Long-Term Confidential Secret Sharing-Based Distributed Storage Systems

Vom Fachbereich Informatik der

Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades

Doktor rerum naturalium (Dr. rer. nat.)

von

Giulia Traverso, M.Sc.

geboren in Soave.

Referenten: Prof. Dr. Johannes Buchmann
Prof. Dr. Reihaneh Safavi-Naini

Tag der Einreichung: 12.03.2019
Tag der mündlichen Prüfung: 24.04.2019

Hochschulkennziffer: D 17

Darmstadt 2019

Dissertation von Giulia Traverso:

Long-Term Confidential Secret Sharing-Based Distributed Storage Systems

Technische Universität Darmstadt, Darmstadt, Germany
Tag der mündlichen Prüfung: 24.04.2019
Jahr der Veröffentlichung: 2019

List of Publications

[T1] Giulia Traverso, Denise Demirel, Johannes Buchmann: Dynamic and Verifiable Hierarchical Secret Sharing. *Information Theoretic Security – 9th International Conference (ICITS 2016)*, LNCS, Volume 10015, pages 24–43, Springer, 2016. [**Part of Chapter 3**].

[T2] Giulia Traverso, Denise Demirel, Johannes Buchmann: Performing Computations on Hierarchically Shared Secrets. *Progress in Cryptology – 10th International Conference on Cryptology in Africa (AFRICACRYPT 2018)*, LNCS, Volume 10831, pages 141–161, Springer, 2018. [**Part of Chapter 3**].

[T3] Giulia Traverso, Denise Demirel, Sheikh Mahbub Habib, Johannes Buchmann: AS^3: Adaptive social secret sharing for distributed storage systems. *14th Annual Conference on Privacy, Security and Trust (PST 2016)*, pages 528–535, IEEE Computer Society, 2016. [**Part of Chapter 4**].

[T4] Denise Demirel, Stephan Krenn, Thomas Lorünser, Giulia Traverso: Efficient and Privacy Preserving Third Party Auditing for a Distributed Storage System. *11th International Conference on Availability, Reliability and Security (ARES) 2016*, pages 88–97, IEEE Computer Society, 2016. [**Part of Chapter 4**].

[T5] Giulia Traverso, Carlos Garcia Cordero, Mehrdad Nojoumian, Reza Azarderakhsh, Denise Demirel, Sheikh Mahbub Habib, Johannes Buchmann: Evidence-Based Trust Mechanism Using Clustering Algorithms for Distributed Storage Systems (Short Paper). *15th Annual Conference on Privacy, Security and Trust (PST 2017)*, pages 277–282, IEEE Computer Society, 2017. [**Part of Chapter 5**].

[T6] Giulia Traverso, Denis Butin, Johannes Buchmann: Coalition-Resistant Peer Rating for Long-Term Confidentiality. *16th Annual Conference on Privacy, Security and Trust (PST 2018)*, pages 1–10, IEEE Computer Society, 2018. [**Part of Chapter 5**].

[T7] Carlos Garcia Cordero, Giulia Traverso, Mehrdad Nojoumian, Sheikh Mahbub Habib, Max Mühlhäuser, Johannes Buchmann, Emmanouil Vasilomanolakis:

Sphinx: a Colluder-Resistant Trust Mechanism for Collaborative Intrusion Detection. *IEEE Access*, pages 72427–72438, IEEE Computer Society, 2018.[**Part of Chapter 5**].

[T8] Johannes Buchmann, Ghada Dessouky, Tommaso Frassetto, Ágnes Kiss, Ahmad-Reza Sadeghi, Thomas Schneider, Giulia Traverso, Shaza Zeitouni: LSTee: An Efficient and Proactive Long-Term Secure Distributed Storage System. Submitted to *14th ACM Asia Conference on Information, Computer and Communications Security (ASIACCS 2019)*. [**Part of Chapter 6**].

To mum, dad and my sister Marta.

Take the first step in faith. You don't have to see the whole staircase, just take the first step.

— Martin Luther King Jr. *(1929-1968)*

Acknowlegdement

First and foremost, I would like to express my deep gratitude to my supervisor Johannes Buchmann. Among many others, the most important lesson I learned from him is that you can have better answers only if you ask better questions. This lesson guided me through my academic development as well as my personal life.

I am especially grateful to Denise Demirel and Denis Butin for their support during these four years. They taught me how papers are written and how to clearly convey my ideas. Without them, I would have never realized how much I love writing!

I want to also thank Guido Salvaneschi for bringing laughter and coffee breaks into my days, as well as for inspiring me with his passion for research and work ethic. Thank you for getting me unstuck more times than I can remember.

I appreciate that Rei Safavi-Naini agreed to co-review my thesis and I would like to extend my thanks to Reiner Hähnle, Sebastian Faust and Marc Fischlin for joining my defense committee.

I wholeheartedly thank my friends Pallavi, Clara, Johannes and Carlos, who made Darmstadt a place I can call home. Special mention goes to Patrick because he was brave enough to actually share a home with me.

I am so blessed because moving to Germany didn't mean that I lost my old friendships. Instead they got stronger and deeper. Chiaramaria, Valeria, Anna, Sara, Elisa, Franca, Maria Chiara, Natalia, Luca, Luca, Roberto, Thiago and Francesco, thank you for keeping these "long-distance relationships"vibrant and alive.

I had the chance to work with three amazing women: Ágnes, Ghada and Shaza. Thank you girls, working with you didn't feel like work. We should do that again!

Mentors were crucial for my personal development. I am profoundly grateful that Cortney McDermott, Sabine Scheunert and Marie Forleo crossed my path. They taught me that boldness means embracing our highest self and acting from there.

Thank you to my beloved boyfriend Alex for supporting me with his love during this journey and for making me a better person through our relationship.

Last but not least, my immense gratitude goes to my family. My mum, dad and my sister Marta are the true pillars of my life. They keep me grounded and give me the courage to pursue my dreams.

Giulia Traverso
Darmstadt, March 2019

Abstract

Secret sharing-based distributed storage systems can provide long-term protection of confidentiality and integrity of stored data. This is achieved by periodically refreshing the stored shares and by checking the validity of the generated shares through additional audit data. However, in most real-life environments (e.g. companies), this type of solution is not optimal for three main reasons. Firstly, the access rules of state of the art secret sharing-based distributed storage systems do not match the hierarchical organization in place in these environments. Secondly, data owners are not supported in selecting the most suitable storage servers while first setting up the system nor in maintaining it secure in the long term. Thirdly, state of the art approaches require computationally demanding and unpractical and expensive building blocks that do not scale well. In this thesis, we mitigate the above mentioned issues and contribute to the transition from theory to more practical secret sharing-based long-term secure distributed storage systems. Firstly, we show that distributed storage systems can be based on hierarchical secret sharing schemes by providing efficient and secure algorithms, whose access rules can be adapted to the hierarchical organization of a company and its future modifications. Secondly, we introduce a decision support system that helps data owners to set up and maintain a distributed storage system. More precisely, on the one hand, we support data owners in selecting the storage servers making up the distributed storage system. We do this by providing them with scores that reflect their actual performances, here used in a broad sense and not tied to a specific metric. These are the output of a novel performance scoring mechanism based on the behavioral model of rational agents as opposed to the classical good/bad model. On the other hand, we support data owners in choosing the right secret sharing scheme parameters given the performance figures of the storage servers and guide them in updating them accordingly with the updated performance figures so as to maintain the system secure in the long term. Thirdly, we introduce efficient and affordable distributed storage systems based on a trusted execution environment that correctly outsources the data and periodically computes valid shares. This way, less information-theoretically secure channels have to be established for confidentiality guarantees and more efficient primitives are used for the integrity safeguard of the data. We present a third-party privacy-preserving mechanism that protects the integrity of data by checking the validity of the shares.

Zusammenfassung

Verteilte Speichersysteme auf Secret-Sharing-Basis können anhaltenden Schutz in Vertraulichkeit und Integrität bieten. Dies kann durch regelmäßiges Aktualisieren der gespeicherten Anteile (Shares) sowie der Kontrolle ihrer Gültigkeit durch zusätzliche Audit Daten erreicht werden. Allerdings ist diese Art von Lösung in den meisten lebensnahen Situationen wie z.B. auch in grossen Unternehmen aus drei Gründen nicht optimal. Erstens stimmen die Zugriffsrechte der modernsten verteilten Speichersysteme auf Secret-Sharing-Basis nicht mit der hierarchischen Organisation in den Unternehmen überein. Zweitens bekommen die Datenbesitzer weder Unterstützung im Auswählen von passenden Speichersystemen noch im Erhalten der langfristigen Sicherheit eben dieser. Drittens fordern modernste Vorgehensweisen rechnerisch aufwändige und unpraktische sowie teure Bestandteile, die ökonomisch nicht sinnvoll sind.

In dieser Doktorarbeit entschärfen wir die oben genannten Probleme und tragen zum Übergang von Theorie zur Praxis von verteilten Speichersystemen auf Secret-Sharing-Basis bei. Erstens zeigen wir, dass verteilte Speichersysteme auf hierarchischen Secret-Sharing-Verfahren gründen können, indem wir schnelle und sichere Algorithmen einführen, deren Zugriffsrechte an die hierarchischen Strukturen eines Unternehmens angepasst und in Zukunft auch verändert werden können. Zweitens stellen wir ein Entscheidungs-Unterstützungs-System vor, das Datenbesitzern dabei hilft, ein verteiltes Speichersystem aufzubauen und zu halten. Konkreter unterstützen wir Datenbesitzer auf der einen Seite dabei, den Speicherserver des verteilten Speichersystems auszuwählen. Dies machen wir durch gegenseitige Bewertung ihrer tatsächlichen Leistung. Das ist der Output eines neuen Bewertungsverfahrens, das auf dem Verhalten von rationalen Teilnehmern als statt auf dem klassischen gut/schlecht-Modell basiert. Auf der anderen Seite unterstützen wir Datenbesitzer beim Auswählen der besten Secret-Sharing- Verfahrensparameter unter Berücksichtigung der Bewertungen der Speicherserver und leiten sie beim Aktualisieren entsprechend der neuesten Bewertungsentwicklungen, um das System langfristig sicherzuhalten. Drittens stellen wir schnelle und finanziell tragbare verteilte Speichersysteme vor, die auf einem Trusted Execution Environment gründen, welches Daten korrekt auslagert und regelmäßig gültige Anteile berechnet. So müssen weniger Kanäle mit informationstheoretischer Sicherheit hergestellt werden, um die Vertraulichkeit zu garantieren

und schnellere Verfahren werden benutzt, um vollständige Schutzmaßnahmen der Daten zu gewährleisten. Darüber hinaus stellen wir ein Third-party Verfahren zur Wahrung der Vertraulichkeit vor, welches den vollständigen langfristigen Schutz von Daten durch die Kontrolle der Gültigkeit der Anteile garantiert.

Contents

List of Figures

List of Tables

1 Introduction

The goal of this thesis is to design practical and affordable long-term secure distributed storage systems that can be employed by hospitals, companies, and institutions to protect the confidentiality and the integrity of the data they issue in an outsourced scenario.

Outsourcing digital data has recently become a common practice for data storage due to its scalability and low costs. When it comes to sensitive data, confidentiality and integrity must be guaranteed for decades. For instance, electronic health records must be protected for at least the entire lifetime of the patient, or even longer to protect the privacy of descendants. The integrity of a health record is crucial to ensure the right treatment of the patient, its confidentiality is crucial to prevent discrimination. These two requirements must hold even when the record is used as input by hospitals or pharmaceutical companies to perform statistics. Also, the access rules to the outsourced records should mirror the inner organization and hierarchy in which hospitals and companies are structured. For instance, if in principle a nurse or a technician can access health records only given the doctor's approval, then this access rule should be fulfilled as well the for the digitized and outsourced data.

As pointed out by Buchmann et al. [29], currently used cryptographic techniques are unsuitable to protect the storage of data for decades. On one hand, there is the threat of a advancement in cryptanalysis, for instance, the design of faster integer factorization algorithms that would compromise the computational hardness of state of the art encryption schemes. On the other hand, there is the threat of quantum computing. Sufficiently large quantum computers can run Shor's algorithm [104], which can break most used public-key cryptography (e.g., RSA [96], the ElGamal cryptosystem [48], and DSA [85]). Even though large quantum computers are not available yet, the significant and continuous progress being made in this direction is undeniable [38, 112]. Furthermore, it should not be underestimated that entities with large storage capabilities, such as intelligence agencies or data providers, might operate maliciously and store large quantities of encrypted data now, only to decrypt them decades later once the underlying mathematical problems of cryptosystems become tractable, which the NSA is feared to be doing with encrypted Internet

traffic [45]. Therefore, encrypting and then storing sensitive data on a storage server is not a viable solution for long-term security.

Secret sharing [102] is suitable for the long-term protection of sensitive data because these schemes provide information-theoretic confidentiality, also known as unconditional security. Hence, the security of secret sharing schemes is neither compromised by cryptanalytic progress nor by quantum computers. Secret sharing schemes generate shares of data such that only specific subsets of shares can reconstruct the original message, while all other subsets of shares reveal no information. In an outsourcing scenario, it is natural to build long-term secure distributed storage systems based on secret sharing schemes. Long-term secure distributed storage systems are composed of several storage servers, where each storage server is provisioned with a share of the data generated through a secret sharing scheme. In particular, long-term secure distributed storage systems are built on Shamir's secret sharing scheme, where the subsets of shares eligible for reconstructing the outsourced data are those whose cardinality is larger than a certain threshold. This solution not only provides confidentiality because subsets of storage servers whose cardinality is smaller than the threshold cannot get any information with respect to the outsourced data. In addition, such systems provide robustness against a certain amount of unreliable or compromised storage servers. However, secret sharing alone does not guarantee confidentiality in the long term. This solution is prone to a mobile adversary that corrupts over time enough storage servers so that it can reconstruct the data by itself with the shares collected.

The protection of the confidentiality of outsourced data is provided by so called *proactive secret sharing* [62]. In proactive secret sharing schemes, the shares distributed to the storage servers are renewed without the intervention of the data owner and without reconstructing the outsourced data. At regular time intervals, fresh new shares that are completely unlinked to the previous ones are generated (so that they still reconstruct to the original data), and the old shares are deleted. Proactive secret sharing is an effective countermeasure against that mobile adversary, which is bounded in the number of storage servers it is capable of corrupting over short windows of time but not over lengthy periods, which is generally assumed in practice. If share renewal is performed often enough, this renders the malicious behavior of a mobile adversary ineffective as it is assumed computationally bounded and cannot collect enough shares to successfully reconstruct the data, thus guaranteeing long-term confidentiality [25]. *Verifiable secret sharing* [34] enables verifying the consistency of the shares with the outsourced data by checking them against some additional audit data by means of commitment schemes. Verifying the consistency of the shares ensures that these reconstruct exactly to the same data that was originally outsourced, thus integrity of the data is preserved. When proactive secret sharing is equipped with commitment schemes, it is possible to check the consistency of the

updated shares every time these are computed. Thus, proactive and verifiable secret sharing can together guarantee the protection of confidentiality and integrity of data in outsourcing scenarios like those of long-term secure storage systems.

However, this approach combining proactive and verifiable secret sharing does not address more practical scenarios where efficient and affordable means to protect the confidentiality and the integrity of data are needed, which is the target of this thesis. Firstly, data owners outsourcing their data are often part of a larger organization and environment rather than single entities. This is the case for companies, hospitals, institutions where employees cover a specific position with specific rights and power within a hierarchical organization. Furthermore, this hierarchical organization continually changes (more levels are added as a company grows and vice versa) and so should be the access capabilities granted to them with respect to the data outsourced in the distributed storage system. Secondly, distributed storage systems have to be built in practice using several storage servers (i.e. data centers) from different storage service providers. This approach is to prevent that the shares are vulnerable to a single point of failure, i.e. the key management of the storage service provider hosting the shares. The storage servers are not equivalent and their performance depends strictly on the choices of the storage service provider they belong to in terms of technological upgrades and maintenance. It is extremely difficult in practice to chose the right storage servers making up the storage system. On the one hand, data owners lack the expertise to recognize good performance and quality of service of the storage servers. On the other hand, a standardized and transparent way to reliably compute performance figures across all storage servers is missing, where here performance is used in a broad sense and is not tied to a specific metric. Storage service providers might commercially benefit from the lack of accurate means to measure their quality of service in order to stay longer in the marketplace and increase their income. Also, even with available performance figures, users lack of the cryptographic knowledge to choose the right secret sharing instantiation and to set up the distributed storage system in the first place. Thirdly, state of the art secret sharing-based distributed storage systems are expensive and somewhat impractical infrastructures that companies, hospitals, and other entities on a budget cannot afford to adopt. Confidentiality of the outsourced data is guaranteed provided that the communication between any two parties involved in the system occurs through an information-theoretically secure channel. Each of these channels requires quantum-key distribution protocols in place, which is very expensive, inefficient, and does not scale well. Integrity is provided by verifiable secret sharing through the usage of several computationally intensive schemes. Furthermore, additional privacy preserving means to ensure that the shares stored have not been corrupted are still missing. In summary, state of the art solutions for secret sharing-based distributed storage systems are not suitable to be used by non-expert data owners in environments with a hierarchical organization among them.

In this thesis, we propose a more practical and affordable long-term secure distributed storage systems that are, while still fulfilling the same security guarantees in terms of confidentiality and integrity of data of state of the art approaches. We address the first problem by building long-term secure distributed storage systems based on *dynamic hierarchical secret sharing schemes*. These are secret sharing schemes whose access rules fit hierarchical organizations and can be flexibly modified. We address the second problem by embedding the distributed storage system into a larger service managed by a trusted third-party authority, whose unique aim is to support data owners in setting up and maintaining a long-term secure distributed storage system. To do that, the trusted authority periodically runs a performance scoring mechanism that outputs accurate performance figures by enforcing the storage service providers to issue accurate ratings, regardless of their economic interests. In addition, given the performance figures, the trusted authority guides the data owners in selecting the best parameters of the secret sharing scheme used and sends alerts when such setup needs to be modified. We address the third problem by introducing an efficient and more affordable long-term secure storage systems where a trusted execution environment is leveraged for the correct computation of the shares, for their renewal, and for the original data retrieval. This approach leads to significantly less information-theoretically secure channels needed to ensure confidentiality of the outsourced data, as well as computationally efficient means to protect their integrity, thus evolving from theory to a more feasible deployment of long-term secure distributed storage systems. To protect integrity of data in the long term, we also propose an alternative solution by introducing the first third party audit mechanism that checks the validity of the shares, while preserving their confidentiality is an information-theoretically manner. In summary, we propose a long-term secure distributed storage system that fit the access rules of hierarchically-organized environments, are less expensive to build, and support data owners in setting up and maintaining such an infrastructure.

Contribution and Outline

In the following, the outline of this thesis together with a summary of the contributions are provided.

Background (Chapter 2). In this chapter, basic knowledge about secret sharing are presented. Then, hierarchical secret sharing schemes are introduced. Finally, notions about game theory are provided, which are needed for the formalization of the rational behaviors of agents in Chapter 5.

Dynamic and Hierarchical Secret Sharing-Based Distributed Storage Systems (Chapter 3). In this chapter, we provide a hierarchical long-term secure distributed

storage system that mirrors the inner organization of companies and institutions and fulfill their needs for flexibility within their hierarchical structure. We define a secret sharing scheme with the above mentioned features as dynamic. More precisely, besides providing confidentiality guarantees through the periodic renewal of the shares, a dynamic secret sharing scheme also allows to add shareholders and modify the access rules in a distributed fashion, i.e. without first reconstructing the outsourced data. We provide such algorithms for Tassa's hierarchical secret sharing schemes and prove their security. Furthermore, we provide algorithms to compute linear operations and multiplications over such hierarchically shared data, together with their security proofs and audit procedures to ensure that the computations are performed correctly. We run evaluations and show empirically that the run time of our algorithms for adding shareholders, modifying the access rules, and performing computations for Tassa's hierarchical secret sharing schemes is comparable with the run time of corresponding algorithms for Shamir's secret sharing scheme. Thus, we show that Tassa's hierarchical secret sharing-based distributed storage systems are a viable solution for the long-term storage of data from entities with an ever-changing organizational structure.

Preventing Security Threats Through Adaptive Social Secret Sharing and Private Third-Party Auditing (Chapter 4). In this chapter, we introduce AS^3, an *adaptive social secret sharing scheme*. An adaptive social secret sharing scheme is a secret sharing scheme that adjusts the distribution of the shares to storage servers according to how their performance evolves over time. The capability of AS^3 to adapt how much reconstruction capability is granted to storage servers is due to the fact that AS^3 is built over Tassa's hierarchical secret sharing schemes, which are dynamic. Furthermore, AS^3 supports the data owner both in first building a distributed storage system and in protecting the confidentiality of the outsourced data in the long-term. This is done by defining how to initialize the parameters of the hierarchical secret sharing scheme, given the performance scores of the storage servers making up the system. Moreover, we support the data owner throughout the entire storage process by setting up alarms triggered when the status of the storage servers might lead to data loss or data breach. In addition, we support the data owner in protecting the integrity of the outsourced data by introducing the first third party audit mechanism that checks the validity of the shares stored by the storage servers without compromising on confidentiality. The mechanism does not leak any information about the shares to be audited and is compatible with proactive secret sharing schemes, thus preserving the stringent information-theoretically confidentiality requirements that secret sharing-based distributed storage systems aim at fulfilling. We provide a batch version of our audit mechanism where the integrity of multiple data can be checked at once to lower communication complexity and make the procedure more efficient.

Coalition-Resistant Performance Scoring Mechanism for Long-Term Confidentiality (Chapter 5). In this chapter, we provide means to compute accurate performance scores for the storage servers. This is achieved by designing a so called *performance scoring mechanism* that is resilient to unreliable inputs submitted by storage servers owned by coalitions of SSPs. We introduce a novel game-theoretic behavioral model to reflect the real-life motivations of SSPs, which do not fit the classical good/bad model. Thus, we treat the computation of accurate aggregate scores through mutual ratings in distributed storage systems as an economic problem. More precisely, we formalize, for the first time, the preferences of the storage servers with respect to gaining or losing a share by introducing a utility function. Assuming the rationality of the SSPs, which want to maximize their income by maximizing the shares distributed to their storage servers, we provide a game-theoretical model of the peer rating strategies of the storage servers and the SSPs. Assuming a honest majority, the performance scoring mechanism outputs aggregate scores that reflect the actual performances of the storage servers by taking advantage of their rationality. A trusted authority is introduced, which acts as a mediator, and rewards and discourages the submission of, respectively, accurate and inconsistent ratings by, respectively, positively and negatively affecting the final aggregate score of the storage server submitting those ratings. It is the trusted authority that provides the data owner with the accurate performance scores. The data owner uses these accurate performance scores to make informed decisions when it comes to choosing the storage servers making up a distributed storage system. Lastly, we show how accurate and inaccurate ratings can be distinguished in practice by instantiating our performance scoring mechanism through machine learning techniques. We provide evaluations to demonstrate that submitters of inaccurate ratings are detected and penalized.

Efficient Proactive Distributed Storage System Based on Trusted Execution Environments (Chapter 6). In this chapter, we propose LSTee, the first long-term secure distributed storage system that relies on a trusted execution environment (TEE) for the generation and the periodical renewal of the shares, as well as for the reconstruction of the originally outsourced message. Leveraging the TEE in this way leads to a more efficient and affordable long-term secure distributed storage system compared to state of the art approaches for two main reasons. First, the commitment schemes used to check the validity of the shares distributed to the storage servers are replaced with more practical signature schemes. It is the TEE that signs each (updated) share before distributing it to the respective storage server. This enables the TEE to check later on their validity (and, thus, the integrity of the outsourced data) prior to performing the renewal. Second, LSTee significantly reduces the number of information-theoretically secure channel to be established among the parties involved. Since all the computations needed to protect the confidentiality and the integrity of

the outsourced data are performed inside the TEE, only an information-theoretically secure channel between the TEE and between the data owner and the TEE and each of the storage server have to be established. Assuming a trustworthy TEE, LSTee provides the same security guarantees in terms of confidentiality and integrity of state of the art approaches. Moreover, we address the challenges of robustness and portability in LSTee by constructing it to be TEE-agnostic, such that secure and seamless migration from a compromised or unavailable TEE to another trustworthy one is supported. Lastly, to show it feasibility, we prototype LSTee on a TEE, instantiated with Intel SGX. We present run times for generation and the renewal of shares as well as for the reconstruction of files.

Conclusion (Chapter 7). This chapter concludes this thesis with a discussion on possible directions for future work.

2 Background

In this chapter, secret sharing is formally introduced. More precisely, threshold secret sharing, proactive secret sharing, verifiable secret sharing, hierarchical secret sharing, in particular Tassa's hierarchical secret sharing schemes, and social secret sharing are defined and their properties are explained.

2.1 Secret Sharing

Secret sharing is a cryptographic primitive enabling a *data owner* to distribute a message among a set of *shareholders*, each of whom is allocated a *share* of the message. More precisely, to distribute a message $m \in \mathcal{M}$ to a set of shareholders $S = \{s_1, \ldots, s_n\}$ the data owner computes shares $\sigma_1, \ldots, \sigma_n \in \Sigma$, where $\mathcal{M}$ is the message space and Σ the space of all possible shares. The message can be reconstructed only when an *authorized* subset $A \subset S$ of these shareholders combine their shares while *unauthorized* subsets $U \subset S$ are prevented from doing it. The *access structure* $\Gamma \in \mathcal{P}(S)$[1] determines both sets, i.e. $A \in \Gamma$ and $U \notin \Gamma$. From now on, the number of shareholders of a subset $R \subset S$ is denoted as $r := |R|$.

In the following, we formalize the security definition of a secret sharing scheme. On a high level, a secret sharing scheme is *secure* if any unauthorized subset of participants learns nothing about the message, while any authorized subset of participants is able to fully reconstruct the secret (the latter property is referred to as accessibility). In particular, in this thesis, we focus on secret sharing schemes that are perfectly secure, i.e. secure in an information-theoretic sense. That is, we do not consider here secret sharing schemes where unauthorized subsets can gain some knowledge about the secret to be reconstructed, even though they cannot fully determine it. We define in the following the security of the secret sharing schemes that we consider by formalizing the notion of perfect security and accessibility.

Definition 2.1 ([107])**.** *Let us assume that $m \in \mathcal{M}$ is the message distributed by a secret sharing scheme among a set S of shareholders according to access structure Γ.*

[1] $\mathcal{P}(S)$ denotes the partition of the set S.

For an authorized subset $A \in S$, i.e. $A \in \Gamma$, let us denote by σ_A the set of shares owned by the shareholders $s_i \in A$, i.e. $\sigma_A := \{\sigma_i \text{ such that } s_i \in A\}$. The accessibility of a secret sharing scheme is the property such that: $H(m|\sigma_A) = 0,\ \forall\ A \in \Gamma$. In contrast, any unauthorized subset $U \in S$, i.e. $U \notin \Gamma$, should not be able to reconstruct the secret. If in addition no information about $m \in \mathcal{M}$ is leaked to the shareholders in U, then the secret sharing scheme is perfectly secure: $H(m|\sigma_U) = H(m),\ \forall\ U \notin \Gamma$.

Note that to have perfect security, it is implicitly assumed that all communications happen through an information-theoretically secure channel, for which eavesdropping is not possible. Another important property that characterizes secret sharing schemes is the length of the shares generated with respect to the message they reconstruct to. To achieve perfects security, the length of the shares has to be at least as large as the length of the message. A perfect secure secret sharing scheme where the shares have precisely the same length as the message is called *ideal*.

We propose in the following a more intuitive definition for secret sharing schemes that uses algorithms. This type of formalization is complaint with the use case of secure distributed storage systems for sensitive data discussed in Chapter 1 and we base all the other definitions proposed in the rest of this thesis on it.

Definition 2.2. *For a message space $\mathcal{M}$, a space of shares Σ, a set of shareholders $S = \{s_1, \ldots, s_n\}$, where $i \in \mathcal{I}$ is the unique ID of shareholder $s_i \in S$, and an access structure $\Gamma \subset \mathcal{P}(S)$, a* secret sharing scheme *satisfying the security requirements of Definition 2.1 is a pair of PPT algorithms* Share *and* Reconstruct *performed by a data owner.*

Share It takes as input a message $m \in \mathcal{M}$ and it outputs n shares $\sigma_1, \ldots, \sigma_n \in \Sigma$, where share σ_i is to be sent to shareholder s_i, for $i = 1, \ldots, n$.

Reconstruct It takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders. It outputs $m \in \mathcal{M}$ if $R \in \Gamma$, and $\perp$ otherwise.

In the following, we present threshold secret sharing schemes and, in particular, Shamir's secret sharing scheme in Section 2.1.1. In Section 2.1.2, we discuss verifiable secret sharing schemes and how to instantiate them.

2.1.1 Shamir's Threshold Secret Sharing Scheme

Among the most studied schemes due to their usage in practical scenarios there are *threshold secret sharing schemes*. In threshold secret sharing schemes, the authorized subsets of shares are those whose cardinality is larger than a certain value, i.e. a threshold usually denoted by t. More precisely, authorized subsets $A \subset S$ are composed of t or more shareholders, that is $|A| \geq t$. Unauthorized subsets $U \subset S$ are

composed of $t-1$ or less shareholders, that is $|U| \leq t-1$. Thus, for threshold secret sharing schemes, the access structure Γ is defined as $\Gamma = \{A \in S \text{ such that} |A| \geq t\}$. In 1979, Shamir introduced a threshold secret sharing scheme based on interpolation of polynomials in [102]. We formalize in the following the Share and Reconstruct protocols of Shamir's (t, n)-threshold secret sharing scheme, where t is the reconstructing threshold and n is the size of the set $S = \{s_1, \ldots, s_n\}$ of shareholders.

Share takes as input a message $m \in \mathbb{F}_q$. It selects a polynomial $f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1}$ of degree $\deg(f(x)) = t-1$, where $a_0 := m$ and coefficients $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$ are chosen uniformly at random. It computes share $\sigma_i \in \mathbb{F}_q$ for shareholder $s_i \in S$ as a point on polynomial $f(x)$, i.e. $\sigma_i := f(i)$, where $i \in \mathcal{I}$ is the ID of shareholder s_i. It distributes share σ_i to shareholder s_i through an information-theoretically secure channel, for $i = 1, \ldots, n$.

Reconstruct takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \in S$ of shareholders. If $R \notin \Gamma$, then it outputs $\perp$. Otherwise, it reconstructs polynomial $f(x)$using Lagrange's interpolation formula as follows:

$$f(x) \equiv \sum_{s_i \in R} f(i) \prod_{s_j \in R, j \neq i} \frac{x-j}{i-j},$$

and outputs message $m \in \mathbb{F}_q$ as $f(0) = m$.

Shamir's secret sharing scheme is secure according to Definition 2.1. More precisely, the requirements of perfect security and accessibility are satisfied because a polynomial of degree $t-1$ in a finite field is fully determined by t of its points, while it cannot be determined when $t-1$ or less of its points are given. It is also ideal because all computations are performed in the same finite field. The full proof can be found in [102]. Note that Shamir's secret sharing scheme, as well as all the other secret sharing primitive discussed in the rest of this thesis, are linear schemes, meaning that the distribution of the shares is a linear mapping and the message space and the shares space are instantiated with the same finite field. Again, in our context and according to Definition 2.2, algorithms Share and Reconstruct are performed by a data owner who wants to outsource a message.

2.1.2 Verifiable Secret Sharing

Verifiable secret sharing was introduced by Chor et al. in [34] to provide means for shareholders to check the validity of shares received from the data owner. More precisely, audit data are generated that allow the shareholders to check whether the shares of each authorized subset of shareholders lead to the same message during the reconstruction algorithm. Verifiable secret sharing was originally meant to

protect shareholders from a malicious data owner providing them with invalid shares that do not reconstruct to the secret the data owner is supposed to share. More precisely, verifiable secret sharing ensures that, even if the data owner is untrusted, the shareholders can still reconstruct a well-defined secret. A verifiable secret sharing scheme is a secret sharing scheme as in Definition 2.2 satisfying the security properties formalized in Definition 2.1 with the following additional requirements.

Definition 2.3 ([93])**.** *In a verifiable secret sharing scheme, the algorithm* Share *of a secret sharing scheme as in Definition 2.2 satisfying perfect security and accessibility as in Definition 2.1 is extended by an additional protocol executed between an untrusted data owner and the shareholders* $S = \{s_1 \dots s_n\}$ *where a majority of them is honest, such that the following properties are fulfilled.*
Completeness If the parties computing the shares, e.g. data owner and shareholders, follow the algorithms correctly, then each shareholder accepts the share with probability 1*.*
Committing If for any two authorized subsets $A_1 \subset S$ *and* $A_2 \subset S$*, i.e.* $A_1, A_2 \in \Gamma$*, the shareholders of* A_1 *and* A_2 *accept their shares, then the following holds except with negligible probability: if* m_i *is the message reconstructed by the shareholders in* A_i *(for* $i = 1, 2$*), then* $m_1 = m_2$*.*

In the framework of secret sharing-based distributed storage system, verifiable secret sharing protects the storage servers from malicious data owner that provision them with invalid shares to later blame them that they failed to properly store them. It also protects the data owners from untrusted storage servers by giving them means to check whether they have tampered with the shares during the reconstruction of the secret.

To provide verifiable secret sharing usually *commitment schemes* are used, which come with two properties. First, bindingness ensures that it is not possible to change the message committed to. Second, hidingness ensures that no information about the message is leaked. Furthermore, there are several commitment schemes with homomorphic properties available, i.e. operations performed on the values committed to can be transferred to operations performed on the commitments. Verifiable secret sharing uses Feldman commitment presented in [46], which is unconditionally binding and computationally hiding, or Pedersen commitment presented in [93], which is computationally binding and unconditionally hiding. In the following, we use Feldmann commitment for the sake of simplicity, but our solutions work with both schemes. In the following, we recall the definition of Feldman commitment and Pedersen commitment (in brackets).

Definition 2.4 ([46],[93])**.** *Feldman (Pedersen) commitment scheme is a triple* (Setup, Commit, Open) *of the following algorithms.*

Setup It takes as input a security parameter λ *and it outputs a prime* q*, a group* $\mathbb{G}$ *of order* q*, and a generator* $g \in \mathbb{G}$ *(distinct generators* $g, h \in \mathbb{G}$*).*

Commit *It takes as input a message $m \in \mathbb{F}_q$ (and randomness $r \in \mathbb{F}_q$) and it outputs commitment $c = g^m$ ($c = g^m h^r$).*

Open *It takes as input a commitment $c \in \mathbb{G}$, a message $m \in \mathbb{F}_q$ (and randomness $r \in \mathbb{F}_q$) and it outputs '1' if $c = g^m$ (if $c = g^m h^r$) and '0' otherwise.*

There exists solutions [50], [47], [70], [6] for verifiable secret sharing providing both information-theoretic confidentiality and bindingness. However, they are not secure against a mobile adversary that is able to collect over time enough shares to retrieve the message. The solution proposed in [6] is an interactive protocol while we only consider non-interactive protocol having less communication complexity.

2.2 Hierarchical Secret Sharing

Hierarchical secret sharing schemes are secret sharing schemes where the shareholders are not equal in their reconstruction capabilities, i.e. the shares generated are not equivalent. This means that some shareholders are distributed highly informative shares (i.e. shares such that few of them are needed to reconstruct the message), while some other shareholders are distributed low informative shares (i.e. shares such that many of them are needed to reconstruct the message). Hierarchical secret sharing schemes address in practice scenarios where there is a hierarchical organization on place, such as companies. These schemes mirror the powers that the members of a company has by providing them with an according reconstruction capabilities through more or less informative shares. In general, shareholders of a hierarchical secret sharing schemes are organized in levels depending on the their power. Each level is associated with a threshold so that perfect secrecy holds when less shareholders attempt to reconstruct the message. In the following, in Section 2.2.1 we survey existing instantiation of hierarchical secret sharing. In Section 2.2.2, we present in detail Tassa's hierarchical secret sharing schemes together with requirements to be fulfilled for the well-definedness of such schemes in Section 2.2.3.

2.2.1 Overview on Existing Solutions

The first solutions for hierarchical secret sharing have been proposed by Shamir in [102] and Kothari in [72]. In Shamir's approach the higher a shareholder is in the hierarchy, the more shares it gets, overloading the most powerful shareholders. In Kothari's solution, shareholders are grouped in sets and for each set an independent secret sharing scheme is instantiated. This requires managing multiple secret sharing schemes and does not allow for cooperation among sets during the reconstruction. The solution proposed by Shamir with respect to hierarchical secret sharing is often referred to as *weighted secret sharing*. *Disjunctive secret sharing* as introduced by Simmons in [105], is the first approach using only one secret sharing scheme and

supporting cooperations of shareholders assigned to different sets, or rather levels in a hierarchy. However, his approach is not ideal meaning that the higher a shareholder in the hierarchy the larger the share to be stored. Brickell in [27] improved this by providing a disjunctive secret sharing scheme that is ideal with respect to the size of the shares, but apart from that rather inefficient. Later, Ghodosi et al. showed in [52] how to achieve efficient schemes for specific access structures. Finally, in [107] Tassa further improved this line of research by providing an efficient disjunctive secret sharing scheme for general access structures. Furthermore, he introduced *conjunctive secret sharing* that does not only allow concurrency among levels, but strictly requires the presence of a minimum amount of shareholders from the highest levels. Both conjunctive and disjunctive secret sharing are good solutions for hierarchical secret sharing and our contribution builds on Tassa's work. None of these approaches provide verifiability, nor do they allow, without reconstructing the shared message, to add or remove shareholders, to modify the conditions for accessing the message, nor to renew shares.

2.2.2 Tassa's Hierarchical Secret Sharing Schemes

The so called *conjunctive* and *disjunctive* schemes proposed by Tassa in [107] are the first hierarchical secret sharing schemes based on Birkhoff interpolation of polynomials. More precisely, shares are either points on a polynomial or points on one of the derivatives of such polynomial. More precisely, a hierarchy is composed of levels $L_0, \ldots, L_\ell$, where L_0 is the highest level, L_ℓ the lowest, and $\ell \leq n$. The cardinality of each level L_h is denoted by n_h and each shareholder is assigned to one level only. In addition, a threshold t_h is associated with each level L_h, for $h \in 0, \ldots, \ell$, such that $0 < t_0 < \cdots < t_\ell$. Tassa individuated two types of access structures, defining, respectively the conjunctive and the disjunctive hierarchical secret sharing. On the one hand, the conjunctive access structure determines that a subset $A \subset S$ is authorized if, *for all levels* L_h, it contains t_h shareholders assigned to levels equal or higher than L_h, for $h = 0, \ldots, \ell$. On the other hand, the disjunctive access structure specifies that a subset $A \subset S$ is authorized if, *for at least one level* L_h, it contains t_h shareholders assigned to levels equal or higher than L_h, for $h = 0, \ldots, \ell$. In the following, we write information relating to disjunctive hierarchical secret sharing in brackets. For conjunctive (disjunctive) hierarchical secret sharing schemes the unique ID of shareholder $s_{i,j} \in S$ is a pair $(i, j) \in \mathcal{I} \times \mathcal{I}$, where $i = 1, \ldots, n_h$ and $j := t_{h-1}$ ($j := t_\ell - t_h$), for $h = 0, \ldots, \ell$ with $t_{-1} := 0$ and $t := t_\ell$. In the following, we formally describe how algorithms Share and Reconstruct of Definition 2.2 are instantiated for Tassa's hierarchical conjunctive (disjunctive) secret sharing schemes. Note that these algorithms are both run by a data owner and satisfy the security properties of Definition 2.1, as formally proved in [107].

Share The algorithm takes as input a message $m \in \mathbb{F}_q$ and generates a polynomial

$f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1}$ of degree $\deg(f(x)) = t-1$, where $a_0 := m$ ($a_{t-1} := m$) and the coefficients $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$ ($a_0, \ldots, a_{t-2} \in \mathbb{F}_q$) are chosen uniformly at random. It outputs share $\sigma_{i,j} \in \mathbb{F}_q$ for shareholder $s_{i,j} \in S$ computed as $\sigma_{i,j} := f^j(i)$, where $f^j(x)$ is the j-th derivative of polynomial $f(x)$ and pair $(i,j) \in \mathcal{I} \times \mathcal{I}$ is the unique ID of shareholder $s_{i,j} \in S$, for $i = 1, \ldots, n_h$ and $h = 0, \ldots, \ell$.

Reconstruct The algorithm takes as input a set of shares held by a subset $R \subset S$ of shareholders. If R is unauthorized, i.e. $R \notin \Gamma$, then it outputs $\perp$. If R is authorized, i.e. $R \in \Gamma$, then it reconstructs polynomial $f(x)$ using Birkhoff interpolation and outputs $m = a_0$ ($m = a_{t-1}$).

The Birkhoff interpolation problem is a generalization of the Lagrange interpolation problem and describes the problem of finding a polynomial $f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1}$ satisfying the equalities $f^j(i) = \sigma_{i,j}$. Given an authorized set $R \in \Gamma$ of shareholders for conjunctive (disjunctive) hierarchical secret sharing schemes, the Birkhoff interpolation problem can be solved as follows. The *interpolation matrix* associated to set R is a binary matrix E where entry $e_{i,j}$ is set to '1' if shareholder $s_{i,j}$ participates with share $\sigma_{i,j}$ and '0' otherwise. Let us denote by $I(E) = \{(i,j) \text{ such that } e_{i,j} = 1\}$ the set containing the entries of E in lexicographic order, i.e. the pair (i,j) precedes the pair (i',j') if and only if $i < i'$ or $i = i'$ and $j < j'$. The elements of $I(E)$ are denoted by $(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)$, where $r := |R|$. Furthermore, we set $\varphi := \{\phi_0, \phi_1, \phi_2, \ldots, \phi_{t-1}\} = \{1, x, x^2, \ldots, x^t\}$ and denote by ϕ_k^j the j-the derivative of ϕ_k, for $k = 0, \ldots, t-1$. Then the matrix $A(E, X, \varphi)$ is defined as follows:

$$A(E, X, \varphi) = \begin{pmatrix} \phi_0^{j_1}(i_1) & \phi_1^{j_1}(i_1) & \phi_2^{j_1}(i_1) & \cdots & \phi_{t-1}^{j_1}(i_1) \\ \phi_0^{j_2}(i_2) & \phi_1^{j_2}(i_2) & \phi_2^{j_2}(i_2) & \cdots & \phi_{t-1}^{j_2}(i_2) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \phi_0^{j_r}(i_r) & \phi_1^{j_r}(i_r) & \phi_2^{j_r}(i_r) & \cdots & \phi_{t-1}^{j_r}(i_r) \end{pmatrix}.$$

Polynomial $f(x)$ can be reconstructed in distributed fashion by computing

$$f(x) = \sum_{k=0}^{t-1} \frac{\det(A(E, X, \varphi_k))}{\det(A(E, X, \varphi))} x^k,$$

where matrix $A(E, X, \varphi_k)$ is obtained from matrix $A(E, X, \varphi)$ by replacing its $(k+1)$-th column with shares $\sigma_{i,j}$ in lexicographic order.

2.2.3 Requirements for Birkhoff Interpolation Matrices Intepolation

In this section the necessary requirements and a sufficient condition for the interpolation matrix E are presented, such that the corresponding Birkhoff interpolation

problem has a unique solution. For the corresponding proofs we refer to [107].

Lemma 2.5. *Let $A \subset S$ be an authorized subset of shareholders, i.e. $A \in \Gamma$, and E the corresponding interpolation matrix, where the entries $e_{i,j}$ of the matrix E satisfy the following condition:*

$$\sum_{j=0}^{k} \sum_{i=1}^{r} e_{i,j} \geq k+1, \quad 0 \leq k \leq d, \tag{2.1}$$

where d is the highest derivative order in the problem and $r := |A|$ is the number of interpolating points.

Before providing the sufficient condition (Theorem 2.7), the following definition is needed.

Definition 2.6 ([107])**.** *In the interpolation matrix E a 1-sequence is a maximal run of consecutive 1s in a row of the matrix E itself. Namely, it is a triplet of the form (i, j_0, j_1) where $1 \leq i \leq r$ and $0 \leq j_0 \leq j_1 \leq d$, such that $e_{i,j} = 1$ for all $j_0 \leq j \leq j_1$, while $e_{i,j_0-1} = e_{i,j_1+1} = 0$. A 1-sequence (i, j_0, j_1) is called supported if E has 1s both to the northwest and southwest of the leading entry in the sequence, i.e. there exist indexes nw and sw, where $i_{nw} < i < i_{sw}$ and $j_{nw}, j_{sw} < j_0$ such that $e_{i_{nw},j_{nw}} = e_{i_{sw},j_{sw}} = 1$.*

Theorem 2.7. *The interpolation Birkhoff problem for an authorized subset A and the corresponding interpolation matrix E has a unique solution, if the interpolation matrix E satisfies (2.1) and contains no supported 1-sequence of odd length.*

In case the Birkhoff interpolation problem is instantiated over a finite field $\mathbb{F}_q$ with $q > 0$ a prime number, then also the following condition has to hold.

Theorem 2.8. *The Birkhoff interpolation problem for an interpolation matrix E has a unique solution over the finite field $\mathbb{F}_q$, if Theorem 2.7 holds and in addition also the following inequality is satisfied:*

$$q > 2^{-d+2} \cdot (d-1)^{\frac{(d-1)}{2}} \cdot (d-1)! \cdot x_r^{\frac{(d-1)(d-2)}{2}}, \tag{2.2}$$

where d is the highest derivative order of the problem.

