approach. That is, we go from bottom up, i.e., from the leaves to the roots for each tree in the configured forrest and recursively verify the security properties. ComposableCrypto is initially handed a list of cryptographic hardness assumptions, that are considered to hold. If a developer is uncertain, which assumptions are state of the art, we provide a list of reasonable assumptions, from which a developer can choose. These range from traditional hardness assumptions like the Diffie-Hellman-problem or the RSA-problem to more recent developments like the group action inverse problem in hard homogeneous spaces.

The verification for a leaf in a configuration is therefore to check the validity of the individual properties against the hardness assumptions, that ComposableCrypto received. After all leaves have been processed, their respective parent nodes are verified, where the properties of their children are taken into consideration. This process is recursively continued until all roots of the forrest are verified. If the security properties, that the security solution necessitates, are fulfilled, the configuration is considered secure. A graphic example of this evaluation can be seen in Figure 7.2.

We point out that a cryptographic component may have security properties, that are irrelevant to the concrete instantiation and can hence be dismissed in computing the security of the roots. Example 65 shows that the concrete access structure implemented by the secret sharing scheme, in which the secret key is stored, does not affect the security properties of the signature scheme. Thus not all security properties of each component have to hold in order for the configuration to yield a secure instantiation of the security solution.

Limits to our Model

Our model verifies that each cryptographic component is instantiated in a secure fashion. Yet we concede that this approach does not consider the different attacker models, that the security proofs for individual components are based on.

It does especially not take practical threats such as side-channel attacks or race conditions into account. Furthermore, even if the attacker models for the security proofs for all selected cryptographic components coincide, it can not be guaranteed that the combination of those instantiations represents a secure solution, since in many cases the security proofs do not include running more than one instance of the scheme in parallel or multiple instances sequentially. It seems intuitive that, if all components can be proven secure in the "universal composability" framework, a secure instantiation of the entire security solution can be

guaranteed. This, however, seems improbable, since impossibility results for particular security proofs in several composability frameworks [72] have been found.

Thus we cannot guarantee the security of a solution consisting of several cryptographic components. ComposableCrypto can, however, detect constructions that would obviously be insecure and alarm a developer to that circumstance. A rigorous security proof can at this time not be generated in our model.

7.3.4 Generating Code

We now illustrate, how deployable code is generated from a forrest, that has been configured with the help of and validated by ComposableCrypto. To accomplish this, we make use of the functionalities, that CogniCrypt provides.

Let us assume that a security solution has been fully configured and validated. That is, the leaves in each tree stemming from a root cryptographic component are cryptographic primitives and the configuration has been validated by ComposableCrypto. This means that the necessary conditions for the security properties of each cryptographic component in the configuration of the security solution are fulfilled. The code for each computing node in a distributed system is generated in three steps:

1. We state a CrySL rule for each cryptographic component and each computing node in the security solution. These can be derived from the concrete choice of cryptographic component and the description of the security solution.

2. We generate the construction code for each component using the code generator that CogniCrypt provides. Furthermore, we generate a REST API according to the OpenAPI standard for each component by applying an API generator as can be found in [36]. These are then connected so that the individual components can interact with each other and thereby achieve their functionalities.

3. The CrySL rules that apply to the root nodes are in a final step combined with those, that are given for the security solution overall to ensure a safe execution of the protocols, that are provided to the developer.

In Figure 7.3 we give a redacted example of a piece of code generated by CogniCrypt that implements a signature scheme defined by three protocols `getkey`, `sign` and `vfy`. We leave out some package imports, since they do not provide further insight.

7.3.5 Distributed Security Solutions

The security solution that the developer wishes to integrate into his project does not have to consist of a single computing node, but may include two or more nodes interacting with each other. This is perfectly illustrated at the example of ELSA, where a client, the evidence service and the shareholders and – depending on the concrete instantiation – the timestamp service combined form a functioning system. This means that the code for each individual computing node has to be generated for the system to properly execute its task.

We solve this challenge by generating the same construction code for each node, that a security solution includes. Yet in a second step, we configure the parameters for each node according to the specifications given by the security solution. This enables us to tailor the instantiation of each computing node to its specific needs.