2.3 Social secret sharing

Introduced by Nojoumian and Stinson in [87], *social secret sharing* is a primitive composed of a secret sharing scheme and a performance scoring mechanism. In the following, we first present performance scoring mechanisms (Section 2.3.1) so that social secret sharing schemes, formalized in Section 2.3.2, can be fully understood.

2.3.1 Performance Scoring Mechanisms

Performance scoring mechanisms [66] are a viable solution to build trust in electronic environments. Performance scoring mechanism are run so that *scores* are computed that rate the entities involved in a certain environment with respect to a given metric. The metric depends on the environment and on the types of tasks the entities are expected to carry out and the term "*performance*"is thus used in a broad sense to convey the idea that a performance scoring mechanism is not tied to the specific metric the scores it outputs are supposed to describe. Performance scoring mechanisms enable trust because entities with higher scores gained a good reputation with respect to carrying out properly a given task and are therefore likely to be chosen for future tasks as well. Indeed, the score tracks the reputation that an entity has built with respect to carrying out a specific task with a certain level of performance.

Performance scoring mechanism are referred to as evidence-based performance scoring mechanisms when they rely on evidence derived from past interactions. More precisely, evidence can be derived from direct interactions between a trustor and a trustee. Direct interactions, however, may be rare in certain cases, e.g. newcomers in service marketplaces. Thus, evidence-based mechanisms also consider evidence derived from indirect interactions. That is, an entity provides another entity with evidence about its past interactions with a third entity. This is usually referred to as exchange of recommendations. In case both direct and indirect interactions are not available, one may rely on evidence derived from virtual cues, e.g., certifications or stereotypes. Evidence-based performance scoring mechanisms are composed of two phases: a collection phase and a processing phase. In the collection phase, trustors observe the behavior of the trustees with respect to a certain metric and issue ratings to describe the performance of that interaction (i.e. the evidence). In the processing phase, the ratings from (possibly) multiple trustors issue after several interactions are processed in a deterministic way proper of the performance scoring mechanism used and the scores of the trustees are computed. Due to the fact that the scores aggregate and process several ratings, we refer to them as *aggregate scores*. For simplicity, because in this thesis we focus on mechanisms that output scores based on evidence, from now on, we refer to evidence-based performance scoring mechanism as simply performance scoring mechanism.

2.3.2 Social Secret Sharing Definition

As already anticipated, social secret sharing is a primitive composed of a secret sharing scheme and of a performance scoring mechanism. In this way, social secret sharing grants more informative shares based on the performance of the shareholders with respect to carrying out a certain task. More precisely, the performance scoring mechanism outputs periodically an aggregate score for each of the shareholders

measuring their performance, where, again, performance is meant in a broad sense and depends on the metric that the scores are supposed to rate. According to these aggregate scores, more or less informative shares are distributed to the shareholders. Social secret sharing have been so far instantiated either through Shamir's weighted secret sharing [87, 88, 90] or through Tassa's hierarchical secret sharing schemes [92]. Together with the usual algorithms Share and Reconstruct proper of general secret sharing schemes, social secret sharing is composed of an additional algorithm called Tune, which is used to determine how to adjust the shares distributed to the shareholders according to their behavior. We formalize the notion of social secret sharing in the following definition.

Definition 2.9. *For a message space $\mathcal{M}$, a space of shares Σ, a set of shareholders $S = \{s_1, \ldots, s_n\}$ where $i \in \mathcal{I}$ is the unique ID of shareholder $s_i \in S$, and an access structure $\Gamma \subset \mathcal{P}(S)$, an* social secret sharing scheme *is a secret sharing scheme with PPT algorithms Share and Reconstruct as in Definition 2.2 satisfying the security properties of Definition 2.1 with an additional PPT algorithm Tune run as well by the data owner as follows.*

Share takes as input a message $m \in \mathcal{M}$ and a vector of weights $w_1, \ldots, w_n \in [0, 1]$, where $\sum_{i=1}^{n} w_i = 1$. It outputs n shares $\sigma_1, \ldots, \sigma_n \in \Sigma$, where share σ_i is to be sent to shareholder $s_i \in S$ and whose reconstruction capability matches weight w_i, for $i = 1, \ldots, n$.

Tune takes as input aggregate scores $\tau_1, \ldots, \tau_n \in [0, 1]$ for, respectively, shareholders $s_1, \ldots, s_n$ computed by a performance scoring mechanism. It outputs weights $w_1, \ldots, w_n$ for shareholders $s_1, \ldots, s_n$ as well as shares $\sigma_1, \ldots, \sigma_n$ whose reconstruction capabilities match the weights.

Reconstruct takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders. It outputs $m \in \mathcal{M}$ if $R \in \Gamma$, and $\perp$ otherwise.

2.4 Notions of Game Theory

In game theory [91], a set of players $P = \{P_1, \ldots, P_n\}$, an action profile $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, where $\mathcal{A}_i$ is the set of actions of player P_i, and a *utility function* $u_i : \mathcal{A} \to \mathbb{R}$ are the components of a *game*. A game is denoted by $\Gamma(P_i, \mathcal{A}_i, u_i)$, for $i = 1, \ldots, n$. The utility function of a player defines its preferences with respect to the actions it takes and to the actions all other players take. That is, given two distinct outcomes $\boldsymbol{a}, \boldsymbol{a}' \in \mathcal{A}$, with $\boldsymbol{a} = (a_1, \ldots, a_n)$ and $\boldsymbol{a}' = (a'_1, \ldots, a'_n)$, if $u_i(\boldsymbol{a}) \geq u_i(\boldsymbol{a}')$, then player P_i prefers $\boldsymbol{a}$ to $\boldsymbol{a}'$.

Games consist of either one or multiple rounds and the latter are referred to as *repeated games*. Among repeated games, there are *infinitely* and *finitely* repeated

games, which consist of, respectively, an infinite or a finite number of rounds. At each round, all players are asked to choose a certain action simultaneously.

Players can rely on different strategies with respect to the actions to choose. A *strategy* for player P_i is a probability distribution $\sigma_i : \mathcal{A}_i \to [0, 1]$ that determines how the actions $a_i \in \mathcal{A}_i$ are chosen, where $\sigma_i \in \mathcal{S}_i$ and $\mathcal{S}_i$ is called the *strategy profile* of player P_i. Since a strategy is a way to choose actions, it is possible to express the expected outcome of a game by using strategies. That is, a game can be denoted by $\Gamma(P_i, \sigma_i, u_i)$, for $i = 1, \ldots, n$ and $\sigma_i \in \mathcal{S}_i$. Given $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ the tuple composed of each player P_i's strategy σ_i, the utility of player P_i when strategy $\boldsymbol{\sigma}$ is played can be denoted by $u_i(\boldsymbol{\sigma})$. We refer to $\boldsymbol{\sigma}$ as the *joint strategy* of players $P_1, \ldots, P_n$.

We denote by $(\sigma'_i, \boldsymbol{\sigma}_{-i})$ the vector of strategies where all players maintained the same strategies of the joint strategy $\boldsymbol{\sigma}$, except for player P_i, which replaced strategy σ_i by another strategy σ'_i. That is, $(\sigma'_i, \boldsymbol{\sigma}_{-i}) = (\sigma_1, \ldots, \sigma_{i-1}, \sigma'_i, \sigma_{i+1}, \ldots, \sigma_n)$. Given a subset $C \subset P$ of players of cardinality $n_C \leq n$, we denote by $(\boldsymbol{\sigma}'_C, \boldsymbol{\sigma}_{-C})$ the vector of strategies where all players maintained the same strategies of the joint strategy $\boldsymbol{\sigma}$, except for player $P_j \in C$, which replaced strategy σ_j by another strategy σ'_j, for $j = 1, \ldots, n_C$. That is, $(\boldsymbol{\sigma}'_C, \boldsymbol{\sigma}_{-C}) = (\sigma'_1, \ldots, \sigma'_{n_C-1}, \sigma'_{n_C}, \sigma_{n_C+1}, \ldots, \sigma_n)$, where, without loss of generality, players in C are the first n_C players.

Players act as rational deciders, meaning that they always play the strategy that maximizes their utilities, which depends on the other players' actions. No player knows what strategies other players adopt at a given round, and can only form beliefs in this respect. The pay-off players get is not only subjected to the way they choose strategies, but also to the type of game that is played.

There are two types of game. A *non-cooperative* (or strategic) game deals with actions chosen by players individually and the pay-off of a player is given solely by its utility function. Instead, a *cooperative* (or coalitional) game deals with the actions a subset of players agree to take collectively and their pay-off is given by splitting the overall utility among themselves. In non-cooperative games, the concept of a *Nash equilibrium* conveys the idea that players choose a strategy by both looking at the best available actions and by taking into account the belief of how the other players might behave. A joint strategy $\boldsymbol{\sigma}$ such that no player P_i alone has an incentive in choosing a strategy σ'_i other than σ_i, while all other players P_j stick to strategy $\sigma_{j \neq i}$ is called a Nash equilibrium.

Definition 2.10 ([91])**.** *A joint strategy $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is called a Nash equilibrium if $u_i(\sigma'_i, \boldsymbol{\sigma}_{-i}) \leq u_i(\sigma_i, \boldsymbol{\sigma}_{-i})$, for each player P_i, where $\sigma'_i \neq \sigma_i$. It is a strict Nash equilibrium if $u_i(\sigma'_i, \boldsymbol{\sigma}_{-i}) < u_i(\sigma_i, \boldsymbol{\sigma}_{-i})$.*

The counterpart of a Nash equilibrium for cooperative games is introduced in [2] and called a *k-resilient equilibrium*.

Definition 2.11 ([2])**.** *A joint strategy $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is a k-resilient equilibrium if $u_i(\boldsymbol{\sigma}'_C, \boldsymbol{\sigma}_{-C}) \leq u_i(\boldsymbol{\sigma}_C, \boldsymbol{\sigma}_{-C})$, for each subset $C \subset P$ of cardinality $n_C \leq k$, where*

$\sigma_i' \neq \sigma_i$, for $P_i \in C$. It is a strongly resilient equilibrium if it is a k-resilient equilibrium for $k \leq n-1$.

Definition 2.10 and Definition 2.11 can also be expressed by using a game outcome $\boldsymbol{a}$ sampled from the action profile instead of the joint strategy $\boldsymbol{\sigma}$. From now on, we refer to the subset C of players deviating from the joint strategy in Definition 2.11 as a *coalition*. Another important concept in game theory is the one of dominated strategies (or actions). A strategy is dominated by another strategy if it always provides the player with a lower pay-off. This concept is formalized in the definition below, denoting by $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ the strategy profile of players $P_1, \ldots, P_n$ and by $\mathcal{S}_{-i} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_{i-1} \times \mathcal{S}_{i+1} \times \cdots \times \mathcal{S}_n$ the strategy profile obtained by removing the set $\mathcal{S}_i$ of possible strategies for player P_i from $\mathcal{S}$.

Definition 2.12 ([91])**.** *Given two strategies $\sigma_i, \sigma_i' \in \mathcal{S}_i$ for player P_i, strategy σ_i is weakly dominated by strategy σ_i' if $u_i(\sigma_i, \boldsymbol{\sigma}_{-i}) \leq u_i(\sigma_i', \boldsymbol{\sigma}_{-i})$ for each $\boldsymbol{\sigma}_{-i} \in \mathcal{S}_{-i}$ and there exist $\boldsymbol{\sigma}'_{-i} \in \mathcal{S}_{-i}$ such that $u_i(\sigma_i, \boldsymbol{\sigma}'_{-i}) < u_i(\sigma_i', \boldsymbol{\sigma}'_{-i})$. Strategy σ_i is strictly dominated if $u_i(\sigma_i, \boldsymbol{\sigma}_{-i}) < u_i(\sigma_i', \boldsymbol{\sigma}_{-i})$ for each strategy $\sigma_i' \in \mathcal{S}_i$.*

3 Dynamic and Hierarchical Secret Sharing-Based Distributed Storage Systems

Distributed storage systems fulfill an important role: they offer highly reliable storage services because data are stored redundantly in many storage servers. Such systems are reliable as a whole because the data can be retrieved even if some of the storage servers involved are not.

There are sensitive data that need protection for decades and whose confidentiality must never be compromised during this time. Such data are for example electronic health records. To cope with the stringent protection requirements of such data, in addition to reliability, distributed storage systems should also provide information-theoretic confidentiality of the data stored, as well as the following functionalities. First, they should allow for hierarchical access rules that mirror the structure of the organizations adopting them. For instance, in the case of electronic health records, distributed storage systems might be adopted by hospitals. Here, access rules can be set up so that a health record is accessed by a nurse only if she gets the approval of the doctor that issued it. The access rules should be dynamic too, meaning that they should be easily modifiable without resetting the system from scratch. For instance, additional nurses or doctors can be employed in the hospital and they all should get their corresponding share of data. In addition, distributed storage systems should support computations on the shared data and be equipped with audit mechanisms that ensure that the computations have been performed correctly. For instance, statistics over electronic health records might provide important insight on how diseases are correlated to certain symptoms and correct computations are crucial for good diagnosis.

Secret sharing-based distributed storage systems are an established approach to the long-term protection of the confidentiality of stored data. Several secret sharing schemes provide information-theoretic confidentiality: they are not vulnerable to cryptanalytic progress nor quantum computing and this makes them suitable for the long-term protection of sensitive data. In the special case of proactive secret sharing schemes, the adversary model can even be strengthened: the shares stored in the

storage servers are periodically renewed and this allows these schemes to cope with an adversary that can over time break into all the storage servers. There are two types of secret sharing schemes that are suitable to build distributed storage systems: Shamir's secret sharing scheme (see Section 2.1) and Tassa's hierarchical secret sharing schemes (see Section 2.2). On the one hand, Shamir's secret sharing scheme satisfies all the above requirements except that it does not support hierarchical access rules. This is due to the fact that all the shares generated are equivalent. On the other hand, Tassa's hierarchical secret sharing schemes allow for hierarchical access rules. However, no algorithms for changing these hierarchical access rules, nor algorithms for computing on hierarchically shared data together with auditing mechanisms, nor for renewing the shares have been proposed until now. Thus, neither solution fully satisfies all the requirements that distributed storage systems should fulfill.

Contributions

In this chapter, we provide a secret sharing-based distributed storage system that has all the required properties described above, meaning it allows for hierarchical and dynamic access rules, for computations on shared data and according audit mechanisms, and for share renewal. We provide these algorithms for Birkhoff interpolation-based secret sharing schemes, in particular Tassa's hierarchical secret sharing schemes. The idea behind the algorithms we present is that the algorithm Reconstruct for Tassa's hierarchical conjunctive and disjunctive secret sharing scheme presented in Section 2.2 uses terms provided by the shareholders that can be computed in distributed fashion. This allows us provide the algorithms for modifying the access rules, changing the set of storage servers making up the storage system, renewing the shares, and computing on shared messages by generalizing the corresponding algorithms for Lagrange interpolation-based secret sharing schemes (like the one by Shamir's). We prove that the computations of such terms needed in algorithm Reconstruct do not leak information and still allows to reconstrcut the shared message. Thus, because of the perfect secrecy and accessibility of Tassa's hierarchical conjunctive and disjunctive secret sharing schemes, we prove that the algorithms we provide are secure and correct as well.

The contributions of this chapter are based on published papers [T1] and [T2]. My contributions were the definition of the dynamic secret sharing primitive and the design of the algorithms for modifying the access rules and for computing on hierarchically shares messages as well as their respective security proofs.

Outline

The outline of this chapter is as follows. In Section 3.1, we discuss related work with respect to secret sharing schemes allowing for dynamic access rules, for computations

on shared data and audit mechanisms, and for share renewal and highlight that none of those schemes provide hierarchical access to secrets. In Section 3.2, we provide a definition of dynamic secret sharing as a scheme where the access rules can be modified, lost shares can be recomputed and renewed and show that Tassa's hierarchical secret sharing schemes are dynamic. Also, we prove that these algorithms that make hierarchical secret sharing dynamic do not lower the underlying security of the schemes. In Section 3.3, we provide algorithms for computing linear operations and multiplications on hierarchically shared data, as well as algorithms to perform the audit of those computations. The security of these algorithms is also proved. In Section 3.4 the efficiency of the algorithms that make Tassa's hierarchical secret sharing schemes dynamic and that allow for computations are compared to the ones for Shamir's secret sharing schemes and experimental validations are also provided. Summary and future work can be found in Section 3.5.

3.1 Related Work

In this section, we review related work with respect to secret sharing schemes allowing to modify the access rules, to renew the shares, to verify their consistency and to perform computations over shared messages that can be audited.

The best known and used secret sharing scheme is the one proposed by Shamir in [102]. Shamir's secret sharing scheme has been extensively investigated and it now provides all the aforementioned features and properties that make it a suitable candidate for distributed storage systems. Herzberg et al. showed in [62] how shares can be periodically refreshed. Based on the work by Desmedt and Jajodia presented in [42], Gupta and Gopinath showed in [58] how to modify the access rules distributedly so that different subsets of shareholder become eligible for the reconstruction of the message. In [87], Nojoumian et al. presented how new shares can be generated without the intervention of the data owner, so that compromised shares can be recovered or new storage servers can join the distributed storage system. Furthermore, it is possible to perform operations over shared messages as discussed by Goldreich et al. in [53]. This enables general multi-party computation, as discussed by Ben-Or et al. in [12], by Chaum et al. in [32], and by Gennaro et al. in [51]. Furthermore, Schabüser et al. showed in [98] how to perform an auditing procedure for computations over shared messages, which is based on the work done by Beaver in [9] and by Damgård and Nielsen in [40].

Hierarchical secret sharing schemes have been investigated. There have been only two steps towards having hierarchical secret sharing schemes suitable for distributed storage systems. The first step was made by Pakniat et al. in [92], where it was shown how to renew the shares. However, their approach was not general enough to describe polynomial-based hierarchical secret sharing schemes as a whole and did not allow for any other functionality. The second step was made by by Kasper

et al. in [69], where conditions on the access structure allowing for multiplication have been investigated. However, they lead to schemes with either an increased length of the shares (which is not optimal for our application to distributed storage systems) or with stronger conditions on the access structure deviating from the original schemes proposed by Tassa. Thus, the polynomial-based Tassa's hierarchical secret sharing schemes currently do not come with any of functionalities available for Shamir's secret sharing scheme that would make it another prominent candidate for secret-sharing based distributed storage systems. Tassa's hierarchical secret sharing schemes are linear and, from a theoretical point of view, this means that they can be equipped with all these functionalities (as one can infer from what is discussed by Desmedt and Jajodia in [42] and by Cramer et al. in [36]). However, practical algorithms carrying out these functionalities have not been proposed yet. And this is what we provide in this work.

3.2 Dynamic and Verifiable Hierarchical Secret Sharing

In this section, we provide algorithms for Tassa's hierarchical secret sharing schemes for modifying the access rules to the message, adding shares, and renewing them periodically. We first define such secret sharing schemes where these three functionalities are added as dynamic secret sharing scheme and formalize this idea in Section 3.2.1. By first proving how to distributedly compute certain operations of Tassa's conjunctive and disjunctive hierarchical secret sharing schemes, in Section 3.2.3 we show they are dynamic by introducing the algorithms Add and Reset to the existing algorithms Share and Reconstruct discussed in Section 2.2. Finally, in Section 3.2.4 it is proven that these new algorithms are correct and do not compromise the security of the underlying schemes.

3.2.1 Definition of Dynamic Secret Sharing

In the framework of dynamic secret sharing, we assume that all communication channels used guarantee reliable delivery of messages, any two shareholders can communicate via a private channel, all shareholders can receive messages sent over a broadcast channel, any shareholder can declare and no shareholder can spoof its identity, and a majority of the shareholders participating in each algorithm is trustworthy such that wrongly generated shares can be detected. Note that these are standard assumption for classical secret sharing schemes that provide verifiability and dynamism and that the latter assumption can be weakened using the complaint mechanism proposed by Gupta and Gopinath in [58]. Furthermore, our algorithms assume a synchronous network, but can easily be adapted to asynchronous networks,

for instance, by using the techniques proposed by Schultz et al. in [100]. In the following, we formally introduce *dynamic secret sharing schemes* as secret sharing schemes that in addition to the algorithm Share and Reconstruct discussed in Section 2.1 allow to perform Add and Reset in distributed fashion.

Definition 3.1. *For a message space $\mathcal{M}$, a space of shares Σ, a set of shareholders $S = \{s_1, \ldots, s_n\}$ where $i \in \mathcal{I}$ is the unique ID of shareholder $s_i \in S$, and an access structure $\Gamma \subset \mathcal{P}(S)$, a dynamic secret sharing scheme is a secret sharing scheme with PPT algorithms Share and Reconstruct run by the data owner as in Definition 2.2 and satisfying the security properties of Definition 2.1 that is equipped by additional PPT algorithms Add and Reconstruct run distributedly by authorized subsets of shareholders that also satisfy the security properties of Definition 2.1. More precisely, a dynamic secret sharing scheme is a tuple of PPT algorithms Share, Add, Reset, and Reconstruct.*

Share takes as input a message $m \in \mathcal{M}$. It outputs n shares $\sigma_1, \ldots, \sigma_n \in \Sigma$, where share σ_i is to be sent to shareholder $s_i \in S$, for $i = 1, \ldots, n$.

Add takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders and the ID i, i.e. $i = n + 1$, of the new shareholder. If R is unauthorized, i.e. $R \notin \Gamma$, it outputs $\perp$. Otherwise, $R \in \Gamma$ and without message reconstruction, it outputs a corresponding share $\sigma_i \in \Sigma$ for the new shareholder s_i.

Reset takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders, a new set of shareholders $S' = \{s'_1, \ldots, s'_{n'}\}$ (that need not be disjoint to S), and an access structure $\Gamma' \subset \mathcal{P}(S')$. If R is unauthorized, i.e. $R \notin \Gamma$, it outputs $\perp$. Otherwise, $R \in \Gamma$ and without message reconstruction, it outputs n' shares $\sigma'_1, \ldots, \sigma'_{n'}$, where share σ'_i is to be sent to each new shareholder $s'_i \in S'$, for $i = 1, \ldots, n'$. The shares $\sigma_1, \ldots, \sigma_n \in \Sigma$ held by the old shareholders are deleted.

Reconstruct takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders. It outputs $m \in \mathcal{M}$ if $R \in \Gamma$, and $\perp$ otherwise.

In addition to the algorithms Share, Add, and Reset, a *verifiable and dynamic secret sharing scheme* provides audit data for verification according to Definition 2.3.

Algorithm Add differs from algorithm Reset in the sense that the access structure remains unchanged and old shareholders keep their shares. This is of practical interest since renewing shares could be a demanding procedure, e.g. in case shares are distributed on smartcards. A new access structure Γ' is given as input to algorithm Reset because here the whole hierarchical organization among the shareholders is completely modified and can be enlarged or shrinked substantially. Algorithm Reset can be run to refresh the shares only, without modifying the access structure nor the set of shareholders. To do that it is sufficient to run it with the old set of shareholder

S and the old access structure Γ as input. Furthermore, algorithm Reset allows also to remove shareholders. We stress that it is important to delete the old shares during algorithm Reset so that the distributed storage system built on dynamic secret sharing is resilient against the mobile adversary. Note that notions of dynamic secret sharing have been already proposed, yet with different meanings and less functionalities with respect to our definition. More precisely, in the work by Blundo et al. in [17], it is the data owner that decides which shareholders reconstruct which secret. This implies the active intervention of the data owner, which is not desirable for long-term storage of data. In the work by Baron et al. in [8], it is not possible to add shareholders without changing all the shares already distributed.

3.2.2 Distributed Computation of Determinants

To fulfill Definition 3.1, the algorithms Add and Reset have to be performed without reconstructing the message $m \in \mathcal{M}$. This is possible since determinant $\det(A(E, X, \varphi_k))$ can be computed in distributed fashion (we refer to Section 2.2 for the definition of this term). From now on we simplify the notation referring to the shareholders within subset $R \subset S$ as s_l and no longer as $s_{(i,j)}$. However, we stress that shareholders in R are not equal from the hierarchical point of view.

Theorem 3.2. *The polynomial* $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1} \in \mathbb{R}_{t-1}[x]$ *can be computed by*

$$f(x) = \sum_{k=0}^{t-1} a_k x^k = \sum_{k=0}^{t-1} \sum_{l=1}^{r} a_{l,k} x^k,$$

where $a_{l,k}$ *is computed by shareholder* $s_l \in R$*, for* $l = 1, \ldots, r$ *and* $R \in \Gamma$ *is an authorized subset of* S*, with* $r =: |R|$*.*

Proof. Let us first recall that *Laplace's expansion formula* computes the determinant $\det(A)$ of an $n \times n$ matrix A as the weighted sum of the determinants of n sub-matrices of A, each of size $(n-1) \times (n-1)$. More precisely $\det(A) = \sum_{j'=1}^{n} a_{i,j'} (-1)^{i+j'} \det(A_{i,j'}) = \sum_{i'=1}^{n} a_{i',j} (-1)^{i'+j} \det(A_{i',j})$, where $A_{i,j}$ results from A by deleting the i-th row and j-th column.

The fact that $A(E, X, \varphi)$ can be computed by each shareholder from public information together with Laplace's expansion formula implies that each shareholder $s_l \in R$, for $l = 1, \ldots, r$, can compute the partial information $a_{l,k}$ for coefficient $a_k = \frac{\det(A(E,X,\varphi_k))}{\det(A(E,X,\varphi))}$, by $a_{l,k} := \sigma_l (-1)^{l-1+k} \frac{\det\left(A_{l-1,k}(E,X,\varphi)\right)}{\det(A(E,X,\varphi))}$, where σ_l is the share held by shareholder s_l, and $A_{l-1,k}(E, X, \varphi)$ is the matrix that results from $A(E, X, \varphi)$ by removing the l-th row and the $(k+1)$-th column. From Laplace's expansion formula it follows that:

$$\sum_{l=1}^{r} a_{l,k} = \sum_{l=1}^{r} \sigma_l (-1)^{l-1+k} \frac{\det(A_{l-1,k}(E, X, \varphi))}{\det(A(E, X, \varphi))} = \frac{\det(A(E, X, \varphi_k))}{\det(A(E, X, \varphi))} = a_k.$$

In conclusion, the coefficients a_k, for $k = 0, \ldots, t-1$, of polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$ are computed as the sum of the partial coefficients $a_{l,k}$, where $a_{l,k}$ is computed by shareholder $s_l \in R$ and $R \in \Gamma$ is an authorized set. Importantly, this also implies that $f(x) = \sum_{l=1}^{r} f_l(x)$, where $f(x) = \sum_{l=1}^{r} f_l(x) = \sum_{l=1}^{r} \sum_{k=0}^{t-1} a_{l,k} x^k$. □

In the following, the notation defined above holds. That is, $a_{l,k}$ is the partial information held by shareholder s_l about the coefficient a_k of polynomial $f(x)$ and $f_l(x) = \sum_{k=0}^{t-1} a_{l,k} x^k$ is the partial Birkhoff interpolation polynomial of shareholder s_l. Note that Theorem 3.2 implies that also derivatives of polynomial $f(x)$ can be computed in a distributed fashion.

Theorem 3.3. *The j-th derivative $f^j(x)$ of polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$ can be computed in distributed fashion as*

$$f^j(x) = \sum_{l=1}^{r} f_l^j(x),$$

where $f_l^j(x)$ is computed by shareholder $s_l \in R$, for $l = 1, \ldots, r$ and $R \in \Gamma$ is an authorized subset of S, with $r =: |R|$.

Proof. To compute the derivative of polynomial $f(x)$ each shareholder $s_l \in R$ first computes its partial Birkhoff interpolation polynomial $f_l(x) = \sum_{k=0}^{t-1} a_{l,k} x^k$. Then it computes the j-th derivative $f_l^j(x) = \sum_{k=j}^{t-1} \frac{k!}{(k-j)!} a_{l,k} x^{k-j}$. Note that due to the sum rule for derivatives, i.e. $(f(x) + g(x))' = f(x)' + g(x)'$, and $f(x) = \sum_{l=1}^{r} f_l(x)$ the j-th derivative $f^j(x)$ of polynomial $f(x)$ can be computed by adding all partial derivatives, i.e. $f^j(x) = \sum_{l=1}^{r} f_l^j(x)$. □

3.2.3 Providing Dynamic and Verifiable Hierarchical Secret Sharing Scheme

In this section, we provide verifiable and dynamic hierarchical secret sharing. That is, we provide algorithm Add and algorithm Reset for Tassa's hierarchical conjunctive and disjunctive secret sharing schemes that generate audit data to verify the shares computed. The verification process is described using Feldman commitments presented in [46]. However, it can easily be adapted to Pedersen commitments presented in [93] to achieve information-theoretic confidentiality. Like in Section 2.2, we focus on conjunctive hierarchical secret sharing and show the differences to disjunctive hierarchical secret sharing in brackets.

Let Γ be an access structure arranged in disjoint levels $L_0, \ldots, L_\ell$, where t_h is the threshold of level L_h for $h = 0, \ldots, \ell$. Let us assume a message space $\mathcal{M}$, a space of shares Σ, and a set of shareholders S where the pair $(i, j) \in \mathcal{I} \times \mathcal{I}$ is the unique ID of shareholder $s_{i,j} \in S$, such that $j = t_{h-1}$ ($j = t_\ell - t_h$) and $t_{-1} = 0$. Then the

algorithms Share, Add, Reset, and Reconstruct for *verifiable and dynamic conjunctive (disjunctive) secret sharing* are defined as follows.
Share takes as input a message $m \in \mathcal{M}$. This algorithm works like the one discussed in Section 2.2 except that some additional audit data is computed and distributed. More precisely, the algorithm randomly chooses two large primes p, q, such that $q|(p-1)$. Let g be a generator of the q-th order subgroup $\mathbb{F}_q$ of $\mathbb{F}_p^*$ and set $\mathcal{M} := \mathbb{F}_q$. After defining the polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$, where $a_0 := m$ ($a_{t-1} := m$) and $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$ ($a_0, \ldots, a_{t-2} \in \mathbb{F}_q$) are chosen uniformly at random, the dealer commits to each coefficient a_k by computing $c_k := g^{a_k} \mod p$, for $k = 0, \ldots, t-1$. It broadcasts the commitments and sends each share $\sigma_{i,j}$ to shareholder $s_{i,j} \in L_h$, for $i = 1, \ldots, n_h$ and $h = 0, \ldots, \ell$ using a private channel. Shareholder $s_{i,j}$ accepts $\sigma_{i,j}$ as its valid share, if and only if

$$g^{\sigma_{i,j}} \equiv \prod_{k=j}^{t-1} c_k^{\frac{k!}{(k-j)!} i^{k-j}} = g^{f^j(i)}.$$

Add takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders and the ID (i', j') of the new shareholder. If R is unauthorized, i.e. $R \notin \Gamma$, it outputs $\perp$. Otherwise, $R \in \Gamma$ and the shareholders compute $\sigma_{i',j'} := f^{j'}(i')$ in distributed fashion. More precisely, each shareholder $s_l \in R$ performs the following steps, for $l = 1, \ldots, r$.

1. It computes the j'-th derivative of its partial Birkhoff interpolation polynomial at $x = i'$, i.e.

$$\lambda_{l,(i',j')} := \sigma_l \sum_{k=j'}^{t-1} \frac{k!}{(k-j')!} (-1)^{l-1+k} \frac{\det(A_{l-1,k}(E, X, \varphi))}{\det(A(E, X, \varphi))} i'^{k-j'}.$$

2. It randomly splits the result into r values, i.e. $\lambda_{l,(i',j')} = \delta_{1,l,(i',j')} + \cdots + \delta_{r,l,(i',j')}$ and sends $\delta_{m,l,(i',j')}$ to shareholder $s_{m,j} \in R$, for $m = 1, \ldots, r$ and $m \neq l$ using a private channel.
3. It collects all values $\delta_{l,m,(i',j')}$ received and computes $\delta_{l,(i',j')} := \sum_{m=1}^{r} \delta_{l,m,(i',j')}$.
4. It sends $\delta_{l,(i',j')}$ to the new shareholder $s_{i',j'}$ using a private channel and broadcasts the audit data $c_0, \ldots, c_{t-1}$ received during the share algorithm.

The new shareholder $s_{i',j'}$ computes its share $\sigma_{i',j'}$ by adding all values $\delta_{l,(i',j')}$ received, i.e. $\sigma_{i',j'} := \sum_{l=1}^{r} \delta_{l,(i',j')}$. It can verify the correctness of its share by checking whether

$$g^{\sigma_{i',j'}} \equiv \prod_{k=j'}^{t-1} c_k^{\frac{k!}{(k-j')!} i'^{k-j'}} = g^{f^{(j')}(i')},$$

using the audit data received from the shareholders.

Reset takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders a new set of shareholders $S' = \{s'_1, \ldots, s'_{n'}\}$, each accompanied with a unique ID (i', j'), and an access structure $\Gamma' \subset \mathcal{P}(S')$ with maximal threshold t'. If R is unauthorized, i.e. $R \notin \Gamma$, it outputs $\perp$. Otherwise, $R \in \Gamma$ and the subset of old shareholders jointly computes shares for the new shareholders in S'. More precisely, each old shareholder $s_l \in R$ performs the following steps, for $l = 1, \ldots, r$.

1. It computes its partial Birkhoff interpolation coefficient

$$a_{l,0} := \sigma_l (-1)^{l-1} \frac{\det(A_{l-1,0}(E, X, \varphi))}{\det(A(E, X, \varphi))}$$

   $\left(a_{l,t-1} = \sigma_l (-1)^{l+t-2} \frac{\det\left(A_{l-1,t-1}(E,X,\varphi)\right)}{\det(A(E,X,\varphi))}\right)$.
2. It chooses a polynomial $f'_l(x) = a'_{l,0} + a'_{l,1}x + a'_{l,2}x^2 + \cdots + a'_{l,t'-1}x^{t'-1}$ of degree $t' - 1$, where $a'_{l,0} = a_{l,0}$ $(a'_{l,t-1} = a_{l,t-1})$ is the partial Birkhoff interpolation coefficient and coefficients $a'_{l,1}, \ldots, a'_{l,t'-1} \in \mathbb{F}_q$ $(a'_{l,0}, \ldots, a'_{l,t'-2} \in \mathbb{F}_q)$ are chosen uniformly at random.
3. It computes subshare $\sigma_{l,(i',j')}$ for shareholder $s'_{i',j'} \in S'$ as $\sigma_{l,(i',j')} := f'^{j'}_l(i')$.
4. It sends subshare $\sigma_{l,(i',j')}$ to shareholder $s'_{i',j'} \in S$ using a private channel and broadcasts the audit data, composed of commitments to each coefficient of polynomial $f'_l(x)$, i.e. $c'_{l,k} := g^{a'_{l,k}}$, for $k = 0, \ldots, t' - 1$, and commitment $c_0 = g^m$ $(c_{t-1} = g^m)$ of the old polynomial $f(x)$.
5. It deletes its share.

Each new shareholder $s_{i',j'} \in S'$ computes its share $\sigma'_{i',j'}$ adding all subshares $\sigma_{l,(i',j')}$ received, i.e. $\sigma'_{i',j'} := \sum_{l=1}^{r} \sigma_{l,(i',j')}$. To verify the correctness of share $\sigma_{l,(i',j')}$, each new shareholder $s_{i',j'} \in S'$ performs the following steps.

1. It checks the function value of each polynomial, i.e.

$$g^{\sigma_{l,(i',j')}} \equiv \prod_{k=j'}^{t'-1} c'^{\frac{k!}{(k-j')!} i'^{k-j'}}_{l,k} = g^{f'^{(j')}_l(i')}, \text{ for } l = 1, \ldots, r.$$

2. It checks whether the free coefficient (last coefficient) of all polynomials $f'_l(i')$ leads to the original message $m \in \mathcal{M}$, i.e.

$$c_0 \equiv \sum_{l=1}^{r} c'_{l,0}$$

   $(c_{t-1} \equiv \sum_{l=1}^{r} c'_{l,t'-1})$.
3. If both equations are satisfied, it accept $\sigma'_{i',j'}$ as its valid share.

Reconstruct takes as input shares held by a subset $R \subset S$ of shareholders. If $R \in \Gamma$, it outputs $m \in \mathcal{M}$ reconstructed using Birkhoff interpolation. It outputs $\perp$ otherwise. Having access to the original audit data $c_0 = g^{a_0}$ $(c_{t-1} = g^{a_{t-1}})$ it is possible to verify whether the reconstructed message $m \in \mathcal{M}$ is a correct opening value for commitment c_0 (c_{t-1}), i.e. $g^m \equiv c_0$ $(g^m \equiv c_{t-1})$.

3.2.4 Correctness and Security

Conjunctive secret sharing has been introduced by Tassa in [107] and it has been proven correct and perfectly secure. We argue that the algorithms Add and Reset we introduced enhance the scheme and do not affect the properties and the security of the original conjunctive secret sharing scheme. To prove that, we first provide a high level idea of the proof of perfect security and accessibility of Tassa's conjunctive secret sharing scheme. Then, we show that our dynamic hierarchical secret sharing scheme maintains perfect security and accessibility, according to Definition 2.1. Furthermore, it is possible to cope with malicious dealers and shareholders including a verification protocol to the algorithm Share, Add, Reset, and Reconstruct. If Pedersen commitments are used in the verification protocol, then unconditional hidingness is maintained while bindingness can only be achieved computationally. Feldmann commitments instead ensure unconditional bindingness, i.e. the correctness of the shares can be guaranteed, but at he expenses of providing only computational hidingness for the shares. Thus, the latter solution is not suitable if data is processed for which long-term or even everlasting confidentiality is required. Similarly, it can be proven that Add and Reset maintain also the same properties of disjunctive secret sharing. However, for readability in the following we focus on conjunctive secret sharing only.

Roughly speaking, reconstructing a distributed message is equal to finding a solution of the Birkhoff interpolation problem for a polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$. Thus, Tassa proved the security of his approach by showing that authorized sets of shareholders $A \in \Gamma$ lead to interpolation matrices E for which the Birkhoff interpolation problem is well posed. Thus, accessibility is provided. Furthermore, any unauthorized set of shareholders $U \notin \Gamma$ leads to an unsolvable system and perfect security is therefore proven.

The introduction of the protocols Add and Reset making the Birkhoff interpolation based secret sharing scheme dynamic does not affect these properties. First, we show that accessibility and perfect security is provided if all shareholders act honestly. This corresponds to the setup of Tassa's security proof. Second, we prove that our scheme even provides verifiability, i.e. can cope with malicious dealers and shareholders.

Theorem 3.4. *The dynamic secret sharing scheme composed of the protocols Share, Add, Reset, and Reconstruct described in Section 3.2.3 is accessible and perfectly secure according to Definition 2.1.*

Proof. The proof for the algorithms Share and Reconstruct follows from Tassa's security proof. The algorithms Add and Reset are discussed individually in the following.

Add If the shareholders follow the protocol correctly, then all shareholders, meaning the old set of shareholders together with the new shareholder, only hold shares of the polynomial $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$ or of one of its derivatives.

This prevents unauthorized subsets from reconstructing the message, meaning that perfect security is achieved. However, the share $\sigma_{i',j'}$ for the new shareholder $s_{i',j'}$ is generated by old shareholders in distributed fashion. More precisely, each old shareholder uses its share to generate a piece of information from which the new shareholder $s_{i',j'}$ can compute its own share $\sigma_{i',j'}$. Therefore, what is left to show is that no information about the other shares is leaked during the generation of the share $\sigma_{i',j'}$. To compute the share of a new shareholder $s_{i',j'}$ each shareholder $s_l \in A$ of an authorized subset $A \in \Gamma$ computes $f_l^{j'}(i')$, where $f_l^{j'}(x)$ is the j'-th derivative of the polynomial $f_l(x)$. Note that this value leaks information about the share of s_l, since $f_l^{j'}(i') = \sigma_l \sum_{k=j'}^{t-1} \frac{k!}{(k-j')!} \frac{(-1)^{l-1+k} \det\left(A_{l-1,k}(E,X,\varphi)\right)}{\det(A(E,X,\varphi))} i'^{k-j'}$ and the latter part $\sum_{k=j'}^{t-1} \frac{k!}{(k-j')!} \frac{(-1)^{l-1+k} \det\left(A_{l-1,k}(E,X,\varphi)\right)}{\det(A(E,X,\varphi))} i'^{k-j'}$ can be computed from public information. Thus, it generates shares to this value using an additive secret sharing scheme (see the work by Doganay et al. in [43]), i.e. computes $f_l^{j'}(i') = \sum_{k,s_k \in A} \delta_{k,l,(i',j')}$, and sends $\delta_{k,l,(i',j')}$ to shareholder $s_k \in A$. Each shareholder s_l then adds all subshares received by the other shareholders, i.e. $\delta_{l,(i',j')} = \sum_{k,s_k \in A} \delta_{l,k,(i',j')}$, and forwards only the result $\delta_{l,(i',j')}$ to the new shareholder. Due to the use of the additive secret sharing scheme perfect security of all shares remains preserved.

Since $\sum_{l,s_l \in A} \delta_{l,(i',j')} = \sum_{l,s_l \in A} \sum_{k,s_k \in A} \delta_{k,l,(i',j')} = \sum_{k,s_k \in A} f_l^{j'}(i') = f^{j'}(i')$ also accessibility is provided. This ensures that the new shareholder holds together with the other shareholders a point of polynomial $f(x)$ or of one of its derivatives and the shares of authorized subsets including the new shareholders can reconstruct the message.

Reset In this algorithm each shareholder $s_l \in A$ of an authorized subset $A \in \Gamma$ uses hierarchical secret sharing to distribute its share to a new set of shareholders. More precisely, it computes its partial Birkhoff interpolation coefficient

$$a_{l,0} := \sigma_l (-1)^{l-1} \frac{\det(A_{l-1,0}(E,X,\varphi))}{\det(A(E,X,\varphi))}$$

of coefficient a_0 and then chooses a polynomial $f'_l(x) = a'_{l,0} + a'_{l,1}x + a'_{l,2}x^2 + \cdots + a'_{l,t'-1}x^{t'-1}$, where $a'_{l,0} = a_{l,0}$, containing this value in the free coefficient. In this way, shares of shares are sent to the new shareholders, since only one point of this polynomial or of one of its derivatives is sent. Therefore, perfect security follows from the perfect security of conjunctive secret sharing. Furthermore, it computes the value to be sent to a new shareholder in accordance to the new access structure and the IDs assigned to each new shareholder. Thus, any unauthorized subset $U \notin \Gamma$ cannot reconstruct the message and perfect security is provided.

Accessibility of this protocol is provided due to the homomorphic property of polynomials. More precisely each new shareholder $s_{i,j}$ receives from each old shareholder s_l share $f'^j_l(i)$ of polynomial $f'_l(x) = a'_{l,0} + a'_{l,1}x + a'_{l,2}x^2 + \cdots + a'_{l,t'-1}x^{t'-1}$, where $a'_{l,0} = a_{l,0}$

is the partial Birkhoff interpolation coefficient of a_0. Since the new shareholder adds all shares received to compute its new share it follows that it holds a point of polynomial $f'(x) = \sum_{l,s_l \in A} f'_l(x) = \sum_{l,s_l \in A}(a'_{l,0} + a'_{l,1}x + \cdots + a'_{l,t'-1}x^{t'-1}) = \sum_{l,s_l \in A} a'_{l,0} + \sum_{l,s_l \in A} a'_{l,1} + \cdots + \sum_{l,s_l \in A} a'_{l,t'-1}x^{t'-1} = a_0 + \sum_{l,s_l \in A} a'_{l,1} + \cdots + \sum_{l,s_l \in A} a'_{l,t'-1}x^{t'-1}$ or of one of its derivatives. So the free coefficient of $f'(x)$ is still a_0, meaning that any authorized subset of the new access structure is still able to retrieve message $a_0 = m$. □

Next we show that our verifiable and dynamic hierarchical secret sharing scheme provides verifiability. According to Definition 2.3, a honest majority of shareholders is assumed. This majority can be identified during Add by checking who reports the same set of commitments to function $f(x)$ and during Reset by checking who reported the same commitments c_0 to the free coefficient of $f(x)$.

Theorem 3.5. *In the presence of a majority of trustworthy shareholders within an authorized subset the verifiable and dynamic secret sharing scheme composed of the protocols Share, Add, Reset, and Reconstruct described in Section 3.2.3 is a verifiable secret sharing scheme according to Definition 2.3.*

Proof. To prove that each authorized subset of shareholders $A \in \Gamma$ reconstruct the same message $a_0 = m$ each shareholder must hold a point of the to-be-found polynomial $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$ or of one of its derivatives. Furthermore, each shareholder must hold the point assigned to its ID $(i, j) \in \mathcal{I} \times \mathcal{I}$, i.e. must receive share $\sigma_{i,j} = f^j(i)$, where $f^j(x)$ is the j-th derivative of the polynomial $f(x)$. In the following, we show for each algorithm that generates shares, i.e. Share, Add, and Reset, that the shareholders receiving these shares are able to verify these conditions.
Share During this algorithm the dealer commits to each coefficient a_k of $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$ by computing a commitment $c_k := g^{a_k} \mod p$, for $k = 0, \ldots, t-1$. It broadcasts the commitments and sends each share $\sigma_{i,j}$ to shareholder $s_{i,j} \in L_h$, for $i = 1, \ldots, n_h$ and $h = 0, \ldots, \ell$. If shareholder $s_{i,j}$ accepts $\sigma_{i,j}$ then the following equation holds

$$g^{\sigma_{i,j}} \equiv \prod_{k=j}^{t-1} c_k^{\frac{k!}{(k-j)!} i^{k-j}} = g^{f^j(i)}.$$

From this it follows directly that incorrect shares can be detected and rejected.