Since we base the interaction between engaged parties on REST APIs, the stubs for these also have to be generated and then connected to the remaining code. Again, we apply an API generator to obtain the REST API stubs. In this step we again distinguish between the computing nodes and their tasks within the system. That is, we only generate the necessary stubs for a node and purposefully ignore all unnecessary stubs. This distinction can be demonstrated at the usage of the timestamp service in ELSA. The timestamp service provides two APIs to external parties: a stamping API, that returns a timestamp on a message it is queried on, and a verification API, that checks the validity of a timestamp with respect to a given document. The evidence service only has access to the stamping functionality, while the client only needs the verifying functionality to check the integrity of a stored document. We hence only generate the stamping stub for the evidence service and the verifying stub for the client.

The developer only has to manually interfere in this last step: The REST API stubs have to be linked to the remaining code of each node. If we take a look at the protocol ELSA.Store, we see that several subprotocols of cryptographic components are used in its execution, these have to be connected to the API stubs manually. It was not feasible to automate this task in a safe manner, as the interaction of the

```
/*********************************************************
 Copyright (c) 2015-2019 TU Darmstadt, Paderborn University
*********************************************************/

import de.cognicrypt.codegenerator.crysl.CrySLCodeGenerator;

public class SecureSigner {
        public static java.security.KeyPair getKey() throws
            NoSuchAlgorithmException {
                java.security.KeyPair pair = null;
                CrySLCodeGenerator.getInstance().includeClass("java
                    .security.KeyPairGenerator").addReturnObject(
                    pair).generate();
                return pair;
        }

        public static java.lang.String sign(java.lang.String msg,
            java.security.KeyPair keyPair) throws
            NoSuchAlgorithmException, InvalidKeyException,
            SignatureException {
                byte[] msgBytes = msg.getBytes(StandardCharsets.
                    UTF_8);
                byte[] res = null;
                java.security.PrivateKey privKey = keyPair.
                    getPrivate();
                CrySLCodeGenerator.getInstance().includeClass("java
                    .security.Signature").addParameter(privKey, "
                    priv").addParameter(msgBytes, "inpba").
                    addReturnObject(res).generate();
                return Base64.getEncoder().encodeToString(res);
        }

        public static boolean vfy(java.lang.String msg, java.
            security.KeyPair keyPair) throws
            NoSuchAlgorithmException, InvalidKeyException,
            SignatureException {
                boolean res = false;
                byte[] msgBytes = Base64.getDecoder().decode(msg);
                java.security.PublicKey pubKey = keyPair.getPublic
                    ();
                CrySLCodeGenerator.getInstance().includeClass("java
                    .security.Signature").addParameter(pubKey, "pub
                    ").addParameter(msgBytes, "sign").
                    addReturnObject(res).generate();
                return res;
        }

}
```

Figure 7.3: Code for a signature scheme generated by CogniCrypt

cryptographic root components relies entirely on the protocols that the security solution is defined by. We shall demonstrate below, why the format in which a security solution is represented in ComposableCrypto does not suffice for automating this task.

7.3.6 Challenges

We briefly discuss the challenges that we faced in constructing the extension ComposableCrypto of CogniCrypt.

The main challenge in implementing the tool ComposableCrypto was unifying the stateless approach of REST APIs with the inherently stateful approach of CogniCrypt and its application of the CrySL rules to ensure a safe execution of the generated code. We overcame this challenge by building a set of CrySL rules that are applied to a computing node in a bottom-up approach, finishing with the rules describing the correct behaviour of the security solution at the top level. The resulting set of CrySL rules capture the correct execution of the security solution as well as the correct usage of all components, that the implementation includes. The calls to individual REST APIs are thereby incorporated and CogniCrypt can ensure a safe state of the security solution.

Breaking the implementation of the security solution down into cryptographic components each representing a single scheme enabled us to efficiently achieve the second challenge, that is, swapping out individual cryptographic schemes to maintain the security of an instantiation. This approach also furthered the capabilities of BuildCompliantSolution in that it is able to complete unfinished configurations or update an existing configuration in which specific components have to replaced.

The last challenge was to verify the security of the selected configuration. The forrest based representation of the configuration of an entire security solution combined with the bottom-up approach of the API generation enables us to build an efficient verifying tool to check whether the security properties, that are necessary for the security solution, are fulfilled. For that, we attributed each cryptographic component with a set of properties, the validity of which is verified depending on general cryptographic hardness assumptions, the security properties of its children in the configuration or is stated in a binary fashion irrespective of the former. We can thus – again in a bottom-up approach – recursively verify the security and indicate the result to the developer.

7.4 Integrating new Security Solutions

We discuss how a developer can integrate his own security solutions into ComposableCrypto, so that other developers can integrate them in their own projects or it can be reused later on.