Add During this algorithm the shareholders $s_l \in A$ of an authorized subset $A \in \Gamma$ compute share $\sigma_{i',j'}$ for a new shareholder $s_{i',j'} \in S$ in distributed fashion. Furthermore, each shareholder broadcasts the commitments to the coefficients $c_k := g^{a_k} \mod p$, for $k = 0, \ldots, t-1$ received from the dealer. Under the assumption that at

least a majority of these shareholders is honest the new shareholder has access to a correct set of commitments and can verify whether

$$g^{\sigma_{i',j'}} \equiv \prod_{k=j'}^{t-1} c_k^{\frac{k!}{(k-j')!} i'^{k-j'}} = g^{f^{j'}(i')}.$$

From this it follows directly that incorrect shares can be detected and rejected.

Reset During this algorithm the shareholders $s_l \in A$ of an authorized subset $A \in \Gamma$ compute shares for a set of new shareholders $S' = \{s'_1, \ldots, s'_{n'}\}$, each accompanied with a unique ID $(i', j') \in \mathcal{I} \times \mathcal{I}$, and an access structure $\Gamma' \subset \mathcal{P}(S')$. Like for the other algorithms it has to be checked that share $\sigma_{i',j'}$ for the shareholder $s'_{i',j'} \in S'$ with ID $(i', j') \in \mathcal{I} \times \mathcal{I}$ are computed as $f'^{j'}(i')$. However, this algorithm has an additional requirement for correctness. The free coefficient of the to-be-found polynomial must be equal to the message m distributed by the dealer. To verify the first condition each shareholder $s_{i',j'}$ of the new access structure checks

$$g^{\sigma_{l,(i',j')}} \equiv \prod_{k=j'}^{t'-1} c'_{l,k}{}^{\frac{k!}{(k-j')!} i'^{k-j'}} = g^{f'^{j'}_l(i')}, \text{ for } s_l \in A,$$

for each share $\sigma_{l,(i',j')}$ received from shareholder l of the old set of shareholders. Finally, it checks that the sum of all shares is a point of a polynomial with free coefficient $a_0 = m$. This can be verified by multiplying all commitments to the individual free coefficients, i.e.

$$c_0 \equiv \prod_{l,s_l \in A} c'_{l,0} = \prod_{l,s_l \in A} g^{a_{l,0}} = g^{a_0} = g^m.$$

Under the assumption that a majority of the old shareholders sent the correct commitments incorrect shares can be detected.

□

Note that our scheme is also ideal, as defined in Section 2.1. This clearly comes from the fact that each shareholder $s_i \in R$ receives a share $\sigma_{i,j} \in \mathbb{F}_q$ that is a field element of the same field as the message $m \in \mathbb{F}_q$.

3.3 Performing Computations on Hierarchically Shared Secrets

We recall that the overall goal of this chapter is to design distributed storage systems based on hierarchical secret sharing schemes so that the rules to access the outsourced data mirror the hierarchical organizations in place in companies or hospitals. Distributed storage systems are normally built on Shamir's threshold

secret sharing scheme, which is a linear scheme based on polynomials. Shamir's secret sharing scheme-based distributed storage systems come with functionalities that are desirable in a long-term storage setting. For instance, they allow to modify the access rules, to change the set of storage servers making up the system, to renew the shares, and to perform computations over shared data without reconstructing them first. Tassa's hierarchical conjunctive and disjunctive secret sharing schemes are promising candidates for hierarchical secret sharing-based distributed storage systems, because they are linear and based on polynomials. However, in order to have viable hierarchical secret sharing-based distributed storage systems, it is necessary to have the same functionalities that regular distributed storage systems already offer. In Section 3.2, we have provided algorithms for modifying the access rules, for changing the set of storage servers making up the storage system, and for renewing the shares.

In this section, we provide algorithms to perform computations over messages shared with Tassa's conjunctive and disjunctive hierarchical secret sharing schemes. More precisely, a message can be reconstructed which is the result of operations performed over previously shared messages. The operations supported are the sum of messages, the multiplication of a message by a scalar, and the product of messages. After describing in Section 3.3.1 the setting we operate in, in Section 3.3.2 and in Section 3.3.3 we provide the algorithms to perform, respectively, linear operations and multiplications over hierarchically shared messages by Tassa's conjunctive and disjunctive schemes. The auditing mechanism for both types of computations can be found in Section 3.3.4.

3.3.1 Setting

Messages $m_1, m_2 \in \mathbb{F}_q$ are distributed to a set S of n shareholders according to the following assumptions.

(A1) The underlying access structure Γ remains the same for both messages m_1, m_2. More precisely, both polynomials $f(x)$ and $h(x)$ used to share m_1 and m_2, respectively, have the same degree. Furthermore, shareholder $s_{i,j}$ with unique ID (i, j) holds share $\sigma_{i,j}(m_1) := f^j(i)$ and $\sigma_{i,j}(m_2) := h^j(i)$.

(A2) The degree $t-1$ of polynomials $f(x)$ and $h(x)$ is chosen such that $2t \leq n$, where n is the total number of shareholders.

(A3) The ID (i, j) of each shareholder $s_{i,j} \in S$ is chosen such that index $i \in \mathcal{I}$ is selected once within the whole hierarchy and such that the corresponding Birkhoff interpolation problem has a unique solution. Also $j < \ell$, i.e. the free coefficients of polynomials $f^\ell(x)$ and $h^\ell(x)$ are not valid shares. The requirements to achieve this can be found in [107].

(A4) The user communicates with the shareholders and the shareholders among each other using private channels.

(A5) A tamper-proof bulletin board is available to allow exchanging data during the preprocessing phase of the multiplication procedure. Note that this is a common assumption for auditable multi-party computation and a more formal definition can be found in [61].

Let us recall that index $j \in \mathcal{I}$ of the unique identity ID of shareholder $s_{i,j} \in S$ is defined as $j := t_{h-1}$ ($j := t_\ell - t_h$), for $h = 0, \ldots, \ell$ and $t_{-1} := 0$. Furthermore, we assume that algorithm Share (presented in Section 2.2) is run separately to share and distribute message m_1 and message m_2 to the n shareholders of set S. More precisely, to share message m_1, algorithm Share selects a polynomial $f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1}$, where $a_0 := m_1$ ($a_{t-1} := m_1$) and $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$ ($a_0, \ldots, a_{t-2} \in \mathbb{F}_q$) are chosen uniformly at random. It distributes to each shareholder $s_{i,j} \in S$ share $\sigma_{i,j}(m_1) = f^j(i)$. To share message m_2, algorithm Share generates a polynomial $h(x) = b_0 + b_1 x + \cdots + b_{t-1}x^{t-1}$, where $b_0 := m_2$ ($b_{t-1} := m_2$) and $b_1, \ldots, b_{t-1} \in \mathbb{F}_q$ ($b_0, \ldots, b_{t-2} \in \mathbb{F}_q$) are chosen uniformly at random. It distributes to each shareholder $s_{i,j} \in S$ share $\sigma_{i,j}(m_2) = h^j(i)$. Afterwards, algorithms LinAdd and Multiply are run by each shareholder individually to perform linear operations and multiplications on their shares of messages m_1 and m_2. Finally, the result $m \in \mathbb{F}_q$ of these operations on m_1, m_2 can be reconstructed by running algorithm Reconstruct defined in Section 2.2 on the shares computed by each shareholder.

3.3.2 Linear Operations

In this section, algorithm LinAdd is presented, which computes share $\sigma_{i,j}(m) \in \mathbb{F}_q$ for shareholder $s_{i,j} \in S$, to be used as input for algorithm Reconstruct to retrieve message $m = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2$, for scalars $\lambda_1, \lambda_2 \in \mathbb{F}_q$. Correctness and perfect secrecy of algorithm LinAdd are proven in Theorem 3.6 below.

LinAdd The algorithm takes as input shares $\sigma_{i,j}(m_1), \sigma_{i,j}(m_2) \in \mathbb{F}_q$ held by shareholder $s_{i,j} \in S$, and scalars $\lambda_1, \lambda_2 \in \mathbb{F}_q$. It outputs share $\sigma_{i,j}(m) := \lambda_1 \cdot \sigma_{i,j}(m_1) + \lambda_2 \cdot \sigma_{i,j}(m_2) \in \mathbb{F}_q$ for shareholder $s_{i,j} \in S$.

Theorem 3.6. *The algorithm LinAdd for conjunctive (disjunctive) hierarchical secret sharing introduced above satisfies accessibility and perfect security according to Definition 2.1. More precisely, on input shares $\sigma_{i,j}(m_1), \sigma_{i,j}(m_2)$ and scalars λ_1, λ_2, the shares computed by LinAdd reconstruct to message m, where $m = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2$.*

Proof. Let $\sigma_{i,j}(m) \in \mathbb{F}_q$ be the shares computed by shareholders $s_{i,j} \in R$ using algorithm LinAdd, where $R \in \Gamma$ is an authorized set. To prove correctness, we have to show that algorithm Reconstruct outputs message $m = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2$ when it takes as input shares $\sigma_{i,j}(m) \in \mathbb{F}_q$. More precisely, we have to show that the shares interpolate to a polynomial $p(x) = c_0 + c_1 x + \cdots + c_{t-1}x^{t-1}$ of degree

$\deg(p(x)) = t-1$, where $c_0 = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2 (c_{t-1} = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2)$. To prove perfect secrecy, we have to show, first, that algorithm LinAdd computes shares for message $m = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2$ without leaking information about the shares for message m_1 and message m_2. Second, we have to show that any unauthorized set $U \notin \Gamma$ gets no information about $m = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2$. In order to do that, we have to show that polynomial $p(x) = c_0 + c_1 x + \cdots + c_{t-1} x^{t-1}$ can be computed in distributed fashion by each shareholder $s_{i,j} \in R$. That is, correctness and perfect secrecy hold if each shareholder can compute a term $p_{(i,j),k}$ without leaking information to any other shareholder and such that:

$$p(x) = \sum_{k=0}^{t-1} c_k x^k = \sum_{k=0}^{t-1} \sum_{s_{i,j} \in R} p_{(i,j),k} x^k,$$

where $c_0 = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2 (c_{t-1} = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2)$.

Let us recall that message $m_1 \in \mathbb{F}_q$ is shared using polynomial $f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$. Due to Birkhoff interpolation resolution formula (see Section 2.2), coefficient a_k of polynomial $f(x)$ can be computed as:

$$a_k = \sum_{l=1}^{r} a_{l,k} = \sum_{l=1}^{r} \sigma_l(m_1)(-1)^{l-1+k} \frac{\det(A_{l-1,k}(E, X, \varphi))}{\det(A(E, X, \varphi))},$$

for $k = 0, \ldots, t-1$, where $\sigma_l(m_1)$, for $l = 1, \ldots, r$, are the shares $\sigma_{i,j}(m_1)$ in lexicographic order ((i,j) precedes the pair (i',j') if $i < i'$ or $i = i'$ and $j < j'$). Similarly, message $m_2 \in \mathbb{F}_q$ is shared through polynomial $h(x) = b_0 + b_1 x + \cdots + b_{t-1} x^{t-1}$. Due to Birkhoff interpolation resolution formula, coefficient b_k of polynomial $h(x)$ can be computed as:

$$b_k = \sum_{l=1}^{r} b_{l,k} = \sum_{l=1}^{r} \sigma_l(m_2)(-1)^{l-1+k} \frac{\det(A_{l-1,k}(E, X, \varphi))}{\det(A(E, X, \varphi))},$$

for $k = 0, \ldots, t-1$, where $\sigma_l(m_2)$, for $l = 1, \ldots, r$, are the shares $\sigma_{i,j}(m_2)$ in lexicographic order. Because of the homomorphic property of polynomials, polynomial $p(x)$ can be computed as the linear combination of polynomial $f(x)$ and polynomial $h(x)$ with scalars $\lambda_1, \lambda_2 \in \mathbb{F}_q$. That is, $p(x) = \lambda_1 \cdot f(x) + \lambda_2 \cdot h(x)$. Therefore,

$$p(x) = \sum_{k=0}^{t-1} \lambda_1 \cdot a_k + \lambda_2 \cdot b_k = \sum_{k=0}^{t-1} \sum_{l=1}^{r} \lambda_1 \cdot a_{l,k} + \lambda_2 \cdot b_{l,k}.$$

This shows that the terms $p_{l,k} = p_{(i,j),k} := \lambda_1 \cdot a_{l,k} + \lambda_2 \cdot b_{l,k}$ computed by the shareholders $s_{i,j} \in R$ interpolate to polynomial $p(x)$ and correctness is provided. Regarding perfect secrecy, the computation of $p_{l,k}$ is performed solely by shareholder $s_l \in R$ using the information it has and without leaking $a_{l,k}$ nor $b_{l,k}$. Thus, no information about shares $\sigma_l(m_1), \sigma_l(m_2)$ is leaked. Moreover, being polynomial $p(x)$ of degree $\deg(p(x)) = t-1$, the original access structure Γ is maintained: subsets $U \subset S$

of shareholders such that $U \notin \Gamma$ not only cannot reconstruct $m = \lambda_1 \cdot m_1 + \lambda_2 \cdot m_2$, but also do not get any information about m_1 nor m_2. Thus, perfect secrecy of the underlying conjunctive (disjunctive) hierarchical secret sharing is still maintained even if algorithm LinAdd is run and the shares computed by this algorithm are used as input for algorithm Reconstruct.

3.3.3 Multiplication

In this section, it is presented how to perform multiplications on shared messages, i.e. how to compute share $\sigma_{i,j} \in \mathbb{F}_q$ for shareholder $s_{i,j} \in S$ to be used as input for algorithm Reconstruct to retrieve message $m = m_1 \cdot m_2$. Unlike the counterpart for linear operations, algorithm Multiply requires more work to be introduced. In particular, Section 3.3.3.1 shows how to compute preliminary shares to be used as input of the preprocessing phase presented in Section 3.3.3.2. These two sections describe operations that can be computed off-line in order to simplify algorithm Multiply, which is performed on-line and is discussed in Section 3.3.3.3.

3.3.3.1 Computation of shares $\sigma_{i,j}(\alpha), \sigma_{i,j}(\beta)$

During the off-line phase a triple (α, β, γ) is generated such that the following conditions hold.

- $\alpha \cdot \beta = \gamma$.
- Assumption (1) of Section 3.3.1 holds, i.e. each shareholder $s_{i,j} \in S$ with ID $(i,j) \in \mathcal{I} \times \mathcal{I}$ holds shares $\sigma_{i,j}(\alpha) := f_\alpha^j(i), \sigma_{i,j}(\beta) := f_\beta^j(i)$, and $\sigma_{i,j}(\gamma) := f_\gamma^j(i)$, where $f_\alpha(x), f_\beta(x)$, and $f_\gamma(x)$ are the polynomials of degree $t-1$ sharing α, β, and γ, respectively.

In the following, we present algorithm RandShares, which computes such a triple for Tassa's conjunctive (disjunctive) hierarchical secret sharing schemes. More precisely, algorithm RandShares computes random shares $\sigma_{i,j}(\alpha), \sigma_{i,j}(\beta)$ reconstructing to messages α, β, respectively. This algorithm is based on the technique used in [40] by Damgård and Nielsen to generate such a triple for Shamir's threshold secret sharing scheme.

We present RandShares to compute shares $\sigma_{i,j}(\alpha)$ for α, but it can be run analogously to generate shares $\sigma_{i,j}(\beta)$ for β.

RandShares The algorithm takes as input values $\alpha_{i,j} \in \mathbb{F}_q$ chosen uniformly at random by shareholders $s_{i,j} \in S$. It outputs shares $\sigma_{i,j}(\alpha)$ of message $\alpha \in \mathbb{F}_q$ for shareholders $s_{i,j} \in S$. To do that, each shareholder $s_{i,j} \in S$ has to perform the following steps.

1. It chooses a secret message $\alpha_{i,j} \in \mathbb{F}_q$ uniformly at random.

2. It runs algorithm Share to generate a polynomial $f_{\alpha_{i,j}}(x)$ of degree $t-1$ defined as $f_{\alpha_{i,j}}(x) := a_{0,(i,j)} + a_{1,(i,j)}x + \cdots + a_{t-1,(i,j)}x^{t-1}$, where $a_{0,(i,j)} = \alpha_{i,j}$ ($a_{t-1,(i,j)} = \alpha_{i,j}$) and coefficients $a_{1,(i,j)}, \ldots, a_{t-1,(i,j)} \in \mathbb{F}_q$ ($a_{0,(i,j)}, \ldots, a_{t-2,(i,j)} \in \mathbb{F}_q$) are chosen uniformly at random. Shares $\sigma_{i',j'}(\alpha_{i,j})$ for shareholders $s_{i',j'} \in S$ with ID $(i',j') \neq (i,j)$ are computed as $\sigma_{i',j'}(\alpha_{i,j}) := f^{j'}_{\alpha_{i,j}}(i')$. Share $\sigma_{i,j}(\alpha_{i,j})$ for shareholder $s_{i,j}$ itself is computed as $\sigma_{i,j}(\alpha_{i,j}) := f^{j}_{\alpha_{i,j}}(i)$.
3. It sends shares $\sigma_{i',j'}(\alpha_{i,j})$ to shareholders $s_{i',j'} \in S$ with ID $(i',j') \neq (i,j)$ using a private channel and keeps share $\sigma_{i,j}(\alpha_{i,j})$.
4. It runs algorithm LinAdd of Section 3.3.2 to compute share $\sigma_{i,j}(\alpha)$ using share $\sigma_{i,j}(\alpha_{i,j})$ and all the shares $\sigma_{i,j}(\alpha_{i',j'})$ received from shareholders $s_{i',j'}$ as $\sigma_{i,j}(\alpha) := \sum_{(i',j')\neq(i,j)} \sigma_{i,j}(\alpha_{i',j'}) + \sigma_{i,j}(\alpha_{i,j})$.

In the following, we prove correctness of algorithm RandShares and we show that perfect secrecy is provided.

Theorem 3.7. *The algorithm RandShares for conjunctive (disjunctive) hierarchical secret sharing introduced above satisfies accessibility and perfect security according to Definition 2.1. More precisely, on input random secret messages $\alpha_{i,j}$, the shares computed by algorithm RandShares reconstruct to a common value α.*

Proof. Let $\sigma_{i,j}(\alpha) \in \mathbb{F}_q$ be the shares computed using algorithm RandShares and held by shareholders $s_{i,j} \in R$, where $R \in \Gamma$ is an authorized set. To prove correctness, we have to show that algorithm Reconstruct outputs a message α when it takes as input shares $\sigma_{i,j}(\alpha)$ held by shareholders of an authorized set R. This means that correctness holds provided that algorithm Reconstruct can be successfully run by shareholders of any authorized set. This is implied by the correctness of algorithm LinAdd, presented in Section 3.3.2. In fact, each share $\sigma_{i,j}(\alpha)$ is computed as a sum of shares $\sigma_{i,j}(\alpha_{i',j'})$ and share $\sigma_{i,j}(\alpha_{i,j})$. Thus, for the homomorphic property of polynomials, shares $\sigma_{i,j}(\alpha)$ is either a point of polynomial $f_\alpha(x) := a_{0,\alpha} + a_{1,\alpha}x + \cdots + a_{t-1,\alpha}x^{t-1} = \sum_{(i,j)} f_{\alpha_{i,j}}(x)$ or a point on one of its derivatives, where $a_{0,\alpha} = \sum_{(i,j)} \alpha_{i,j}$ ($a_{t-1,\alpha} = \sum_{(i,j)} \alpha_{i,j}$). Because of the underlying conjunctive (disjunctive) hierarchical secret sharing scheme, any authorized set R of shareholders can run algorithm Reconstruct over their shares and retrieve message $\alpha := \sum_{(i,j)} \alpha_{i,j}$. This proves correctness. With respect to perfect secrecy, the underlying conjunctive (disjunctive) hierarchical secret sharing scheme guarantees that shares $\sigma_{i,j}(\alpha)$ are computed without leaking information about the secret messages $\alpha_{i,j}$. Furthermore, this implies that unauthorized sets of shareholders not only cannot successfully run algorithm Reconstruct to retrieve α, but also no information about it is gained.

3.3.3.2 Preprocessing

We introduce the preprocessing phase enabling the multiplication between two shared messages. Preprocessing has been common practice for multi-party computation

since it has been introduced by Beaver in [9], because it lowers the communication complexity of the on-line phase where the shares of the result of a product of two shared messages are computed. The preprocessing algorithm by Beaver was designed for messages shared with Shamir's secret sharing scheme. We adapt it here for messages shares with Tassa's conjunctive (disjunctive) hierarchical secret sharing schemes.

PreMult This algorithm takes as input shares $\sigma_{i,j}(\alpha), \sigma_{i,j}(\beta)$ for each shareholder $s_{i,j} \in S$ computed by RandShares described in Section 3.3.3.1 and outputs for each shareholder $s_{i,j} \in S$ a triple of shares $\sigma_{i,j}(\alpha), \sigma_{i,j}(\beta), \sigma_{i,j}(\gamma) \in \mathbb{F}_q$, such that for each triple it holds that $\sigma_{i,j}(\gamma) = \sigma_{i,j}(\alpha\beta)$. More precisely, each shareholder $s_l \in R$ from an authorized subset $R \in \Gamma$ performs the following steps.

1. It uses its shares $\sigma_l(\alpha)$ and $\sigma_l(\beta)$ and the unique ID (i,j) of shareholder $s_{i,j}$ to compute the values $\lambda^m_{l,(i,j)}$ and $\mu^m_{l,(i,j)}$ defined as:

$$\lambda^m_{l,(i,j)} := \sigma_l(\alpha) \sum_{k=m}^{j} \frac{k!}{(k-m)!}(-1)^{l-1+k} \frac{\det(A_{l-1,k}(E,X,\varphi))}{\det(A(E,X,\varphi))} i^{k-m},$$

   and

$$\mu^m_{l,(i,j)} := \sigma_l(\beta) \sum_{k=m}^{j} \frac{k!}{(k-m)!}(-1)^{l-1+k} \frac{\det(A_{l-1,k}(E,X,\varphi))}{\det(A(E,X,\varphi))} i^{k-m},$$

   where $m = 0, \ldots, j$ and $A(E,X,\varphi)$ and $A_{l-1,k}(E,X,\varphi)$ are the matrices defined in Section 2.2.
2. It randomly splits $\lambda^m_{l,(i,j)}$ and $\mu^m_{l,(i,j)}$ into r values, i.e. $\lambda^m_{l,(i,j)} = \lambda^m_{1,l,(i,j)} + \cdots + \lambda^m_{r,l,(i,j)}$ and $\mu^m_{l,(i,j)} = \mu^m_{1,l,(i,j)} + \cdots + \mu^m_{r,l,(i,j)}$.
3. It sends $\lambda^m_{u,l,(i,j)}$ and $\mu^m_{u,l,(i,j)}$ to shareholder $s_u \in R$, for $u = 1, \ldots, r$ and $u \neq l$, using a private channel.
4. It collects all values $\lambda^m_{l,u,(i,j)}$ and $\mu^m_{l,u,(i,j)}$ received from shareholder $s_u \in R$, for $u = 1, \ldots, r$ and $u \neq l$, and computes $\delta^m_{l,(i,j)} := \sum_{u=1}^{r} \lambda^m_{l,u,(i,j)}$ and $\varepsilon^m_{l,(i,j)} := \sum_{u=1}^{r} \mu^m_{l,u,(i,j)}$, for $m = 0, \ldots, j$.
5. It sends $\delta^m_{l,(i,j)}$ and $\varepsilon^m_{l,(i,j)}$ to shareholder $s_{i,j}$ using a private channel.

Then, all shareholders within the set S compute their shares. More precisely, each shareholder $s_{i,j} \in S$ performs the following steps.

1. It computes $\delta^m_{(i,j)} := \sum_{l=1}^{r} \delta^m_{l,(i,j)}$ and $\varepsilon^m_{(i,j)} := \sum_{l=1}^{r} \varepsilon^m_{l,(i,j)}$ using the values $\delta^m_{l,(i,j)}$ and $\varepsilon^m_{l,(i,j)}$, for $m = 0, \ldots, j$, received from shareholder $s_l \in R$, for $l = 1, \ldots, r$.
2. It computes share $\sigma_{i,j}(\gamma)$ as

$$\sigma_{i,j}(\gamma) := \sigma_{i,j}(\alpha\beta) = \sum_{m=0}^{j} \binom{j}{m} \delta^{j-m}_{(i,j)} \cdot \varepsilon^m_{(i,j)}.$$

Theorem 3.8. *The algorithm PreMult for conjunctive (disjunctive) hierarchical secret sharing introduced above satisfies accessibility and perfect security according*

to Definition 2.1. More precisely, on input the shares $\sigma_{i,j}(\alpha)$ and $\sigma_{i,j}(\beta)$, the shares computed by algorithm PreMult *reconstructs to γ, where $\gamma = \alpha\beta$.*

Proof. Let $\sigma_{i,j}(\alpha\beta)$ be the share computed by shareholder $s_{i,j} \in R$ using algorithm PreMult, where $R \in \Gamma$ is an authorized set. Correctness of algorithm PreMult is provided if the shares held by shareholders in R it outputs interpolate to a polynomial $p(x) = c_0 + c_1 x + \cdots + c_{2(t-1)} x^{2(t-1)}$, where $c_0 = \alpha\beta (c_{2(t-1)} = \alpha\beta)$. Polynomial $p(x)$ is defined as $p(x) = f_\alpha(x) \cdot f_\beta(x)$, given that α is shared using polynomial $f_\alpha(x)$ and β is shared using polynomial $f_\beta(x)$. We have to show that, for each share $\sigma_{i,j}(\gamma)$ computed by algorithm PreMult, it holds that $\sigma_{i,j}(\gamma) = \sigma_{i,j}(\alpha\beta)$, where $\sigma_{i,j}(\alpha)$ and $\sigma_{i,j}(\beta)$ were randomly selected from shareholder $s_{i,j}$. In this case $\sigma_{i,j}(\gamma)$ can be written as:

$$\sigma_{i,j}(\alpha\beta) = p^j(i) = [f_\alpha(i) \cdot f_\beta(i)]^j = \sum_{m=0}^{j} \binom{j}{m} f_\alpha^{j-m}(i) \cdot f_\beta^m(i)$$

The terms $f_\alpha^j(i)$ and $f_\beta^j(i)$ constitute the random values $\sigma_{i,j}(\alpha)$ and $\sigma_{i,j}(\beta)$ selected by shareholder $s_{i,j} \in S$. It is left to check that $\delta_{(i,j)}^m$ and $\varepsilon_{(i,j)}^m$ correspond to $f_\alpha^m(i)$ and $f_\beta^m(i)$, respectively. From the second step, we recall that $\delta_{l,(i,j)}^m = \sum_{u=1}^r \lambda_{l,u,(i,j)}$. Thus, it follows that:

$$\delta_{(i,j)}^m = \sum_{l=1}^{r} \delta_{l,(i,j)}^m = \sum_{l=1}^{r} \sum_{u=1}^{r} \lambda_{l,u,(i,j)} = \sum_{l=1}^{r} f_{\alpha,l}^m(i) = f_\alpha^m(i),$$

where polynomial $f_{\alpha,l}^m(x)$ is the m-th derivative of polynomial $f_{\alpha,l}(x) = \sum_{k=0}^{t-1} \alpha_{l,k} x^k$, where $\alpha_{l,k}$ is the reconstructing term of Birkhoff interpolation formula (see Section 2.2). Note that the last equality of the expression above holds because the coefficients of $f_\alpha(x)$ can be computed in distributed fashion, as shown in Theorem 3.2. The equality $\varepsilon_{(i,j)}^m = f_\beta^m(i)$ can be shown analogously. Moreover, since polynomial $p(x) = c_0 + c_1 x + \cdots + c_{2(t-1)} x^{2(t-1)}$ is the product of polynomials $f_\alpha(x)$ and $f_\beta(x)$, then $c_0 = a_0 b_0 = \alpha\beta (c_{2(t-1)} = a_{t-1} b_{t-1} = \alpha\beta)$. Thus, correctness holds. To prove perfect secrecy, we have to show that no information is leaked when share $\sigma_{i,j}(\alpha\beta)$ is generated for shareholder $s_{i,j} \in S$. Regarding the terms $\delta_{(i,j)}$ and $\varepsilon_{(i,j)}$, we have to show that they do not leak information about shares $\sigma_l(\alpha)$ and $\sigma_l(\beta)$ of shareholder $s_l \in R$, respectively. That is the case because shareholder $s_l \in R$ uses additive secret sharing, discussed by Doganay et al. in [43] to split $\lambda_{l,(i,j)}^m$ and $\mu_{l,(i,j)}^m$ into r random values $\lambda_{u,l,(i,j)}$ and $\mu_{u,l,(i,j)}$, respectively. Furthermore, perfect secrecy holds also because index $i \in \mathcal{I}$ of each identity ID $(i,j) \in \mathcal{I} \times \mathcal{I}$ is used once, as required by Assumption (A3) of Section 3.3.1. Otherwise, points $f_\alpha^m(i)$ and $f_\beta^m(i)$ might correspond to already existing shares $\sigma_{i,m}(\alpha)$ and $\sigma_{i,m}(\beta)$ for α and β, respectively, already computed for shareholder $s_{i,m} \in S$. Also, the fact that $j < \ell$ prevents information leakage, because otherwise the shareholders has knowledge about all the derivatives of $f_\alpha(i)$ and $f_\beta(i)$. Moreover, because each share $\sigma_{i,j}(\gamma)$ is

a point on polynomial $p(x)$ or on one of its derivatives, the underlying conjunctive (disjunctive) hierarchical secret sharing scheme ensures that unauthorized subsets gain no information about α, β, γ.

3.3.3.3 The Multiplication Algorithm

In this section, algorithm Multiply is presented, which computes share $\sigma_{i,j}(m)$ for shareholder $s_{i,j} \in S$ through the shares $\sigma_{i,j}(m_1)$ and $\sigma_{i,j}(m_2)$ of message m_1 and message m_2, respectively. Share $\sigma_{i,j}(m)$ is used as input for algorithm Reconstruct to retrieve message $m = m_1 \cdot m_2$. Algorithm Multiply uses algorithm LinAdd (see Section 3.3.2) to compute message m as linear combinations of the shares for message m_1 and message m_2. More precisely, it builds on the preprocessing phase performed by algorithm PreMult described in Section 3.3.3.1, which in turn builds on algorithm RandShares described in Section 3.3.3.2. Note that, according to Assumption (A1) in Section 3.3.1, for algorithm Multiply to work the values α, β, and γ have to be shared according to the same access structure Γ as message m_1 and message m_2.

Multiply The algorithm takes as input shares $\sigma_{i,j}(m_1), \sigma_{i,j}(m_2) \in \mathbb{F}_q$ generated during algorithm Share and shares $\sigma_{i,j}(\alpha), \sigma_{i,j}(\beta), \sigma_{i,j}(\gamma) \in \mathbb{F}_q$ held by shareholder $s_{i,j} \in S$ generated during algorithm PreMult. It outputs share $\sigma_{i,j}(m) \in \mathbb{F}_q$ for message $m = m_1 \cdot m_2$, which is computed performing the following steps.

1. Shareholder $s_{i,j}$ computes share $\sigma_{i,j}(\delta) := \sigma_{i,j}(m_1) - \sigma_{i,j}(\alpha)$ and share $\sigma_{i,j}(\varepsilon) := \sigma_{i,j}(m_2) - \sigma_{i,j}(\beta)$ using algorithm LinAdd.
2. Shareholders from an authorized set $R \in \Gamma$ run algorithm Reconstruct with shares $\sigma_{i,j}(\delta), \sigma_{i,j}(\varepsilon)$ as input to publicly reconstruct values δ, ε using the bulletin board.
3. Shareholder $s_{i,j} \in S$ computes the share $\sigma_{i,j}(m) := \sigma_{i,j}(\gamma) + \varepsilon \cdot \sigma_{i,j}(m_1) + \delta \cdot \sigma_{i,j}(m_2) - \delta\varepsilon$ using algorithm LinAdd.

Theorem 3.9. *The algorithm Multiply for conjunctive (disjunctive) hierarchical secret sharing introduced above satisfies accessibility and perfect security according to Definition 2.1. More precisely, on input shares $\sigma_{i,j}(m_1), \sigma_{i,j}(m_2)$, the shares computed by Multiply reconstruct to message m, where $m = m_1 \cdot m_2$.*

Proof. The correctness relies on the correctness of algorithm LinAdd, presented in Section 3.3.2. In fact, share $\sigma_{i,j}(m)$ is defined as the linear combination of shares $\sigma_{i,j}(\gamma), \sigma_{i,j}(m_1), \sigma_{i,j}(m_2)$ for messages γ, m_1, m_2, respectively, and scalars δ, ε. More precisely, in the first step the scalars δ and ε are computed in distributed fashion using algorithm LinAdd, such that $\delta = m_1 - \alpha$ and $\varepsilon = m_2 - \beta$. After those values have been reconstructed in the second step, in the third step each shareholder computes a share to message m by computing $\sigma_{i,j}(m) = \sigma_{i,j}(\gamma) + \varepsilon \cdot \sigma_{i,j}(m_1) + \delta \cdot \sigma_{i,j}(m_2) - \delta\varepsilon$ using algorithm LinAdd. Therefore, if algorithm Reconstruct takes as input shares $\sigma_{i,j}(m) \in \mathbb{F}_q$ held by shareholders $s_{i,j} \in R$, where $R \in \Gamma$ is an authorized set, then it

retrieves:

$$m = \gamma + \varepsilon \cdot m_1 + \delta \cdot m_2 - \delta\varepsilon$$

$$= \gamma + (m_2 - \beta) \cdot m_1 + (m_1 - \alpha) \cdot m_2 - (m_2 - \beta)(m_1 - \alpha)$$

$$= \gamma + m_1 \cdot m_2 - \beta \cdot \alpha$$

Since $\alpha \cdot \beta = \gamma$ this leads to

$$m = m_1 \cdot m_2,$$

showing that algorithm Multiply is correct.

Thus, algorithm Reconstruct interpolates to a polynomial $p(x) = c_0 + c_1 x + \cdots + c_{t-1}x^{t-1}$ of degree $\deg(p(x)) = t - 1$ and retrieves message $m_1 \cdot m_2$ as $c_0(c_{t-1})$. The perfect secrecy of algorithm Multiply is implied by the perfect secrecy of algorithm LinAdd (proven in Section 3.3.2) and by the perfect secrecy of the preprocessing phase, which is discussed in Section 3.3.3.2.

3.3.4 Auditing Procedure for Conjunctive (Disjunctive) Hierarchical Secret Sharing Schemes

In this section, we present auditing procedures for computations on messages shared hierarchically. More precisely, we adapt the auditing procedure for computations on messages shared by Shamir's threshold secret sharing scheme presented in [98] to computations on messages shared by Tassa's conjunctive (disjunctive) secret sharing schemes. In Section 3.3.4.1, we present algorithms Audit.Setup and Audit.Share, which describe the steps to be performed during the setup phase and after algorithm Share, respectively. Then, in Section 3.3.4.2 we present algorithm Audit.LinAdd which is run to audit algorithm LinAdd presented in Section 3.3.2. Lastly, in Section 3.3.4.3, we present algorithm Audit.RandShares, algorithm Audit.PreMult, and algorithm Audit.Multiply, run to audit, respectively, algorithm RandShares presented in Section 3.3.3.1, algorithm PreMult presented in Section 3.3.3.2, and algorithm Multiply presented in Section 3.3.3.3. For consistency with the other algorithms presented in this chapter, Feldman commitment is used. However, the algorithm can be easily adapted to Pedersen commitment. Note that the assumptions made in Section 3.3.1 still hold.

3.3.4.1 Setup and Share

Algorithm Audit.Setup sets up the cryptographic primitives, i.e. commitment schemes and bilinear maps,[2] needed for the auditing procedures. This can be run by any party. However, the parameters must be made publicly available for the dealer of the

[2]For a formal definition of bilinear maps we refer to [19].

input messages and the auditor running the auditing procedures. Then, to allow operations to be audited, the dealer commits to messages shared by running Audit.Share.

Audit.Setup The algorithm takes as input a security parameter λ and it outputs two large primes p, q such that $q|(p-1)$. It also outputs a generator g of the q-th order subgroup $\mathbb{F}_q$ of $\mathbb{F}_p^*$.

Audit.Share The dealer of messages $m_1, m_2 \in \mathbb{F}_q$ calls algorithm Commit.Share presented in Section 2.1 during algorithm Share and computes commitment $c(m_1) := g^{m_1} \mod p$ to message m_1 and commitment $c(m_2) := g^{m_2} \mod p$ to message m_2. It publishes the commitments on the bulletin board.

3.3.4.2 Linear Operations

In the following, algorithm Audit.LinAdd run by the auditor to verify the result of linear operations over shared messages is presented. We assume that either the shareholders or the message dealer published the used scalars $\lambda_1, \lambda_2 \in \mathbb{F}_q$ on the bulletin board.

Audit.LinAdd The algorithm takes as input the commitments to the input values $c(m_1), c(m_2)$ and the scalars $\lambda_1, \lambda_2 \in \mathbb{F}_q$ from the bulletin board and the claimed result m. If $g^m = c(m_1)^{\lambda_1} \cdot c(m_2)^{\lambda_2}$ it returns '1' and '0' otherwise.

3.3.4.3 Multiplication

Algorithm Audit.RandShares is run by an auditor together with the shareholders $s_{i,j} \in S$ to verify that shares $\sigma_{i,j}(\alpha)$ and $\sigma_{i,j}(\beta)$ were computed correctly by generating commitments $c_{k,\alpha}, c_{k,\beta}$, for $k = 0, \ldots, t-1$, to the coefficients of the polynomials sharing messages α, β, respectively. In the following, we present algorithm Audit.RandShares to compute commitment $c_{k,\alpha}$, but it can be run analogously to generate commitment $c_{k,\beta}$, for $k = 0, \ldots, t-1$.

Audit.RandShares The algorithm takes as input shares $\sigma_{i,j}(\alpha)$ and performs the following steps.

1. Each shareholder $s_{i,j} \in S$ running algorithm Share to share the secret message $\alpha_{i,j} \in \mathbb{F}_q$ among all other shareholders $s_{i',j'} \in S$ for $(i', j') \neq (i, j)$ calls algorithm Commit.Share and computes commitments $c_{k,\alpha_{i,j}} := g^{a_{k,(i,j)}} \mod p$, to coefficient $a_{k,(i,j)}$ of polynomial $f_{\alpha_{i,j}}(x)$, for $k = 0, \ldots, t-1$. It publishes the commitments on the bulletin board.
2. Each shareholder $s_{i,j} \in S$ has valid input $\sigma_{i,j}(\alpha_{i',j'})$, for $(i', j') \neq (i, j)$, to compute share $\sigma_{i,j}(\alpha)$ if and only if

$$g^{\sigma_{i,j}(\alpha_{i',j'})} \equiv \prod_{k=j}^{t-1} c_{k,\alpha_{i',j'}}{}^{\frac{k!}{(k-j)!} i^{k-j}} = g^{f^{j}_{\alpha_{i',j'}}(i)}.$$

If the above equality is not satisfied, then it outputs '0' and aborts. Otherwise, it publishes '1' on the bulletin board and Step 3) can be performed.

3. The auditor uses commitments $c_{k,\alpha_{i,j}}$ published by shareholders $s_{i,j} \in S$ on the bulletin board to compute commitments $c_{k,\alpha} := \prod_{(i,j)} c_{k,\alpha_{i,j}}$, for $k = 0, \ldots, t-1$. It publishes the commitments on the bulletin board.

In the following, algorithm Audit.PreMult is run by an auditor together with shareholders $s_{i,j} \in S$ to check the validity of terms $\delta_{l,i,j}$ and $\varepsilon_{l,i,j}$ for the computation of shares $\sigma_{i,j}(\alpha\beta)$. More precisely, it is explained what audit data have to be generated such that shareholder $s_{i,j}$ can detect inconsistent input sent by other malicious shareholders during algorithm PreMult performing the preprocessing phase. In fact, algorithm PreMult is performed in distributed fashion by the shareholders of an authorized set $R \in \Gamma$. That is, each shareholder $s_{i,j} \in S$ receives input from each shareholder contained in R to compute share $\sigma_{i,j}(\alpha\beta)$. If one of the inputs is not valid, then shareholder $s_{i,j}$ cannot compute a valid share for $\alpha\beta$.

Audit.PreMult The algorithm takes as input from the bulletin board commitments $c_{k,\alpha}, c_{k,\beta}$, for $k = 0, \ldots, t-1$, to the coefficients of the polynomials $f_\alpha(x), f_\beta(x)$ sharing α and β, respectively, generated during algorithm Audit.RandShares. More precisely, each shareholder $s_{i,j} \in S$ performs the following steps.

1. It computes

$$g^{\sum_{l=1}^{r} \delta_{l,i,j}} \equiv \prod_{k=j-1}^{t-1} c_{k,\alpha}{}^{\frac{k!}{(k-j+1)!} i^{k-j+1}} = g^{f_\alpha^{(j-1)}(i)},$$

and

$$g^{\sum_{l=1}^{r} \varepsilon_{l,i,j}} \equiv \prod_{k=j-1}^{t-1} c_{k,\beta}{}^{\frac{k!}{(k-j+1)!} i^{k-j+1}} = g^{f_\beta^{(j-1)}(i)}.$$

2. If one of the above equalities is not satisfied, then it outputs '0' and aborts.
3. Otherwise, if both of the above equations hold, it accepts $\delta_{l,i,j}$ and $\varepsilon_{l,i,j}$, for $l = 1, \ldots, r$, as valid input.
4. It calls algorithm Commit.PreMult and computes commitments $c_{i,j}(\alpha) := g^{\sigma_{i,j}(\alpha)}$, $c_{i,j}(\beta) := g^{\sigma_{i,j}(\beta)}$, and $c_{i,j}(\gamma) := g^{\sigma_{i,j}(\gamma)}$ for $\sigma_{i,j}(\alpha), \sigma_{i,j}(\beta)$, and $\sigma_{i,j}(\gamma)$, respectively.
5. It publishes commitments $c_{i,j}(\alpha), c_{i,j}(\beta)$, and $c_{i,j}(\gamma)$ on the bulletin board and outputs '1'.

In the following, algorithm Audit.Multiply run by an auditor to check that shares $s_{i,j}(m)$ of message $m = m_1 \cdot m_2$ have been computed correctly is presented.

Audit.Multiply The algorithm takes as input the values δ, ε, the commitments to the shares of the multiplicative triple, i.e. $c_{i,j}(\alpha), c_{i,j}(\beta)$, and $c_{i,j}(\gamma)$, for $s_{i,j} \in S$, the commitments to the input values, i.e. $c(m_1), c(m_2)$, and the claimed result m. Then, it first audits that the equation $\alpha\beta = \gamma$ was fulfilled and then that m has been computed correctly. More precisely, an auditor performs by the following steps.

1. It computes the reconstruction vector $(w_1, \ldots, w_r)$ for shareholders $s_1, \ldots, s_r \in R$, with $R \in \Gamma$ authorized set, which computed the input for γ during PreMult. For conjunctive (disjunctive) hierarchical secret sharing schemes the interpolation vector $(w_1, \ldots, w_r)$ is composed of the entries $w_l := (-1)^{l-1} \frac{\det\left(A_{l-1,0}(E,X,\varphi)\right)}{\det(A(E,X,\varphi))}$ $\left(w_l := (-1)^{l+t-2} \frac{\det\left(A_{l-1,t-1}(E,X,\varphi)\right)}{\det(A(E,X,\varphi))}\right)$ according to the notation of Section 2.2.
2. It computes the following commitments:

$$c(\alpha) := \prod_{l=1}^{r} c_l(\alpha)^{w_l}; \quad c(\beta) := \prod_{l=1}^{r} c_l(\beta)^{w_l}; \quad c(\gamma) := \prod_{l=1}^{r} c_l(\gamma)^{w_l},$$

   where $c_l(\alpha), c_l(\beta), c_l(\gamma)$, for $l = 1, \ldots, r$, are commitments $c_{i,j}(\alpha), c_{i,j}(\beta), c_{i,j}(\gamma)$, respectively, in lexicographic order.
3. It checks whether multiplicative triple (α, β, γ) is correct by computing $e(c(\alpha), c(\beta)) = e(c(\gamma), g)$[3]. If the equation does not hold, then it outputs '0' and aborts the algorithm.
4. Otherwise, it takes from the bulletin board commitments $c(m_1), c(m_2)$ and the values δ, ε reconstructed during algorithm Multiply.
5. it check whether $c(\alpha)^{-1} \cdot c(m_1) = g^{\delta}$ and $c(\beta)^{-1} \cdot c(m_1) = g^{\varepsilon}$ and $g^m = c(\gamma) \cdot c(m_1)^{\varepsilon} \cdot c(m_2)^{\delta} \cdot g^{-\delta\varepsilon}$. If all of the equations hold, then it returns '1' and '0' otherwise.

3.3.5 Summary of Security Aspects

In Section 3.3.2 and in Section 3.3.3, we have proven that algorithm LinAdd, algorithm RandShares, algorithm PreMult, and algorithm Multiply do not compromise the perfect secrecy and correctness of the underlying conjunctive (disjunctive) hierarchical secret sharing scheme. Furthermore, the adversary these algorithms can cope with is active, i.e. not only it knows data private to shareholders (like the passive adversary), but also it can make them deviate from the protocols. More precisely, assumptions (A1)-(A4) of Section 3.3.4.1 set requirements for, respectively, the access structure, the threshold, the identities of the shareholders, and the channels through which shareholders communicate. These assumptions together with verifiable secret sharing ensure that a honest majority of shareholders is able to correctly reconstruct the message, while maintaining the secrecy of their shares, even if all other shareholders are corrupted by the adversary and cheat. Assumption (A5) prevents the adversary

[3]Here the definition of bilinear maps is used.