Consider for that the set of all primary cryptographic schemes, that are used in the protocols of a security solution. That is, all schemes, that are explicitly called to in the execution of the protocols. These schemes form the roots of the forrest of components to be configured. In ELSA these are:

- a signature scheme Sig to ensure the integrity and authenticity of stored documents,
- a vector commitment scheme VC to be applied to all stored documents to provide a succinct integrity measure, that is regularly updated,
- a secret sharing scheme $\mathcal{S}$ to store the documents and decommitment values attributed to them in an information-theoretically hiding fashion,
- a timestamp scheme TS to be applied by the evidence service in maintaining the proofs of integrity,
- a channel protocol Ch to provide secure communication between all engaged parties.

The principal components of a security solution are collected in a JSON file, that can be imported into ComposableCrypto to integrate said solution.

A developer, who wants to integrate his own security solution into ComposableCrypto and thereby CogniCrypt, has to provide a set of CrySL rules. These define an acceptable execution and order of the protocols, that a security solution is defined by. The syntax for these rules follows that of the standard CrySL rules, that are used to ensure the correct application of the cryptographic components. These rules will – as we discussed in Section 7.3 – later on be combined with those, that arise from the configuration of the cryptographic components, when the imported security solution is integrated into a code project. The CrySL rules for the specific components should not be provided by the developer as they depend on the concrete schemes, that are selected during the configuration step.

Integrating a security solution into ComposableCrypto is hence a fairly straightforward task.

7.5 Importing Cryptographic Components and Implementations

A developer may want to introduce new cryptographic components or schemes into ComposableCrypto. This may be due to a security solution using a type of cryptographic scheme, that is not yet represented in the implementations that CogniCrypt provides, or a new, more secure implementation of an existing scheme becoming available.

Cryptographic components in ComposableCrypto are represented in a JSON file, that contains the type of scheme it represents (e.g., a signature scheme or a hash function), the handle, the name with which it is displayed, the Java class in which the implementation can be found and the security properties. For a signature scheme this may include ε-existential unforgeability (see Section 2.2.5), or for a hash function ε-extractable bindingness (see Section 2.2.2). If the cryptographic component is not a primitive, that is, it uses other components to achieve its functionalities, a list of children with according type must be given. The security properties of a cryptographic component can be stated in two flavours: (1) boolean, that is, they are either fulfilled or not fulfilled, or (2) dependent on external conditions, that is, cryptographic hardness assumptions or properties of its subcomponents determine whether a given security property holds true.

Importing new or updated cryptographic components is thus an easy task for a developer, since he has to provide the JSON file, that classifies the respective component, and the Java class, in which the implementation is to be found.

8 Conclusions and Outlook

In research question 1, we raised the question on how the three aspects of data security, that is, confidentiality, integrity and authenticity, can be guaranteed in a long-term storage architecture so that multiple clients can interact with the same instance simultaneously.

In research question 2, we asked how secure, private channels between two parties, that share a common source of symmetric bits, can be achieved against an adverary that grows in computational power and cryptoanalytical capabilities over time. This ties into research question 1, since secure and private communication is a central concern in long-term storage architectures.

Research question 3 raised the challenge whether the preprocessing phase of secret sharing based MPC protocols can be expedited without disproportionate additional effort while maintaining the security guarantees, that the MPC protocol provides. This was motivated by most performance improvements to secret sharing based MPC protocols coming with a benefit to the online phase and neglecting the preprocessing phase or worse shifting workload from the online to the preprocessing phase.

With research question 4 we inquired how the key exchange mechanism proposed by Meyer and De Feo [45] can be extended so that it achieves active security. This key exchange mechanism is set in the context of a hard homogeneous space with the secret key being shared in a Shamir sharing scheme. We furthermore asked whether the field of applicable secret sharing schemes can be widened.

Research question 5 asked how a complex security solution such as ELSA or MCELSA can be integrated into CogniCrypt so that developers can integrate it into a code project and that the resulting implementation represents a secure instantiation of said security solution.

We addressed research question 1 in Section 3 and presented MCELSA, a long-term storage architecture that provides unconditional confidentiality and prolonged integrity as well as authenticity to documents stored within an instance of it. MCELSA constitutes an evolution of ELSA in that an instance of MCELSA serves multiple clients simultaneously, whereas ELSA can handle but one client.

The two main challenges in transferring ELSA to a multi-client setting were establishing a method of distributed access management and the distribution of the maintenance tasks, that are necessary to uphold the security of the stored documents. The first challenge was mastered by tasking the shareholders, that store the shares of the documents and the accompanying decommitment values, with determining individual access privileges. Thus a client must be deemed authorised for his query by an authorised set of shareholders for his query to be successfully executed. We overcame the second challenge, that is, distribution of maintenance tasks, by having the shareholders determine a minimal list of clients that assist in renewing the vector commitments placed on stored documents. This keeps the number of vector commitments to be maintained as low as possible and does not overburden individual clients.

The optimisations that we applied are reflected in the performance benchmarks, that we gave. They show that MCELSA outperforms ELSA in terms of storage demand as well as computational effort for the engaged parties. We tested several scenarios to demonstrate the real world applicability of MCELSA.

In Section 4, we answered research question 2 in that we adopted the two-stage adversary of [17] to account for unbounded adversarial resources and modelled channel security following the common notions for the computational setting like [12, 11].

We considered atomic channel protocols in which it is assumed that a transmitted ciphertext is fully received on the other side. Depending on the network, however, ciphertexts may be fragmented. It has been shown in attacks on actual channel protocols like SSH and IPSec [3, 47] that this fragmentation behavior could potentially be exploited. A more formal treatment of ciphertext fragmentation can be found in [20, 2]. One can also consider, on top, the possibility that the channel protocol itself may distribute input messages arbitrarily over ciphertexts, leading to the notion of stream-based channels [57]. It would be interesting to see how the requirement of unconditional security affects such models.

We followed earlier work and used a game-based definition for the security of channels, where keying material is provided by external means. If one now uses, say, a secure QKD protocol to generate the keys, then it remains yet to prove formally that the combined protocol is secure (albeit no attack on the joint execution is obvious). This is called compositional security. In stronger, simulation-based notions for key exchange and channels such as [32, 46] compositional guarantees usually follow immediately. Compositional security for game-based notions of key exchange, as here, have been discussed in [27]. Again, both types, simulation-based and game-based models, usually only consider computationally bounded adversaries, leaving open the question if they still hold in the information-theoretic setting.

We answered research question 3 in the positive in Section 5. For that, we took an existing MPC protocol and introduced an additional helper party P_h. The helper party takes on executing the preprocessing for the parties in the original protocol. To do so, P_h receives a list of parties and a description of the circuit, that is to be evaluated in the online phase. P_h then samples the auxiliary data for the online phase and shares each item according to the secret sharing scheme, that the MPC protocol is based on, among the original parties.

We proved that the resulting protocol with P_h executing the preprocessing phase on behalf of the parties engaged in the original protocol, possesses the same security guarantees as the original protocol. Thus the introduction of P_h does not impact the security of the MPC protocol.

The helper party P_h can be instantiated in one of two fashions, that is, a trusted helper party or an untrusted one. This enables the parties to execute the online phase with the auxiliary data as it was received from P_h or to first apply the cut-and-choose approach to verify that it was generated honestly.

To demonstrate that our proposed approach is not of a purely theoretical nature, we gave three feasible instantiations, that can be deployed in a real world setting. We implemented an instance of P_h in the context of the secret sharing based MPC protocol SPDZ [43] and tested the performance in two scenarios, i.e., Beaver triples generated and shared per second and the time elapsed for preprocessing of a Vickrey auction with up to 50 parties bidding in it. The results achieved by our implementation significantly improved upon those of the SPDZ protocol and evolutions thereof, i.e., MASCOT [70] and Overdrive [71].

We addressed research question 4 in Section 6 and presented an actively secure key exchange mechanism in the context of hard homogeneous spaces, in which the secret key is being secret shared among a set of shareholders. For that, we transferred the piecewise verifiable proof and a zero-knowledge protocol for the group action inverse problem proposed in [14] to the setting of threshold secret sharing. These protocols we then applied to the passively secure KEM proposed by De Feo and Meyer [45]. We proved the simulatability of the emerging protocols under the assumption of the decisional parallelisation problem.

The KEM was then extended into a signature scheme, that is also simulatable, via the Fiat-Shamir-transformation.

We furthermore characterised the properties of a secret sharing scheme necessary to base our key exchange mechanism on it. We gave several examples of secret sharing schemes, that are compatible with our protocols, to demonstrate its capabilities and limits. We thereby showed that cryptographic schemes with secret shared private key in the setting of hard homogeneous spaces can be applied to a much wider field of secret sharing schemes than the traditional Shamir scheme.