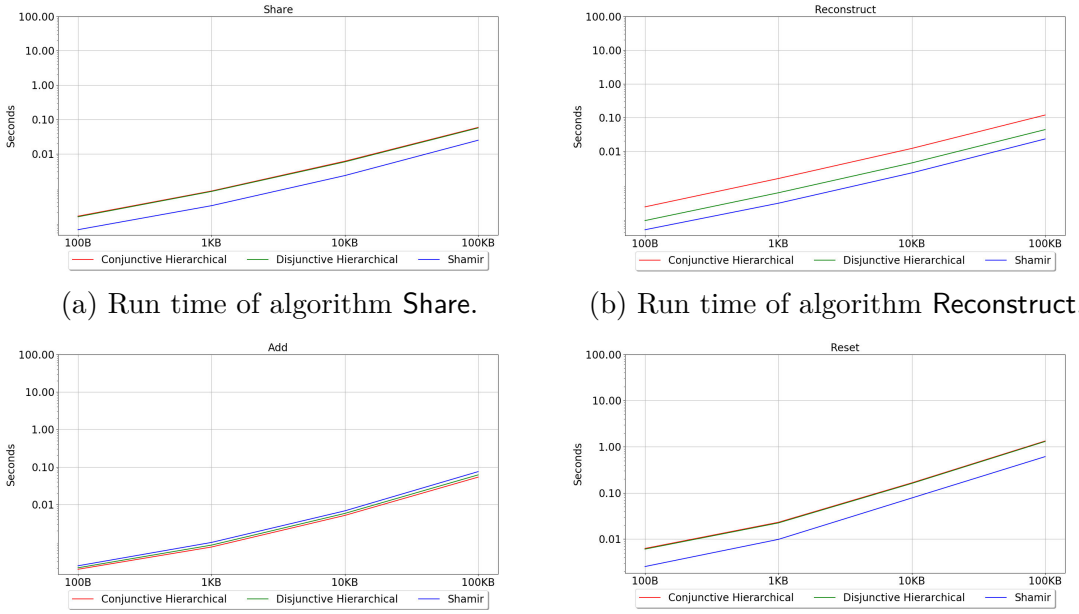
from tampering with the bulletin board and, together with the auditing procedure, ensures correctness when operations on data are performed. Moreover, as it is shown in Section 3.2, conjunctive (disjunctive) hierarchical secret sharing schemes support proactive secret sharing. This means that, provided that the shares are refreshed periodically, our protocols can cope with a mobile adversary, which is only bounded in the amount of shareholders it can corrupt within a certain time interval, but not over time. Lastly, the auditing procedure presented in Section 3.3.4 enhances the overall security because it allows to detect misbehaviors. The protocols described use Feldman commitment, which ensures only computationally hidingness. However, the auditing procedure can be easily adapted to Pedersen commitment to achieve unconditionally hidingness, which preserves even perfect secrecy of the underlying conjunctive (disjunctive) hierarchical secret sharing scheme.

3.4 Evaluation

In this section, we discuss the efficiency of the algorithms to add shares, reset the access structure, and perform operations for Tassa's hierarchical secret sharing schemes with respect to their counterparts for Shamir's secret sharing scheme. We first discuss on a high level why the efficiency of those algorithms for Tassa's hierarchical secret sharing schemes are comparable to the ones for Shamir's secret sharing scheme and then show experimental evaluations supporting our point.

Besides polynomials' evaluation, algorithm Share of Tassa's hierarchical conjunctive (disjunctive) secret sharing schemes requires also to compute up to $t-1$ polynomials' derivatives, which is not required for the sharing algorithm in Shamir's secret sharing scheme. However, the additional multiplications due to derivation are balanced by the fewer multiplications needed when evaluating derivatives of polynomials. Algorithm Reconstruct is the most expensive algorithm and requires Tassa's scheme to perform Gaussian elimination to find a solution to a system of t linear equations. However, in the secret sharing framework the only coefficient that matters is the free (last) coefficient for conjunctive (disjunctive) secret sharing. Therefore for message reconstruction only two determinants have to be computed. This leads to a complexity of $\mathcal{O}(t^3)$ for matrix A of dimension $t \times t$ in case the LU decomposition technique is used [3]. Solving a system of t linear equations is the underlying problem also to reconstruct the message in a Shamir's secret sharing scheme. Algorithm Add and algorithm Reset are very similar to the respective counterpart for Shamir's secret sharing scheme, as one can see, respectively, in the work by Nojoumian et al. in [90] in the work by Gupta and Gopinath in [58]. More precisely, for Tassa's hierarchical conjunctive (disjunctive) secret sharing schemes the determinants $\det(A(E, X, \varphi))$, $\det(A_{l-1,k}(E, X, \varphi))$, and $\det(A_{l-1,0}(E, X, \varphi))$ ($\det(A_{l-1,0}(E, X, \varphi))$) have to be computed and these are expensive operations. However, these terms need only public values to be determined and thus can be computed off-line, making the on-line phase

of algorithms Add and Reset equivalent to the corresponding algorithms for Shamir's secret sharing. In Figure 3.1, a comparison for algorithms Share, Reconstruct, Add, and Reset is provided.



(a) Run time of algorithm Share.

(b) Run time of algorithm Reconstruct.

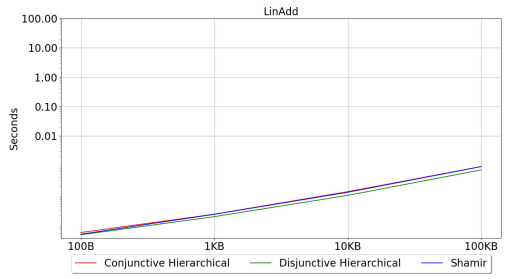
(c) Run time of algorithm Add.

(d) Run time of algorithm Reset.

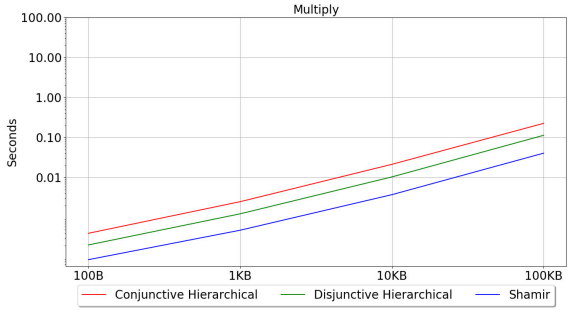
Figure 3.1: Run time of Share, Reconstruct, Add, and Reset for Shamir's secret sharing scheme and Tassa's conjunctive and disjunctive secret sharing schemes.

Algorithm LinAdd and algorithm Multiply require that the shareholders perform steps very similar to the corresponding algorithms for Shamir's secret sharing (see for instance the work by Ben-Or et al. [12] and by Schabhüser et al. in [98]). In fact, those algorithms consist of the equivalent linear operations with coefficients λ_1 and λ_2 for algorithm LinAdd and with coefficients δ and ε for algorithm Multiply. Algorithm RandShare is an adaptation of the algorithm proposed by Damgård and Nielsen proposed in [40] to fit the hierarchical setting. More precisely, they both consist of running algorithm Share and LinAdd for each shareholder and, as we have already discussed, these two are equivalent in terms of operations. Algorithm PreMult requires more work with respect to the preprocessing phase compared to Shamir's threshold secret sharing. Algorithm PreMult is computed in distributed fashion because additional information is needed to compute the shares. Despite the fact that only additions and polynomials' evaluation are performed to compute such additional information, algorithm PreMult increases the communication cost and requires secure channels. For Shamir's threshold secret sharing scheme this additional information needs not to be computed and the communication complexity is, thus, lower. For the same reasons, the auditing procedure during the on-line phase

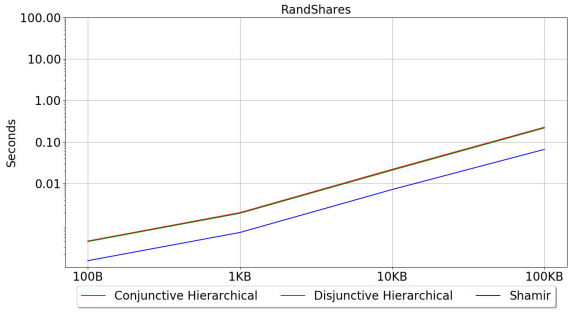
of Tassa's schemes has computational complexity similar to the one for Shamir's scheme while the auditing procedure during the off-line phase is more expensive. In Figure 3.2, a comparison for algorithms LinAdd, Multiply, RandShares, and PreMult is provided.



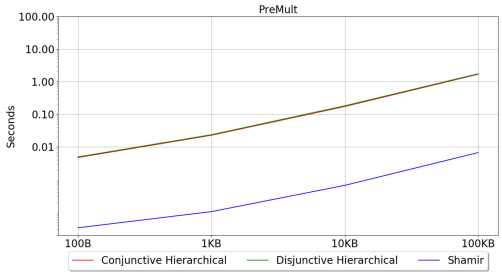
(a) Run time of algorithm LinAdd.



(b) Run time of algorithm Multiply.



(c) Run time of algorithm RandShares.



(d) Run time of algorithm PreMult.

Figure 3.2: Run time of LinAdd, Multiply, RandShares, and PreMult for Shamir's secret sharing scheme and Tassa's conjunctive and disjunctive secret sharing schemes.

In the following, we describe in detail the evaluations displayed in Figure 3.1 and in Figure 3.2. The simulations where performed for the following values of pair (t, n): $(2, 3), (3, 5), (4, 7), (5, 9)$, and $(6, 11)$. With respect to Tassa's conjunctive and disjunctive secret sharing schemes, for every pair (t, n), all possible hierarchical configurations with up to t levels were tested and the average of the run time is what is displayed. Thus, the higher the threshold t, the more hierarchical configurations to be tested. What we have tested is the run times of algorithms Share, Reconstruct, Add, Reset, LinAdd, Multiply, RandShares, and PreMult when files 100 Byte, 1 Kbyte, 10 Kbyte, and 100 Kbyte are processed with Shamir's threshold secret sharing, Tassa's conjunctive hierarchical secret sharing, and Tassa's disjunctive hierarchical secret sharing. Also, for each (t, n)-configuration, for each algorithm and for each secret sharing primitive we run the experiment 10 times and then plotted the average run time. We have used a 256-byte encoding, meaning that every 256 characters from the file were first converted into an integer of the finite field used in the secret sharing

primitives. The finite field that was selected in $\mathbb{F}_{2^{2203}-1}$ because this is the minimal prime upper bound for the encoding that we have chosen. The simulations have been performed on an Intel (Quad-Core) i5-8250U CPU clocked at 1.6GHz,with 8 GB of RAM and 64bit Windows 10. As one can see, the run times of the algorithms instantiated with Tassa's conjunctive and disjunctive hierarchical secret sharing schemes are comparable with the run times with Shamir's threshold secret sharing scheme, where Note the breaks in the plots are due to the increase in size of the file to be processed.

More precisely, with respect to algorithm Share, for Shamir's secret sharing scheme, that means to compute $t-1$ multiplication for each share to be generated for a polynomial of degree $t-1$. Besides polynomials' evaluation, algorithm Share for Tassa's conjunctive and disjunctive hierarchical secret sharing schemes requires also to compute up to $t-1$ polynomials' derivatives. However, the additional multiplications due to derivation are partially balanced by the fewer multiplications needed when evaluating derivatives of polynomials and this results into a slightly longer run time. Algorithms Reconstruct of both Shamir's secret sharing scheme and Tassa's conjunctive and disjunctive hierarchical secret sharing schemes require Gaussian elimination to solve a system of t linear equations in order to reconstruct the polynomial used to share the message. Then, the message is retrieved through polynomial evaluation. However, in the secret sharing framework the only coefficient that matters is the free coefficient for Shamir's secret sharing scheme and for Tassa's conjunctive hierarchical secret sharing scheme and the last coefficient for Tassa's disjunctive hierarchical secret sharing scheme. For Tassa's conjunctive (disjunctive) hierarchical secret sharing schemes, this implies that only determinants $\det(A(E,X,\varphi_0))$ $(\det(A(E,X,\varphi_{t-1})))$ and $\det(A(E,X,\varphi))$ have to be computed, where the latter can be computed in advance off-line. Also, Tassa's disjunctive's Reconstruct algorithm is faster than the Tassa's conjunctive counterpart because the matrix $A(E,X,\varphi_{t-1})$ has more zeros than $A(E,X,\varphi_0)$, leading to a faster computation of the determinants (see Chapter 2.2). The Reconstruction algorithm of Shamir's secret sharing scheme requires $t-1$ multiplications. Figure 3.1 shows that the three run times are comparable, especially those of Shamir's secret sharing scheme and Tassa's conjunctive secret sharing scheme. With respect to algorithms Add and Reset, the run times of Shamir's secret sharing scheme and Tassa's conjunctive and disjunctive hierarchical secret sharing schemes are similar because the steps to be performed on-line are equivalent. That is because, the Lagrange interpolation constant for Shamir's secret sharing schemes can be computed in advance since it does not involved secret information(see algorithm Add by Nojoumian et al. in [90] algorithm Reset by Gupta and Gopinath in [58]). The same thing holds for Tassa's conjunctive (disjunctive) secret sharing schemes where term $\frac{\lambda_{l,(i',j')}}{\sigma_l}$ for algorithm Add and term $\frac{a_{l,0}}{\sigma_l}(\frac{a_{l,t-1}}{\sigma_l})$ can be computed in advance. With respect to Figure 3.2, algorithms LinAdd and Multiply consist of the on-line computations to perform, respectively, linear operations and multiplications over

(hierarchically) shared secrets. Since algorithm LinAdd is entirely on-line and the operations on the shares are the same, the run times for Shamir's secret sharing scheme and for Tassa's conjunctive and disjunctive hierarchical secret sharing schemes are similar. With respect to algorithm Multiply, the difference in run time is due to the difference in run time of algorithm Reconstruct of the corresponding schemes. The run times of algorithms RandShares and PreMult for Tassa's conjunctive and disjunctive hierarcical secret sharing scheme is significantly greater than the ones for Shamir's secret sharing scheme due to the derivatives for the different levels. However, these algorithms are the off-line steps to be computed in advance before algorithm Multiply. Thus, overall, for the on-line computations of the algorithm, the run times of Tassa's conjunctive and disjunctive hierarchical secret sharing schemes are comparable to that of Shamir's secret sharing scheme.

To perform algorithms Audit.LinAdd, Audit.Multiply and Audit.RandShares, the auditor takes steps very similar to the corresponding auditing procedure for Shamir's secret sharing schemes, because algorithms LinAdd, Multiply and RandShares are defined similarly. Instead, algorithm Audit.PreMult requires the computation of commitments in a distributed fashion, which increases the communication and the computation cost. However, we recall that the preprocessing phase is off-line and can be performed in advance. Regarding the on-line phase, which is the time critical phase, the schemes of both Shamir and Tassa perform equally well. The auditing algorithms were not tested due to the computationally intensive commitment schemes.

3.5 Summary and Future Work

In this chapter, we showed that Tassa's hierarchical conjunctive and disjunctive secret sharing schemes are a promising candidate for distributed storage systems. They provide the same functionalities that Shamir's secret sharing scheme provides and, in addition, they fit scenarios with an underlying hierarchical structure. We first introduced the definition of dynamic secret sharing as a secret sharing schemes where additional shares can be computed, the access rules can me modified, and the shares can be renewed in distributed fashion without the intervention of the data owner. Then we provided the corresponding algorithms to perform the above mentioned functionalities when the message is shared using Tassa's hierarchical conjunctive and disjunctive secret sharing schemes. Second, we showed how to practically compute linear operations and multiplications over shared messages when Tassa's conjunctive and disjunctive hierarchical secret sharing schemes are used by providing the algorithms to do that. For all the algorithms that we presented, we also proved their correctness and showed that they do not lower the security of the underlying Tassa's hierarchical conjunctive and disjunctive secret sharing schemes. For the algorithms to perform operations on hierarchically shared data, we also provided auditing mechanism to check that those operations were actually

performed correctly. We also investigated the efficiency of the algorithms we proposed. In general, they are not less efficient than the corresponding algorithms for Shamir's secret sharing scheme, given that the most expensive operations for Tassa's hierarchical conjunctive and disjunctive secret sharing schemes can be performed off-line in advance. We provided experimental evaluations to empirically support this.

As future work, we want to implement a distributed storage system built on Tassa's hierarchical secret sharing schemes that involes real data centers from different SSPs. The challenge here is the technology of multi-cloud, which seems not to be ready yet to support a single point of orchestration across different commercial providers so that to enable the data owner to outsource, withdraw, and monitor its data. Once further improvements with respect to the multi-cloud will be achieved, we plan to implement hierarchical secret sharing-based distributed storage systems.

4 Preventing Security Threats Through Adaptive Social Secret Sharing and Private Third-Party Auditing

Distributed storage systems built on secret sharing schemes are suitable for the long-term confidentiality protection of stored data due to the information-theoretic nature of such cryptographic primitives. In particular, this is achieved because the shares of the stored data are periodically renewed distributedly by performing proactive secret sharing. This is a viable solution for information-theoretic long-term confidentiality distributed storage systems from a purely cryptographic perspective. However, long-term confidentiality relies in practice on high-performing storage servers carrying out proactive secret sharing, where performance is understood in a broad sense, including for instance reliability. The performance of storage servers may vary over time and data owners require guidance to select the individual storage servers making up the distributed storage system. Social secret sharing (see Section 2.3) offers a solution: an aggregate performance score is assigned to each storage server and is periodically updated to provide accurate performance figures. Data owners can check which storage servers are assigned the best aggregate scores and select them accordingly. In case the updated aggregate performance scores vary significantly over time, the distributed storage system has to be adapted accordingly. This means two things. First, that the access rules of the underlying secret sharing scheme should exclude low-performing storage servers to include better-performing ones. Second, that the access rules should be adapted the updated aggregate performance scores so that proactive secret sharing can always be carried out reliably. However, state of the art social secret sharing schemes [86, 90, 92] do not allow for modifying the underlying secret sharing scheme, nor to include or exclude storage servers without the intervention of the data owner. This makes the current social secret sharing solutions unsuitable for long-term protection of the confidentiality of data in distributed storage systems.

A further requirement for the long-term protection of data, especially when it is critical data, is to be able to check whether the data stored in the storage servers have been compromised or not. In the best case, such auditing could itself be outsourced

to a third party that does not to be trusted by the data owner. That is, also the auditing mechanism should guarantee privacy, even if the auditor collaborates with a set of storage servers. However, the privacy preserving third party auditing mechanisms presented so far only tackle single-server storage solutions or rely on encryption and no such protocol exists at all for a distributed storage system setting. Thus, currently, it is not possible to double check the integrity of the outsourced data while protecting its confidentiality in an information-theoretic way.

Contributions

In this chapter, we provide the first social secret sharing solution that allows to distributedly modify the access rules of the underlying secret sharing scheme as well as to modify the set of storage servers according to their aggregate scores. We do this by first defining a new primitive called *adaptive social secret sharing*, which requires that the underlying secret sharing scheme is dynamic, according to Definition 3.1. Building social secret sharing on dynamic secret sharing allows to modify the access rules and the set of storage servers making up the storage system without the intervention of the data owner. We instantiated adaptive social secret sharing through Shamir's secret sharing scheme (see section 2.1) and Tassa's hierarchical conjunctive and disjunctive secret sharing schemes (see Section 2.2), which are dynamic, by also setting up alarm triggers when instable states with too many low-performing storage servers occur. Our solution outperforms state of the art ones in terms of security because it can respond to the degradation of the storage servers. Furthermore, the instantiation with Tassa's conjunctive and disjunctive hierarchical secret sharing schemes achieves optimal storage consumption because all shares distributed to the storage servers are of the same length.

In addition, we present an information-theoretically private auditing mechanism for distributed storage systems which is compatible with proactive secret sharing and current distributed storage solutions. This means that a third party can audit the integrity of sensitive data, while still fulfilling the stringent confidentiality requirements they come with. More precisely, we define the syntax and the security requirements of such mechanisms, in particular by formalizing the notion of privacy and extractability of such third party auditing mechanism. We then present an instantiation based on additively homomorphic threshold secret sharing schemes, specifically with Shamir's secret sharing scheme. In contrast to state of the art solutions [20, 37, 101, 111], our auditing mechanism does not require encryption and is completely keyless. This is achieved by exploiting the non-collusion assumptions intrinsic to secret sharing-based distributed storage systems. We introduce also a batch version of our auditing mechanism, so multiple data from different data owners can be audited at the same time with constant communication complexity. Moreover, we show how storage servers can protect themselves against malicious data owners distributing them incorrect shares.

Contributions to this chapter come from papers [T3] and [T4]. My contributions were the definition of the adaptive secret sharing primitive and the comparison of the instantiation of this primitive through Tassa's conjunctive and disjunctive hierarchical secret sharing schemes with state of the art approaches. Furthermore, I modified the third-party auditing mechanism so that it can cope with malicious data owners that generate invalid shares when outsourcing their data.

Outline

The contributions of this chapter are organized into two main sections. In Section 4.1, the adaptive social secret sharing solution for distributed storage system is presented. More precisely, first we define adaptive social secret sharing as a primitive, then we provide an instantiation based on Shamir's secret sharing scheme and an instantiation based on Tassa's hierarchical conjunctive and disjunctive secret sharing schemes and argue how these outperform the state of the art instantiations of social secret sharing. in Section 4.2, the privacy preserving third party auditing for distributed storage systems is presented. More precisely, first we define auditable distributed storage systems as a primitive, specifying the privacy and security requirements for completeness, privacy, and extractability. Then we instantiate this primitive by using Shamir's secret sharing scheme, presenting a simple solution for auditing one message only and for auditing multiple messages at the same time. Afterwards, we present how our auditing mechanism can be used instead of verifiable secret sharing to protect the storage servers against malicious data owner and we survey related work. Summary and future work can be found in Section 4.3.

We first highlight the drawbacks and open problems of state of the art social secret sharing-based distributed storage system, both with respect to protection of confidentiality of data and with respect to storage consumption. Then we introduce AS^3, an adaptive social secret sharing scheme that allows to modify the setup of the underlying secret sharing primitive to keep up with the ever changing performance of the storage servers withing the system. It supports data owners not only in first setting up a distributed storage system, but also in maintaining it over time. Then, we present the first information-theoretic secure audit mechanism that allows a third party to audit the integrity of the data stored in a secret sharing-based distributed storage system. We also introduce a batch version of such mechanism, where multiple shares from different data can be verified at once.

4.1 Adaptive Social Secret Sharing for Distributed Storage Systems

In this section, we present how to instantiate distributed storage systems when the storage servers involved have different reconstruction capabilities and how to adapt them to the always changing performance of these storage servers. The distributed storage systems that we propose are based on dynamic secret sharing (according to Definition 3.1 introduced in Section 3.2.1) and on social secret sharing, introduced in Section 2.3. This allows not only to grant more informative shares to the best performing storage servers, but also to change the distribution of the shares and the access rules to the outsourced data in case of decrease or increase of the performance of those storage servers. The rest of the section is organized as follows. In Section 4.1.1, we describe the framework we operate in and formalize the requirements that a distributed storage system has in order to guarantee the availability of the outsourced data. In Section 4.1.2, we present the AS^3 protocol, based on social and dynamic secret sharing, and describe in detail the algorithms it is composed of. In Section 4.1.3, we compare AS^3 with state of the art solutions for distributed storage systems based on social secret sharing and show how our solution outperforms them in terms of security, availability, and storage consumption.

4.1.1 Framework

Social secret sharing, introduced in Section 2.3, is used when data owners want to outsource their data in distributed fashion so that the storage servers involved are granted with reconstruction capabilities that mirror their performances. More precisely, distributed storage systems are composed of storage servers from different commercial storage service providers whose performance might change over time. The performance of each storage server is expressed by an aggregate performance score which determines the weight of a storage server with respect to the reconstruction of the data. This weight determines how informative the share distributed is. In addition, the performance of the selected storage servers is monitored such that the reconstruction power of each storage server can be adapted accordingly. In the following, we define adaptive social secret sharing schemes as social secret sharing schemes that in addition of algorithms Share, Tune, and Reconstruct of social secret sharing schemes (see Definition 2.9) allow to perform algorithm Reset of dynamic secret sharing schemes of Definition 3.1. That is, an adaptive social secret sharing scheme is composed of algorithms that fulfill the security properties of accessibility and perfect security formalized in Definition 2.1. Adaptive social secret sharing is the primitive based on which we construct our solution AS^3 for distributed storage systems. We also formalize the setup on which such solution is instantiated and determine the requirements that distributed storage systems have to fulfill in order

to be viable solutions for long-term storage of data.

Definition 4.1. *For a message space $\mathcal{M}$, a space of shares Σ, a set of shareholders $S = \{s_1, \ldots, s_n\}$ where $i \in \mathcal{I}$ is the unique ID of shareholder $s_i \in S$, and an access structure $\Gamma \subset \mathcal{P}(S)$, an* adaptive social secret sharing scheme *is a social secret sharing scheme according to Definition 2.9 with an additional PPT algorithm* Reset *run by an authorized set of shareholders that also satisfies the security properties of Definition 2.1. More precisely, an adaptive social secret sharing scheme is a tuple of PPT algorithms* Share, Tune, Reset, *and* Reconstruct *defined as follows.*

Share *takes as input a message $m \in \mathcal{M}$ and a vector of weights $w_1, \ldots, w_n \in [0, 1]$, where $\sum_{i=1}^{n} w_i = 1$. It outputs n shares $\sigma_1, \ldots, \sigma_n \in \Sigma$, where share σ_i is to be sent to shareholder $s_i \in S$ and whose reconstruction capability matches weight w_i, for $i = 1, \ldots, n$.*

Tune *takes as input aggregate performance scores $\tau_1, \ldots, \tau_n \in [0, 1]$ for, respectively, shareholders $s_1, \ldots, s_n$ computed by a performance scoring mechanism. It outputs weights $w_1, \ldots, w_n$ for shareholders $s_1, \ldots, s_n$.*

Reset *takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders, a vector of weights $w_1, \ldots, w_n$ for shareholders $s_1, \ldots, s_n$, a new set of shareholders $S' = \{s'_1, \ldots, s'_{n'}\}$ (where S' needs not be disjoint from S and n' needs not to be different from n) with a vector of bootstrapped weights $w'_1, \ldots, w'_n$, and an access structure $\Gamma' \subset \mathcal{P}(S')$. If R is unauthorized, i.e. $R \notin \Gamma$, it outputs $\perp$. Otherwise, $R \in \Gamma$ and without message reconstruction, it outputs n' shares $\sigma'_1, \ldots, \sigma'_{n'}$, where share σ'_i is to be sent to each new shareholder $s'_i \in S'$, for $i = 1, \ldots, n'$. The shares $\sigma_1, \ldots, \sigma_n \in \Sigma$ held by the old shareholders are deleted.*

Reconstruct *takes as input a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders. It outputs $m \in \mathcal{M}$ if $R \in \Gamma$, and $\perp$ otherwise.*

For our framework, we assume that the set $\mathcal{S} = \{s_1, \ldots, s_n\}$ of shareholders in Definition 4.1 are n storage servers owned by different storage service providers. Like for the original definition of social secret sharing presented in Section 2.2, also algorithm Share of adaptive social secret sharing allows the data owner to store their messages either by using weighted secret sharing or hierarchical secret sharing. The reason is that Shamir's threshold secret sharing scheme, and therefore its weighted version, and Tassa's hierarchical conjunctive and disjunctive scheme are dynamic and can be equipped with algorithm Reset, as presented in Section 3.2. In both cases the initial weights $w_1, \ldots, w_n$ for storage servers $s_1, \ldots, s_n$ are determined and algorithm S.Share of the underlying secret sharing scheme is called with the vector of weights as input. Afterwards, the storage servers run periodically algorithm Tune to determine and update the weights of each storage server and to accordingly adjust

the shares by calling the algorithm S.Reset. More precisely, algorithm Reset allows the user to add, remove, and replace storage servers by first computing weights for the newcomers and then calling algorithm S.Reset of the underlying secret sharing scheme with the new storage servers and weights as input. Finally, at any time the user can retrieve the message by running algorithm Reconstruct, which calls the algorithm S.Reconstruct of the underlying secret sharing scheme.

Our scheme is parameterized by the following values. (1) The total number of storage servers n which is input to algorithm Share and can be changed by calling algorithm Reset. (2) The weights $w_1, \ldots, w_n$ of storage servers $s_1, \ldots, s_n$, respectively, which are initialized by algorithm Share and updated regularly by algorithm Tune. If we use weighted secret sharing as underlying secret sharing scheme we also use (3a) threshold t required to reconstruct the message while for hierarchical secret sharing we have (3b.1) the total number ℓ of levels and (3b.2) thresholds $t_1, \ldots, t_\ell$ for levels $L_1, \ldots, L_\ell$, respectively. In addition, d denotes the total number of different subsets of storage servers that are able to reconstruct the message.

There are two aspects that distributed storage systems should guarantee: confidentiality and availability of the outsourced data. As we have discussed in the introduction of this chapter, confidentiality and availability rely on the performance of the storage servers involved. These two properties rely on algorithm Reset and algorithm Reconstruct that can be performed in distributed fashion by authorized subsets of storage servers. More precisely, a certain threshold of storage servers is needed to perform algorithms S.Reset and S.Recontruct of the underlying dynamic secret sharing scheme and correspondingly to run algorithms Tune, Reset, and Recontruct of the adaptive social secret sharing scheme. Thus, the parameters must be chosen such that the system can cope with low-performing storage servers. On the other hand, storage servers with high aggregate performance scores are more likely to perform well the next time one of the above algorithms is called, but this is not guaranteed. There is always the possibility that problems occur and they respond late or not at all. In this worst case scenario, the algorithms still have to be run and the operations of updated still have to be carried out. In order for this to happen, the shares must be distributed so that, in case the highest-performing storage servers are faulty, the other storage servers form at least one authorized set that can correctly run algorithms Reset and Reconstruct. We present in the next session how algorithm Tune is instantiated so that this requirement holds.

4.1.2 AS^3: An Instantiation of Adaptive Social Secret Sharing

In this section, we present AS^3, our solution for distributed storage systems. AS^3 is an adaptive social secret sharing scheme that preserves confidentiality and availability of the outsourced data. We detail how algorithm Share, algorithm Tune, algorithm

Reset, and algorithm Reconstruct are instantiated for AS^3 in, respectively, Section 4.1.2.1, Section 4.1.2.2, Section 4.1.2.3, and Section 4.1.2.4.

4.1.2.1 Algorithm Share

When algorithm Share is run, a set of n storage servers is selected and their vector of weights $w_1, \ldots, w_n$ is bootstrapped. For simplicity and without loss of generality we assume that all storage servers are treated as newcomers and have all the same weight equal to 1, i.e. $w_1 = \cdots = w_n = 1$. This means that, for weighted secret sharing, all storage servers $s_1, \ldots, s_n$ receive one share only. For both conjunctive and disjunctive hierarchical secret sharing, this means that only one level L_0 is spanned and that all shareholders $s_1, \ldots, s_n$ are assigned to level L_0. The threshold t and the threshold t_0 of the underlying weighted secret sharing scheme and hierarchical secret sharing scheme, respectively, are selected so that $n = 2t - 1$ to ensure honest majority.

4.1.2.2 Algorithm Tune

Algorithm Tune is responsible for adjusting the weights of the storage servers according to the updated aggregate performance scores computed by the performance scoring mechanism in the distributed storage system, where $0 \leq x \leq n$ is an integer. More precisely, the algorithm takes as input a vector $\tau_1, \ldots, \tau_n$ of aggregate performance scores describing the performance of storage servers $s_1, \ldots, s_n$, respectively. This algorithm is composed of the following steps.

1) Estimation of x. It is estimated the amount x of possibly faulty storage servers given the updated aggregate scores. Given a performance threshold $k \in [0, 1]$ selected by the data owner at the moment of outsourcing the data, where $0 \leq k \leq 1$, x is the amount of storage servers s_i with aggregate score lower than the threshold, i.e. $x = |\{s_i \in S | \tau_i < k\}|$. If $x < n$, then the weights are computed at step 2). Otherwise a failure message is transmitted and the protocol is aborted.

2) Computations of the weights. Let the integer $\gamma > 0$ denote the granularity parameter with which the aggregate scores $\tau_1, \ldots, \tau_n$ are mapped into weights $w_1, \ldots, w_n$. More precisely, interval $[0, 1]$ is divided into γ disjoint subintervals, $I_1 = [0, \frac{1}{\gamma}), I_2 = [\frac{1}{\gamma}, \frac{2}{\gamma}), \ldots, I_\gamma = [1 - \frac{1}{\gamma}, 1]$. If the aggregate score τ_i lies in interval I_j, then weight w_i set to j, where $j = 1, \ldots, \gamma$ is an integer. Thus, there are γ possible values that weights $w_1, \ldots, w_n$ can take. For weighted secret sharing, storage server s_i receives $w_i = j$ shares, for $i = 1, \ldots, n$. For hierarchical secret sharing, γ corresponds to the amount ℓ of levels $L_1, \ldots, L_\ell$ spanned and storage server s_i whose aggregate score τ_i lies in I_j is assigned to level L_j, for $i = 1, \ldots, n$, as initially proposed for social secret sharing (see Section 2.3).

3) Thresholds selection. We denote by W the vector of length n containing all the weights $w_1, \ldots, w_n$ sorted from the smallest value to the highest value, i.e. $W := \{W[1], \ldots, W[n]\}$ where $W[i] \leq W[i+1]$, for $i = 1, \ldots, n$. For the weighted secret sharing, t is set to $1 + \sum_{i>n-x} W[i]$. For disjunctive secret sharing, t_1 is set to $x + 1$ and t_{h+1} to $t_h + 1$, for $h = 1, \ldots, \ell - 1$.

4) Availability check. As we have anticipated in Section 4.1.1, in the worst case scenario the x faulty storage servers are those with the highest aggregate performance score, i.e. they store the shares with the highest reconstruction capabilities. In this case, it must be ensured that there is still at least one subset of non faulty storage servers able to retrieve the message, i.e. $d \geq 1$. We denote by $\mathcal{S}' \subset \mathcal{S}$ the subset of all storage servers where the x most powerful storage servers are discarded (in this case, the ones corresponding to the last x entries of vector W).

- For weighted secret sharing, d is defined as $d := |\{A \subset \mathcal{S}' | \sum_{S_i \in A} w_i \geq t\}|$ and trivially algorithm Reset and algorithm Reconstruct can be run if $d \geq 1$.
- For hierarchical secret sharing, we additionally assume that $n'_h \leq n_h$ is the amount of storage servers from $\mathcal{S}'$ assigned to level L_h, for $h = 1, \ldots, \ell$. For disjunctive secret sharing, algorithm Reset and algorithm Reconstruct can be run in the worst case scenario if $\exists\, h$ such that $\sum_{i \leq h} n'_i \geq t_h$. For conjunctive secret sharing, if $\forall\, h = 1, \ldots, \ell$ it holds that $\sum_{i \geq h} n'_h \geq t_h$.

If the above constraints are not satisfied, then the new shares are not computed and distributed, because otherwise the data would be irreversibly lost. Instead, a warning message is sent to the data owner to make it aware of the possibly dangerous situation caused by the presence of too many faulty storage servers. The user is strongly recommended to reboot new storage servers to prevent the loss of the message.

4.1.2.3 Algorithm Reset

Algorithm Reset takes as input threshold t or thresholds $t_1, \ldots, t_\ell$ adjusted by algorithm Tune, computes and distributes the shares to the storage servers in the storage system according to weights $w_1, \ldots, w_n$, and, eventually, the shares of newcomers. With weighted secret sharing scheme, if the threshold t has been modified, then algorithm S.Reset discussed in Section 2.1 is run and the shares are distributed according to $w_1, \ldots, w_n$. With hierarchical secret sharing scheme, if thresholds $t_1, \ldots, t_\ell$ have been modified, then algorithm S.Reset introduced in Section 3.2.3 is run and the shares are distributed according to weights $w_1, \ldots, w_n$.

4.1.2.4 Algorithm Reconstruct

Algorithm Reconstruct is called when the user wants to retrieve the message. Depending on the underlying secret sharing scheme, a specific algorithm S.Reconstruct is run. Specifically, algorithm S.Reconstruct described in Section 2.1 is called for weighted secret sharing, while algorithm S.Reconstruct described in Section 2.2 is called for both conjunctive and disjunctive hierarchical secret sharing.

4.1.3 Related Work and Comparison with AS^3

In this section, we provide an evaluation of the related work with respect to social secret sharing, i.e. the weighted social secret sharing by Nojoumian and Stinson [88,90] (who first introduced the concept of social secret sharing) and the hierarchical social secret sharing by Pakniat et al. [92]. We briefly describe the approaches and highlight their shortcomings. Furthermore, we show the countermeasures provided by our adaptive weighted and adaptive hierarchical social secret sharing scheme AS^3 presented in Section 4.1.2.

4.1.3.1 Weighted Social Secret Sharing

In [88, 90, 92], algorithm Share sets the aggregate scores $\tau_1, \ldots, \tau_n$ to zero. The weights $w_1, \ldots, w_n$ are defined as integers and no bound on their value is provided. Also, it is claimed that they are selected according to an distribution that was never specified by the authors. Then, shares are generated and distributed to the storage servers according to their weights. Algorithm Tune calls the performance scoring mechanism in place, which updates the aggregate scores $\tau_1, \ldots, \tau_n$ based on the performance of the storage servers $s_1, \ldots, s_n$. Then, algorithm Tune adjusts the weights $w_1, \ldots, w_n$ as follows. The storage servers whose weight increased get one additional share while one share is taken from those whose weight decreased. Furthermore, each w_i is bounded by a parameter z much smaller than the threshold t of the scheme. However, this approach has left many issues unsolved.

(1) Initialization of weights and threshold. It is not specified how the weights $w_1, \ldots, w_n$ are initialized because no detail is given regarding which distribution to choose. Furthermore, it is not specified how to select threshold t given the weights. Note that care must be taken when this parameter is selected, because it is of major importance to ensure both confidentiality and availability. In fact, on the one hand, threshold t determines the maximum number $(t-1)$ of colluding storage servers the scheme can cope with. On the other hand, it determines how many shares held by different storage servers are needed to reconstruct the data.

(2) Translation of aggregate scores into weights. The final aggregate scores $\tau_1, \ldots, \tau_n$ should in principle be computed in a deterministically way (proper of

the specific instantiation of the social secret sharing scheme) given the weights $w_1, \ldots, w_n$ as input. However, a formula or algorithms to do so is not provided. More precisely, it is not defined if increase or decrease of weight w_i happens because the aggregate score τ_i is, respectively, above or below a certain threshold or because the aggregate score τ_i itself is, respectively, greater or smaller than the aggregate score of the previous round. In the former case, it is left open how to choose these thresholds. In the latter case, storage servers that behave better compared to the last round would be rewarded even though they have a low aggregate score and can therefore be considered untrustworthy. Furthermore, it is not clear which ranges of different aggregate scores are mapped to the same weight. Note that having small ranges, e.g. such that each single aggregate score is mapped to one weight, leads to a huge amount of shares, causing a high computational overhead and storage space consumption.

(3) Selection of upper bound z. No indication is given with respect to the choice of the upper bound z for the weights $w_1, \ldots, w_n$. The parameter z is introduced to prevent the storage servers from having a weight high enough to reconstruct the message by themselves, violating confidentiality. However, the parameter z must be chosen such that threshold t is much larger than any single initial weight w_i. Thus, there is no guarantee that all the weights of all the storage servers together can actually reach threshold t, namely there is no guarantee that the message can be reconstructed. Another problem is that parameter z makes the performance scoring mechanism less effective: the weights $w_1, \ldots, w_n$ cannot go beyond z no matter how high the aggregate scores $\tau_1, \ldots, \tau_n$ are. Thus, approaching z, it is less and less rewarding for the storage servers to be high-performing. Choosing such a high threshold t leads to a high computation overhead, which is not necessary when the weights are small.

(4) Instable states. Wrong choices for threshold t and bound z might lead to "instable states". In this framework, we refer to an instable state as a state in which there is a risk that the data cannot be reconstructed because there are not subsets of storage servers that have enough reconstruction capability as to successfully run algorithm Reconstruct. The current solution does not provide any measures to prevent or detect such states.

(5) Dynamism of parameters. Threshold t and bound z are selected by algorithm Share and kept unchanged for the entire lifetime of the storage. This has several drawbacks. If, for instance, the amount of storage servers and/or the amount of shares generated got increased, then threshold t and bound z must be increased as well to prevent malicious storage servers from reconstructing the data.

Our weighted AS^3 scheme addresses the issues summarized above. With respect to

(1), algorithm Share specifies how to initialize the weights $w_1, \ldots, w_n$ and threshold t. Furthermore, algorithm Tune defines how to map the aggregate scores into weights thereby addressing issue (2). With respect to (3) our scheme employs an accuracy parameter γ that keeps bounded the total amount of shares generated, instead of bounding the weights. This approach optimizes the storage space consumption and leads to a low computation overhead. Furthermore, the fact that there is no upper bound z encourages the storage servers to always be high-performing. A very critical part of all social secret sharing schemes is reaching instable states as described in shortcoming (4). We address this by defining the parameter d and providing rules to ensure that there are enough subsets of storage servers able to successfully run algorithm Reconstruct in case some of them are faulty and have a breakdown. Note that the compliance with these requirements is checked *before* new shares are generated and distributed. This prevents that the data is revealed or lost. In addition, the scheme includes countermeasures to prevent instable states. For instance, algorithm Share sets the total amount of storage servers n such that the Byzantine model [50] is satisfied, i.e. such that with the initial estimation of untrustworthy storage servers the message can be retrieved. Finally (5), our scheme is the first that provides dynamism with respect to the parameters selected, i.e. n, t, γ. Algorithm Tune and algorithm Reset increase or decrease threshold t at each point in time in order to protect confidentiality and to ensure availability. This especially allows to always choose threshold t tightly thereby optimizing the computational overhead and the storage space consumption.

4.1.3.2 Hierarchical Social Secret Sharing

Hierarchical social secret sharing has been introduced by Pakniat et al. in [92]. The underlying scheme is the disjunctive secret sharing scheme. Algorithm Share initializes the aggregate scores $\tau_1, \ldots, \tau_n$, the weights $w_1, \ldots, w_n$, and the levels $L_1, \ldots, L_\ell$. The total number ℓ of levels determines the accuracy for which the aggregate scores are mapped into levels. Depending on the weights, the storage servers are assigned to a specific level and shares are generated and distributed accordingly. Algorithm Tune calls the performance scoring mechanism which updates the aggregate scores $\tau_1, \ldots, \tau_n$ based on the behavior of the storage servers $s_1, \ldots, s_n$. Then algorithm Tune adjusts the weights $w_1, \ldots, w_n$ and reassigns the storage servers to the corresponding levels.

In this approach, the accuracy with which aggregate scores are mapped into levels is provided. However, this does not fully address (3), because the total number of levels ℓ also bounds how much storage servers can be incentivized or penalized. In fact, storage servers assigned to level L_1 cannot be incentivized any further and storage servers assigned to level L_ℓ cannot be penalized any further, no matter how, respectively, high-performing or low-performing they are. Besides this partial

countermeasure to (3), this hierarchical social secret sharing scheme leaves the same issues open as identified for the weighted social secret sharing scheme. More precisely, it is not clear how to initialize the weights $w_1, \ldots, w_n$, how many levels $L_1, \ldots, L_\ell$ to span, and how to choose the corresponding thresholds $t_1, \ldots, t_\ell$ (1). Furthermore, it is not defined how the aggregate scores are mapped into weights (2), instable states are not detected nor prevented (4), and the parameter selection is not dynamic (5). Our AS^3 based on hierarchical secret sharing schemes addresses these shortcomings similarly as described for weighted secret sharing.

4.2 Privacy Preserving Third Party Auditing for a Distributed Storage System

We recall that in the previous Section 4.1 our goal was to protect the long-term confidentiality of the outsourced data by adapting the instantiation of the underlying secret sharing scheme of a distributed storage system according to the evolution of the storage servers in terms of their performance. We achieved that by first formalizing these requirements with the definition of adaptive social secret sharing scheme and by providing an instantiation based on Tassa's hierarchical conjunctive and disjunctive secret sharing scheme referred to as AS^3.

In this section, we tackle a second aspect regarding the long-term protection of data in distributed storage systems. That is, to check whether the data stored in the storage servers have been tampered with or not. Periodically checking that the shares stored in each storage server are correct ensures the integrity of the outsourced data. Thus, an auditing mechanism is needed. Because of the information-theoretic nature of secret sharing-based distributed storage systems, also the auditing mechanism used should mirror this type of protection. More precisely, the auditing mechanism should guarantee privacy even if the auditor is an untrusted third party that collaborates with a subset of storage servers. We provide here the first solution for information-theoretic private third party auditing for distributed storage systems.

The rest of the section is structured as follows. In Section 4.2.1 we define auditable distributed storage systems and formalize the security and privacy requirements needed to protect the outsourced data. In Section 4.2.2, we provide a concrete instantiation of auditable distributed storage systems based on Shamir's secret sharing scheme. In Section 4.2.3, we show how and in what scenarios a variant of our protocol can be used as a replacement for verifiable secret sharing schemes to protect against malicious data owners. In Section 4.1.3, we survey related work.

4.2.1 Definition of Auditable Distributed Storage Systems

We formalize the notion of auditable distributed storage systems as well as the security and privacy properties that such systems have to fulfill. Note that we keep

this discussion more general than necessary for our concrete instantiation presented in Section 4.2.2. For instance, all our definitions allow for keyed instantiations of the audit mechanism too, even though our concrete instantiation is fully keyless. In the following, the participants involved are the data owner, denoted by D, the auditor, denoted by A, and a set of storage servers $S = \{S_1, \ldots, S_n\}$. We define *auditable distributed storage systems* as schemes consisting of algorithms Setup, KGen, Store, Reconstruct, and Verify, and an interactive protocol $\langle A.\mathsf{Audit}; S_i.\mathsf{Audit}\rangle$ between the auditor A and the storage servers $S_1, \ldots, S_n$.