In Section 7, we presented our integration of the long-term storage architecture ELSA into CogniCrypt thereby addressing research question 5. More generally, we discussed how CogniCrypt can enable a developer to integrate a complex security solution into a code project in a secure manner. For that, we implemented the extension ComposableCrypto of CogniCrypt. Once an existing security solution has been selected, it guides a developer through configuring the concrete parameters and cryptographic building blocks, that are to be employed in the implementation of said security solution. ComposableCrypto then verifies that the selected configuration fulfills the necessary requirements in terms of functionality and security properties of the individual schemes, that were chosen. If so, CrySL rules are automatically derived and the concrete code for the security solution is generated. We also provide the tool BuildCompliantSolution, that automatically configures the forrest of cryptographic components for a security solution, if a valid configuration exists.

A developer can furthermore integrate his own solution into ComposableCrypto by providing a list of necessary root components and CrySL rules, that describe the correct behaviour of the security solutions and valid execution patterns of its protocols.

If desired, new cryptographic components can also be introduced to our extension ComposableCrypto. For that, a description of the new component including the name, type of scheme and location of the implementation have to be provided. This way, implementations for schemes, that were previously not provided by CogniCrypt, can be made available. Components, that were already integrated, can also be updated in order to provide more secure or efficient implementations.

Bibliography

[1] Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001.

[2] Martin R. Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G. Paterson. A surfeit of SSH cipher suites. 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 1480–1491, Vienna, Austria, October 24–28, 2016. ACM Press.

[3] Martin R. Albrecht, Kenneth G. Paterson, and Gaven J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26, Oakland, CA, USA, May 17–20, 2009. IEEE Computer Society Press.

[4] Ryan Babbush, Jarrod Ryan McClean, Michael Newman, Craig Michael Gidney, Sergio Boixo, and Hartmut Neven. Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX Quantum*, 2:010103, 2021.

[5] Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro, editors, *Sequences II: Methods in Communication, Security, and Computer Science*, pages 329–334, New York, NY, 1993. Springer New York.

[6] Sebastian P. Bayerl, Tommaso Frassetto, Patrick Jauernig, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, Emmanuel Stapf, and Christian Weinert. Offline model guard: Secure and private ml on mobile devices. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 460–465, 2020.

[7] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[8] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision resistance. *Journal of Cryptology*, 28(4):844–878, October 2015.

[9] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[10] Mihir Bellare, Oded Goldreich, and Anton Mityagin. The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309, 2004. `https://eprint.iacr.org/2004/309`.

[11] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 7(2):206–241, 2004.

[12] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

[13] Thomas Beth, Hans-Joachim Knobloch, and Marcus Otten. Verifiable secret sharing for monotone access structures. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 189–194, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[14] Ward Beullens, Lucas Disson, Robi Pedersen, and Frederik Vercauteren. CSI-RAShi: distributed key generation for CSIDH. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings*, volume 12841 of *Lecture Notes in Computer Science*, pages 257–276. Springer, 2021.

[15] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

[16] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. Hybrid key encapsulation mechanisms and authenticated key exchange. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*, pages 206–226, Chongqing, China, May 8–10, 2019. Springer, Heidelberg, Germany.

[17] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. Transitioning to a quantum-resistant public key infrastructure. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 384–405, Utrecht, The Netherlands, June 26–28, 2017. Springer, Heidelberg, Germany.

[18] G. R. Blakley. Safeguarding cryptographic keys. *Proceedings of AFIPS 1979 National Computer Conference*, 48:313–317, 1979.

[19] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008: 13th European Symposium on Research in Computer Security*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206, Málaga, Spain, October 6–8, 2008. Springer, Heidelberg, Germany.

[20] Alexandra Boldyreva, Jean Paul Degabriele, Kenneth G. Paterson, and Martijn Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 682–699, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[21] Xavier Bonnetain and André Schrottenloher. Quantum security analysis of CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 493–522, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[22] Luis T A N Brandao, Michael Davidson, and Apostol Vassilev. NIST roadmap toward criteria for threshold schemes for cryptographic primitives, Jul 2020.

[23] Gilles Brassard, Claude Crépeau, Dominic Mayers, and Louis Salvail. A brief review on the impossibility of quantum bit commitment, 1997.

[24] Ferdinand Brasser, Tommaso Frassetto, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, and Christian Weinert. VoiceGuard: Secure and Private Speech Processing. In *Proc. Interspeech 2018*, pages 1303–1307, 2018.

[25] Johannes Braun, Johannes Buchmann, Ciaran Mullan, and Alex Wiesmaier. Long term confidentiality: a survey. Cryptology ePrint Archive, Report 2012/449, 2012. `https://eprint.iacr.org/2012/449`.

[26] Johannes Braun, Johannes A. Buchmann, Denise Demirel, Matthias Geihs, Mikio Fujiwara, Shiho Moriai, Masahide Sasaki, and Atsushi Waseda. LINCOS: A storage system providing long-term integrity, authenticity, and confidentiality. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *ASIACCS 17: 12th ACM Symposium on Information, Computer and Communications Security*, pages 461–468, Abu Dhabi, United Arab Emirates, April 2–6, 2017. ACM Press.

[27] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 2011: 18th Conference on Computer and Communications Security*, pages 51–62, Chicago, Illinois, USA, October 17–21, 2011. ACM Press.

[28] Johannes Buchmann, Ghada Dessouky, Tommaso Frassetto, Ágnes Kiss, Ahmad-Reza Sadeghi, Thomas Schneider, Giulia Traverso, and Shaza Zeitouni. SAFE: A secure and efficient long-term distributed storage system. Cryptology ePrint Archive, Report 2020/690, 2020. `https://eprint.iacr.org/2020/690`.

[29] Ahto Buldas, Matthias Geihs, and Johannes A. Buchmann. Long-term secure commitments via extractable-binding commitments. In Josef Pieprzyk and Suriadi Suriadi, editors, *ACISP 17: 22nd Australasian Conference on Information Security and Privacy, Part I*, volume 10342 of *Lecture Notes in Computer Science*, pages 65–81, Auckland, New Zealand, July 3–5, 2017. Springer, Heidelberg, Germany.

[30] Christian Cachin and Ueli M. Maurer. Unconditional security against memory-bounded adversaries. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 292–306, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany.

[31] Fabio Campos and Philipp Muth. On actively secure fine-grained access structures from isogeny assumptions. In Jung Hee Cheon and Thomas Johansson, editors, *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022*, pages 375–398. Springer, Heidelberg, Germany, September 28–30, 2022.

[32] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[33] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 55–72, Nara, Japan, February 26 – March 1, 2013. Springer, Heidelberg, Germany.

[34] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. ASTRA: High throughput 3PC over rings with application to secure prediction. Cryptology ePrint Archive, Report 2019/429, 2019. `https://eprint.iacr.org/2019/429`.

[35] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4PC framework for privacy preserving machine learning. In *ISOC Network and Distributed System Security Symposium – NDSS 2020*, San Diego, CA, USA, February 23–26, 2020. The Internet Society.

[36] OpenAPI Tools Contributors. Openapi generator.

[37] Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. `https://eprint.iacr.org/2006/291`.

[38] Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: Threshold post-quantum signatures. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *Lecture Notes in Computer Science*, pages 128–153, Oxford, UK, December 16–18, 2019. Springer, Heidelberg, Germany.

[39] Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 169–186, Paris, France, April 15–17, 2020. Springer, Heidelberg, Germany.

[40] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 769–798, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[41] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy*, pages 1102–1120, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

[42] Ivan Damgård, Helene Haagh, Michael Nielsen, and Claudio Orlandi. Commodity-based 2PC for arithmetic circuits. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *Lecture Notes in Computer Science*, pages 154–177, Oxford, UK, December 16–18, 2019. Springer, Heidelberg, Germany.

[43] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[44] Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 75–90, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany.

[45] Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 187–212, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany.

[46] Jean Paul Degabriele and Marc Fischlin. Simulatable channels: Extended security that is universally composable and easier to prove. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 519–550, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

[47] Jean Paul Degabriele and Kenneth G. Paterson. On the (in)security of IPsec in MAC-then-encrypt configurations. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 493–504, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[48] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, February 8–11, 2015. The Internet Society.

[49] Yevgeniy Dodis, Willy Quach, and Daniel Wichs. Speak much, remember little: Cryptography in the bounded storage model, revisited. Cryptology ePrint Archive, Report 2021/1270, 2021. `https://eprint.iacr.org/2021/1270`.

[50] 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.

[51] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

[52] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 356–383, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[53] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015.

[54] Oriol Farràs and Carles Padró. Ideal hierarchical secret sharing schemes. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 219–236, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany.

[55] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th Annual ACM Symposium on Theory of Computing*, pages 554–563, Montréal, Québec, Canada, May 23–25, 1994. ACM Press.