Definition 4.2. *For a data owner D, an auditor A, a message $m \in \mathcal{M}$, a space of shares Σ, a set of storage servers $S = \{S_1, \ldots, S_n\}$ where $i \in \mathcal{I}$ is the unique ID of storage server $S_i \in S$, and an access structure $\Gamma \subset \mathcal{P}(S)$, an auditable distributed storage system is a tuple of the PPT algorithms Setup, KGen, Store, Reconstruct, and Verify, and the interactive protocol $\langle A.\mathsf{Audit}; S_i.\mathsf{Audit}\rangle$.*

Setup takes as input a security parameter λ in unary and the number of storage servers n. It outputs system parameters spar.

KGen takes as input the system parameters spar. It outputs a key pair (sk_D, pk_D) for the data owner D.

Store takes as input a message $m \in \mathcal{M}$, the system parameters spar, and the data owner's secret key sk_D. It outputs a shares $\sigma_i \in \Sigma$, for $i + 1, \ldots, n$, to be sent to, respectively, storage server $S_i \in S$, for $i = 1, \ldots, n$.

Reconstruct takes as input the system parameters spar, the data owner's secret key sk_D, and a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders. It outputs the reconstructed message $m' \in \mathcal{M}$ if $R \in \Gamma$, and $\perp$ otherwise.

Verify takes as input the system parameters spar, a message $m \in \mathcal{M}$, the data owner's secret key sk_D, and a full set of shares $\sigma_1, \ldots, \sigma_n$. It outputs 1 or 0, depending on, respectively, whether or not the shares are valid shares leading to the reconstruction of message m.

$\langle A.\mathsf{Audit}; \{S_i.\mathsf{Audit}\}_{i=1}^n\rangle$ with respect to the auditor A, it takes as input the system parameters spar and the data owner's public key pk_D. With respect to each storage server S_i, it takes as input the system parameters spar, the data owner's public key pk_D, and its share σ_i. It outputs 1 if the storage servers passed the audit, and 0 otherwise.

Depending on the concrete instantiation of the auditable distributed storage system, the generation of publicly available system parameters with algorithm Setup can be done by a trusted third party, which in practice can be realized through a

joint computation of multiple parties. Afterwards, a data owner D who wants to securely store its data on the selected storage servers $S_1, \ldots, S_n$ computes its key pair using KGen. A message can then be outsourced to the distributed storage system using Store, and later be retrieved using Reconstruct. The received shares from the different storage servers can be verified using Verify. Finally, the interactive auditing protocol $\langle A.\mathsf{Audit}; \{S_i.\mathsf{Audit}\}_{i=1}^{n} \rangle$ is executed between a potentially external auditor A and the storage servers.

We now define the *security and privacy properties* required for auditable distributed storage systems. More precisely, besides completeness, an auditable distributed storage system should guarantee that even if a subset of the storage server in the distributed storage system colludes, no information about the stored data is leaked. Furthermore, the auditor should accept an execution of the audit protocol if and only if all storage servers have access to valid shares for the outsourced message that was originally stored by the data owner. This property is referred to as the extractability property. For practical purposes, we do not require here that the auditor and the data owner are the same entity. Therefore, it is of prime importance that even a malicious auditor has no chance to learn any information about the data stored by a data owner. This should hold even if the malicious auditor colludes with a subset of storage servers, and even in case the storage servers jointly deviate from the original protocol specification. In the following, we formalize the notion of completeness in Section 4.2.1.1, the notion of privacy in Section 4.2.1.2, and the notion of extractability in Section 4.2.1.3.

4.2.1.1 Completeness

The *completeness* property captures the intuitive requirement that if all parties are honest and follow the protocol specifications, the auditor should always output 1.

Definition 4.3. *An auditable distributed storage system composed on n storage servers is* complete *if the auditor outputs* 1 *with probability* 1 *and accepts the shares as valid, given that all the parties involved, i.e. data owner and storage servers, behaved honestly.*

The completeness property of the auditable distributed storage systems presented in Definition 4.2 implies that algorithm Reconstruct should always return the correct original message if it receives as input a authorized subset of shares, i.e. that the underlying secret sharing scheme fulfills the completeness property in the sense of Definition 2.1. The same holds for algorithm Verify, which should always output 1 if all shares are consistent with the message, i.e. it should be complete in the sense of Definition 2.3.

4.2.1.2 Privacy

Informally, an auditable distributed storage system is said to be *private* if no adversary can infer any information about the stored message m from a bounded number of shares and arbitrary many runs of the auditing protocol. More formally, we define the privacy game as follows. We let the adversary Adv choose two messages, m_0, m_1, one of which is outsourced to a distributed storage system by storing its shares in the storage servers. Furthermore, the adversary Adv can corrupt at most $t-1$ out of the n storage servers and the auditor A. This message is then audited upon the adversary Adv's request. At the end of the privacy game, the adversary Adv should not be able to tell which of the two messages m_0 and m_1 was distributed in the first place. We denote by $\varepsilon \in [0,1]$ the probability enforced by the adversary Adv that the auditor accepts the shares.

Experiment $\mathsf{Privacy}_{\mathsf{Adv}}(\lambda, n, t)$
  $b \xleftarrow{\$} \{0,1\}$
  $(spar) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$
  $(sk_D, pk_D) \xleftarrow{\$} \mathsf{KGen}(spar)$
  $(m_0, m_1, state) \xleftarrow{\$} \mathsf{Adv}(spar, pk_D)$
  $(\sigma_1, \ldots, \sigma_n) \xleftarrow{\$} \mathsf{Store}(m_b, spar, sk_D)$
  $b' \xleftarrow{\$} \mathsf{Adv}^{\mathcal{O}(\cdot)}(state)$
    where $\mathcal{O}(\cdot) = \mathcal{O}(spar, pk_D, \sigma_1, \ldots, \sigma_n, \cdot)$ behaves as follows:
      On input $(\texttt{corrupt}, S_i)$:
        mark S_i as corrupt
        return σ_i to the adversary
      On input $(\texttt{corrupt}, A)$:
        mark A as corrupt
        return ε to the adversary
      On input $(\texttt{audit})$:
        simulate all honest parties in an execution of
          the auditing protocol
    Let t' be the total number of corruption requests.
  return $(b \stackrel{?}{=} b') \ \wedge \ (t-1 \stackrel{?}{\leq} t')$

Figure 4.1: Privacy definition game for an auditable distributed storage system.

Definition 4.4. *An auditable distributed storage system composed of n storage servers is* t-private*, if for every PPT adversary Adv there exists a negligible function negl such that the following holds:*

$$\left| \Pr\left[\mathsf{Privacy}_{\mathsf{Adv}}(\lambda, n, t) = 1\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda)\,,$$

where the privacy game $\mathsf{Privacy}_{\mathsf{Adv}}(\lambda, n, t)$ *is defined in Figure 4.1.*

Note that this definition implies the privacy of the distributed storage system itself. For instance, the adversary can break into enough storage servers as to collect up to $t-1$ shares and must not be able to tell which message was outsourced to the distributed storage system.

4.2.1.3 Extractability

Informally, an auditable distributed storage system is said to be *extractable* if it can only pass a run of the auditing protocol if every storage server actually stored the share received from the data owner. This property is formalized in the spirit of proofs of knowledge discussed by Belgol in [10]. More precisely, we require that a set of storage nodes can only pass the audit protocol, if the original outsourced data is *still present in the system.* That is, there must be an algorithm E that can extract the messages from the single storage node. Therefore, algorithm E is given *rewindable black-box access* [7] to the storage servers. This means that algorithm E does not have access to the code of the (potentially malicious) storage server S_i, but it can reset and rerun the system arbitrarily and can also control their random tapes. Also, algorithm E can force the servers to rerun the audit protocol again on the same random tape. We stress that this approach is in line with a long line of research on zero-knowledge proofs of knowledge.

Experiment $\mathsf{Extractability}_{\mathsf{Adv}}(\lambda, n, t)$
$(spar) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, n)$
$(sk_D, pk_D) \xleftarrow{\$} \mathsf{KGen}(spar)$
$(m, \mathcal{I}) \xleftarrow{\$} \mathsf{Adv}(spar, pk_D)$
where $|\mathcal{I}| \leq t-1$
$(\sigma_1, \ldots, \sigma_n) \xleftarrow{\$} \mathsf{Store}(m, spar, sk_D)$
$\{\mathsf{t}_i\}_{i=1}^{\ell} \xleftarrow{\$} \mathsf{E}_1^{S_1,\ldots,S_n}$
where $\{S_i\}_{i\in\mathcal{I}}$ are controlled by $\mathsf{Adv}(\{\sigma_i\}_{i\in\mathcal{I}})$,
and E_1 has rewindable black-box access to the system
$(\sigma'_1, \ldots, \sigma'_n) \xleftarrow{\$} (\mathsf{E}_2^1(\{\mathsf{t}_{i\downarrow 1}\}_{i=1}^{\ell}, \ldots, \mathsf{E}_2^n(\{\mathsf{t}_{i\downarrow n}\}_{i=1}^{\ell})$
where $\mathsf{t}_{i\downarrow j}$ is the i^{th} transcript reduced to only the
messages that were exchanged with S_j
return $\mathsf{Verify}(spar, m, sk_D, \sigma'_1, \ldots, \sigma'_n)$

Figure 4.2: Extractability definition game for an auditable distributed storage system.

Definition 4.5. *An auditable distributed storage system composed of n storage servers is* t-extractable *with retrievability error $\rho : \mathbb{N} \to [0, 1]$, if there exists a state*

less algorithm $\mathsf{E} = (\mathsf{E}_1, (\mathsf{E}_2^1, \ldots, \mathsf{E}_2^n))$ *such that for every adversary* Adv *that makes the auditor accept with probability* $\varepsilon > \rho$ *it holds that:*

$$\Pr[\mathit{Extractability}_{\mathsf{Adv}}(\lambda, n, t)] = 1\,,$$

where the expected running time of E *is upper-bounded by* $\mathrm{poly}(\lambda)/(\varepsilon(\lambda) - \rho(\lambda))$.

Definition 4.5 is similar to the one for distributed proofs of knowledge proposed by Ben-Or et al. in [11], where multiple provers need to convince a verifier that they jointly know a secret piece of information and it is not, in general, an information-theoretically secure property (contrary to its instantiation in the following section). However, our definition differs in several subtle aspects from this definition, as discussed in the following.

- First and foremost, in our framework, the definition of distributed proofs of knowledge of [11] would only have guaranteed that all valid witnesses exist *in the system*. That is, it is only guaranteed that the witness (i.e., all shares) can be extracted from the set of provers (i.e., the storage servers). This would entail the possibility for malicious storage nodes to delete their shares, as long as these shares can be recomputed from the remaining ones. However, this contradicts the intuition of our auditing protocol, as it needs to be guaranteed that every single node correctly stored its share. For this reason it is necessary to split the extractor into two phases, where the actual "extraction" may only use the interaction of the extractor with the respective server.
- Second, in (distributed) proofs of knowledge, the knowledge extractor takes as input also the value for which knowledge the witness needs to be proven. This is sensible, as the extractor should in particular be able to play the role of the verifier. However, in our case the verifier (i.e., the auditor) does not know this value (i.e., m), and thus it would be unnatural to give m to E.
- Finally, the standard definition of (distributed) proofs of knowledge [11] is fully independent of valid witnesses, i.e., the σ_i in our definition. However, in our definition it is crucial to start from a valid set of shares. In fact, depending on the instantiation the adversary may only be allowed to corrupt a limited number of servers (parametrized by the allowed size of $\mathcal{I}$). For the other servers to be able to honestly play their parts in the protocol it is crucial for them to receive their valid shares as inputs.

4.2.2 Instantiation Based on Shamir Secret Sharing

In the following, we present an auditable distributed storage system that is based on an arbitrary additively homomorphic threshold secret sharing scheme. For clarity of presentation, we instantiate the auditable distributed storage system with Shamir's threshold secret sharing scheme presented in Section 2.1. Our Shamir's threshold secret sharing-based instantiation of the auditable distributed storage

system presented in Definition 4.2 is a completely keyless system because of the nature of secret sharing-based distributed storage systems. Furthermore, since our system does not require any pre-processing of messages, it can be applied directly to any data that has already been outsourced to a distributed storage system based on Shamir's threshold secret sharing scheme.

The rest of this section is organized as follows. In Section 4.2.2.1, we discuss two natural auditable distributed storage systems and discuss their shortcoming to further motivate our solution. Then, in Section 4.2.2.2, our instantiation is presented and is formally proven secure in Section 4.2.2.3. In Section 4.2.2.4, we show how to significantly increase the efficiency of our basic protocol using a batch technique. For instance, the use of a batching technique allows to audit at once multiple messages, potentially across multiple data owners. In Section 4.2.3, it is discussed how single storage servers can be blamed in case of failure.

4.2.2.1 Canonical Instantiations

In the following, we briefly discuss two seemingly natural instantiations for auditable distributed storage systems, and explain their shortcomings.

On the one hand, one might first encrypt the message using an arbitrary encryption scheme, and then store replicas of it on n storage servers. To audit, each storage server sends a hash of the ciphertext to the auditor, who accepts if and only if all storage servers return the same hash value. This scheme would be very efficient and directly support a batch version (also across different data owners) by simply hashing an entire batch of ciphertexts. Furthermore, the privacy of the scheme does not depend on a non-collusion assumption between the servers. However, the extractability property requires a non-collusion assumption between any 2 (not t) storage nodes. If this is not the case, it cannot be guaranteed that each storage server actually stores a full replica of the data, because a storage server could just send the correct hash value to another one that stores nothing at all but that can then simply forwards the hash value to the auditor. Moreover, this instantiation is not keyless and is computationally secure only, which makes it not suitable for long-term storage of confidential data.

On the other hand, to achieve information-theoretic privacy, one might first share the data using a secret sharing scheme, then apply standard proof of retrievability and proof of data possession techniques (see 4.2.4) to the single shares, and then audit every server independently. On the positive side, the extractability property in this case does not depend on a non-collusion assumption, while the privacy property clearly does. However, the scheme is not keyless, and furthermore none of the existing publicly verifiable private data checking techniques supports batch verification across multiple data owners. Furthermore, this approach is not compatible with existing storage solutions, does not (directly) support proactive steps, and also introduces a storage overhead compared to regular secret sharing schemes.

Instead, our instantiation based on Shamir's threshold secret sharing scheme provides information-theoretic confidentiality , allows for efficient batch audits across different data owners, does not require any storage overhead, supports proactivity, and is backward compatible with existing solutions. This is achieved by leveraging the non-collusion assumption and built in redundancy of the underlying distributed storage system to also prove privacy and extractability of the resulting scheme.

4.2.2.2 An Audit Protocol for Single Messages

In this section, we present our protocol for auditable distributed storage systems when these are built on Shamir's threshold secret sharing scheme. Thus, algorithm Store and algorithm Reconstruct correspond, respectively, to algorithm Share and algorithm Reconstruct of Shamir's threshold secret sharing scheme presented in Section 2.1. The high-level idea of our auditing mechanism for distributed storage systems is the following. The storage servers involved in the system jointly compute a distributed random value. That is, each storage server obtains a share of the randomness used to blind the actual shares of the message. And the blinding of each share of the message is done by adding the corresponding share of the randomness to it. If all those sums are consistent, the auditor accepts the audit, otherwise it rejects.

Setup takes as input a security parameter λ and the number of servers n. It outputs the system parameters:

$$spar = (q, t, n, (S_1, \ldots, S_n)) \,,$$

where $q > n$ is a prime number defining the field $\mathbb{F}_q$ as both the message space and the shares space, $t \leq \frac{n-1}{2}$ is an integer defining the minimum amount of shares required to reconstruct the secret, and $i \in \mathcal{I}$ is the unique identifier of storage server S_i, for $i = 1, \ldots, n$.

KGen takes as input the system parameters $spar$. It outputs a key pair (sk_D, pk_D) for the data owner D. The key pair is computed setting each key as the probability ε that the auditor passes the audit:

$$(sk_D, pk_D) = (\varepsilon, \varepsilon) \,.$$

Store takes as input a message $m \in \mathbb{F}_q$, the system parameters $spar$, and the data owner's secret key sk_D. It outputs a share σ_i to be sent to storage server S_i for $i = 1, \ldots, n$. The shares $\sigma_1, \ldots, \sigma_n$ are computed as follows:

$$(\sigma_1, \ldots, \sigma_n) \xleftarrow{\$} \mathsf{Shamir.Share}(m, q, t, n) \,,$$

where Shamir.Share is Shamir's secret sharing algorithm (see Section 2.1). Note that each S_i receives the corresponding σ_i over a secure channel.

Reconstruct takes as input the system parameters $spar$, the data owner's secret key sk_D, a set of shares $\sigma_1, \ldots, \sigma_r$ held by a subset $R \subset S$ of shareholders. It outputs $\perp$ if $r < t$. Otherwise, it checks whether there exists a unique interpolation polynomial $f(x)$ of degree t such that $f(i) = \sigma_i$ for all $i = 1, \ldots, r$. If this is the case, it outputs $m' = f(0)$; otherwise it outputs $\perp$.

Verify takes as input the system parameters $spar$, a message $m \in \mathbb{F}_q$, the data owner's secret key sk_D, and a full set of shares $\sigma_1, \ldots, \sigma_n$. It outputs 1, if and only if:

$$m \stackrel{?}{=} \mathsf{Reconstruct}(spar, sk_D, \sigma_1, \ldots, \sigma_n).$$

Otherwise, it outputs 0.

$\langle A.\mathsf{Audit}(spar, pk_D); \{S_i.\mathsf{Audit}(spar, pk_D, \sigma_i)\}_{i=1}^{n} \rangle$ consists of the following steps.

1. The storage servers $S_1, \ldots, S_n$ jointly compute a distributed uniformly random value in $\mathbb{F}_q$ by running algorithm RandShares introduced in Section 3.3.3.1 and adapting it to the access structure of Shamir's threshold secret sharing scheme. That is, at the end of this interaction, every server storage server S_i has a share α_i of a Shamir-shared random value α with threshold t.
2. The auditor A broadcasts a challenge value $c \stackrel{\$}{\leftarrow} \mathbb{F}_q$ to the storage servers $S_1, \ldots, S_n$.
3. Each storage server S_i computes locally $\eta_i = c\sigma_i + \alpha_i$ by running algorithm Linear introduced in Section 3.3.2.
4. Each storage server S_i returns η_i to the auditor A.
5. The auditor A outputs 1 if and only if the provided $\eta_1, \ldots, \eta_n$ are all consistent, i.e., if:

$$\mathsf{Reconstruct}(spar, sk_D, \eta_1, \ldots, \eta_n) \neq \perp .$$

We stress that the above auditing mechanism for distributed storage systems is fully key-less. In particular, the data owner D does not need to store any information, which relieves it from any complex key management issues when accessing the data from different devices. Our protocol actually proves that all shares held by the different storage servers have a valid (degree-t) interpolation polynomial, i.e., that the shares are consistent. Consistency with the originally shared message (and therefore integrity of the shares) follows from the assumption that the majority of the storage servers is honest, and thus their shares are consistent with the original message.

4.2.2.3 Security Proofs

In the following, we provide detailed proofs that the system described above satisfies the security properties of completeness, privacy, and extractability defined,

respectively, in Section 4.2.1.1, Section 4.2.1.2, and Section 4.2.1.3.

Theorem 4.6. *The above scheme is complete.*

Proof. This property follows trivially from the completeness of Shamir's threshold secret sharing scheme, the completeness of algorithm RandShares, and the completeness of algorithm Linear for the additivity of shares for the same threshold. □

Theorem 4.7. *The above scheme is t-private according to Definition 4.4.*

Proof. On the one hand, if the auditor A is not corrupted, then privacy follows from the fact that Shamir's secret sharing scheme does not reveal any information if at most $t-1$ shares are known. On the other hand, if the auditor A is corrupted, still the responses received from honest storage servers do not contain any information about their original shares (and thus about the message). That holds because algorithm RandShares used for computing the shared randomness guarantees that all shares α_i obtained by honest servers are uniformly random. □

Theorem 4.8. *The above scheme is t-extractable with retrievability error $\rho = 1/q$ according to Definition 4.5.*

Proof. Assume the adversary can make the auditor A accept the audit with probability more than ρ. First, the extractor E_1 uses its rewindable black-box access to obtain two different sets of transcripts with the same shared randomness but different challenges c_1 and c_2. This can be done by rewinding the system of servers to the end of Step 1 in the auditing protocol, or to the beginning of the protocol. The precise mode of operation of extractor E_1 is identical to the knowledge extractor in the proof that every Σ-protocol is a proof of knowledge, and can be found, e.g., in Damgård [39]. From there it follows that the running time of extractor E_1 is bounded above by $\text{poly}(\lambda)/(\varepsilon(\lambda) - \rho(\lambda))$. Let the transcripts be given by $\mathsf{t}_1 = (c_1, (\eta_{1,1}, \ldots, \eta_{1,n}))$ and $\mathsf{t}_2 = (c_2, (\eta_{2,1}, \ldots, \eta_{2,n}))$. We then have $\mathsf{t}_{1,i} = (c_1, \eta_{1,i})$, and $\mathsf{t}_{2,i} = (c_2, \eta_{2,i})$.

Second, the extractor E_2^i outputs

$$\sigma_i' = (\eta_{1,i} - \eta_{2,i}) \cdot (c_1 - c_2)^{-1} \in \mathbb{F}_q,$$

where this computations can be done in polynomial time.

What now remains to be shown is that these shares are indeed consistent shares for the original message m. To see this, it holds that $\eta_{j,1}, \ldots, \eta_{j,n}$ are consistent shares for message R_j by Step 5 for $j = \{1, 2\}$. Thus, by the linearity of Shamir's secret sharing scheme, $\sigma_1', \ldots, \sigma_n'$ are consistent shares for $m' = (R_1 - R_2) \cdot (c_1 - c_2)^{-1}$. Since by construction $n \geq 2t + 1$, it follows that there are at least t storage servers that replied honestly in both transcripts. Clearly, the corresponding share σ_i' is consistent also with the original message m. This holds because $\eta_{j,i} = c_j \sigma_{j,i} + \alpha_i$ and thus $\sigma_i' = \sigma_i$. As those (at least) t shares uniquely determine the shared message, we have that $m' = m$. □

4.2.2.4 Batch-Auditing of Stored Messages

The main drawback of the auditing mechanism for distributed storage systems presented in Section 4.2.2.2 is that it requires to compute a fresh distributed random value α for each message to be audited. We show how to reduce these communication costs to a practically negligible amount if a large number of data blocks are audited at the same time. This is particularly interesting in our setting, where no pre-processing is necessary and where it is possible to simultaneously audit a large batch of messages across different data owners. For instance, the auditor can audit the storage solution of a company as a whole without running our auditing mechanism individually for each employee. This is possible because the employees do not need to use different private keys during algorithm Store. The only requirement is that all messages to be batch-audited have been shared for the same system parameters *spar*, i.e., using the same threshold t and the same storage servers $S_1, \ldots, S_n$.

Let us assume that each of the messages $m_1, \ldots, m_\ell$ has been distributed and stored according to Section 4.2.2.2, resulting in shares $\sigma_{1,i}, \ldots, \sigma_{\ell,i}$ on each storage server S_i, for $i = 1, \ldots, n$.

$\langle A.\mathsf{Audit}(spar, pk_D); \{S_i.\mathsf{Audit}(spar, pk_D, \{\sigma_{j,i}\}_{j=1}^{\ell})\}_{i=1}^{n}\rangle$ works just as in the basic case discussed in Section 4.2.2.2, except that in Step 3 each storage server S_i computes its response η_i as:

$$\eta_i = \sum_{j=1}^{\ell} c^j \sigma_{j,i} + \rho_i.$$

That is, the response is computed as a polynomial hash of the shares stored by the server. Similarly to the proof of Theorem 4.8, it can be shown that for every constant ℓ the resulting protocol is a t-extractable system (see Definition 4.5) with retrievability error $(\ell-1)/q$ in two steps. In the first step, rewinding allows to extract sufficiently many transcripts as in the case of Σ_m-protocols discussed by Benhamouda et al. in [13]. The second step essentially corresponds to solving a linear system of ℓ equations. We stress that the communication complexity is exactly the same as for the basic auditing protocol for one message, and that, in particular, it is independent of the batch size ℓ. The computational complexity in the batch setting consists of the joint computation of a single distributed random value, computing a sum for each server, and calling Verify once on the auditor's side.

Note that both the basic auditing protocol presented in Section 4.2.2.2 and the batch version it is required that $n \geq 2t+1$, i.e. that the majority of the storage servers is honest. In this setting, it is possible for the auditor to only detect inconsistencies among the shares stored across the storage servers only. However, the auditor cannot identify which shares (and thus which storage servers) caused the inconsistency. Instead, this is possible whenever $n \geq 3t + 1$. Also, once the malicious storage

servers are identified, one can either replace them or recompute consistent shares by adapting algorithm Add presented in Section 3.2.3 to the right access structure.

4.2.3 Protecting Against Malicious Data Owners

So far, in Section 4.2.1 and in Section 4.2.2, we have provided, respectively, a definition and an instantiation of auditable distributed storage systems. Such auditable mechanism for distributed storage systems protects the integrity of the stored data through a third party auditor, which checks periodically that the shares are stored correctly. The underlying assumption is the the data owner is honest, i.e. that it does not deviate from algorithm Store and distributes to the storage servers valid shares. In this section, we do not assume this any longer and, instead, allow malicious data owners to distribute inconsistent shares of data to the storage servers, and then try to harm their reputation and trustworthiness by announcing an unjustified complaint.

As already discussed in Section 2.1, verifiable secret sharing is a countermeasure against malicious data owners. However, this solution is highly inefficient. For instance, Shamir's secret sharing scheme is usually implemented with $|q|$ being comparatively small for efficiency reason. That is, larger files are first split into multiple (say: ℓ) blocks of several bytes each before they are actually shared. In this case, $\ell(t+1)$ commitments (either Feldman or Pedersen's commitments) need to be computed and distributed by the data owner for a single file. Furthermore, each storage server has to compute at least ℓ full-length and $\ell(t+1)$ non-full-length exponentiations in a group with ≈ 256-bit order. Given that for files in the range of several megabytes ℓ would thus be at least 2^{15} to 2^{20}, this makes the existing verifiable secret sharing schemes impracticable to protect storage servers against malicious data owners in a distributed storage systems.

We show how our auditing protocol from Section 4.2.2 can be extended to a batch verification protocol for Shamir's scheme assuming the following rational behavior of the storage servers. That is, we assume that each storage server aims at remaining in the set of storage servers. However, each server may be interested in excluding data owners from the distributed storage system, or in blaming an honest storage server for misbehaving (potentially in collaboration with the data owner). We believe that this assumption is reasonable from an economic point of view, as storage servers do not profit from being excluded from a system because of misbehavior. Instead, storage servers may benefit from a reduced reputation of competitors. In the following, we provide a high-level description of the verification protocol based on the auditing mechanism of Section 4.2.2.

1. After having received shares $\sigma_{j,i}$ for $j = 1, \ldots, \ell$ to each storage server S_j for $i = 1, \ldots, n$, the storage server jointly (randomly or deterministically) agree on

a set of $2t+1$ storage servers, say $S_{i_1}, \ldots, S_{i_{2t+1}}$.
2. The selected storage servers $S_{i_1}, \ldots, S_{i_{2t+1}}$ initiate parallelized instances of the batch auditing protocol described in Section 4.2.2.4, in which storage server S_{i_u} takes over the roles of both of a storage server and the auditor, for $u = 1, \ldots, 2t+1$.
3. Each storage server $S_{i,u}$ broadcasts the result (either 1 or 0) of its audit, for $u = 1, \ldots, 2t+1$.
4. Storage server S_i accepts its batch $\sigma_{1,i}, \ldots, \sigma_{\ell,i}$ of shares if it received at least t result messages 1, and rejects its shares otherwise, for $i = 1, \ldots, n$.

Note that as a direct consequence of our assumption and because of the results from Section 4.2.2, in the above protocol every storage server acts honestly in the role of a server and might only cheat when acting as an auditor, as otherwise the misbehavior would be detected. The following result holds.

Theorem 4.9. *The protocol above is complete and committing according to Definition 2.3 of a verification protocol assuming the rational behavior of the storage servers defined above.*

Proof. Completeness follows immediately from the completeness of our auditing procedure and the fact that by assumption storage servers behave honestly when playing the role of a storage server in our protocol. Also, it is also easy to see that no information about the unknown shares is leaked to a set of at most t collaborating shares.

To see that the protocol is committing, note that because t received 1 messages, at least one such message was sent from an honest node. Now, by Theorem 4.8, it follows that with overwhelming probability all shares are consistent for all $j = 1, \ldots, \ell$ and $i = 1, \ldots, n$, as otherwise the valid shares could not be extracted from the storage servers. □

Similar to the batch-auditing protocol presented in Section 4.2.2.4, the storage servers' communication complexity of this batch verification protocol is independent of the batch size. In particular, the computational complexity for each storage server essentially consists on computing $2t+1$ weighted sums over the shares to be verified, where this sums can be computed in parallel for efficiency reasons. The data owner does not have to perform any computations.

Summing up, the above protocol significantly reduces the complexity of verifying the shares distributed by a data owner compared to currently used solutions with Pedersen or Feldman commitment, assuming a rational behavior of the storage servers. This results in the first practically verifiable secret sharing based distributed storage system.

4.2.4 Related Work

In the following, we discuss related work for multi server storage verifiable schemes.

Schwarz and Miller [101] propose a scheme that allows a client to verify the storage of (n, m) erasure-coded data across multiple storage servers even if they collude. The scheme can also be used to verify storage on a single storage server and relies on a special construct, called algebraic signatures. The solution was intended for application in peer to peer networks and allows the creation of very large-scale verifiable distributed storage systems. The authors also propose performance optimizations to achieve checking throughputs of hundreds of Mbytes/sec. Their approach is the first to show that the distributed setting allows for very efficient solutions for remote data checking with low computational and low storage overhead and minimal communication requirements. However, the scheme only receives an informal security analysis and requires secret keys. Moreover, privacy of data is not considered and therefore third party auditing is not possible.

Other extensions to remote data checking include extending the data possession guarantee to multiple storage servers based on replication without encoding each replica separately. For example, the multiple-replica provable data possession scheme proposed by Curtmola et al. in [37] allows a client that stores t replicas of a file in a storage system to verify through a challenge-response protocol that (1) each unique replica can be produced at the time of the challenge and that (2) the storage system uses t times the storage required to store a single replica. Multiple-replica provable data possession schemes extend previous work on data possession proofs for a single copy of a file in a client/server storage system. Using multiple-replica provable data possession schemes to store t replicas is computationally much more efficient than using a single-replica provable data possession scheme to store t separate, unrelated files (e.g., by encrypting each file separately prior to storing it). Another advantage of multiple-replica provable data possession schemes is that it can generate further replicas on demand, at little expense, when some of the existing replicas fail.

In [111] and [20] solutions based on erasure coding are introduced which also include a sound security treatment. The solution of Wang et al. [111] applies a special encoding scheme to construct a systematic Reed-Solomon (RS) code together with a challenge-response protocol which not only detects the retrievability state as a binary value, but also provides the localization of data error in an efficient way. Additionally, the scheme supports secure and efficient dynamic operations on data blocks, including: update, delete, and append. Furthermore, an extensive security and performance analysis shows that the proposed scheme is highly efficient and resilient against Byzantine failures, malicious data modification attacks, and even server colluding attacks.

In [20] integrity-protected error correcting codes are introduced together with a data checking framework called HAIL. Integrity-protected error correcting codes are cryptographic primitives that act both as a message authentication as well as

an error-correcting code. Integrity-protected error correcting codes achieve cross server redundancy through error-correcting codes and represents a corruption resilient message authentication schemes on the underlying data. The construction of an integrity-protected error correcting codes in [20] is based on a $(n, l, n-l+1)$ Reed-Solomon code and can be adapted for systematic Reed-Solomon codes (l is the number of elements in the file vector $\mathbf{f}$). To tag a file f, it is encoded under the RS code, and then a suitable pseudo random function is added to the last s code symbols (for $1 \leq s \leq n$ being a system parameter), obtaining a message authentication code on each of those s code symbols. When reconstructing a file, a codeword is considered valid if at least one of its last s symbols are valid message authentication codes under universal hashing on the decoded file f.

In summary, only few approaches have been proposed for remote data checking in a distributed setting comprising multiple servers, and what all these schemes have in common is that they do not give formal privacy guarantees. The situation is similar for most single-server instantiations, except for a few protocols, e.g., Gritti et al. [55, 56]. Note that simply encrypting the data before storing it into the cloud would mitigate this problem, but could not offer information-theoretic privacy, which arguably is desirable for long-term archiving of confidential data.

4.3 Summary and Future Work

In this chapter, we have shown how confidentiality and integrity of the data outsourced to a distributed storage system can be maintained in the long-term.

On the one hand, long-term protection of the confidentiality of the data relies on high-performing storage servers reliabily carrying out proactive secret sharing periodically. A performance scoring mechanism must be in place in order to provide data owner with up to dated performance figures, so that they can select the storage servers making up the distributed storage system accordingly. Then social secret sharing can be instantiated to distribute more informative shares to the better performing storage servers. However, as the performance of the storage servers changes over time and so the aggregate performance scores assigned to them, current social secret sharing solutions do not allow to modify the underlying secret sharing scheme. Thus, it is not possible to modify the access rules nor to discard low-performing storage servers and include better ones into the system. That is, current solutions for social secret sharing-based distributed storage systems do not fully use the performance scoring mechanism in place to enhance the confidentiality protection of the data. We defined a new primitive, called adaptive social secret sharing, that demands the usage of a dynamic secret sharing scheme to set up the distributed storage system. This way, the flexibility needed to modify the access rules and the set of storage servers is provided. We then instantiated this primitive by using Shamir's secret sharing scheme and Tassa's hierarchical conjnctive and disjunctive

secret sharing schemes, which are dynamic. In our instantiation, we also included alarms that are triggered whenever instable states where the performance of the storage servers is below a certain threshold are reached. This way, it is possible to control when to reset the underlying secret sharing scheme. We show how our instantiations outperform state of the art solutions for social secret sharing not only in terms of the protection of confidentiality, but also in terms of storage consumption.

On the other hand, long-term protection of integrity of the data relies on checking periodically that the shares stored into the storage servers are consistent with the outsourced data. When the third party in charge of the auditing mechanism is not trusted, then such auditing mechanism has to be privacy preserving too. The third party auditing mechanisms presented so far either relied on encryption or are suitable for a single-storage server setting. The auditing mechanism needed for our setting should not only be privacy preserving in an information-theoretic sense, but also should run for multiple storage servers. We provide the first auditable mechanism for distributed storage systems that is information-theoretic private. We first defined the syntax of auditable distributed storage systems and then we provided two instantiations based on Shamir's secret sharing scheme. One for a single message and one for multiple messages from (possibly) multiple data owners. The non-collusion assumptions and the homomorphic property of Shamir's secret sharing schemes allow to perform the auditing without compromising the confidentiality of the data, provided that the auditor does not collude with more than a threshold of storage servers. The auditing mechanism for distributed storage servers that we proposed can also be adapted to a verifiable secret sharing scheme to prevent malicious data owners from distributing to the storage servers inconsisten shares in the first place.

As future work, we plan to implement both our instantiation for adaptive social secret sharing and for auditable distributed storage systems. In particular, regarding the third party auditing mechanism, we plan to investigate how the non-collusion assumption of the auditor can be relaxed, while still relying on one third party auditor and not many of them. In particular, we want to simulate the third party auditor with a trusted execution environment to see how the auditing mechanism in place can be simplified.

5 Coalition-Resistant Performance Scoring Mechanism for Long-Term Confidentiality

As discussed more extensively in Chapter 1, the long-term confidentiality of data outsourced to distributed storage systems relies in practice on high-performing storage servers, reliably carrying out proactive secret sharing. The performance of the storage servers varies over time, and data owners require guidance to select the individual storage servers making up the distributed storage system. The performance (understood in a broad sense, notably including reliability) of the individual SSPs is the main criterion for inclusion in the distributed storage system. However, data owners do not have access to comprehensive and comparable performance figures for candidate storage servers. Data owners can be guided in their choice if a trusted third party measures and publishes performance figures for the storage servers. However, for a large number of storage servers and frequent measurements, the workload becomes unwieldy for a single entity. Aggregated peer rating is an alternative approach that distributes the workload. In this approach, the storage servers involved in the distributed storage system provide mutual ratings for each others' performances based on the interactions they had with each other. However, SSPs benefit commercially in having the storage servers they own included in the distributed storage system of a data owner. They might as well benefit commercially from enforcing the storage servers they own to provide selfish ratings for the other storage servers to undermine competitors. Moreover, SSPs might even form coalitions among themselves, making the selfish ratings issued by their storage servers even more harming for the storage servers owned by other SSPs. Thus, the aggregate scores computed through mere averaging the ratings are unreliable. Distributed storage systems based on those aggregate scores might fail to successfully carry out proactive secret sharing and, thus, put the confidentiality of the data at risk.

Contribution

In this chapter, we provide the first performance scoring mechanism that outputs aggregate scores within an accuracy interval, given that the SSPs are interested in maximizing the amount of shares to store and given that the majority of the storage servers are honest and submit accurate ratings. We show this by modeling the performance scoring mechanism as a cooperative game with infinite rounds and a trust authority as a mediator. The trust authority (from now on referred to as TA) is responsible for collecting all the mutual ratings and penalizes the submitters of the inaccurate ones, and proving that it reaches a k-Nash equilibrium. For each storage server, at each round the performance scoring mechanism is run, its aggregate score is computed as a combination of three terms. The first term is the weighted average of the mutual ratings from the other storage servers. The second term is the average of the penalties and incentives that the storage servers got for rating, respectively, inaccurately and accurately the other storage servers. The third term of the aggregate score of the storage server at the previous round the performance scoring mechanism was run. The term leading to aggregate scores that are within the selected accuracy interval with high probability is the second one, which exploits the income-driven nature of commercial SSPs. In order to define how this second term is computed, we formalize, for the first time, the computation of aggregate scores through mutual ratings in distributed storage systems as an economic problem and provide a game-theoretical model of the peer rating strategies of the storage servers and the SSPs. In the framework of game theory, SSPs and, consequently, storage servers are seen as rational agents trying to increase their utility by having high aggregate scores. This second term, which decreases and increases the final aggregate scores, and the fact that the storage servers do not know how many rounds the performance scoring mechanism is run (i.e. it is modeled as an infinitely repeated game) are necessary to output aggregate scores within the accuracy interval. We argue this by showing that, without, the aggregate scores do not mirror the performance of the storage servers. Furthermore, we provide an instantiation and an evaluation of the performance scoring mechanism to show how, in practice, inaccurate ratings are detected and the second term lowers the aggregate scores of the storage servers submitting inaccurate ratings. The performance scoring mechanism instantiation uses machine learning techniques to compute the above mentioned first term.

Contributions to this chapter come from papers [T5], [T6], and [T7]. My contributions in these works were the problem statement of rationals storage servers and the confidentiality threats entailed, the formalization of the preferences and strategies of storage servers through game theory, and the new performance scoring mechanism modeled as an infinitely-repeated and incentivized game with a mediator.

Outline

The outline of this chapter is as follows. Section 5.1 and Section 5.2 explain the game-theoretic background necessary to define and use the second term of which the aggregate scores computed by the performance scoring mechanism are composed of. More precisely, in Section 5.1, we provide a formalization based on game-theory of the strategies that storage servers may adopt when mutually rating each other in a distributed storage system. In Section 5.2, we prove that performance scoring mechanisms where accurate ratings are not incentivised and with a finite amount of rounds output aggregate scores that are inconsistent with the actual performance of the storage servers. Section 5.3 presents in detail the performance scoring mechanism and how the three terms each aggregate score is composed of are defined. We prove that this performance scoring mechanism output accurate aggregate scores even when coalitions among the SSPs and the corresponding storage servers are formed, provided an honest majority. In Section 5.4, we instantiate the performance scoring mechanism by machine learning techniques and experimentally evaluate our findings. In Section 5.5, we discuss related work with respect to evidence-based performance scoring mechanisms, in particular the ones in peer-to-peer environments and using machine learning techniques, and with respect to how rational players are tackled in secet sharing schemes. Summary and future work can be found in Section 5.6.

5.1 Game Theoretic Model of Rating Strategies

We recall that the performance scoring mechanism presented in Section 5.3 computes the aggregate scores of the storage servers as a convex combination of three terms. The second term penalizes the submission of inaccurate ratings and of incentivizing the submission of accurate ratings by, respectively, lowering and increasing the value of the final aggregate scores. In this section, we present why lowering and increasing the value of their aggregate scores is of crucial for the income-driven storage servers and the SSPs they are owned by. For a storage server having a high aggregate score compared to the other storage servers increases the chance to not only be kept as part of the distributed storage system, but also to get multiple shares distributed. Staying in the distributed storage system with as many storage servers as possible and getting several shares to store and refresh means a higher utility for the SSPs. Therefore, having high aggregate scores ultimately leads to a higher income.

In the following, in Section 5.1.1, we argue that the rational adversary model is actually the one to be considered in this context and we compare it with the traditional malicious adversary model. In Section 5.1.2, the formalization of the rating strategies of the storage servers and SSPs according to game theory is presented.

5.1.1 The Rational Adversary Model

For the computation of aggregate scores in secret sharing-based distributed storage systems, the rational adversary is the one a performance scoring mechanism has to cope with and, in the following, we motivate why this is the case.

The performance scoring mechanism that we envision for a secret sharing-based distributed storage system is one based on peer-rating. That is, each storage server is a trustor and a trustee at the same time (as defined in Section 2.3.1) as it issues ratings to rate the other storage servers of the distributed storage system with respect to a certain performance metric and is also rated in the same way by them. These ratings are collected by a trusted third-party authority that processes them and outputs the corresponding aggregate score for each of the storage servers. In this context, an adversary is someone that prevents the performance scoring mechanism in place from outputting accurate aggregate scores, i.e. aggregate scores that reflect the performance of the storage severs according to a certain predetermined metric. That means, an adversary can be any storage server of the distributed storage system that issues inconsistent ratings with respect to the performance of the other storage servers so that their final aggregate scores are inaccurate. We argue that the reason behind this type of behavior is that the storage servers and the SSPs they are owned by can benefit economically from issuing inconsistent ratings. More precisely, SSPs are economical entities whose goal is staying in the marketplace for as long as possible while maximizing their payoff at the same time. This is provided as long as the storage servers owned by the SSPs are assigned to comparatively high aggregate scores with respect to the aggregate scores assigned to the storage servers owned by the competitors. Thus, for a SSP, attempting at lowering the aggregate scores of storage servers owned by competitors and at increasing the aggregate scores of the storage servers it own is a viable strategy to achieve the above mentioned goals. Because the aggregate scores are computed by a performance scoring mechanism based on peer-rating, a SSP can easily achieve this by enforcing the storage servers to always issue very positive ratings when the storage servers to be rated are those it owns and very negative ratings when the storage servers to be rated are those owned by the competitors. We believe that this type of adversarial behavior is rational rather than malicious because it is driven by economical purposes rather than by the willingness to break the protection of the data outsourced to the secret sharing-based distributed storage system.

The rational adversary model is similar to the usual malicious adversary model because they might both jeopardize confidentiality and, more in general, the long-term protection of the data outsourced to a secret sharing-based distributed storage system. The difference is that jeopardizing the protection of the data is the primary goal of a malicious adversary, which acts according to this. Instead, jeopardizing

the protection of data is not the primary goal of a rational adversary, but can end up being a byproduct of its strategy to increase its payoff. More precisely, if a rational adversary is successful with the above mentioned strategy, then the aggregate scores that the performance scoring mechanism outputs do not match the performance of the storage servers they are assigned to. Therefore, a data owner making up a secret sharing-based distributed storage system might select storage servers that do have high aggregate scores assigned to, but that in reality are low performing. This might entail a distributed storage systems that is not able to carry out properly the protocols and algorithms inherent to a secret sharing-based long-term storage system, such as the share renewal, putting the confidentiality of the data at risk. However, the rational adversary model differs from the malicious adversary model because the countermeasures that mitigate the latter do not work with the former. More precisely, the countermeasures mitigating a malicious attacker willing to break the protection of data outsourced to secret sharing-based distributed storage system are of cryptographic nature. One of these is actually periodically renewing the shares so that the the adversary does not have enough time to corrupt enough storage servers so that to reconstruct the data by itself. However, this solution does not work for a rational adversary because it does not prevent it from submitting inconsistent ratings when the performance scoring mechanism is run. That is why different countermeasures are needed to prevent a rational adversary to accidentally jeopardize data protection. Because of the economic motivation driving the actions of a rational adversary, game theory offers a natural framework where such countermeasures can be designed.

5.1.2 Formalization

In game theory, the agents involved are referred to as players. In our scenario, the players are the n SSPs involved in the distributed storage system. From now on, we refer to those SSPs as players $P_1, \ldots, P_n$. Players $P_1, \ldots, P_n$ are interested in maximizing their economic return when they offer long-term storage to data owners via a distributed storage system. This entails preferences with respect to the shares that data owners may distribute or withdraw from them, depending on the aggregate scores of players $P_1, \ldots, P_n$. In other words, the aim of players $P_1, \ldots, P_n$ is to increase over time the number of shares they store because this leads to greater income.

In the following, we formalize the preferences of players $P_1, \ldots, P_n$ with respect to gaining or losing shares by defining relevant utility functions. Afterwards, we formalize the preferences of these players with respect to their aggregate scores, because gaining or losing a share depends on the aggregate score assigned to a player. In particular, it depends on the comparison between the aggregate score of a player and the aggregate scores of all other players. Further utility functions related to

aggregate scores are defined for players $P_1, \ldots, P_n$.

We first formalize this scenario for non-cooperatives situations, where players act individually. Then, we formalize this scenario for cooperative situations, where players form coalitions among each other. Players in a coalition act in coordination for mutual benefit.

In a non-cooperative setting, we denote by $U_i(\boldsymbol{\sigma})$ the utility function of a player P_i with respect to gaining or losing a share when the joint strategy is $\boldsymbol{\sigma}$. The utility function $U_i(\boldsymbol{\sigma})$ is defined as follows.

U1) If player P_i gains a share, then $U_i(\boldsymbol{\sigma}) = 1$.
U2) If player P_i neither gains nor loses a share, then $U_i(\boldsymbol{\sigma}) = 0$.
U3) If player P_i loses a share, then $U_i(\boldsymbol{\sigma}) = -1$.

Utility $U_i(\boldsymbol{\sigma})$ is directly related to the economic pay-off of player P_i, because the amount of shares players store and manage is proportional to their economic return. However, the amount of shares gained or lost by players ultimately depends on the given aggregate score. The data owner periodically checks the aggregate scores of $P_1, \ldots, P_n$ and accordingly arranges how the shares are distributed among them.

Periodically, the aggregate scores are updated through the aggregation of the players' mutual ratings. Let us denote by r the last round where the aggregate scores are updated before the data owner checks them. We denote the aggregate scores of players $P_1, \ldots, P_n$ at round r by $\tau_1^r, \ldots, \tau_n^r$, where $0 \leq \tau_i^r \leq 1$, for $i = 1, \ldots, n$. Player P_i eventually gains a share if its aggregate score τ_i^r is high enough to convince the data owner to do so. We formalize this idea of "high enough" as having an aggregate score that is on average higher than all other aggregate scores. Vice versa, having an aggregate score "low enough" to convince the data owner to withdraw a share means that a player P_i is assigned an aggregate score τ_i^r that is on average lower than the aggregate scores of all other players. We formalize this by defining the utility function $u_i(r)$ with respect to the aggregate scores (where the joint strategy $\boldsymbol{\sigma}$ is omitted for simplicity) of a player P_i as follows, for $i = 1, \ldots, n$.

u1) If $\tau_i^r > \frac{1}{n-1} \sum_{j=1, j \neq i}^{n} \tau_j^r$, then $u_i(r) = 1$.
u2) If $\tau_i^r = \frac{1}{n-1} \sum_{j=1, j \neq i}^{n} \tau_j^r$, then $u_i(r) = 0$.
u3) If $\tau_i^r < \frac{1}{n-1} \sum_{j=1, j \neq i}^{n} \tau_j^r$, then $u_i(r) = -1$.

Utility $u_i(r)$ is not directly related to the economic pay-off of player P_i, because having $u_i(r) = 1$ at round r does not necessarily imply that one share is distributed to player P_i. Also, even if its aggregate score is higher than all other aggregate scores at a given round, it does not mean that all other aggregate scores are so low as to convince the data owner to rearrange the shares' distribution. However, consistently

getting high aggregate scores is the only way to obtain additional shares. The output of aggregate scores $\tau_1^r, \ldots, \tau_n^r$ at round r is the result of a repeated non-cooperative game among players $P_1, \ldots, P_n$ of r rounds.

Instead, if players $P_1, \ldots, P_n$ form coalitions, then the computation of the aggregate scores $\tau_1^r, \ldots, \tau_n^r$ at round r is the result of a repeated cooperative game of r rounds. In this case, the pay-off from gaining shares is split among the members of a coalition. Thus, the goal for player P_i is that at least one of his coalition partners gains a share and none of them loses any share. We assume that each player belongs at most to one coalition and that the coalitions are at most as numerous as the players, in which case there will be no coalition because each player plays alone. For a coalition C_k, we denote $J = \{j \mid P_j \in C_k \ \wedge \ U_j(\boldsymbol{\sigma}) = 1\}$, and $J' = \{j' \mid P_{j'} \in C_k \ \wedge \ U_{j'}(\boldsymbol{\sigma}) = -1\}$. That is, we see the indexes of players in the coalition C_k as the union of two subsets, J and J', where J are the indexes of the subsets of players in C_k that gained a share and J' are the indexes of the subset of players in C_k that lost a share. The utility function $U_i'(\boldsymbol{\sigma})$ of player $P_i \in C_k$ with respect to gaining or losing shares within the coalition is defined as follows.

U1') If $\sum_{j \in J} U_j(\boldsymbol{\sigma}) > \sum_{j' \in J'} U_{j'}(\boldsymbol{\sigma})$, then $U_i'(\boldsymbol{\sigma}) = 1$.
U2') If $\sum_{j \in J} U_j(\boldsymbol{\sigma}) = \sum_{j' \in J'} U_{j'}(\boldsymbol{\sigma})$, then $U_i'(\boldsymbol{\sigma}) = 0$.
U3') If $\sum_{j \in J} U_j(\boldsymbol{\sigma}) < \sum_{j' \in J'} U_{j'}(\boldsymbol{\sigma})$, then $U_i'(\boldsymbol{\sigma}) = -1$.

In other words, utility $U_i'(\sigma)$ for a player $P_i \in C_k$ is positive if the amount of players within coalition C_k that gained a share is greater than the amount of players within C_k that lost a share. Vice versa, utility $U_i'(\sigma)$ is negative if the amount of players within coalition C_k that gained a share is smaller than the amount of players within C_k that lost a share. The definition of utility $U_i'(\boldsymbol{\sigma})$ with respect to shares shapes the definition of utility $u_i'(r)$ with respect to the aggregate scores, mirroring what holds for non-cooperative games. The goal of player $P_i \in C_k$ is to maximize the average of the aggregate scores assigned to the players within the same coalition. That is, having a "high enough"aggregate score for a player $P_i \in C_k$ means that the average of the aggregate scores of the players within coalition C_k is higher than the average of the aggregate scores of players outside C_k. Vice versa, having a "low enough"aggregate score means that the average of the aggregate scores of the players within coalition C_k is lower than the average of the aggregate scores of players outside C_k. We denote $M = \{m \mid P_m \in C_k\}$, and $L = \{l \mid P_l \notin C_k\}$. The utility function $u_i(r)$ for a player $P_i \in C_k$ with respect to aggregate scores, where coalition C_k has cardinality n_k, is defined as follows, for $i = 1, \ldots, n$.

u1') If $\frac{1}{n_k} \sum_{m \in M} \tau_m^r > \frac{1}{n-n_k} \sum_{l \in L} \tau_l^r$, then $u_i'(r) = 1$.
u2') If $\frac{1}{n_k} \sum_{m \in M} \tau_m^r = \frac{1}{n-n_k} \sum_{l \in L} \tau_l^r$, then $u_i'(r) = 0$.
u3') If $\frac{1}{n_k} \sum_{m \in M} \tau_m^r < \frac{1}{n-n_k} \sum_{l \in L} \tau_l^r$, then $u_i'(r) = -1$.

We defined the utility of gaining and losing shares as, respectively, 1 and -1 for the sake of simplicity. Instead, defining the utility of gaining or losing shares as the number of, respectively, gained or lost shares is also possible but entails additional formalism. In the remainder of this chapter, for simplicity, we use shorthand notation such as $i \in C_k$ to denote $i \in \{j \mid P_j \in C_k\}$ (and similarly for $i \notin C_k$).

As we have just discussed, a way to obtain more shares is to obtain a comparatively high aggregate score. This can be achieved by behaving rationally (i.e., selfishly) during the performance scoring mechanism. SSPs send potentially selfish mutual ratings to the TA. The TA collects these ratings and computes the aggregate scores. Based on them, the data owner distributes the shares to high-performing SSPs. As mentioned earlier, we use performance in a broad sense, including qualities such as reliability. The performance metric used in practice is independent of our model and results. We formalize this distributed storage system scenario through a game-theoretic perspective modelling the rating strategies of the SSPs.

5.2 Score Inaccuracy of Unincentivised Ratings

In this section, we show why a naive approach with finite amount of rounds and where accurate ratings are not incentivised and inaccurate ratings are not penalized leads to the computations of aggregate scores that do not match the actual performance of the storage servers. As we have already anticipated, by taking into account the rationality of the players formalized in Section 5.1, the performance scoring mechanism presented in Section 5.3 is modelled as an incentivised and infinitely repeated game. We recall that the incentives and penalties are given to the storage servers through the second term which each aggregate score is composed of.

We now show that an *unincentivised* performance scoring mechanism (which does not reward accuracy) with a finite number of rounds does not yield accurate aggregate scores. In an unincentivised performance scoring mechanism, the TA computes the aggregate score of player P_i by simply taking into account the rating that each player $P_{j \neq i}$ gives for player P_i, for $i = 1, \ldots, n$. The TA computes the aggregate scores through these ratings (usually by averaging them) and outputs them to the data owner upon request. On the basis of those aggregate scores, the data owner decides whether to rearrange the distribution of the shares and whether to exclude players from the distributed storage system. However, the unincentivised aggregation and average of these ratings do not lead to accurate aggregate scores. In the following, we show this both when single players issue selfish ratings (Section 5.2.1) and when they form coalitions (Section 5.2.2).

5.2.1 Score Inaccuracy in Non-Cooperative Games

If the aggregate scores are computed through repeated non-cooperative games, the players $P_1, \ldots, P_n$ do not form coalitions. Thus, here, *selfish* ratings means that players are giving low ratings even to high-performing players. The data owner checks the aggregate scores $\tau_1^r, \ldots, \tau_n^r$, at round, say, r. It is *known* to all players that r is the last round the performance scoring mechanism is run before the data owner eventually reallocates the shares. We denote by $\tau_1^{r'}, \ldots, \tau_n^{r'}$ the aggregate scores of players $P_1, \ldots, P_n$ at round r', which occurs strictly earlier than round r, and assume without loss of generality that $\tau_1^{r'} \geq \cdots \geq \tau_n^{r'}$. We assume that the aggregate score $\tau_i^{r'}$ of player P_i is the t^{th} highest score. Player P_i can choose one among the following (mixed) strategies:

- $\sigma 1$) At all rounds, give low ratings to all other players $P_{j \neq i}$.
- $\sigma 2$) At all rounds, give low ratings to the players $P_1, \ldots, P_{t-1}$ with the 1^{st}, 2^{nd}, $\ldots$, $(t-1)^{\text{th}}$ highest aggregate scores $\tau_1^{r'} \geq \cdots \geq \tau_{t-1}^{r'}$.
- $\sigma 3$) From round r' on, give low ratings to all other players $P_{j \neq i}$.
- $\sigma 4$) From round r' on, give low ratings to players $P_1, \ldots, P_{t-1}$ with the 1^{st}, 2^{nd}, $\ldots$, $(t-1)^{\text{th}}$ highest aggregate scores $\tau_1^{r'} \geq \cdots \geq \tau_{t-1}^{r'}$.
- $\sigma 5$) Always give ratings reflecting the actual measured performance of all other players $P_{j \neq i}$.

Theorem 5.1 shows that strategy $\sigma 1$ weakly dominates all other strategies for each player P_i when the goal is to maximize its aggregate score τ_i^r at round r.

Theorem 5.1. *Let $u_1, \ldots, u_n$ be the utilities of, respectively, players $P_1, \ldots, P_n$ satisfying u1)–u3) when the aggregate scores $\tau_1^r, \ldots, \tau_n^r$ at last round r are computed. Then, $\sigma 1$ weakly dominates all other available strategies, i.e. $u_i(\sigma_i', \boldsymbol{\sigma}_{-i}) \leq u_i(\sigma 1, \boldsymbol{\sigma}_{-i})$, for $\sigma_i' \neq \sigma 1$ and $i = 1, \ldots, n$.*

We first outline the structure of the proof. In short, we first show that strategy $\sigma 1$ is weakly dominant for player P_i with the t-th highest aggregate score $\tau_i^{r'}$. This is due to the fact that each other player $P_{j \neq i}$ is assumed to be rational as well and strategy $\sigma 1$ has to be played by player P_i to contrast the undermining effect of all other players. Second, it is proven that this holds even for the case where player P_i has the lowest aggregate score or the highest aggregate score.

We now formally prove Theorem 5.1 by showing that strategy $\sigma 1$ always dominates a less selfish strategy in the following three steps.

If P_i is given the t^{th} highest aggregate score $\tau_i^{r'}$ by the TA at round $r' < r$, then we have $\tau_i^{r'} \leq \tau_j^{r'}$, for $j = 1, \ldots, t-1$. Depending on the aggregate scores $\tau_1^{r'}, \ldots, \tau_{t-1}^{r'}$ and on t, $u1), u2), u3)$ can all occur. In particular, if either $u2)$ or $u3)$ occurs, then utility $u_i(r') \neq 1$. A way for player P_i to prevent this is to attempt to

lower the aggregate scores of players $P_1, \ldots, P_{t-1}$ from round $r' + 1$ on to increase the chances that $u1$) occurs at the final round r. That is, strategy $\sigma 4$ is for player P_i weakly dominant with respect to $\sigma 5$, assuming player $P_{j \neq i}$ plays strategy $\sigma 5$. Furthermore, the more rounds player P_i gives low ratings for players $P_1, \ldots, P_{t-1}$, the more their aggregate scores decrease over time and, thus, the higher the chance that $u1$) occurs at round r. A weakly dominant strategy for player P_i is to start giving low ratings as early as possible and thus $\sigma 2$ weakly dominates $\sigma 4$, assuming player $P_{j \neq i}$ plays $\sigma 5$. Getting the t^{th} highest aggregate score means that player P_i is the t^{th} best-performing player and that player $P_{j \neq i}$ played $\sigma 5$. In case player $P_{j \neq i}$ plays a strategy other than $\sigma 5$, then player P_i might be given by the TA an aggregate score lower than the t^{th} highest aggregate score $\tau_i^{r'}$, which increases the chances that $u3$) occurs. Strategy $\sigma 5$ is weakly dominated by $\sigma 2$ and $\sigma 4$ for players $P_{t+1}, \ldots, P_n$, which are, respectively, given the lowest aggregate scores $\tau_{t+1}^{r'}, \ldots, \tau_n^{r'}$. Hence players $P_{t+1}, \ldots, P_n$ give low ratings for players $P_1, \ldots, P_t$, including player P_i itself. In order to compensate low ratings from worse-performing players, player P_i has to give low ratings for players $P_{t+1}, \ldots, P_n$ as well. However, because player P_i already plays $\sigma 2$, it eventually assigns low ratings to each player $P_{j \neq i}$. In case players $P_{t+1}, \ldots, P_n$ play $\sigma 4$, then player P_i can respond by playing $\sigma 3$. However, to maximize the desired effect, it is more effective for player P_i to give low ratings as soon as possible. Thus, it plays $\sigma 1$. In conclusion, player P_i plays $\sigma 1$ because this increases the chances that, at round r, $u1$) occurs and $u_i(r) = 1$. Therefore, $\sigma 1$ weakly dominates $\sigma 2 - \sigma 5$.

If P_i is given the lowest aggregate score $\tau_i^{r'}$ by the TA at round $r' < r$, then $\tau_i^{r'} < \tau_j^{r'}$, for each player $P_{j \neq i}$. Thus, $u3$) occurs for player P_i. In order to avoid this, a possible option for player P_i is to try to lower the aggregate scores of all the players that are given higher aggregate scores, which means to give low ratings for player $P_{j \neq i}$ from round $r' + 1$ on. Thus, $\sigma 3$, which in this case is identical to $\sigma 4$, weakly dominates $\sigma 5$. However, in order to decrease the probability of $u3$) to occur, it is better for player P_i to give low ratings to all other players as early as possible. Thus $\sigma 1$ weakly dominates $\sigma 3$. Since $\sigma 1$ is here identical to $\sigma 2$, $\sigma 1$ weakly dominates $\sigma 2$ (by Definition 2.12). Due to the transitivity of weak strategy dominance, $\sigma 1$ also weakly dominates $\sigma 5$. In conclusion, player P_i plays $\sigma 1$ regardless of what each other player $P_{j \neq i}$ plays because $\sigma 1$ weakly dominates $\sigma 2 - \sigma 5$. This case where player P_i is the worst-performing player can be obtained by induction from *1)* where P_i is given the t^{th} highest aggregate score, with t increasing towards n.

If P_i is given the highest aggregate score $\tau_i^{r'}$ by the TA at round $r' < r$, then $\tau_i^{r'} > \tau_j^{r'}$, where player $P_{j \neq i}$ played $\sigma 5$. So for player P_i, sticking to $\sigma 5$ is sufficient for $u1$) to occur at round r. However, for the same reasons discussed in *1)*, player $P_{j \neq i}$ is rational and gives low ratings to the players given higher aggregate scores, which always includes player P_i. Thus, $u1$) is unlikely to occur. To balance out this effect, player P_i's can give low ratings to the lower performing players, which in this case

means to give low ratings to all other players. Thus, player P_i plays $\sigma 3$. However, for reasons discussed in *1)*, $\sigma 1$ weakly dominates $\sigma 3$ in case each other player $P_{j\neq i}$ plays $\sigma 2$ rather than $\sigma 4$. In conclusion, if player P_i is given the highest aggregate score, then $\sigma 1$ weakly dominates $\sigma 2 - \sigma 5$.
Because the above analysis can be applied to each player P_i, for $i = 1, \ldots, n$, we conclude that strategy $\sigma 1$ weakly dominates strategy $\sigma 2 - \sigma 5$ for all players $P_1, \ldots, P_n$. Thus, $u_i(\sigma_i', \boldsymbol{\sigma}_{-i}) \leq u_i(\sigma 1, \boldsymbol{\sigma}_{-i})$, for $\sigma_i' \neq \sigma 1$ and $i = 1, \ldots, n$. □

Since each player P_i plays strategy $\sigma 1$, and thus gives low ratings to each other player $P_{j\neq i}$ at all rounds, the following corollary ensues.

Theorem 5.2. *When computed through unincentivised ratings, the aggregate scores $\tau_1^r, \ldots, \tau_n^r$ the data owner checks at last round r are inaccurate, i.e. they do not describe the actual performance of, respectively, players $P_1, \ldots, P_n$.*

In particular, in this case where no coalitions are formed, all aggregate scores $\tau_1^r, \ldots, \tau_n^r$ are low. In terms of SSPs and shares, the data owner might believe that all SSPs are underperforming and eventually withdraw all shares from all SSPs. This implies that $U_i(r) = -1$, for $i = 1, \ldots, n$, which is the worst possible situation for an SSP.

5.2.2 Score Inaccuracy in Cooperative Games

In a performance scoring mechanism modelled as a cooperative game, players can form coalitions to cooperatively obtain high aggregate scores and coordinate on players to undermine. Here, *selfish* rating means that players give high ratings to coalition partners and low ratings to all other players, regardless of the actual witnessed performance. The strategies a player P_i can adopt combine the strategies listed in Section 5.2.1 with the possibility of increasing the aggregate scores of fellow coalition members by giving them high ratings. As for Theorem 5.1, one can show that the weakly dominant strategy (adapted from $\sigma 1$) is the one where each player simultaneously gives high ratings to fellow coalition members and low ratings to all other players at all rounds. The proof, omitted here due to space constraints, is similar to the one of Theorem 5.1. Thus, Corollary 5.2 holds also for the case where coalitions among players are formed.

Theorem 5.1 and Corollary 5.2 state the impossibility of getting accurate aggregate scores in unincentivised and finitely repeated games. They mirror the impossibility result for rational secret sharing presented in [59]. The performance scoring mechanism described in this section is a finitely repeated game because all players know that round r is the last one. However, the same impossibility result holds for infinitely repeated games, where it is unknown to the players that round r is the last one. The proof of Theorem 5.1 is not tied to round r because it is weakly dominant for the players to play strategy $\sigma 1$ as soon as possible. This is true in particular when players do not know which round is the final one, because they possibly have even

less chances to cope with undermining ratings of the other players. This problem is solved in the next section, where it is shown that in order to obtain accurate aggregate scores, the performance scoring mechanism has to be modelled *both* as an *incentivised* and *infinitely repeated* game.

5.3 Score Accuracy for Incentivized Ratings: A Performance Scoring Mechanism Resilient to Coalitions

In this section, we present in details the main contribution of this chapter. That is a performance scoring mechanism that outputs aggregate scores within an accuracy interval with high probability, given that the players involved are rational agents interested in maximizing their payoff by maximizig their aggregate scores and assuming that half of them behave honestly. We have already anticipated that the aggregate scores are computed as the convex combination of three terms. It is the second of these terms that exploits the rationality of the players (formalized through the definitions of their utilities and preferences in Section 5.1) and encourages the players to submit accurate ratings by incentivizing them and penalizing the inaccurate ones. The performance scoring is modelled as a game that is incentivized because of the presence of this second term. Furthermore, the performance scoring mechanism can be seen as an infinitely repeated game for the storage servers. This is due to the fact that the storage servers do not know after how many rounds the performance scoring mechanism is run before issuing the updated aggregate scores. Otherwise, they could rate inaccurately without consequence, as shown in Section 5.2.

The following assumptions hold:

Assumption 1. A coalition can be formed by the storage servers owned by one or multiple SSPs and when these SSPs collude then all the storage servers they own are part of that coalition.

Assumption 2. A storage server can belong to at most one coalition.

Assumption 3. The behavior of storage servers is assumed to be consistent among all storage servers. That is, a storage server does not choose to behave differently according to the provenance of the storage servers it is interacting with. On the contrary, a storage server may submit dishonest evidence for other storage servers according to their provenance.

Assumption 4. All evidence is submitted to a central TA responsible for computing the aggregate performance scores from the ratings. The TA is always honest and does not tamper with the evidence.

Assumptionw 1-3 emphasize the fact that what we focus on here is coping with the possibility of unreliable evidence (i.e. unreliable ratings) submitted by storage servers with the aim of maximizing the overall trustworthiness of the SSP they belong to.

5.3.1 A New Performance Scoring Mechanism

We design a performance scoring mechanism modelled as an *incentivised*, *infinitely repeated*, and *cooperative*[4] game with a mediator (the TA). Each round of the repeated game consists of running a performance scoring mechanism distributedly. The TA runs the performance scoring mechanism at random intervals, *independently* of the intervals in which share renewal for proactive secret sharing is performed. We emphasise that the performance scoring mechanism is entirely separate from the proactive secret sharing operations (such as share renewal), which are performed as usual. The players do not know how many rounds the performance scoring mechanism is run before the data owner checks the aggregate scores. Thus, even if the performance scoring mechanism is run a finite amount of times, it can be modelled as an infinitely repeated game. This enables the TA to not only collect and processes ratings, but also encourages the submission of accurate ratings and penalizes the players submitting selfish ratings. This allows the definition of a strategy leading to a k-Nash equilibrium and pushes the players to follow it, because incentives and penalties, respectively, positively and negative influence the final aggregate score of the players.

We now provide a high-level description of our performance scoring mechanism. Each aggregate score τ_i^r at round r is computed by the performance scoring mechanism as a convex combination of a first component τ_i', a second component τ_i'', and a third component τ_i^{r-1}, for $i = 1, \ldots, n$. The first component τ_i' is computed through steps 1)–3) of the mechanism and is the aggregate score of all ratings submitted by all players for the player being rated for the current round r. The second component τ_i'' is computed through steps 4)–6) of the mechanism and is the aggregate score of all incentives and penalties given to player P_i by the TA for, respectively, the accurate and selfish ratings submitted for the current round r. For the first and the second component, the round r is omitted to simplify notation. The third component τ_i^{r-1} is the aggregate score of player P_i at the previous round $r-1$. Thus, it was previously computed and is an input of the performance scoring mechanism run at the current round r. The third component keeps track of the evolution of the performance of a player over time and increasingly rewards good performance and penalizes poor performance. The final aggregate score τ_i^r at round r is computed during step 7) of the mechanism.

We now show how to compute the aggregate scores $\tau_1^r, \ldots, \tau_n^r$ at round r for players

[4]It is sufficient to model the mechanism as a cooperative game, because non-cooperative games are a particular case where coalitions have only one player each.

$P_1, \ldots, P_n$, where round r is not necessarily the round where the data owner checks the aggregate scores. We defined each aggregate score τ_i^r as a real number between 0 and 1, i.e. $0 \leq \tau_i^r \leq 1$ for $i = 1, \ldots, n$. The same holds for the first and the second component denoted by, respectively, τ_i' and τ_i'', and for the aggregate score τ_i^{r-1} of player P_i at round $r-1$. The rating submitted by player P_i to evaluate player P_j for the computation of its aggregate score τ_j^r is denoted by $\rho_{i,j}^r$, where $0 \leq \rho_{i,j}^r \leq 1$. We denote by $\overline{\tau}_1', \ldots, \overline{\tau}_n'$ the *targeted first components*, i.e. they are the first components $\tau_1', \ldots, \tau_n'$ when computed through ratings that perfectly match the actual performance of players $P_1, \ldots, P_n$, respectively. We now provide a definition of accurate ratings.

Definition 5.3. *Let $\overline{\tau}_1', \ldots, \overline{\tau}_n'$ be the targeted first components of the aggregate scores of players $P_1, \ldots, P_n$ and let $t_\varepsilon > 0$. A rating $\rho_{i,j}^r$ submitted by player P_i to evaluate player P_j is called t_ε-accurate if $\overline{\tau}_i' - t_\varepsilon \leq \rho_{i,j}^r \leq \overline{\tau}_i' + t_\varepsilon$. We refer to t_ε as the accuracy threshold. In cases where t_ε is clear from the context, we simply write accurate.*

The accuracy of the ratings leads to accurate first components $\tau_1', \ldots, \tau_n'$, which in turn (as we discuss in detail later) leads to accurate aggregate scores $\tau_1, \ldots, \tau_n$. Clearly, the accuracy of the first component τ_j' depends on how many accurate ratings $\rho_{i,j}^r$ were submitted from all the other players $P_{i \neq i}$. In other words, the accuracy of the first component depends on the ratio between the accurate ratings submitted by honest players and inaccurate ratings submitted by colluding rational players. Supposing that all colluding players form a unique coalition in order to maximize their effect, then what is the biggest coalition tolerated? This is what we investigate in the following, starting from the definition of *weight*.

Definition 5.4. *Let us denote by P_j a player whose first component τ_j' has to be computed. The weight $w_{i,j}^r$ with respect to this computation that a player $P_{i \neq j}$ has by submitting rating $\rho_{i,j}^r$ is defined as:*

$$w_{i,j}^r := \frac{\tau_i^{r-1}}{\sum_{l \neq j} \tau_l^{r-1}}, \tag{5.1}$$

where $\sum_{i \neq j} w_{i,j}^r = 1$. Note that weight $w_{i,i}^r$ is implicitly undefined because player P_i cannot rate itself.Let us denote by C_k a coalition of cardinality k of players $P_1, \ldots, P_k$ (relabelling the players if necessary, without loss of generality). Denoted by w_{j,C_k}^r, the weight of a coalition C_k with respect to the computation of the first component τ_j' of player P_j is defined as:

$$w_{j,C_k}^r := \sum_{i \in C_k} w_{i,j}, \tag{5.2}$$

if player $P_j \notin C_k$. Instead, if player $P_j \in C_k$, then w_{j,C_k}^r is defined as:

$$w^r_{j,C_k} := \sum_{i \in C_k, i \neq j} w_{i,j}. \tag{5.3}$$

The weight w^r_{j,C_k} of a coalition C_k determines how much this coalition affects the final trust value τ^r_j of player P_j. What we want to determine here is the biggest weight that a coalition can have with respect to all the other players so that the final trust value τ^r_j of player P_j is still accurate. To formalize this we introduce Definition 5.5 and Definition 5.6.

Definition 5.5. *Let w^r_{j,C_k} be the weight of a coalition C_k with respect to the computation of the first component τ'_j of a player P_j at round r. The maximal weight of coalition C_k is denoted by $w^r_{C_k}$ and defined as:*

$$w^r_{C_k} := \max_{1 \leq j \leq n} w^r_{j,C_k}. \tag{5.4}$$

Definition 5.6. *A coalition C_k is called admissible with respect to the computation of the aggregate score τ^r_j of player P_j if the inaccurate ratings $\rho^r_{1,j}, \ldots, \rho^r_{k,j}$ submitted by players $P_1, \ldots, P_k \in C_k$ do not affect the accuracy of the aggregate score τ^r_j, i.e. $\overline{\tau}'_i - t_\varepsilon \leq \tau^r_j \leq \overline{\tau}'_j + t_\varepsilon$.*

We now present the performance scoring mechanism with the TA as a mediator for the computation of the aggregate scores $\tau^r_1, \ldots, \tau^r_n$ of players $P_1, \ldots, P_n$ at round r.

1) The TA selects integers $a, b, c > 0$ such that $a + b + c = 1$. These are the weights associated to $\tau'_i, \tau''_i, \tau^{r-1}_i$, respectively, for the computation of the aggregate score τ_i at round r. These weights are chosen by the TA depending on whether it is considered more important to be high-performing at the current round or to have a consistent performance over time. They will be used in the computation of the aggregate score in step 7).
2) The TA receives ratings $\rho^r_{j,i}$ from each player $P_{j \neq i}$ to evaluate player P_i. In case the TA does not receive all the expected ratings, it broadcasts a complaint message and aborts the protocol. This is to make sure that all players participate at each round and that no player can benefit from not submitting a rating.
3) The TA computes the first component τ'_i for player P_i as follows:

$$\tau'_i = \sum_{j \neq i} w^r_{j,i} \cdot \rho^r_{j,i}. \tag{5.5}$$

   It computes all first components $\tau'_1, \ldots, \tau'_n$ before continuing.
4) The TA selects the incentive α, with $\frac{1}{\eta} t_\varepsilon < \alpha \leq t_\varepsilon$ and the penalty β, with $-t_\varepsilon \leq \beta < -\frac{1}{\eta} t_\varepsilon$, where $\eta > 0$ is a scalar. Incentive α and penalty β are defined in this way to, respectively, raise and lower the aggregate score of a player as much as possible while remaining within the accuracy interval (Definition 5.3).

This way, α and β shift the aggregate score, respectively, above and below the score corresponding to the performance while still providing a description of the performance itself.

5) The TA computes the second partial score $\tau_i^{''}$ for player P_i as $\tau_i^{''} = \frac{1}{n-1} \sum_{j=1}^{n-1} o_{i,j}$, where $o_{i,j}$ is defined as follows:

$$o_{i,j} = \begin{cases} \alpha, & \text{if } |\tau_j' - \rho_{i,j}^r| \leq t_\varepsilon \\ 0, & \text{if } t_\varepsilon < |\tau_j' - \rho_{i,j}^r| \leq 2t_\varepsilon \\ \beta & \text{if } |\tau_j' - \rho_{i,j}^r| > 2t_\varepsilon \end{cases} \tag{5.6}$$

According to Definition 5.3, a t_ε-accurate rating $\rho_{i,j}^r$ varies around the targeted first partial score $\overline{\tau}_j'$, which is only a reference point. A t_ε-accurate first component τ_j' may shift the centre of the accuracy interval above or below $\overline{\tau}_j'$. A rating accurate with respect to the targeted first component $\overline{\tau}_j'$ has to be compared to the computed first component τ_j' now. Thus, rating $\rho_{i,j}^r$ might be at distance greater than t_ε from τ_j', while still being accurate. Hence we considered above three cases instead of two, where in the second case the rating is considered neutral. The performance scoring mechanism is run multiple times before the data owner checks the aggregate scores. When this is iterated several times, as in a repeated game, the penalties and incentives based on a misleading accuracy interval are balanced out and mitigate the eventual misrepresentation of τ''.

6) The TA computes the aggregate score τ_i^r of player P_i at round r as $\tau_i^r = a \cdot \tau_i' + b \cdot \tau_i^{''} + c \cdot \tau_i^{r-1}$.

We have seen in Step 6) of the performance scoring mechanism presented above that the aggregate score τ_i^r of player P_i computed at round r depends on the aggregate score τ_i^{r-1} at round $r-1$. That means, recursively, that the aggregate score of a player at the current round r depends on all the aggregate scores that that player was assigned to in all the previous rounds. We chose to defined the aggregate score τ_i^r at round r like that because in our use case we have to store data safely for lengthy periods of time and so we are interested in a steady and consistent behavior of the players involved. This way of defining the aggregate scores gives us a hint of the value of the targeted first component $\overline{\tau}_i'$. The targeted first component $\overline{\tau}_i'$ represents the value that matches exactly the performance of player P_i at round r, which is of course unknown. However, assuming that at the previous rounds $r-1, r-2, \ldots, 1$ the aggregate scores $\tau_i^{r-1}, \tau^{r-2}, \ldots, 1$ were accurate, i.e. the there were no coalitions of rational players that were not admissible, it is possible to compute an estimation of the targeted first component $\overline{\tau}_i'$ at round r. We refer to this *estimated* value as $\tilde{\tau}_i'$ and we define it as follows:

$$\tilde{\tau}_i' := \sum_{\ell=1}^{r-1} c^{r-\ell} \tau_i^{r-\ell}, \tag{5.7}$$

where $c < 1$ is the weight used in Step 6) of the performance scoring mechanism to compute the aggregate score τ_i^r at round r. The definition of the estimated targeted first component $\tilde{\tau}_i'$ is aligned with how the aggregate score τ_i^r is computed, because it takes into account all the aggregate scores at the previous rounds where the latest rounds are the most prominent ones. Given the estimated targeted first component, we defined the *weight bound* ε^r as follows.

Definition 5.7. *Given the estimated targeted first components $\tilde{\tau}_1', \ldots, \tilde{\tau}_n'$ at round r of player $P_1, \ldots, P_n$ and the accuracy threshold t_ε, the weight bound with respect to the computation of the aggregate scores $\tau_1^r, \ldots, \tau_n^r$ to be computed at round r is denoted by ε^r and defined as:*

$$\varepsilon^r := \begin{cases} \min_{1 \leq j \leq n} \frac{t_\varepsilon}{\tilde{\tau}_j'}, & \text{if } \tilde{\tau}_j' \geq 0.5 \\ \min_{1 \leq j \leq n} \frac{t_\varepsilon}{1-\tilde{\tau}_j'}, & \text{if } \tilde{\tau}_j' < 0.5 \end{cases} \tag{5.8}$$

We present a strategy σ_i for player $P_i \in C_k$ to use when the performance scoring mechanism is run to compute the aggregate scores $\tau_1^r, \ldots, \tau_n^r$ of players $P_1, \ldots, P_n$ at round r. This strategy takes into account the weight bound ε^r and provides criteria for player P_i to decide whether to submit an accurate rating $\rho_{i,j}^r$ for player $P_{j \neq i}$, and it holds for any formed coalition C_k.

a) Player $P_i \in C_k$ submits to the TA an accurate rating $\rho_{i,j}^r$ for player P_j if $w_{j,C_k} \leq \varepsilon^r$.
b) Otherwise, it submits a selfish rating $\rho_{i,j}^r$ to the TA.

5.3.2 Resilience Against Coalitions

Let us denote by $\Gamma(P_i, \sigma_i, u_i')$, for $i = 1, \ldots, n$, the repeated game, where at every round the performance scoring mechanism described in Section 5.3.1 is run and where player P_i has utility $u'(r)$ (see Section 5.1) and plays strategy σ_i defined in Section 5.3.1, for $i = 1, \ldots, n$. We now prove that game $\Gamma(P_i, \sigma_i, u_i')$ can cope with any coalition C_k whose weight is bounded by the weight bound ε^r. That is, it computes accurate aggregate scores even if a coalition C_k of players $P_1, \ldots, P_k$ with $w_{j,C_k} \leq \varepsilon^r$ submits selfish ratings $\rho_{1,j}, \ldots, \rho_{k,j}$ to rate player P_j. We assume that there is only one coalition C_k and that the players that are outside this coalition are therefore honest and submit accurate ratings. This assumption does not weaken our result. In fact, it strengthens it because one can assume that multiple coalitions are formed that in order to maximise their effect collude altogether to form C_k. We also assume that the majority of the players is honest.

Theorem 5.8. *Let the infinitely repeated cooperative game $\Gamma(P_i, \sigma_i, u_i')$, for $i = 1, \ldots, n$ be run at round r to compute aggregate scores $\tau_1^r, \ldots, \tau_n^r$, where u1')–u3') are satisfied. Furthermore, let $t_\varepsilon > 0$ be the accuracy threshold and ε^r the weight*

bound. Let C_k with cardinality k be the largest coalition that is formed among the players $P_1, \ldots, P_n$. If $w_{C_k} \leq \varepsilon^r$, then the game reaches a k-resilient equilibrium with respect to the computation of the aggregate scores $\tau_1^r, \ldots, \tau_n^r$. That is, coalition C_k is admissible for each aggregate score τ_j^r to be computed, for $j = 1, \ldots, n$.

Proof. What we need to show is that above condition on the maximum weight w_{C_k} of coalition C_k implies the accuracy of the first component τ_j' of player P_j, for $j = 1' \ldots, n$. The accuracy of the aggregate score τ_j^r follows from the accuracy of the first component. The second component $\tau"_j$ boosts or lowers by definition the aggregate score to be computed so that to encourage that the first component is computed accurately. Also, we assume that the aggregate score τ_j^{r-1} is accurate (see Definition 5.7).

First of all, the assumption that $w_{C_k} \leq \varepsilon^r$ holds for every player P_j that the players in coalition C_k evaluate. More precisely, by definition of maximum weight w_{C_k} we have that:

$$\varepsilon^r \geq w_{C_k} \geq w_{j,C_k}, \qquad \forall P_j. \tag{5.9}$$

This means that option a) of strategy σ_i is always verified and, thus, that player $P_{i \neq j}$ chooses to submit accurate ratings $\rho_{i,j}^r$ to the TA. Now we need to prove that the weight bound ε^r allows to compute accurate aggregate scores and we want to prove that regardless of the ratings submitted to be as general as possible. That is, we do not prove the accuracy of the aggregate scores as dependent of the ratings submitted and so we investigate the wost case scenario.

Let us assume that the first component τ_j' of player $P_j \notin C_k$ has to be computed and that the estimated targeted first component $\tilde{\tau}_j'$ is 1, i.e. it is assumed that player P_j is high-performing, and that coalition C_k of players $P_1, \ldots, P_k$ with weight, respectively, $w_{1,j}^r, \ldots, w_{k,j}^r$ works against player P_j. In the worst case scenario, players $P_1, \ldots, P_k$ submit the lowest possible ratings for player P_j, that is $\rho_{1,j}^r = \rho_{2,j}^r = \cdots = \rho_{k,j}^r = 0$. Instead, each player $P_\ell \notin C_k$ behaves honestly and submits rating $\rho_{\ell,j} = \tilde{\tau'}_j = 1$. According to Definition 5.7, this means that $\varepsilon^r = \frac{t_\varepsilon}{\tilde{\tau}_j'}$ and that the following holds.

$$\begin{aligned}
|\tau_j' - \tilde{\tau}_j'| &= |\sum_{i \in C_k} w_{i,j}^r \rho_{i,j}^r + \sum_{\ell \notin C_k} w_{\ell,j}^r \rho_{\ell,j}^r - \tilde{\tau}_j'| \\
&= |w_{j,C_k} \sum_{i \in C_k} \rho_{i,j}^r + (1 - w_{j,C_k}) \sum_{\ell \notin C_k} \rho_{\ell,j}^r - \tilde{\tau}_j'| \\
&= |(1 - w_{j,C_k})\tilde{\tau}_j' - \tilde{\tau}_j'| \\
&= w_{j,C_k} \tilde{\tau}_j' \\
&\leq w_{C_k} \tilde{\tau}_j' \\
&\leq \varepsilon^r \tilde{\tau}_j' \\
&= t_\varepsilon.
\end{aligned} \tag{5.10}$$

This means that the weight bound ε^r on w_{C_k} implies that $|\tau'_j - \tilde{\tau}'_j| \leq t_\varepsilon$. Thus, with high probability, that $\overline{\tau}'_j - t_\varepsilon \leq \tau^r_j \leq \overline{\tau}'_j + t_\varepsilon$, i.e. that the aggregate score τ^r_j computed is accurate with respect to the targeted first component $\tilde{\tau}'_j$, i.e. with the actual performance of the storage server. Similarly, one can prove that Equation (5.10) holds for $\varepsilon^r = \frac{t_\varepsilon}{\tilde{\tau}'_j}$ for the opposite worst case scenario, where $\tilde{\tau'}_j$ is 0 and the players within coalition C_k work in favor of player P_j and submit rating $\rho^r_{i,j} = 1$.

We considered the two worst case scenarios where the rating $\rho^r_{i,j}$ submitted by a player $P_i \in C_k$ can be the most distant from the estimated targeted first component $\tilde{\tau}'_j$ (or the targeted first component $\overline{\tau}'_j$). Rating $\rho^r_{i,j}$ maximizes the weight of the coalition with respect the computation of the first component τ'_j and thus the accuracy of the aggregate score of all the other intermediate cases where $0 < \tilde{\tau}'_j < 1$ holds automatically.

This concludes our proof because by assumption it holds that coalition C_k is admissible with respect to the computation of all aggregate scores $\tau^r_1, \ldots, \tau^r_n$ and, thus, player $P_i \in C_k$ never deviates from strategy σ_i, otherwise it will be detected and penalized according to the performance scoring mechanism described as the infinitely repeated cooperative game $\Gamma(P_i, \sigma_i, u'_i)$, for $i = 1, \ldots, n$. □

5.4 Machine Learning-Based Instantiation of the Performance Scoring Mechanism

In this section, we present an instantiation of the performance scoring mechanism described in Section 5.3.1 followed by extensive evaluations that experimentally validate that the performance scoring mechanism tolerates coalitions of players submitting inaccurate ratings. We recall that the aggregate score τ^r_i of player P_i at round r is computed as the convex combination of a first component τ'_i that averages the rating $\rho^r_{i,j}$ submitted by player $P_{j \neq i}$, a second component τ''_i evaluating how accurate are the ratings $\rho^r_{j,i}$ submitted by player P_i to evaluate player $P_{j \neq i}$, and the aggregate score τ^{r-1}_i of player P_i at round $r-1$. While the computation of the second component τ''_i is standard, devising what is the first component τ'_i from the ratings submitted by player $P_{j \neq i}$ is less obvious. To do that we use machine learning techniques that allow distinguish so called classes of accuracy among the ratings submitted so that these can be weighted accordingly for the computation of the first component. This section is organized as follows. In Section 5.4.1 we survey the machine learning techniques that we use to compute the first component τ'_i of the aggregate score τ^r_i for player P_i. In Section 5.4.2, we present how the aforementioned machine learning techniques are used to compute the first component τ'_i. In Section 5.4.3, we show how the second component τ''_i is computed, focusing on how incentives and penalties for, respectively, accurate and inaccurate ratings are selected. Extensive evaluations of the performance scoring mechanism using machine

learning techniques to compute the first component τ_i' can be found in Section 5.4.4. We recall that both the instantiation of the performance scoring mechanism and the evaluation relies on Assumption 1, Assumption 2, Assumption 3, and Assumption 4 described in Section 5.3.

5.4.1 Used Machine Learning Techniques Description

In this section, we review the machine learning technique of K-means clustering. Note that for the definition of this technique, as well as for the definition of the technique of mixture of Gaussians in the next section, we refer to [16].

5.4.1.1 K-Means Clustering

Let us define a data set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of n points in a D-dimensional Euclidean space, with $S_i = (x_{1_i}, \ldots, x_{D_i})$, for $i = 1, \ldots, n$. The *K-means clustering* is the problem of grouping these points into K clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$. These clusters are identified such that the distances of points within the same cluster are smaller than the distances to points outside the cluster. This means that together with the clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$, so called *center points* $M_1, \ldots, M_K$ are identified, where $M_j = (x_{1_j}, \ldots, x_{D_j})$, for $j = 1, \ldots, K$. Each center point M_j satisfies the property that the sum of the squares of the distances of each data point P_i to the closest point M_j is a minimum. This concept can be formalized by the so called *distortion measure* J, an objective function defined as follows:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{K} r_{i,j} ||P_i - M_j||^2,$$

where $||P_i - M_j||$ is the distance between points P_i, M_j and where $r_{i,j} = 1$ if point P_i is assigned to cluster $\mathcal{C}_j$ and $r_{i,l} = 0$ for $l \neq j$. Thus, the K-means clustering problem consists of finding values $r_{i,j}$ and centers M_j, for $i = 1, \ldots, n$ and $j = 1, \ldots, K$, such that the distortion measure J is minimized. This is achieved by means of so called EM algorithms, consisting of an expectation step E, where the values $r_{i,j}$ are adjusted, and a maximization step M, where, instead, the points M_j are adjusted. In fact, the distortion measure J can be minimized through multiple iterations, where after each iteration an expectation step and a maximization step are performed. A concrete instantiation of the above strategy can be found in [16] and [79]. Details about how to make this K-means clustering more efficient can be found in [94] and [82].