[56] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[57] Marc Fischlin, Felix Günther, Giorgia Azzurra Marson, and Kenneth G. Paterson. Data is a stream: Security of stream-based channels. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 545–564, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[58] Marc Fischlin, Felix Günther, and Philipp Muth. Information-theoretic security of cryptographic channels. In Weizhi Meng, Dieter Gollmann, Christian Damsgaard Jensen, and Jianying Zhou, editors, *ICICS 20: 22nd International Conference on Information and Communication Security*, volume 11999 of *Lecture Notes in Computer Science*, pages 295–311, Copenhagen, Denmark, August 24–26, 2020. Springer, Heidelberg, Germany.

[59] Matthias Fitzi, Juan A. Garay, Shyamnath Gollakota, C. Pandu Rangan, and K. Srinathan. Round-optimal and efficient verifiable secret sharing. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 329–342, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

[60] Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 32–46, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

[61] M. Geihs, O. Nikiforov, D. Demirel, A. Sauer, D. Butin, F. Günther, G. Alber, T. Walther, and J. Buchmann. The status of quantum-key-distribution-based long-term secure internet communication. *IEEE Transactions on Sustainable Computing*, 2019.

[62] Matthias Geihs and Johannes Buchmann. ELSA: Efficient long-term secure storage of large datasets. In Kwangsu Lee, editor, *ICISC 18: 21st International Conference on Information Security and Cryptology*, volume 11396 of *Lecture Notes in Computer Science*, pages 269–286, Seoul, Korea, November 28–30, 2019. Springer, Heidelberg, Germany.

[63] Matthias Geihs, Nikolaos Karvelas, Stefan Katzenbeisser, and Johannes Buchmann. Propyla: Privacy preserving long-term secure storage. In *Proceedings of the 6th International Workshop on Security in Cloud Computing*, SCC '18, pages 39–48, New York, NY, USA, 2018. ACM.

[64] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[65] Jiaxin Guan and Mark Zhandry. Simple schemes in the bounded storage model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 500–524, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[66] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[67] Julius Hardt. Integrating elsa into cognicrypt - steering developers towards the correct usage of rest-based security solutions, 2021.

[68] Javier Herranz and Germán Sáez. Verifiable secret sharing for general access structures, with application to fully distributed proxy signatures. In Rebecca Wright, editor, *FC 2003: 7th International Conference on Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 286–302, Guadeloupe, French West Indies, January 27–30, 2003. Springer, Heidelberg, Germany.

[69] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[70] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. 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 830–842, Vienna, Austria, October 24–28, 2016. ACM Press.

[71] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[72] Dafna Kidron and Yehuda Lindell. Impossibility results for universal composability in public-key models and with fixed inputs. *Journal of Cryptology*, 24(3):517–544, July 2011.

[73] Nishat Koti, Shravani Patil, Arpita Patra, and Ajith Suresh. MPClan: Protocol suite for privacy-conscious computations. Cryptology ePrint Archive, Report 2022/675, 2022. `https://eprint.iacr.org/2022/675`.

[74] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively secure 4PC for secure training and inference. Cryptology ePrint Archive, Report 2021/755, 2021. `https://eprint.iacr.org/2021/755`.

[75] Stefan Krüger. *CogniCrypt - the secure integration of cryptographic software*. PhD thesis, University of Paderborn, Germany, 2020.

[76] Stefan Krüger, Johannes Späth, Karim Ali, Eric Bodden, and Mira Mezini. Crysl: An extensible approach to validating the correct usage of cryptographic apis. *IEEE Trans. Software Eng.*, 47(11):2382–2400, 2021.

[77] Robin Künzler, Jörn Müller-Quade, and Dominik Raub. Secure computability of functions in the IT setting with dishonest majority and applications to long-term security. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 238–255. Springer, Heidelberg, Germany, March 15–17, 2009.

[78] Arjen K Lenstra. *The Handbook of Information Security*, chapter Key lengths. Wiley, 2004.

[79] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, September 2001.

[80] ARM Limited. Arm security technology – building a secure system using trustzone technology. `https://community.arm.com/cfs-file/__key/telligent-evolution-components-attachments/01-2671-00-00-00-00-53-99/PRD29_2D00_GENC_2D00_009492C_5F00_trustzone_5F00_security_5F00_whitepaper.pdf`, 2009.

[81] 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.

[82] Jake Loftus and Nigel P. Smart. Secure outsourced computation. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT 11: 4th International Conference on Cryptology in Africa*, volume 6737 of *Lecture Notes in Computer Science*, pages 1–20, Dakar, Senegal, July 5–7, 2011. Springer, Heidelberg, Germany.