5.4.1.2 Mixture of Gaussians

We review the machine learning technique of *mixture of Gaussians*. This technique is meant to model real data set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of points, which otherwise could

not be fully described by a single Gaussian distribution. Consider K Gaussian distributions $\mathcal{N}_1(\mu_1, \sigma_1^2), \ldots, \mathcal{N}_K(\mu_K, \sigma_K^2)$, where μ_i is the mean and σ_i^2 is the variance, for $i = 1, \ldots, K$. Denoted by $p(\mathcal{PS})$, a mixture of Gaussians with respect to data set $\mathcal{S}$ is defined as a linear combinations of the given Gaussian distributions:

$$p(\mathcal{S}) = \sum_{j=1}^{K} \pi_j \mathcal{N}(\mu_j, \sigma_j^2),$$

where each Gaussian $\mathcal{N}(\mu_j, \sigma_j^2)$ is said a *component* of the mixture and each parameter π_j is the respective *mixing coefficient*. In addition, if $\sum_{j=1}^{K} \pi_j = 1$, then also $p(\mathcal{S})$ is a probability distribution. It is possible to find a Gaussian distribution $\mathcal{N}_{\mathcal{S}}(\mu, \sigma^2)$ computed as the approximation of the mixture of Gaussians $p(\mathcal{S})$. That is, distribution $\mathcal{N}_{\mathcal{S}}(\mu, \sigma^2)$ is the closest distribution to $p(\mathcal{S})$ that is not a mixture of Gaussians, according to the Kullback-Leibler distance (see [74]). The mean μ and the variance σ^2 of $\mathcal{N}_{\mathcal{S}}(\mu, \sigma^2)$ can be easily computed as:

$$\mu = \sum_{j=1}^{K} \pi_j \mu_j \quad \text{and} \quad \sigma^2 = \sum_{j=1}^{K} \pi_j (\sigma_j^2 + \mu_j^2) - \mu^2,$$

where $\pi_1, \ldots, \pi_K$ correspond to the mixing coefficients of the mixture of Gaussians $p(\mathcal{S})$, as described in [108].

For the sake of readability, we summarize in Table 5.1 the notations that follow throughout the rest of the section.

5.4.2 Computation of the First Component τ_i'

This section describes how the first component τ_i' of the aggregate score τ_i^r of player P_i at round r is computed, which takes into account the ratings submitted by all players $P_{j \neq i}$. The computation of the first component τ_i' is performed in an Euclidean space of dimension $D = 2$. For readability, we divide this computation into steps.

5.4.2.1 Collection of the evidence

Each player P_j submits point $S_{j,i} = (x_{S_{j,i}}, y_{S_{j,i}})$ to rate player P_i to the TA. The first coordinate of point $S_{j,i}$ is $x_{S_{j,i}} = \tau_j^{r-1}$, where $0 \leq \tau_j^{r-1} \leq 1$ is the aggregate score of player P_j at round $r-1$. The second coordinate of point $S_{j,i}$ is $y_{S_{j,i}} = \rho_{j,i}$, where $0 \leq \rho_{j,i} \leq 1$ is the rating submitted by player P_j to rate player P_i.

5.4.2.2 Representation of τ_i'

Since the ratings with respect to the performance of player P_i is represented as a value between 0 and 1, the first component τ_i' is also a value between 0 and 1. The idea is to define the data set $\mathcal{S}^{(i)} = \{S_{1,i}, \ldots, S_{i-1,i}, S_{i+1,i}, \ldots, S_{n,i}\}$ of points submitted

Table 5.1: Summary of the notation used to instantiate the accurate performance scoring mechanism through machine learning techniques.

P_i, P_j	players
n	total number of players
τ_i^r	aggregate score of player P_i at round r
τ_i'/τ_i''	first/second component of player P_i at round r
$\tau_i^{(r-1)}$	aggregate score of player P_i at round $r-1$
$S_{j,i}$	data point describing how player P_j rates player P_i
$x_{P_{j,i}}/y_{P_{j,i}}$	x/y-coordinate of data point $S_{j,i}$
$\rho_{j,i}$	evidence submitted by player P_j with respect to player P_i
$\mathcal{C}_1, \ldots, \mathcal{C}_K$	clusters/classes of credibility
$M_1, \ldots, M_K$	center points of clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$
$y_{M_1}, \ldots, y_{M_K}$	y-coordinate of $M_1, \ldots, M_K$
$w_{j,i}$	weight of player P_j when rating player P_i
$\pi_1, \ldots, \pi_K$	mixing coefficients of $M_1, \ldots, M_K$
α, β	incentive, penalty for, respectively, accurate and inaccurate ratings
$o_{j,i}$	score gained or loss by player P_j when rating player P_i
a, b, c	coefficients for τ_i', τ_i'', and τ_i^{r-1} respectively

by player $P_{j\neq i}$. Then, the machine learning techniques of K-means clustering (see Section 5.4.1.1) and mixture of Gaussians (see Section 5.4.1.2) are used to extract the first component τ_i' from coordinate $y_{S_{j,i}}$ of each point in the data set $\mathcal{S}^{(i)}$.

5.4.2.3 Classes of credibility

K classes of evidence are distinguished with respect to their credibility by the K-means clustering algorithm. Informally, we use the term credibility to underline the fact that these classes are distinguished based on the reputation already gained by the raters, whose ratings are believed to be more accurate, thus, credible. The points in the data set $\mathcal{S}^{(i)}$ are grouped into K clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$. In fact, each point in the data set $\mathcal{S}^{(i)}$ is a tuple corresponding to the values "aggregate score of the rater" and the values "the submitted rating". Therefore, the clustering algorithm finds

classes which take into account both values. The center points $M_1, \ldots, M_K$ of clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$ simplify these classes of credibility with fewer, yet more informative points.

5.4.2.4 Assigning a weight $w_{j,i}$ to point $S_{j,i}$

Each point $S_{j,i}$ is submitted by player P_j, which has aggregate score at round $r-1$ τ_j^{r-1}. We use this aggregate score to weight point $S_{j,i}$. We define weight $w_{j,i}$ as:

$$w_{j,i} = \frac{F(\tau_j^{r-1})}{\sum_{j=1}^{n-1} F(\tau_j^{r-1})},$$

where $F : [0,1] \to \mathbb{R}$ is a positive and increasing step function over subintervals of interval $[0,1]$, which assigns higher scores to larger aggregate scores τ_j^{r-1} at round $r-1$. The meaning of such definition for function $F(x)$ is to simplify the weights that a point $S_{j,i}$ can have into fewer possible values. Thus, two aggregate scores $\tau_j^{r-1} < \tau_k^{r-1}$ at round $r-1$ are considered equivalent with respect to the weights $w_{j,i}$ and $w_{k,i}$ for the points $S_{j,i}$ and $S_{k,i}$ if they lie on the same subinterval.

5.4.2.5 Computation of the first component τ_i'

The first component τ_i' is computed as a weighted combination of coordinates $y_{M_1}, \ldots, y_{M_K}$ of the center points $M_1, \ldots, M_K$, respectively. Center points $M_1, \ldots, M_K$ are not equivalent: together with the classes of credibility they distinguish, they depend on the cardinality of the respective clusters. The idea is to associate values $\pi_1, \ldots, \pi_K$ to center points $M_1, \ldots, M_K$ in quantitative and qualitative manner. Values $\pi_1, \ldots, \pi_K$ are regarded as the mixing coefficients of a mixture $p(\mathcal{S}^{(i)})$ of K Gaussian distributions $\mathcal{N}_1(\mu_1, \sigma_1^2), \ldots, \mathcal{N}_K(\mu_K, \sigma_K^2)$. More precisely, the points within each cluster $\mathcal{C}_l$ can be seen as following a Gaussian distribution with $\mu_l = M_l$, for $l = 1, \ldots, K$. That is because the mean and the variance of a Gaussian distribution convey information about where the points are mostly concentrated and how they are spread, which is comparable to the information conveyed by the clusters. Weights $w_{j,i}$ are used to compute the mixing coefficients $\pi_1, \ldots, \pi_K$. In more detail, $\pi_l = \sum_{j=1}^{n_l} w_{l_j,i}$, where n_l is the cardinality of cluster $\mathcal{C}_l$ and $w_{l_j,i}$ is the weight assigned to point $S_{l_j,i} \in \mathcal{C}_l$, for $l = 1, \ldots, K$. Note that $\sum_{l=1}^{K} \pi_l = 1$ and thus the mixture $p(\mathcal{S}^{(i)})$ is a probability distribution. The weighted sum of means $\mu_1, \ldots, \mu_K$ represents the mean μ of the closets Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ approximating mixture $p(\mathcal{S}^{(i)})$ (see Section 5.4.1.2). And this is exactly what we aim at: since the means $\mu_1, \ldots, \mu_K$ are the center points $M_1, \ldots, M_K$, we can now compute the first component τ_i' as the weighted sum of coordinates $y_{M_1}, \ldots, y_{M_K}$ according to the mixing coefficients $\pi_1, \ldots, \pi_K$. That is,

$$\tau_i' = \sum_{l=1}^{K} \pi_l \cdot y_{M_l} \in [0,1].$$

Basically, the first component is computed as coordinate y_μ of the mean μ of the fitting Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. One can argue that the first component τ_i' could be computed directly after clusters $\mathcal{C}_1, \ldots, \mathcal{C}_K$ were distinguished, without passing through the step of computing the mixture of Gaussians. This is, in fact, what one would practically do when computing τ_i'. However, we highlight that this computation is possible because the center points of the clusters model are the means of Gaussian distributions.

We recall that the technique proposed to compute the first partial trust value $\tau_i^{(1)}$ for storage server S_i assumes an honest majority among the storage servers within $\mathcal{S}$. Otherwise, the cluster with the highest weight is the one with a high density of untrustworthy storage servers. Note that this is a common assumption within the framework of distributed storage.

5.4.3 Computation of the Second Component τ_i''

To make more evident the effect of incentives and penalties in the evaluation presented in Section 5.4.4, the second component τ'' is computed slightly differently than how it was originally defined in Section 5.3. More precisely, the second component τ'' for a storage server S_i is still defined as $\tau_i'' = \frac{1}{n-1} \sum_{j=1}^{n-1} o_{i,j}$, but the reliability score $o_{i,j}$ assigned to $|\tau_j' - \rho_{i,j}^r|$ is now a value in the set.

$$\mathbf{O} = \{x : x = \alpha_1 + (-\alpha_1 \times m), m \in \{0, 1, 2, \ldots, (\alpha_2 - 1)\}\},$$

where $\alpha_1 \in \{0, 1\}$ specifies, both, a maximum reward and a series of penalty values; and $\alpha_2 \in \mathbb{Z}^+$ specifies the total number of elements, or steps, in $\mathbf{O}$. Intuitively, the first value of $\mathbf{O}$ is α_1, the second value is 0 and subsequent values are multiples of $-\alpha_1$, with a total of α_2 elements. The element $o_{i,j} \in \mathbf{O}$ is assigned to $|\tau_j' - \rho_{i,j}^r|$ at index $\lfloor |\tau_j' - \rho_{i,j}^r| \cdot \alpha_2 \rfloor$. With such an interpretation, if $|\tau_j' - \rho_{i,j}^r|$ is between the range $\left[0, \frac{1}{\alpha_2}\right]$, the first element of $\mathbf{O}$, namely α_1, is assigned as $o_{i,j}$.

5.4.4 Evaluation

The performance scoring mechanism described in Section 5.3.1 instantiated using the machine learning techniques of K-means clustering algorithm and of mixture of Gaussians as discussed in Section 5.4.2 is capable of detecting coalitions of rational players and penalizing them.

We recall that, according to what a weakly dominant strategy for a rational behavior is (see Section 5.2) and to the assumptions stated in Section 5.3, the players either behave honestly by submitting always accurate ratings or behave rationally (or dishonestly) by submitting always inaccurate ratings. More precisely, an accurate rating is in accordance with the actual performance observed (the targeted first

component $\overline{\tau}'_i$ of a player P_i). An inaccurate ratings boosts or lowers the actual performance observed depending on whether the player in colludes or not in the same coalition as the submitter of the rating.

In our experiments, rational players submit inaccurate ratings where they try to boost or lower the targeted first component of the player to be rated according to a Beta distribution. We choose the Beta distribution as it models rational players that alter the ratings conservatively most of the time and aggressively a few times with low probability. In the description of the performance scoring mechanism in Section 5.3, this behavior was not assumed specifically but totally possible because Theorem 5.8 was proven by taking into account only the worst case scenarios where ratings where totally diverged from the targeted first component. For this evaluation, we have chosen to model the ratings through the Beta distribution to probe the detection capabilities of our mechanism because it is harder to detect a rational player when its ratings are not completely opposite to the actual performance. In particular, given a random sample $\Delta_r \sim Beta(a, b)$, a rational player P_i submits rating $\rho^r_{i,j}$ to compute the first component τ'_j of player P_j as:

$$\rho^r_{i,j} = \overline{\tau}'_j \pm \Delta_r, \tag{5.11}$$

where a and b are chosen parameters for a Beta distribution according to one of the combinations shown in Figure 5.1. In Equation (5.11), a sample Δ_r of the chosen Beta distribution is added to the targeted first component $\overline{\tau}'_j$ if player P_j belongs to the same coalition as player P_i and it is subtracted otherwise.

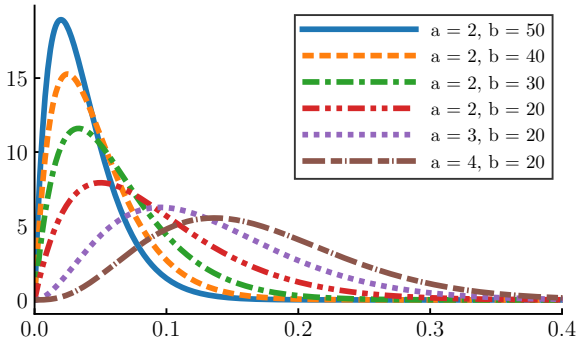


Figure 5.1: Family of Beta distributions that specifies how much ratings are boosted or lowered by rational players.

For the generation of ratings by rational players, we choose the parameters $a = 3$, $b = 20$ for the Beta distribution in Equation (5.11). These parameters model a distribution with an expected value of $\mathbb{E}[Beta(3, 20)] = 0.13$. This implies that, most of the time rational players increase or decrease the rating of others by 0.13 points.

In other words, the increase or decrease can range from small values close to 0.0 up to the high value 0.35 (with low probability), modeling the mostly conservative behavior of rational players discussed above.

An honest players P_i submits rating $\rho^r_{i,j}$ to compute the first component τ'_j of player P_j according to a Gaussian distribution. In particular, given a random sample $\Delta_h \sim \mathcal{N}(0, t_\varepsilon)$, rating $\rho^r_{i,j}$ submitted by a honest player P_i to rate P_j is computed as:

$$\rho^r_{i,j} = \overline{\tau}'_j + \Delta_h, \tag{5.12}$$

where the sample Δ_h from a Gaussian distribution is added to the targeted first component $\overline{\tau}'_j$ to account for the potential inaccuracies in the empirical observations player P_j might have had. In order to prevent the samples of the Gaussian distribution used by honest players from greatly overlapping with the samples of the Beta distribution used by rational players, thus making honest and dishonest behaviors almost indistinguishable, in all experiments, t_ε is chosen such that $t_\varepsilon < \mathbb{E}[Beta(a, b)]$. For the generation of ratings by honest players, we choose the parameter $t_\varepsilon = 0.05$ for the Gaussian distribution in Equation (5.12).

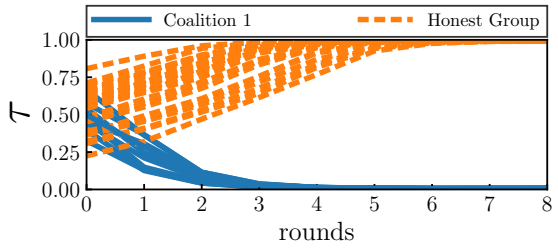
Furthermore, the function $F(x)$ used in Section 5.4.2.4 to compute the weight $w^r_{i,j}$ of a point $S_{i,j}$ submitted by player P_i to rate player P_j used in the experiments is defined as follows:

$$F(x) = \begin{cases} 1 & \text{if } 0.00 \leq \tau_i^{r-1} \leq 0.25 \\ 2 & \text{if } 0.25 < \tau_i^{r-1} < 0.50 \\ 3 & \text{if } 0.50 < \tau_i^{r-1} \leq 0.75 \\ 4 & \text{if } 0.75 < \tau_i^{r-1} \leq 1.00. \end{cases},$$

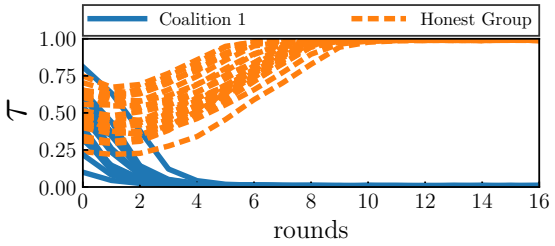
For the remaining parameters, we chose incentive and penalty $\alpha = 0.05$ and$\beta = 15$ and the coefficients of the convex combination as $a = b = c = \frac{1}{3}$ (these a, b shall not be confused with the parameters of the Beta distribution). The number of clusters to individuate is $K = 3$. Once these parameters are selected, the algorithm is deterministic except for how rational players alter ratings and for how honest players diverge from the targeted first component. Furthermore, the players are bootstrapped with an initial aggregate score that follow the Gaussian distribution $\mathcal{N}(0.50, 0.40)$, so that they can take any value between 0 and 1. All experiments are repeated 50 times. Instead of showing aggregated graphs of all experiments, we choose to show one descriptive scenario that represents the experiments well.

As shown in Figure 5.2, the x-axis of each plot is the number of rounds (or, the number of times) our performance scoring mechanism is run and the aggregate scores are updated, which are represented by the y-axis. Note that the initial aggregate scores at round 0 are the ones bootstrapped according to the above mentioned Gaussian distribution. The aggregate scores computed by our performance scoring mechanism are the ones from round 1 on. In each of the four plots of Figure 5.2,

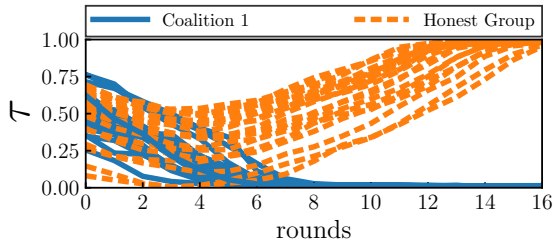
setups of different rational and honest players are tested. Figure 5.2a shows that it is easy to detect a single coalition where 10 out of the total 40 players are rational. This corresponds to a tolerance of 25.0% of rational players. Figures 5.2b and Figure 5.2c show what happens when, respectively, 15 and 19 players out of 40 are rational and collude. Although there is some negative influence from the coalition for a few rounds, the rational players are still detected and their aggregate scores drop eventually to zero. This corresponds to a tolerance of, respectively, 37.5% and 47.5% of rational players. Instead, if 20 out of 40 players are rational and collude, then the performance scoring mechanism does not detect and punish inaccurate ratings and the aggregate scores of the honest players drop to zero, as shown in Figure 5.2d.



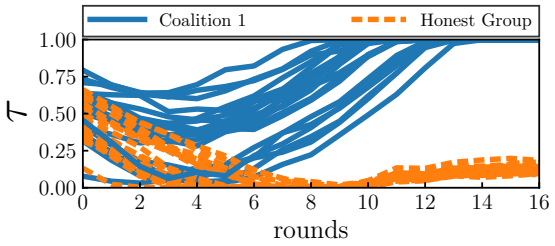
(a) 40 players, 25% are rational.



(b) 40 players, 37.5% are rational.



(c) 40 players, 47.5% are rational.



(d) 40 players, 50% are rational.

Figure 5.2: Change of aggregate scores for our performance scoring mechanism when 25%, 37.5%, 47.7% and 50.0% of the players form a rational coalition.

5.5 Related Work

Our approach bridges two research fields. For one, it is an evidence-based reputation mechanism where aggregate scores are computed by peers. It also formalises the interaction of rational players, which enable data protection through secret sharing. We survey both angles in turn in, respectively, Section 5.5.1 and Section 5.5.2.

5.5.1 Evidence-based Reputation Mechanisms

We have already introduced in Section 2.3 the concept if performance scoring mechanism, which in literature are also often referred to as reputation mechanism. We have already said that this thesis focuses on evidence-based reputation mechanisms, where the aggregate scores are the output of ratings issued after that a trustor has had an interaction with the trustee. In particular, we are interested in computational mechanisms that consider past evidence (via direct or indirect interactions) about trustee's behavior to estimate the trustworthiness of that trustee in the future. In the following, evidence-based reputation mechanisms based on various statistical techniques (especially the Beta probability) and machine learning techniques are surveyed in, respectively, Section 5.5.1.1 and Section 5.5.1.2, and peer-to-peer reputation mechanisms are discussed in Section 5.5.1.3.

5.5.1.1 Bayesian Evidence-based Reputation Mechanisms

Bayesian trust models [65] [28], [83], [60], [95] leverage Bayesian probabilities [18] to estimate the future behavior (i.e. the trust value) of the trustee. In particular, the Beta probability density function is used to estimate the future behavior based on the evidence collected from the past interactions. For instance, the reputation system proposed in [65] calculates trust values following the Beta distribution. However, the system is not able to filter out unfair evidence, making the system ineffective when evidence is not genuine. A robust reputation system is introduced in [28], which deals with honest behavior of the participants. The idea is to learn from the observation of others before having to learn by direct interaction. In other words, reputation ratings are incorporated into the view of others. An extension of the Bayesian probabilistic model is the event-based trust mechanism proposed in [83], which handle so called event-structure frameworks [84]. More precisely, the work provides a formal framework based on information divergence to measure the quality of probabilistic trust mechanisms. Furthermore, the so called trust-aware model introduced in [60] addressing service-oriented environments formalizes a Bayesian service selection model focusing on monitoring and exploring desired service composition. Specifically, the work shows how one can reward/punish the services dynamically with incomplete knowledge of the composition. CertainTrust [95], an extended Bayesian trust model considering context-dependent parameters.

5.5.1.2 Evidence-based Performance Scoring Mechanisms using Machine Learning Techniques

Machine learning plays an important role in the area of trust research. In fact, nowadays, an increasing amount of evidence (or data) is generated by large-scale web applications, e.g. social media, e-commerce, recommender systems. Machine learning techniques are used by researchers to model more and more complex scenarios by

answering two fundamental questions in trust research. The first question is about the initial trustworthiness estimation of a target entity in the absence of past behavior. The second question is about capturing and detecting dynamic behavior of the target entity in different interactions.

In order to address the first question, stereotyping models (e.g. [76]) use the trustor's past experience with other similar entities. These models harness trust-relevant features using machine learning techniques [77] (e.g. Linear Discriminant Analysis (LDA), Decision Tree (DT), and M5 model tree) to extract connections between potential interactions and past interactions. In large-scale open systems like social networks, behavior of an entity may vary in different interactions with different entities to maximize their profits. Approaches based on Hidden Markov Model [81] are essential to effectively capture and detect dynamic behavior patterns.

In order to address the second question, Tang et al. [106] address the issue of dynamic behavior. The authors analyze the trust evolution by investigating the dynamics of the preference of the users on line in review sites like Epinions[5]. It turns out that trust is strongly correlated to the similarity of the preference of the users. In order to capture their preference evolution, hence dynamic trust, the authors use machine learning approaches like the latent factor model [16]. Moreover, in evidence-based trust mechanisms, evidence is often provided by different sources. Honesty of the information source is key for reliable trust estimation and, thus, it is essential to know whether the information source is unbiased or biased. Existing evidence-based trust models use unsupervised approaches, like statistical deviation [75], to identify feedbacks that are very different from others. The assumption here is that the biased feedback is a small subset of all feedbacks.

In this chapter, we address the second question by using machine learning techniques to design our evidence-based trust mechanism. The unsupervised clustering algorithms are able to identify evidence created by dishonest participants. Furthermore, in contrast to related work, with the techniques of fitting mixture of Gaussians to clusters, we identify unreliable evidence submitted by colluding participants. This confers our trust mechanism the capability to downgrade the trustworthiness of groups of participants that have chosen to collaborate in a dishonest manner.

5.5.1.3 Peer-to-Peer Reputation Mechanisms

We review mechanisms promoting accurate ratings in peer-to-peer systems, where accurate/inaccurate ratings are rewarded/penalized. In particular, we focus on mechanisms where agents interact with one or multiple external third parties, either to learn the reputation of their peers or to aggregate their ratings. These are referred to as *third-party-aided peer reputation mechanisms* and our mechanism falls into this category. Jurca et al. [67] consider a scenario involving peers as well as broker agents

[5] http://epinions.com/

where incentives are issued through payment for reputation information. Afterwards, an agent decides whether to engage with peers. This model uses a game-theoretic approach, but addresses peer interaction and not how the reputation information of peers is aggregated and computed. The same authors also investigate how the lack of incentives leads to biased recommendations and ratings on online reviews platforms such as TripAdvisor [68]. The scoring system discussed by Miller et al. [80] is referred to as a peer-prediction model and is most suitable for rating commercial items in online platforms. It involves a centre common to all peers that processes the ratings forwarded by the peers themselves and has no independent information. The model uses a peer's rating to update a probability distribution for the rating of a different peer. Based on these ratings, the centre rewards or penalizes the targeted peer. The problem of coalitions is mentioned but not solved. Our work differs from the ones listed above in the following ways. Our performance scoring mechanism is the first to be provably resilient against peer coalitions. Moreover, it provides a relation between the admissible variations of the performance scores and the size of coalitions. This way, incentives and penalties are implemented before the performance scores are computed (and not later), leading to performance scores that match the desired accuracy level.

5.5.2 Protection Against Rational Parties in Secret Sharing

We review mechanisms for coping with rational players in secret sharing schemes. The notion of *rational secret sharing* was first introduced by Halpern and Teague [59]. Here, the utility of the players is tied to their goal of being the only ones to know the secret, while rationality captures their unwillingness to collaborate with other players to reconstruct the secret. The paradox of rational secret sharing is that the secret is lost because players will never actively collaborate by providing their own share during reconstruction. Halpern and Teague solve this problem by using game theory to incentivise the players to collaborate and they show how their solution reaches a Nash equilibrium if the players are not allowed to form coalitions. Later, Gordon and Katz proposed [54] a protocol also supporting two players only, which was not possible before. Abraham et al. introduced [2] the notion of a k-resilient Nash equilibrium and a secret sharing scheme reaching this equilibrium is designed. As in [2], we consider only players that act rationally according to their ultimate goal and arbitrarily behaving players with unknown utilities are not considered. Instead, this type of players are discussed by Lysyanskaya and Triandopoulos [78] and by Asharov and Lindell [5]. All so far mentioned protocols for rational secret sharing assume that communication happens simultaneously, either through a broadcast channel or through secure private channels. Kol and Naor proposed [71] a rational secret sharing scheme with a non-simultaneous broadcast channel that is also coalition-resistant. In all preceding cases, rational secret sharing protocols are treated as infinitely repeated games, as in our approach. This allows a reward mechanism that forces

the players to not behave selfishly and reach a social optimum. Nojoumian and Stinson proposed [89] a model where players are associated with a score recording the number of times they provided their shares during the reconstruction of the secret. A high score means that players might be included in future secret sharing schemes. Our model differs from the approaches discussed in this paragraph because we consider rationality in terms of economic return for SSPs, which in our scenario boils down to maximising share storage. We do not focus on the performance of the players during the reconstruction of the secret shared data. Instead, we focus on the general performance of the players, with measurements uncoupled from share renewal operations. Our performance scoring mechanism helps the data owner to select high performing storage servers in the first place. All approaches discussed above tackle the threats of private secret acquisition and uncooperative behaviour during secret reconstruction. In contrast, uniquely, we counter the threat of inaccurate peer rating to undermine storage competitors.

5.6 Summary and Future Work

The long-term confidential storage of sensitive data can be realised through proactive secret sharing, performed in a distributed storage system consisting of several storage servers owned by multiple commercial SSPs. Data owners can be guided in the selection of high-performing SSPs through performance scoring mechanisms based on mutual peer ratings.

In this chapter, we addressed the problem of modelling performance scoring mechanisms such that they are resilient against coalitions of rational storage servers, which for the most part of this chapter were referred to as players. We first modelled storage servers as rational agents aiming at maximizing their shares storage and formalized their rating strategies. Second, we showed that performance scoring mechanisms output aggregate scores that do not reflect actual performance if the players are not incentivized to submit accurate ratings. Third, we introduced a novel performance scoring mechanism modelled as an infinitely repeated and cooperative game with a TA as a mediator. The TA incentivizes accurate ratings and detects and penalizes inaccurate ratings with respect to a margin that depends on coalition sizes. Using our game-theoretic formalism, we proved that such a performance scoring mechanism outputs accurate aggregate scores. It thus provides viable guidance for the selection of high-performing storage servers in distributed storage systems. Fourth, we instantiated the performance scoring mechanism by using machine learning techniques to process the ratings submitted by the players and distinguish the accurate ones from the inaccurate ones. Experimental evaluations show empirically that when the rational players collude into a single large coalition so that the majority of the players is still honest then the performance scoring mechanism detects and penalizes the colluders, as expected.

Our framework formalizes the accuracy of the aggregate scores when there is one large coalition only and does not consider multiple smaller coalitions at the same time. Also, it does not take into account the fact that rational players may decide to submit inaccurate ratings with a certain probability only, so that to lower the chance to be detected. Although some experiments we run show that our performance scoring mechanism is still resilient in both above situations, we have not formalized yet these two frameworks yet and we plan to do that as future work. Furthermore, as another future work, we plan to investigate annd formalize the number the number of rounds it takes to output accurate aggregate scores depending on how large the cardinality of the coalitions formed is.

6 Efficient Distributed Storage Systems Based on Trusted Execution Environment

Long-term secure storage of sensitive data must protect the confidentiality and integrity of such data for decades. For instance, electronic health records must be protected for at least the entire lifetime of the patient, or even longer to protect the privacy of descendants. Confidentiality of a health record is crucial to prevent discrimination, while integrity of the record is crucial to ensure the right treatment for the patient.

State of the art secret sharing-based distributed storage systems rely on *proactive secret sharing* and on *commitment schemes* (see Chapter 2) to protect, respectively, the confidentiality and the integrity of the outsourced data. Proactive secret sharing ensures confidentiality because, at regular time intervals, fresh new shares that are completely unlinked to the previous ones are generated so that they still reconstruct to the original data. The integrity of the outsourced data can be protected in the long-term when proactive secret sharing is equipped with commitment schemes. This way, it can be verified the validity of the original shares generated by the data owner when first outsourcing the data (like originally verifiable secret sharing was meant to be). In addition, this enables to check the validity of the updated shares every time these are computed.

However, building such an infrastructure turns out to be costly and still hard to achieve in practice, both in terms of confidentiality and integrity. On the one hand, proactive secret sharing requires that each pair of storage servers are linked by an information-theoretically secure channel, which remains very difficult and expensive to build. So far, only NICT in Japan have provided such infrastructure [24], where quantum key distribution [97] is used to build point-to-point information-theoretically secure channel between any two nodes. This is, however, one of the rare examples of information-theoretically secure channels that have been established in practice with QKD, since this technology is still very expensive and comes with scalability limitations. On the other hand, commitment schemes are computationally intensive, and are in practice never used for large files (e.g., CT scans or genomic data).

Contributions

In this chapter, we investigate more efficient protection of confidentiality and integrity in long-term secure distributed storage systems. We propose LSTee, the first proactive solution for long-term secure storage systems that relies on a trusted execution environment (TEE) (which is an integrity-protected and isolated environment with respect to both hardware and software) for the generation, renewal and reconstruction of valid shares. More precisely, in LSTee, the shares are generated in the TEE and then securely distributed and provisioned to the storage servers. The shares are not stored into the TEE, but they are brought back into it only for periodic renewal and reconstruction when the data is requested by the data owner. This approach leads to significantly optimized secret sharing protocols for generating and renewing the shares, where commitment schemes are replaced with more practical public-key cryptography for verifying the integrity of the shares. Furthermore, the amount of information-theoretically secure channel needed is significantly reduced. The amount of information-theoretically secure channel needed by LSTee is linear in the number of storage servers as opposed to quadratic for the state of the art approaches discussed above. Despite enabling more practical computations and lower communication costs of the infrastructure, assuming a trusted TEE, LSTee provides the same security (long-term confidentiality and integrity) guarantees as state of the art long-term secure distributed storage systems. LSTee is TEE-agnostic, i.e. it is not tied to a specific TEE, and supports seamless migration from a compromised or unavailable TEE to another trustworthy. Moreover, we prototype our protocols on a TEE, instantiated with Intel SGX and show that LSTee is practical, even for large files We present runtimes for secret sharing and reconstructing a file with varying file sizes and parameters, as well as for renewing the shares for long-term security.

Contributions to this chapter come from paper [T8]. Due to the collaborative nature of this work, all co-authors contributed to the main contribution of this chapter. Besides, my most explicit role in this work was to systematically pointing out the limitations and constraints of state of the art secret sharing-distributed storage systems so as to develop at the end a viable solution that needs fewer information-theoretically secure channel that is also complaint with the original framework. Moreover, the security analysis of LSTee and the comparison of LSTee with related work with respect to number of information-theoretically secure channel and communication traffic were solely my input.

Outline

The rest of the chapter is organized as follows. In Section 6.1, relevant preliminaries are described. In Section 6.2, a detailed description of state-of-the-art long-term

secure storage systems based on secret sharing primitives and commitment schemes is provided. In Section 6.3, we present our solution LSTee. In Section 6.4, we present the security analysis of LSTee. We compare with related work with respect to communication and computation complexities in Section 6.5, and instantiate LSTee with a specific TEE (Intel SGX) and evaluate its performance in Section 6.6. We conclude with directions for future work in Section 6.7.

6.1 Trusted Executions Environments

Trusted Execution Environments (TEEs) are increasingly recognized as a key component for the deployment of security-sensitive and privacy-preserving applications such as digital content protection [22] or financial services [15]. A TEE is an execution environment with its own hardware and software components, running in a totally isolated manner alongside the operating system of a computing server, which is referred to as the TEE operator. It guarantees that the code and the data inside the TEE itself are protected with respect to confidentiality and integrity. A TEE has its proprietary assets and communicates with the external environment through TEE APIs. Commercial TEE solutions include Intel SGX [64] and ARM TrustZone [4]. Because this is TEE on which the instantiation of LSTee of Section 6.6 is based, in the following, we only present how Intel SGX works.

Intel Software Guard Extensions (SGX) is Intel's widely available TEE where confidential data can be processed securely on untrusted systems [64]. To do this, SGX introduces the concept of *enclaves*, which are software components executed in isolation from all software running on the system, including the untrusted operating system or hypervisor. SGX assumes the CPU hardware to be the only trustworthy hardware component of the system. While data is stored and processed unencrypted in the CPU's caches and registers, it is encrypted and integrity-protected once it is moved out of the CPU, e.g., into DRAM. A host process (usually the OS) loads an enclave from its virtual memory and manages its creation which means that the enclave's initial data and code content may be manipulated. Therefore, to ensure that confidential data is not sent to a malicious or corrupted enclave, the authenticity and integrity of an enclave are verified through *remote attestation* (see more details below). An external party verifies whether an enclave was created correctly by checking the cryptographic hash of the enclave's initial memory that is signed by the platform's secret key. It then communicates the confidential data to the enclave over a secure channel only after the enclave has been attested. Once inside the enclave, data can be securely processed in isolation and later on encrypted with an enclave-specific secret key before it is written to untrusted storage, e.g., DRAM.

In the following, we briefly describe what remote attestation protocols are in

order to better understand how SGX works and our LSTee solution in Section 6.3. Remote attestation is a security service that is realized by means of an interactive challenge-response protocol to allow a *verifier* to capture the state of a potentially untrusted remote *prover*. The verifier initiates the attestation protocol by sending a request/challenge to the prover. Upon receiving the request, the prover computes a digest, e.g., a cryptographic hash, of its memory, signs it, and sends it back to the verifier. Based on that, the verifier then decides whether the prover is in a trustworthy state or not. However, if the prover is compromised it may evade detection by sending the verifier the expected benign measurement. Therefore, a trusted component or trust anchor at the prover, that is trusted to faithfully measure the software state of the prover and send the measurement back to the verifier, is required.

6.2 State of the Art Proactive Secret Sharing-Based Distributed Storage Systems

We recall that long-term secure storage systems based on secret sharing utilize several storage servers owned by different commercial SSPs where each of the storage servers receives a share of the data from the data owner. The main goal of such infrastructure is to *proactively* protect the confidentiality and the integrity of the outsourced data in the *long term*. We describe in Section 6.2.1 the requirements and assumptions underlying state of the art long-term secure distributed storage systems. We present in Section 6.2.2 the well-established proactive secret sharing scheme by Herzberg et al. [62] and discuss its shortcomings in Section 6.2.3 (other, more recent solutions, are rather incremental with respect to the solution by Herzberg et al. and are discussed and compared to our solution in Section 6.5).

6.2.1 Requirements and Assumptions

In the following, in Section 6.2.1.1 and Section 6.2.1.2 we detail, respectively, the network requirements and the adversary model that state of the art approach for long-term secure distributed storage systems can cope with.

6.2.1.1 Network Requirements

The communication between the nodes and the data owner is characterized by the following requirements.

1. There is an information-theoretically secure channel between any two storage servers and between any data owner and each storage server.
2. There exists a reliable broadcast channel including all the nodes of the long-term secure storage system.
3. The messages sent are reliably delivered.

4. Authentication measures are in place (no spoofing possible).

Thus, a *synchronous* network with access to a common global clock is assumed.

6.2.1.2 Adversary Model

The threat model in long-term secure storage systems is that of a *mobile*, *active* and *computationally bounded* adversary. A *mobile* adversary can, *over time*, break into as many storage servers as it wants, and thus can collect enough shares to reconstruct the data on its own. However, a mobile adversary is limited in the number of storage servers it can corrupt (and thus the number of shares it can collect) within a *certain period of time*. In particular, within any time period we assume it cannot break into more than $t-1$ storage servers, which means that $n \geq 2t-1$ for preserving confidentiality and integrity with honest majority (i.e., we require at least t honest servers), where n is the total amount of storage servers included in the system. That is why periodically refreshing the shares of the storage servers by performing proactive secret sharing is an effective countermeasure against this type of attacker. It is also assumed that the attacker leaves the storage servers it broke into when these are performing proactive secret sharing, so that the protocol can actually be used effectively to derive fresh new shares unlinked to the old ones.

An *active* adversary is an adversary that does not remain a passive eavesdropper. Instead, it deliberately tries to gain unwarranted access to information about the data by making the corrupted storage servers deviate from the protocols they are supposed to run, modifying the data stored on them or disconnecting them, etc. Note that, for simplicity, storage servers subjected to accidental crashes or power failures are considered corrupted and under the control of the adversary.

A *computationally bounded* adversary, in the context of long-term secure storage systems, does not have enough computational power to break cryptographic schemes instantiated with state-of-the-art choices of security parameters at a given point in time.

Furthermore, the adversary can connect to the broadcast channel mentioned above in the network requirements, observe and corrupt all messages that the storage servers or the data owner broadcast and can also inject its own. However, the adversary cannot prevent a benign storage server from receiving any of the messages sent through the broadcast channel. A malicious data owner can also be assumed, who can intentionally distribute corrupted shares to storage servers to blame them later for not preserving the integrity of the document shares. Finally, it is assumed that, as soon as attacks or deviations from the regular protocols are detected, a reboot mechanism is performed so that the adversary is removed from the storage servers it corrupted. This allows, for instance, to perform the share renewal effectively without leaking information.

6.2.2 The Existing Solution

A countermeasure against the mobile adversary is to periodically refresh the shares. More precisely, the shares are renewed at regular intervals of time so that the updated shares are completely independent of the old ones. At any point in time, i.e., between two share renewal phases, the mobile adversary is limited to corrupt $t-1$ storage servers as described in Section 6.2.1.2. After the next share renewal phase, the updated shares are entirely independent from the old shares, meaning the attacker has to start over and corrupt the storage servers again. This process will always leave the adversary with no information about the outsourced data and confidentiality is therefore maintained. Share renewal is also referred to as proactive secret sharing, as introduced by Herzberg et al. in [62]. At a high level, storage servers distributedly compute some randomness values (also refereed to as subshares) and add them to the old shares to generate the updated shares. These subshares are computed by using the underlying Shamir's secret sharing scheme.

To thwart an active adversary, verifiable secret sharing is used, where commitments to the coefficients of the polynomial used to compute the shares are broadcasted to all storage servers, and are used to detect deviations from the regular protocols. During the first share generation phase Share, commitments are used by the storage servers to check whether the data owner distributed consistent shares or not. During share renewal Renew, each storage server acts simultaneously as a receiver and a dealer of subshares. Commitments are used to check whether the subshares received are consistent with the outsourced document and, therefore, can be used to compute valid updated shares. In this way, integrity of the outsourced document is also preserved because valid shares reconstruct to the original document. Proactive secret sharing supports both Feldman's and Pedersen's commitment schemes used in verifiable secret sharing (see Section 2.1 for more details).

In the following, we present the proactive secret sharing protocol by Herzberg et al. [62] to periodically renew the shares of the outsourced data. It is presented in its verifiable version, where Pedersen's commitment is used to cope against malicious storage servers sending invalid subshares during the protocol. Pedersen's commitment scheme is unconditionally hiding. This means that, contrary to the computationally hiding Feldman's commitment scheme, it does not allow an adversary to store the broadcasted commitments until the underlying mathematical problem (in Feldman's case, the Discrete Logarithm Problem) becomes easily solvable due to cryptanalytic advances or the construction of a sufficiently large quantum computer. Thus, commitments computed using Pedersen's scheme cannot be broken to retrieve the coefficients of the polynomial used to compute the shares and, thus, to retrieve the outsourced data.

In order to use a commitment scheme during Renew, it has to be in place also during Share and Reconstruct. Thus, in the following, we also show how Share and Reconstruct described in Section 2.1 are modified with Pedersen's commitment

scheme.

Share chooses two large primes p, q, are randomly chosen, such that $q|(p-1)$. Then it takes as input a message $m \in \mathbb{F}_q$ to be outsourced to the long-term secure storage system composed of storage servers $S_1, \ldots, S_n$, where $i \in \mathcal{I}$ is the unique identifier ID of storage server S_i by performing the following steps. Let g, h be distinct generators of the q-th order subgroup $\mathbb{F}_q$ of $\mathbb{F}_p^*$ and set $\mathbb{F}_q$ as both the message space and the share space. The reconstructing threshold t is chosen so that $n \geq 2t-1$, so that there is an honest majority among the storage servers making up the long-term secure storage system. Polynomials $f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1}$ and $\varphi(x) = b_0 + b_1 x + \cdots + b_{t-1}x^{t-1}$ are defined, where $a_0 := m$ and $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$ and $b_0, b_1, \ldots, b_{t-1} \in \mathbb{F}_q$ are chosen uniformly at random. In the following, b_0 is denoted as s to distinguish it from the other coefficients. The data owner computes share σ_i for storage server S_i as $\sigma_i := (f(i), \varphi(i))$, for $i = 1, \ldots, n$. The data owner commits to each coefficient pair (a_k, b_k) by computing $c_k := g^{a_k} h^{b_k} \mod p$, for $k = 0, 1, \ldots, t-1$. It broadcasts the commitments and sends share σ_i to storage server S_i through an information-theoretically secure channel. Each storage server S_i accepts σ_i as its valid share if and only if

$$g^{f(i)} h^{\varphi(i)} \equiv \prod_{k=0}^{t-1} c_k^{i^k} = g^m h^s \prod_{k=1}^{t-1} (g^{a_k} h^{b_k})^{i^k}.$$

Renew takes as input a subset of t valid shares $\sigma_1, \ldots, \sigma_t$ and outputs updated shares σ_i' for storage server S_i, for $i = 1, \ldots, n$. Each storage server S_i performs the following steps.

1. It select polynomials $f_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t-1}x^{t-1}$ and $\varphi_i(x) = b_{i,0} + b_{i,1}x + \cdots + b_{i,t-1}x^{t-1}$, where $a_{i,0} = b_{i,0} = 0$ and both coefficients $a_{i,1}, \ldots, a_{i,t-1} \in \mathbb{F}_q$ as well as $b_{i,1}, \ldots, b_{i,t-1} \in \mathbb{F}_q$ are chosen uniformly at random.
2. It computes subshare $\sigma_{j,i} := (f_i(j), \varphi_i(j))$ for storage server S_j with identity $j \neq i$, and subshare $\sigma_{i,i} := (f_i(i), \varphi_i(i))$.
3. It computes commitment $c_{i,k}$ to coefficients $a_{i,k}$ and $b_{i,k}$ as $c_{i,k} := g^{a_{i,k}} h^{b_{i,k}}$, for $k = 0, 1, \ldots, t-1$.
4. It sends subshare $\sigma_{j,i}$ to storage server S_j through an information-theoretically secure channel for $j \neq i$, keeps subshare $\sigma_{i,i}$ private, and broadcasts the commitments $c_{i,0}, c_{i,1}, \ldots, c_{i,t-1}$.
5. It receives subshare $\sigma_{i,j} = (f_j(i), \varphi_j(i))$ and commitments $c_{j,0}, c_{j,1}, \ldots, c_{j,t-1}$ from storage server S_j, for $j \neq i$.
6. It accepts $\sigma_{i,j}$ as a valid subshare if and only if

$$g^{f_j(i)} h^{\varphi_j(i)} \equiv \prod_{k=0}^{t-1} c_k^{i^k} = g^m h^s \prod_{k=1}^{t-1} (g^{a_k} h^{b_k})^{i^k}.$$

7. If all subshares are valid, then it computes its updated share σ'_i as $\sigma'_i := (f(i) + \sum_{j=1}^{n} f_j(i), \varphi(i) + \sum_{j=1}^{n} \varphi_i(j))$. Otherwise, it broadcasts a complaint message against the sender(s) S_j of the invalid subshare $\sigma_{i,j}$ and aborts the protocol.
8. It deletes old share σ_i.