[83] Akash Malhotra. Amd ryzen pro 5000 series mobile processor security features. `https://www.amd.com/system/files/documents/amd-security-white-paper.pdf`, 2021.

[84] Ueli M. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, January 1992.

[85] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[86] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany.

[87] Michele Mosca, Douglas Stebila, and Berkant Ustaoglu. Quantum key distribution in the classical authenticated key exchange framework. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 136–154, Limoges, France, June 4–7, 2013. Springer, Heidelberg, Germany.

[88] Jörn Müller-Quade and Dominique Unruh. Long-term security and universal composability. *Journal of Cryptology*, 23(4):594–671, October 2010.

[89] Philipp Muth, Matthias Geihs, Tolga Arul, Johannes Buchmann, and Stefan Katzenbeisser. ELSA: efficient long-term secure storage of large datasets (full version) *. *EURASIP J. Inf. Secur.*, 2020:9, 2020.

[90] Philipp Muth and Stefan Katzenbeisser. Assisted mpc. Cryptology ePrint Archive, Paper 2022/1453, 2022. `https://eprint.iacr.org/2022/1453`.

[91] National Institute of Standards and Technology. FIPS PUB 180-4: Secure hash standard (SHS), 2015.

[92] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In Stanislaw Jarecki, editor, *Topics in Cryptology – CT-RSA 2020*, volume 12006 of *Lecture Notes in Computer Science*, pages 254–283, San Francisco, CA, USA, February 24–28, 2020. Springer, Heidelberg, Germany.

[93] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[94] Chris Peikert. He gives C-sieves on the CSIDH. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 463–492, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[95] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st Annual ACM Symposium on Theory of Computing*, pages 73–85, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[96] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018.

[97] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim, editors, *ASIACCS 18: 13th ACM Symposium on Information, Computer and Communications Security*, pages 707–721, Incheon, Republic of Korea, April 2–6, 2018. ACM Press.

[98] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.

[99] 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.

[100] Peter Scholl, Nigel P. Smart, and Tim Wood. When it's all just too much: Outsourcing MPC-preprocessing. In Máire O'Neill, editor, *16th IMA International Conference on Cryptography and Coding*, volume 10655 of *Lecture Notes in Computer Science*, pages 77–99, Oxford, UK, December 12–14, 2017. Springer, Heidelberg, Germany.

[101] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[102] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[103] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.

[104] Tom Shrimpton. A characterization of authenticated-encryption as a form of chosen-ciphertext security. Cryptology ePrint Archive, Report 2004/272, 2004. `https://eprint.iacr.org/2004/272`.

[105] Nigel P. Smart and Titouan Tanguy. TaaS: Commodity MPC via triples-as-a-service. Cryptology ePrint Archive, Report 2019/957, 2019. `https://eprint.iacr.org/2019/957`.

[106] Markus Stadler. Publicly verifiable secret sharing. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 190–199, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

[107] Tamir Tassa. Hierarchical threshold secret sharing. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 473–490, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

[108] Tamir Tassa. Hierarchical threshold secret sharing. *Journal of Cryptology*, 20(2):237–264, April 2007.

[109] Rune Thorbek. Proactive linear integer secret sharing. Cryptology ePrint Archive, Report 2009/183, 2009. `https://eprint.iacr.org/2009/183`.

[110] Giulia Traverso, Denise Demirel, and Johannes Buchmann. Performing computations on hierarchically shared secrets. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 18: 10th International Conference on Cryptology in Africa*, volume 10831 of *Lecture Notes in Computer Science*, pages 141–161, Marrakesh, Morocco, May 7–9, 2018. Springer, Heidelberg, Germany.

[111] Giulia Traverso, Denise Demirel, and Johannes A. Buchmann. Dynamic and verifiable hierarchical secret sharing. In Anderson C. A. Nascimento and Paulo Barreto, editors, *ICITS 16: 9th International Conference on Information Theoretic Security*, volume 10015 of *Lecture Notes in Computer Science*, pages 24–43, Tacoma, WA, USA, August 9–12, 2016. Springer, Heidelberg, Germany.

[112] Martín A. Gagliotti Vigil, Johannes A. Buchmann, Daniel Cabarcas, Christian Weinert, and Alexander Wiesmaier. Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: A survey. *Computers & Security*, 50:16–32, 2015.

[113] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22:265–279, 1981.

[114] A. D. Wyner. The wire-tap channel. *The Bell System Technical Journal*, 54(8):1355–1387, Oct 1975.

[115] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.