Reconstruct takes as input t valid shares $\sigma_1, \sigma_2, \ldots, \sigma_t$ to reconstruct polynomial $f(x)$ and polynomial $\varphi(x)$ by using Lagrange interpolation. Message m is retrieved as $f(0) = m$ and value s is retrieved as $\varphi(0) = s$. Having access to the commitments originally broadcasted by the data owner, it is possible to check the validity of the reconstructed message m and value s by verifying that it is a correct opening value for commitment c_0, i.e., $g^m h^s \equiv c_0$.

The above proactive secret sharing scheme is a secret sharing scheme according to Definition 2.2 that provides the security guarantees of accessibility and perfect security formalized in Definition 2.1, equipped with an additional algorithm Renew run by the storage servers in distributed fashion that still satisfy the security properties of Definition 2.1. Note that, for simplicity, we have shown the above protocols for only a message chunk m of a larger message M to be outsourced. Thus, the sharing, renewing and reconstructing protocols have to be performed for each of chunk m that message M was divided into. We formalize this in Section 6.3.2 when presenting LSTee.

6.2.3 Shortcomings of the Existing Solution

The protocols Share, Renew, and Reconstruct described above suffer from two major shortcomings that hinder their deployment in practice. On the one hand, all shares and subshares must be sent through information-theoretically secure channel. This means that n information-theoretically secure channel from *each* data owner to all storage servers $S_1, S_2, \ldots, S_n$ are needed during Share and that $\frac{n(n-1)}{2}$ information-theoretically secure channel between each pair of storage servers are needed to run Renew once. These can be reduced to $\frac{t(t-1)}{2} + t(n-t)$, since the reconstructing threshold is t, then protocol Renew (as well as protocol Reconstruct) can be successfully carried out when a subset of at least t storage servers perform steps 1.-4. However, the establishment of these information-theoretically secure channel remains very expensive. They require the usage of quantum key distribution (QKD) protocols, such as BB84 [14] or E91 [44], which are very costly to implement, or they require off-line one-time pad (OTP) key material exchange, which is difficult to achieve in practice. For more details, we refer to the security analysis in Section 6.4.

On the other hand, commitment schemes are also highly impractical to be computed for the size of data that is normally outsourced and, thus, are never implemented, leaving the integrity unverified. Commitments cannot be computed over the hashes of data because the homomorphic property needed to perform operations in the

exponent would be lost, and thus, the commitment cannot be verified. To share large files with Shamir's secret sharing scheme, these are first split into multiple chunks, say ℓ, and then for each chunk m Shamir's secret sharing scheme is applied. This implies that, just for protocol Share, on the data owner's side $\ell(t-1)$ commitments have to be computed and broadcasted, where ℓ normally ranges between 2^{15} and 2^{20} for a large file of several megabytes. Furthermore, in order to check the validity of the share, each storage server has to compute at least ℓ full-length and $\ell(t+1)$ non-full-length exponentiations in a group with order around 256 bits. This is computationally very expensive and makes this approach hard to be adopted in practice. With LSTee, we attempt to mitigate these shortcomings and reduce the described requirements by relying on trusted execution environments (TEEs) in our solution, which we present next.

6.3 LSTee: Our Solution

LSTee leverages TEEs for the generation, renewal and reconstruction of consistent shares in an integrity-protected and isolated environment. This guarantees that the shares are instantiated, updated and reconstructed as mandated by the underlying protocols and that the consistency of the shares with the original document is preserved. The integrity of the shares at the storage servers is protected by means of computationally secure integrity proofs, i.e., signatures, that are issued and distributed to the storage servers by the TEE together with the respective shares. In this way, the TEE attests the shares before performing share renewal or the reconstruction of the original document, and identifies which storage server is compromised by its corrupted share by means of signature verification. LSTee involves several parties: the data owner(s), the TEE operator, i.e. a computing server equipped with the TEE and its hardware-based trust anchor, and storage servers. We first present our network requirements and adversary model in Section 6.3.1, then a high-level overview of LSTee's optimized secret sharing protocols in Section 6.3.2 and outline our optimizations with respect to the existing Share, Renew and Reconstruct described in Section 6.2.2. Thereafter, we detail the protocols in Section 6.3.3 and elaborate on TEE migration in Section 6.3.5.

6.3.1 Requirements and Assumptions

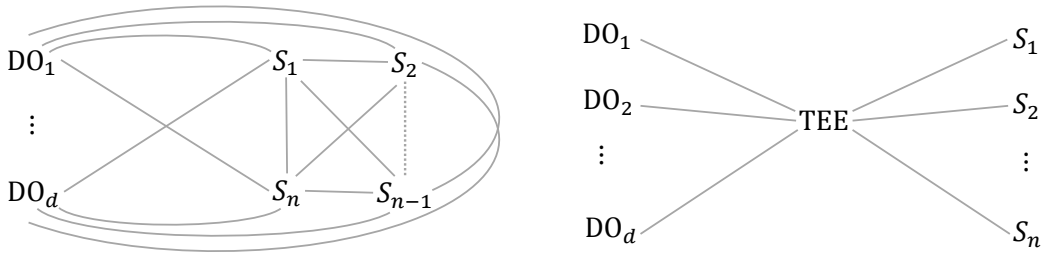
In this section, we detail network requirements (Section 6.3.1.1) and security assumptions with respect to the adversary (Section 6.3.1.2) needed for LSTee.

6.3.1.1 Network Requirements

As shown in Figure 6.1b, network requirements for LSTee differ from that of the state-of-the-art long-term secure storage systems (see Section 6.2.1) because a broadcast channel is no longer needed, thus, we can disregard all assumptions regarding this. Note that LSTee does not require a *synchronous* network with access to a common global clock, since Renew is initiated and performed by the TEE. Our network assumptions are as follows.

1. There is an information-theoretically secure channel between every storage server and the TEE.
2. There is an information-theoretically secure channel between any data owner and the TEE.
3. The messages sent are reliably delivered.
4. Authentication measures are in place.

Only data owners and the storage servers are directly connected to the TEE operator, while we eliminate the need for a direct information-theoretically secure channel between any two storage servers or between data owners and storage servers. The architecture of our system is depicted in Figure 6.1b, where $S_1, S_2, \ldots, S_n$ denote n storage servers, $\mathrm{DO}_1, \mathrm{DO}_2, \ldots, \mathrm{DO}_d$ denote d data owners, TEE denotes the trusted execution environment, and lines are information-theoretically secure channel.



(a) Architecture of existing solutions.

(b) Architecture of our solution.

Figure 6.1: Architecture of the state of the art protocols for long-term secure distributed storage systems and our protocol based on a trusted execution environment.

6.3.1.2 Adversary Model

We assume an adversary model with identical capabilities to that described in Section 6.2.1 for state of the art long-term secure storage systems. Furthermore, we assume an untrusted TEE operator, that has full control over the machine deploying a trusted TEE and can execute arbitrary code with supervisor privileges. In face of memory corruption vulnerabilities, we assume deployment of common code-reuse

defenses, such as control-flow integrity [1,30], or fine-grained code randomization [41, 73]. A number of architectural side-channel attacks [23,33,110] leaking confidential data from TEE-based solutions have been recently shown and several defenses [21, 35,103] have been proposed to thwart them. We consider mitigating TEE-based side-channel leakage as an orthogonal problem and out of scope for this chapter.

6.3.2 LSTee Optimized Protocols: Overview

In LSTee, the TEE performs the Share, Renew, and Reconstruct protocols in a privacy-preserving manner.

The data owner is not required to be online after sending its data to the TEE, and only communicates with the TEE operator at the beginning of the Share and Reconstruct. Note that LSTee's protocols require that the *trust anchor* at the untrusted TEE operator attests the TEE's integrity first and then initiates it. Then, the TEE establishes information-theoretically secure channel with the other parties and runs the protocols. For simplicity, we do not describe the TEE initiation steps below.

6.3.2.1 Protocol Share

Our share initialization protocol is initiated by the data owner and runs between the data owner, the TEE, and the storage servers. The data owner sends the pertinent document to the TEE via an information-theoretically secure channel. The data owner also selects the storage servers making up the storage system as well as the threshold number t of shares necessary to reconstruct the document. For all storage servers, the TEE generates the document shares along with their integrity proofs and distributes them to each of the selected storage servers individually via information-theoretically secure channel. At this point, each storage server holds its document share and an integrity proof ready for the first round of share renewal Renew. Contrary to state of the art Share of Section 6.2.2, where each data owner establishes an information-theoretically secure channel with each of the storage servers, we optimize this by requiring that data owners establish information-theoretically secure channel with the TEE only. The TEE, in turn, establishes an information-theoretically secure channel with each of the storage servers.

6.3.2.2 Protocol Renew

Our share renewal protocol is performed periodically as in the state of the art (see Section 6.2.2). However, our protocol is initiated by the TEE sending an update request to each storage server. Upon receiving the document share and its integrity proof, the TEE verifies the integrity of the share. If the share is valid, the TEE generates a subshare required to update the share, otherwise reports malicious

behavior to the service provider. The updated share is computed by adding the subshare to the old share and sending the result to the corresponding storage server along with a new proof of integrity. This is performed by the TEE for each storage server S_i, for $i = 1, \ldots, n$ individually. Therefore, this eliminates the need for information-theoretically secure channel between every two storage servers and significantly reduces the number of information-theoretically secure channel compared with Renew protocol from Section 6.2.2.

6.3.2.3 Protocol Reconstruct

Our document reconstruction protocol is initiated by the data owner sending a request to the TEE, which, in turn, collects t shares from t storage servers to reconstruct the document. Upon receiving the shares, the TEE verifies their integrity, then reconstructs the document. Similar to our Share, we eliminate the need for information-theoretically secure channel between data owners and each of the storage servers. Instead, our Reconstruct only requires that each data owner establishes a single information-theoretically secure channel with the TEE while the TEE establishes the channels with the storage servers.

6.3.3 LSTee Optimized Protocols: Detailed

In LSTee, the TEE, encompassed in the TEE operator, performs the secret sharing protocols, while public-key signatures are used to verify the integrity of the provisioned shares before share renewal and document reconstruction protocols are performed. Therefore, commitment schemes are no longer required. In Table 6.1, we present the commonly used symbols throughout the paper with their description.

A document M can be expressed as the concatenation of multiple chunks of size B, i.e. $M = m_1||m_2||\ldots||m_N$, such that the number of chunks is $N = \frac{|M|}{B}$. Let n be the total number of storage servers $S_1, S_2, \ldots, S_n$ selected by the data owner, id be the unique identifier ID of storage server S_i for $i = 1, 2, \ldots, n$, t be the threshold number of shares necessary to reconstruct the document, $\sigma_{i,\ell}$ be a share of chunk m_ℓ, for $\ell = 1, 2, \ldots, N$. $Sh_i = \sigma_{i,1}||\sigma_{i,2}||\ldots||\sigma_{i,N}$, document share, is a concatenation of the shares $\sigma_{i,\ell}$, for $\ell = 1, 2, \ldots, N$ distributed to storage server S_i for $i = 1, 2, \ldots, n$. The TEE T_1 owns a pair (pk_{T_1}, sk_{T_1}) of public and secret keys. All communication between the data owner, the TEE and the storage servers described in the protocols below occurs through information-theoretically secure channels.

Share involves the data owner, the TEE, and storage servers $S_1, S_2, \ldots, S_n$. It is initiated by the data owner, who sends document M, the number of storage servers n, their IDs and the reconstructing threshold t to the TEE through a information-theoretically secure channel. The TEE executes the following steps.

1. It receives M, n, t and storage servers' identifiers IDs from the data owner.
2. It splits M into ℓ chunks, $M = m_1||m_2||\ldots||m_N$.

3. For each document chunk $m_\ell \in \mathbb{F}_q$, where $\mathbb{F}_q$ is a finite field with $q > n$ elements, for $\ell = 1, 2, \ldots N$:
   a. it selects a polynomial $f_\ell(x) = a_{0,\ell} + a_{1,\ell}x + \cdots + a_{t-1,\ell}x^{t-1}$ such that $a_{0,\ell} := m_\ell$ and $a_{1,\ell}, \ldots, a_{t-1,\ell} \in \mathbb{F}_q$ are chosen uniformly at random;
   b. it compute n document chunk shares such that $\sigma_{i,\ell} := f_\ell(i)$ for storage server S_i, for $i = 1, \ldots n$.
4. It aggregates N document chunk shares into a unique document share Sh_i as $Sh_i := \sigma_{i,1}|| \ldots ||\sigma_{i,N}$.
5. It computes signature P_i of document share Sh_i with the private key sk_{T_1} as $P_i = sign(sk_{T_1}, Sh_i)$, for $i = 1, \ldots, n$.
6. It distributes document share Sh_i along with its signature P_i to storage server S_i through a information-theoretically secure channel, for $i = 1, \ldots, n$.

Storage servers trust that the TEE generates consistent shares. Therefore, document share Sh_i and its signature P_i are securely provisioned to storage server S_i until the first round of the share renewal protocol Renew.

Renew involves the TEE and the storage servers $S_1, \ldots, S_n$ and is periodically initiated by the TEE that performs the following steps.

1. It sends update requests to storage servers $S_1, \ldots, S_n$.
2. It receives n document share Sh_i and its signatures P_i sent by storage server S_i through a information-theoretically secure channel, for $i = 1, \ldots, n$.
3. It attests the integrity of document share Sh_i by verifying its signature P_i as follows: $ver(pk_{T_1}, P_i, Sh_i) \stackrel{?}{=} \text{true}$ for $i = 1, \ldots, n$. If the document share Sh_i is valid, then it proceeds with step 4. Otherwise, it recovers the original document M from t valid shares, notifies the service provider of the corruption of storage server S_i and executes protocol Share to recompute document share Sh_i.
4. For each document chunk m_ℓ, for $\ell = 1, \ldots, N$:
   a. it selects a polynomial $g_\ell(x) = b_{0,\ell} + b_{1,\ell}x + \cdots + b_{t-1,\ell}x^{t-1}$, where $b_{0,\ell} = 0$ and coefficients $b_{1,\ell}, \ldots, b_{t-1,\ell} \in \mathbb{F}_q$ are chosen uniformly at random;
   b. it computes subshare $r_{i,\ell}$ for storage server S_i as $r_{i,\ell} := g_\ell(\text{id})$, for $i = 1, \ldots, n$;
   c. it computes the updated share $\sigma'_{i,\ell}$ of message chunk m_ℓ for storage server S_i as $\sigma'_{i,\ell} := \sigma_{i,\ell} + r_{i,\ell}$, for $i = 1, \ldots, n$.
5. It aggregates N document chunk shares into a unique document share Sh_i as $Sh'_i = \sigma'_{i,1}||\sigma'_{i,2}|| \ldots ||\sigma'_{i,N}$.
6. It computes signature P'_i for document share Sh'_i with the private key sk_{T_1} as follows: $P'_i = sign(sk_{T_1}, Sh'_i)$, for $i = 1, \ldots, n$.
7. It distributes document share Sh'_i along with its signature P'_i to storage server S_i through a information-theoretically secure channel, for $i = 1, \ldots, n$.

Storage server S_i stores the new document share Sh'_i and its signature P'_i and deletes

the old Sh_i and P_i, for $i = 1, \ldots, n$.

Table 6.1: Summary of the notation used to define distributed storage systems based on a trusted execution environment.

n	total number of storage servers
S_i	storage server, for $i = 1, \ldots, n$
d	number of data owners
DO_j	data owner for $j = 1, \ldots, d$
id	The unique identifier ID of storage server S_i
t	reconstructing threshold of shares
M	document to be outsourced
m_ℓ	document chunk
B	size of document chunk m_ℓ
$N = \frac{\lvert M \rvert}{B}$	number of chunks per document
$\sigma_{i,\ell}$	share of chunk m_ℓ, held by storage server S_i
$Sh_i = \sigma_{i,1} \lvert\lvert \sigma_{i,2} \lvert\lvert \ldots \lvert\lvert \sigma_{i,N}$	document share, is a concatenation of the shares $\sigma_{i,\ell}$, for $\ell = 1, 2, \ldots, N$ held by storage server S_i for $i = 1, 2, \ldots, n$
P_i	proof of integrity of share Sh_i held by storage server S_i
$f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}$	polynomial of degree $t-1$
$a_0, a_1, \ldots, a_{t-1}$	coefficients of the polynomial from $\mathbb{F}_q$
c_k	commitment to coefficient a_k for $k = 0, 1, \ldots, t-1$
(pk_{T_1}, sk_{T_1})	Pair of public and secret keys owned by TEE T_1
$sign(sk, m)$	signature generation with secret key sk on message m
$ver(pk, P, Sh) \stackrel{?}{=} \text{true}$	verification of signature P on share Sh with public key pk

Reconstruct involves the data owner, the TEE, and the storage servers $S_1, \ldots, S_n$

and is initiated by the data owner, who requests document M from the TEE. The TEE performs the following steps.

1. It receives reconstruction request from the data owner.
2. It requests t document shares Sh_i from t storage servers, i.e., $i = 1, \ldots, t$.
3. It receives document share Sh_i and its signature P_i through an information-theoretically secure channel, for $i = 1, \ldots, t$.
4. It attests the integrity of document share Sh_i by verifying signature P_i as follows: $ver(pk_{T_1}, P_i, Sh_i) \stackrel{?}{=} \text{true}$. If share Sh_i is valid, it proceeds with step 5. Otherwise, it requests another document share Sh_j from a different storage server $S_{j \neq i}$ to reconstruct the original document, and notifies the service provider of the corruption of storage server S_i.
5. It uses shares $\sigma_{1,\ell}, \ldots, \sigma_{t,\ell}$ to reconstruct polynomial $f^*_\ell(x)$ using Lagrange interpolation, where $f^*_\ell(x)$ is the polynomial obtained by the sum of polynomial $f_\ell(x)$ used in share initialization Share and the polynomials $g_\ell(x)$ used for every round the share renewal protocol Renew was run. The ℓ-th document chunk m_ℓ is retrieved as $f^*_\ell(0) = m_\ell$, for $\ell = 1, \ldots N$.
6. It retrieves document M by concatenating the N document chunks m_ℓ as $M = m_1 || m_2 || \ldots || m_N$.
7. It sends M to data owner over an information-theoretically secure channel.

6.3.4 Alternative Renew Protocol

An alternative for protocol Renew presented in Section 6.3.3 would be to perform a combination of protocol Reconstruct and protocol Share (without the participation of the data owner). In this section, we describe this alternative and discuss its advantages and disadvantages. The protocol is initiated by the TEE, which performs the following steps.

1. It requests t document shares Sh_i from t storage servers S_i for $i = 1, \ldots, t$.
2. It receives document share Sh_i and its signature P_i through an information-theoretically secure channel, for $i = 1, \ldots, t$.
3. It attests the integrity of document share Sh_i by verifying its signature P_i as follows: $ver(pk_{T_1}, P_i, Sh_i) \stackrel{?}{=} \text{true}$. If share Sh_i is valid, then it proceeds with step 4. Otherwise, it requests another document share Sh_j from a different storage server $S_{j \neq i}$ to reconstruct the original document, and notifies the service provider of the corruption of storage server S_i.
4. For every $\ell = 1, 2, \ldots N$:
   a. it uses shares $\sigma_{1,\ell}, \ldots, \sigma_{t,\ell}$ to reconstruct polynomial $f^*_\ell(x)$ using Lagrange interpolation. The ℓ-th document chunk m_ℓ is retrieved as $f^*_\ell(0) = m_\ell$;
   b. it selects a polynomial $f'_\ell(x) = a'_{0,\ell} + a'_{1,\ell}x + \cdots + a'_{t-1,\ell}x^{t-1}$ where $a'_{0,\ell} := m_\ell$ and $a'_{1,\ell}, \ldots, a'_{t-1,\ell} \in \mathbb{F}_q$ are chosen uniformly at random;
   c. it computes share $\sigma'_{i,\ell} := f'_\ell(i)$ for storage server S_i.

5. It aggregates N document chunk shares into a unique document share Sh'_i as $Sh'_i = \sigma'_{i,1}||\sigma'_{i,2}|| \ldots ||\sigma'_{i,N}$.
6. It computes signature P'_i of document share Sh'_i with the private key sk_{T_1} as follows: $P'_i = sign(sk_{T_1}, Sh'_i)$, for $i = 1, \ldots, n$.
7. It distributes document share Sh'_i along with its signature P'_i to storage server S_i through an information-theoretically secure channel, for $i = 1, \ldots, n$.

The advantage of this protocol is that, in the best case scenario, when the storage servers are honest, t information-theoretically secure channel are sufficient to reconstruct the message, while in our original protocol, we require all shares to be sent from $S_1, S_2, \ldots, S_n$. However, we note that in the worst case scenario when $t-1$ servers are corrupted, the TEE might request all n storage servers through n information-theoretically secure channel. Moreover, our original protocol Renew requires less computation effort, since it does not rely on Lagrange interpolation for each message chunk for reconstructing the message. On the other hand, this alternative protocol requires in the best case (when the chosen t servers are honest), verification of $n-t$ less signatures, since the update is performed with the actual reconstruction of the secret. Moreover, in our original protocol, we detect servers trying to compromise the integrity of the shares by verifying all n signatures on the shares, while in this alternative protocol, we only verify t signatures and leave the other $n-t$ corruptions potentially undetected.

6.3.5 TEE Migration Protocol

Our share renewal protocol Renew allows for TEE migration, which is crucial for long-term secure storage systems. Besides, TEE migration is also required to mitigate Denial-of-Service (DoS) attacks or other attacks compromising the currently used TEE T_1 or its private key, in which case the storage service has to migrate to another TEE T_2. Moreover, migration allows for updating the security parameters of the signature scheme used, e.g., according to NIST recommendations [49]. In long-term secure storage systems, it has to be taken into account that signatures might be broken later. However, by keeping the security parameters up-to-date, the adversary does not gain advantage by forging signatures for an outdated security parameter. Protocol Renew can be modified to support seamless TEE migration in LSTee as follows. TEE T_2 requires its own public and private keys pk_{T_2} and sk_{T_2}, respectively, and the public key of TEE T_1, pk_{T_1} to verify the integrity of the shares integrity of step 3. To compute the new signatures, TEE T_2 uses its own private key and proceeds in the next update phase with its own public key.

The protocol augmented with TEE migration is initiated by TEE T_2, which performs the following steps.

1. It requests document share Sh_i from storage server S_i, for $i = 1, \ldots, n$.

2. It receives n document shares Sh_i and their signatures P_i sent by storage servers S_i through a information-theoretically secure channel, for $i = 1, \ldots, n$.
3. It attests the integrity document share Sh_i by verifying its signatures P_i as follows: $ver(pk_{T_1}, P_i, Sh_i) \stackrel{?}{=} \text{true}$, for $i = 1, \ldots, n$. If share Sh_i is valid, then it proceeds with step 4. Otherwise, it recovers the original document M from other t valid shares, notifies the service provider of the corruption of storage server S_i and executes Share to recompute the document share Sh_i.
4. For every $\ell = 1, \ldots N$:
   a. it selects a polynomial $g_\ell(x) = b_{0,\ell} + b_{1,\ell}x + \cdots + b_{t-1,\ell}x^{t-1}$, where $b_{0,\ell} = 0$ and coefficients $b_{1,\ell}, \ldots, b_{t-1,\ell} \in \mathbb{F}_q$ are chosen uniformly at random;
   b. it computes subshare $r_{i,\ell}$ for storage server S_i as $r_{i,\ell} := g_\ell(\text{id})$, for $i = 1, 2, \ldots, n$;
   c. it computes the updated share $\sigma'_{i,\ell} := \sigma_{i,\ell} + r_{i,\ell}$, for $i = 1, 2, \ldots, n$.
5. It aggregates N document chunk shares into a unique document share Sh'_i as $Sh'_i = \sigma'_{i,1}||\sigma'_{i,2}|| \ldots ||\sigma'_{i,N}$.
6. It computes signature P'_i of document share Sh'_i with the private key sk_{T_2} as follows: $P'_i = sign(sk_{T_2}, Sh'_i)$, for $i = 1, \ldots, n$.
7. It distributes document share Sh'_i along with signature P'_i to storage server S_i through a information-theoretically secure channel, for $i = 1, \ldots, n$.

Storage server S_i stores the new document share Sh'_i and its signature P'_i and deletes the old share Sh_i and signature P_i.

6.4 Security Analysis

We discuss next the security guarantees of LSTee with respect to our trust assumptions and the overall scenario in which LSTee is embedded as a service. More precisely, in Section 6.4.1 we argue why LSTee provides the same confidentiality and integrity protection guarantees as state of the art secret sharing-based distributed storage systems. In Section 6.4.2, we explain our assumptions with respect to the trustworthiness of the used TEE. In Section 6.4.3 and in Section 6.4.4, we discuss the security of, respectively, the signatures used for attestation and of the information-theoretic channels to be established. In Section 6.4.5, we discuss the overall service in which we envision LSTee to be embedded.

6.4.1 Security Guarantees

LSTee provides the same long-term confidentiality and integrity guarantees as state of the art secret sharing-based long-term secure storage systems with Pedersen's commitment, while significantly optimizing the computation overhead and the number of information-theoretically secure channel required. *Long-term confidentiality*

is provided by the deployment of proactive secret sharing, where shares are distributed to multiple storage servers and periodically renewed, thus mitigating a mobile but computationally bounded adversary. The shares' confidentiality during the execution of the Share, Renew, and Reconstruct is preserved because they are processed inside a TEE and communicated over information-theoretically secure channel (see Section 6.3.3). *Long-term integrity* is provided because shares computed during protocols Share and Renew are created and signed by the TEE. More precisely, (public-key) signature schemes are deployed to verify the integrity of these shares prior to protocols Renew and Reconstruct in the TEE. Share renewal allows LSTee to update the security parameters of the signature scheme according to NIST recommendations, thus providing long-term security guarantees. Furthermore, if up to $t-1$ shares are found to be corrupted, the TEE can still recover the original document, or recompute correct shares and redistribute them, thus preserving message integrity and providing robustness. In short, the main primitives LSTee deploys to provide the above guarantees for our optimized long-term secure storage system are: a TEE, (public-key) signature schemes, and information-theoretically secure channel, which we discuss in more detail next.

6.4.2 Trustworthiness of the TEE

As explained in Section 6.3.3, protocols Share, Renew, and Reconstruct are performed in LSTee by a TEE deployed in a remote computing server. For LSTee to cope with the same cryptographic adversary as the state of the art solution discussed in Section 6.2.1.2, further assumptions are made with respect to the trustworthiness of the TEE. More precisely, we made standard assumptions with respect to the trustworthiness of the TEE to guarantee that it provides the desirable state-of-the-art security guarantees described above. In particular, this ensures that algorithms Share, Renew, and Reconstruct are accessible and provide perfect security according to Definition 2.1. Before processing any confidential data in the TEE, the hardware-based trust anchor in the TEE operator attests the integrity of the TEE, by computing a cryptographic hash of its content (code and data) using the secret key of the trust anchor. If the TEE is in a trustworthy state (verified by the communicating entity), the TEE is initiated and information-theoretically secure channel are established with the other parties to run the protocols. LSTee relies on the TEE behaving honestly and performing the computations as mandated by our protocols. While leveraging the TEE allows us to achieve significant gains in terms of efficiency, the TEE, however, becomes a single point of failure in LSTee. If it is compromised or unavailable due to a deny of service attack for example, neither the confidentiality nor the integrity of the outsourced data can be guaranteed. To mitigate this, LSTee allows for a smooth migration to another trustworthy TEE while requiring only the public key of the originally used TEE (see Section 6.3.5). We stress that the TEE, in and of itself,

does not provide long-term security guarantees since TEEs are intended for isolated execution where the TEE memory and disk storage still require standard memory encryption. Therefore, naively storing the encrypted data without secret sharing in a TEE does not provide the desirable long-term security guarantees and would quickly outgrow the constrained TEE memory capacity.

6.4.3 Security of the Signature Scheme

In compliance with the NIST recommendation [49], LSTee uses computationally secure signature schemes to protect the integrity of the outsourced data and to detect compromised storage servers. Replacing (expensive) computationally secure commitment schemes with (affordable) computationally secure signature schemes does not lower the security guarantees. Furthermore, LSTee is not tied to a specific signature scheme. This means that when the signature scheme used becomes vulnerable to attacks run on quantum computers, LSTee can easily switch to a post-quantum signature scheme. The private key used for the signatures in LSTee is sealed in the TEE operator, i.e., encrypted with the TEE's secret key. However, it is also possible to avoid this sealing mechanism by generating a fresh pair of private-public keys at every occurrence of Renew as follows: the shares that the Renew outputs at a certain round are signed with the currently used private key. These signed shares are provisioned to the storage servers. On the next round of Renew, the signature of each share is first verified by the TEE with the counterpart public key before proceeding with the share update. However, the updated shares are then signed by the TEE using a new fresh private-public key pair generated at the beginning of Renew. In this case, only the public key needs to be sealed, while the private key is never stored or sealed in the TEE operator. To prevent replay attacks where a compromised storage server may send the TEE an old benign share and its signature, timestamps or version numbers are included in the signatures computed by the TEE at every round of Renew.

6.4.4 Information-Theoretic Secure Channels

Security of LSTee relies on information-theoretically secure point-to-point channels in the network (see Section 6.3), which is aligned with the state of the art solution discussed in Section 6.2.1.1. Such information-theoretically secure channel can be constructed by using QKD protocols, which require quantum channels between any two nodes or, alternatively, by using one-time pad with pre-distributed key material. Long-term secure distributed storage systems with information-theoretically secure channel have been realized recently in Japan [25]. However, they are still impractical and expensive to establish, especially because the nodes cannot be more than 100 km far from each other. The difficulty in establishing such channels remains a general

and fundamental limitation that long-term secure distributed storage systems face to provide their stringent security guarantees, is not specific to LSTee, and is out of scope for this work. Nevertheless, the merit of LSTee lies in significantly reducing the required number of such information-theoretically secure channel while still achieving the same security guarantees as state of the art approaches.

6.4.5 LSTee as Part of a Larger Service

We envision LSTee as a larger service whose aim is to help data owners safely store their outsourced data. Thus, no bugs are intentionally injected or undetected, and continuous TEE upgrades are assumed. Furthermore, reports providing precise analytics about how often the storage servers got corrupted or broke down are periodically issued to help data owners make informed decisions about which storage servers to include in their long-term secure storage system. The need for such a service providing guidance to data owners through rigorously-derived performance figures about storage serves in the cloud was also pointed out by NIST [63]. In case the data owner decides to replace corrupted or low-performance servers with newer storage servers, LSTee enables this seamlessly by simply computing the new shares in the TEE during Renew. Again, we neither need distributed computation of commitments nor a broadcast channel due to the trustworthiness of the TEE.

6.5 Comparison with Related Work

We first present related work relevant to the contributions of this chapter. Next, we compare the communication and computation of LSTee with that of existing proactive secret sharing protocols in synchronous networks in Sections 6.5.1 and 6.5.2, respectively.

Besides Shamir's secret sharing scheme (see Chapter 2.1), distributed storage systems can also be built over Tassa's hierarchical secret sharing schemes (see Chapter 2.2) where shares with different reconstruction capabilities are generated. In a distributed storage scenario, the more informative shares generated by polynomial evaluation (vs. less informative shares generated by polynomial derivative evaluation) are ideally distributed to the more reliable storage servers [90]. We recall from Chapter 3 that Tassa's hierarchical secret sharing schemes are suitable for the long-term protection of the confidentiality of outsourced data because shares can be renewed periodically, similar to Shamir's scheme.

Two main approaches exist for share renewal: those with *synchronous networks* and those with *asynchronous networks*. For *synchronous networks*, besides the seminal approach by Herzberg et al. [62] (see Section 6.2.2), Desmedt and Jajoda [42] proposed protocols that enable dynamically adding and removing storage servers.

However, verification was not possible, and thus inconsistent shares from corrupted storage servers could not be detected. Wong et al. [113] mitigated this by enabling the storage servers in their protocols to verify their new shares once they got distributed. However, it is assumed that all the storage servers are honest during the redistribution phase. Gupta and Gopinath improved on the work by Wong et al. in [57] by allowing a honest majority among the storage servers during the redistribution phase, by using Feldman's commitment scheme in their protocol. They proposed a revised protocol in [58] using Pedersen's commitment to achieve information-theoretic confidentiality of the outsourced data. Brendel and Demirel in [26] optimized the number of information-theoretically secure channel needed in [58] by clustering the storage servers into groups that can communicate securely only within the cluster.

In contrast to synchronous networks, proactive secret sharing in *asynchronous networks* does not have access to a common global clock. Hence, the initiation of share renewal cannot be synchronized among all nodes, as pointed out by Cachin et al. in [31]. They proposed a formal model for cryptosystems in asynchronous proactive networks by providing an abstract timer accessible to all storage servers, while assuming a mobile adversary that can corrupt up to less than a threshold of storage servers that are in the same local time. In [99], the time intervals in which the share renewal is performed are defined by the events of the protocol itself. Instead, Zhou et al. [114] proposed a conservative estimation of the time in which the share renewal is executed.

6.5.1 Communication

One of the most significant challenges in deploying long-term secure storage systems in practice is establishing information-theoretically secure channel between all relevant nodes. LSTee makes a considerable advancement in this direction by reducing the number of information-theoretically secure channel needed to carry out the long-term secure storage of outsourced data. We compare LSTee with state of the art approaches [57, 58, 62] plus Tassa's hierarchical secret sharing-based distributed storage systems presented in Chapter 3 that provide the same security guarantees, i.e. they assume an honest majority ($n \geq 2t - 1$), protect the integrity of the outsourced document (not only its confidentiality), and do not relax the security requirements of the point-to-point channels between nodes.

As shown in Table 6.2, LSTee significantly outperforms all previous protocols in terms of the number of information-theoretically secure channel needed within the long-term secure storage system. When d data owners use the same long-term secure storage system composed of n storage servers, for Share and Reconstruct, LSTee needs, respectively, $n + d$ and $t + d$ information-theoretically secure channel compared with the nd information-theoretically secure channel needed in state-of-the art approaches. That is because in LSTee the data owners do not communicate directly with the storage servers and, instead, they establish a information-theoretically secure channel

Protocol	Share	Renew	Reconstruct
[57, 58, 62] Ch. 3	nd	$\frac{t(t-1)}{2} + t(n-t)$	td
LSTee	$n+d$	n	$t+d$

Table 6.2: Amount of information-theoretically secure channel needed to establish a long-term secure storage system for d data owners, where n is the total number of storage servers and t is the reconstructing threshold.

Protocol	Share	Renew	Reconstruct
[57, 58, 62] Ch. 3	$2nN$ shares, tN comm.	$2tN$ shares, t^2N comm.	$2tN$ shares
LSTee	M, nN shares, n sign.	$2nN$ shares, $2n$ sign.	tN shares, t sign., M

Table 6.3: Comparison of the communication during Share, Renew, and Reconstruct, where M is the message to be outsourced, n is the total number of storage servers, N is the number of message chunks, t is the reconstructing threshold, "comm."stands for "commitments"and "sign."stands for "signatures".

with the TEE, which works as a mediator node. Only the TEE is connected to all the storage servers through n information-theoretically secure channel and this number clearly does not depend on the number of data owners (see Figure 6.1b). Furthermore, in LSTee, the n information-theoretically secure channel already established to run Share are sufficient to carry out the Renew.

In contrast to LSTee, all other approaches require additional channels between the storage servers to perform share renewal distributedly. Renew and Reconstruct are compared assuming the optimized version of the protocols, where only t storage servers are, respectively, actively generating and distributing randomness values and provisioning the data owner with the necessary number of reconstructing shares. The quantity $\frac{t(t-1)}{2} + t(n-t)$ is always larger than n, assuming a reasonable threshold $t \geq 2$ (otherwise secret sharing becomes useless). In case $n \gg t$, then $t(n-t) \geq n$, and in case $t \sim n$, then $\frac{t(t-1)}{2} \geq n$.

Besides the number of channels, the volume of traffic communicated during Share, Renew, and Reconstruct is compared in Table 6.3. More precisely, to protect integrity, the state of the art approaches require a broadcast channel where *commitments* to the coefficients of polynomials of degree $t-1$ are communicated. A broadcast channel is not needed in LSTee, instead, to protect integrity, only signatures on the generated document share for each of the n shares are sent through the information-theoretically secure channel. For large files, where the number of chunks N is larger than the number of storage servers n, fewer signatures than commitments are communicated to protect integrity. To protect confidentiality, in LSTee fewer shares than the state of the art approaches are sent through the information-theoretically secure channel during Share and Reconstruct. This is because Pedersen's commitment used by the state of the art approaches uses two polynomials instead of one. However, for LSTee

the document M must also be communicated because the data owner outsources the generation of the shares and the document retrieval to the TEE. During Renew, LSTee needs to communicate more shares than the state of the art approaches because first the old shares have to be sent to the TEE by all storage servers and then the renewed shares have to be distributed.

6.5.2 Computation

LSTee achieves a lower computation overhead compared to [57, 58, 62] and Chapter 3. More precisely, during Renew, LSTee requires that the TEE computes only one polynomial of degree $t-1$ from which the n randomness values are generated and summed to the corresponding old shares. Instead, t (or $2t$ for [58] due to the usage of Pedersen's commitment scheme) polynomial evaluations for the computation of tn (or $2tn$) randomness values have to be performed in [57, 58, 62] and Chapter 3 even for the optimized protocol where t servers generate randomness values. Furthermore, in LSTee only a signature $P'_i = sign(sk_{T_1}, Sh'_i)$ for each of the n renewed document shares $Sh'_1, Sh'_2, \ldots, Sh'_n$ are computed for integrity check, as opposed to the t^2N (or $2t^2N$ for [58]) commitments that are computed for *each chunk* in which the document is divided. We highlight that signatures are computationally feasible to compute and are commonly adopted, while it is still an open problem how commitment schemes can be efficiently implemented. Similarly for Share, LSTee requires nN shares to be computed through polynomial evaluations and n signatures $P_1, P_2, \ldots, P_n$ for integrity check. Instead, in all other approaches, nN (or $2nN$ for [58]) shares and additionally tnN (or $2tnN$ for [58]) commitments have to be computed.

6.6 Instantiation and Evaluation

In this section, we describe our instantiation of the TEE using Intel SGX, which is the only TEE implementation that can be used with common Intel processors and hardware and does not require a certificate from the manufacturer.

In our implementation, we used an open-source implementation of Shamir's secret sharing protocol by Penney[6] and used $\mathbb{F}_{2^8+1}$ as finite field. It implements the standard Share and Reconstruct and allows for specifying the number of servers n and the threshold t. We have modified this implementation to include the Renew for proactive secret sharing as described in Section 6.3.3. We also modified the code such that it uses the CPU's hardware random generator as its randomness source through the `rdrand` instruction. For the public-key signatures, we used the RSA implementation of OpenSSL 1.0.2g with 3072-bit keys. We embedded this program in an SGX enclave using the Graphene-SGX framework [109]. We highlight that,

[6] `https://github.com/fletcher/c-sss`

despite the RSA signature is not resistant to quantum attacks, LSTee is not tied to a specific signature scheme. Thus, when the RSA signature will become vulnerable, it can easily be replaced by a post-quantum signature scheme.

Performance. We measure the performance of all three protocols on Intel SGX, i.e., of Share, Renew and Reconstruct as described in Section 6.3.3. In our experiments, we vary the number of shares n and the reconstructing threshold t and report performance results for documents containing between 100 bytes and 30 MBytes of random data. We ran our tests on an Intel i7-7700 CPU clocked at 3.60 GHz, with 8 GB of RAM and Ubuntu 16.04.5 OS.

Figures 6.2, 6.3 and 6.4 show the time required to perform the three algorithms Share, Renew, and Reconstruct using different values for our parameters, namely $\{n = 3, t = 2\}$, $\{n = 5, t = 3\}$, $\{n = 7, t = 4\}$, $\{n = 9, t = 5\}$, and $\{n = 11, t = 6\}$.

In our experiment, the time required to perform the secret-sharing computations themselves scales linearly in our whole document size range. The time required to sign a document scales linearly as well, but there is also a constant-time component of approximately 2 ms. Similarly, verifying a signature has a linear component and a constant-time component of approximately 60 μs. Creating and renewing shares (Figures 6.2, 6.3) requires multiple signatures, so the time required does not increase significantly until 3 KB, while it increases linearly after 10 KB. For a 30 MB document, creating the shares takes between 21 s and 90 s depending on the parameters n and t; renewing the shares takes between 27 s and 109 s. Reconstructing the shares (Figure 6.4) only requires signatures verification, so it increases linearly in the whole range. Reconstructing the shares of a 30 MB document takes between 23 s and 128 s depending on the parameters.

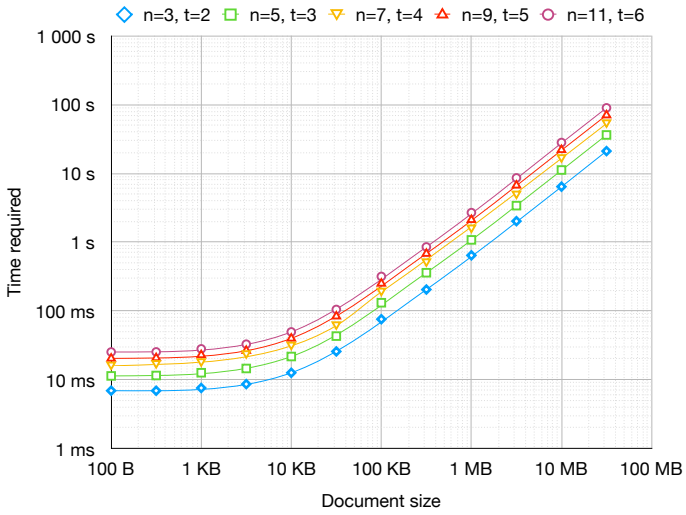


Figure 6.2: Time required to create a set of shares.

Our prototype implementation stores the whole document and the shares in enclave memory, which results in a bound on the size of the document in our experiments. Modifying the implementation to stream the document shares to the enclave eliminates this bound entirely. We can directly see that the computation time scales linearly, since the generation and verification of signatures contribute with only a linear overhead to the overall runtime of the protocols.

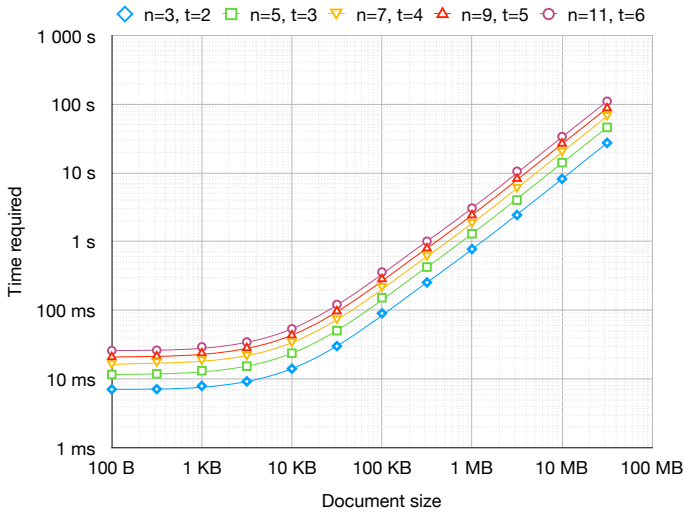


Figure 6.3: Time required to renew a set of shares.

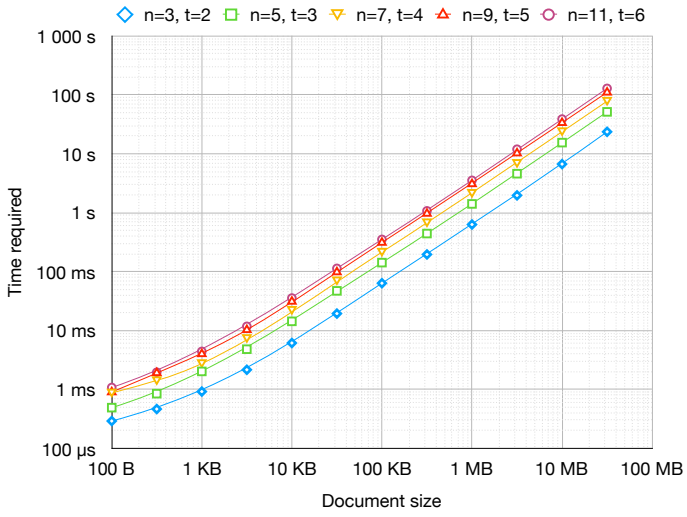


Figure 6.4: Time required to reconstruct a set of shares.

6.7 Summary and Future Work

Long-term secure protection of the confidentiality and integrity in secret sharing-based distributed storage systems can be carried out by proactive secret sharing in combination with verifiable secret sharing. This way, the validity of updated shares that are computed periodically to protect the confidentiality can be verified. This infrastructure relies on point-to-point information-theoretic secure communication channels between any two storage servers and the computation of commitment schemes.

In this chapter, we introduced LSTee, a TEE-based long-term secure distributed storage system that provides a significantly more efficient means to protect the confidentiality and the integrity of outsourced data compared to state of the art approaches. LSTee uses TEEs for the secure generation and update of shares of the outsourced data at regular time intervals. LSTee needs significantly fewer information-theoretically secure channel to securely carry out the share renewal protocol because information-theoretically secure channel need to be established only between each data owner and the TEE operator and between each storage server and the TEE. Furthermore, commitment schemes are no longer required because the validity of shares with respect to the original document is preserved by the TEE, while their integrity is protected by signatures. This guarantees that the updated shares computed by the TEE reconstruct the original data. These shares are then signed and distributed to the respective storage server through a information-theoretically secure channel. The signatures allow the TEE to check whether the storage server and the share were compromised before running again the renewing protocol. We validated LSTee by providing a proof-of-concept implementation and evaluated the computation runtimes of the protocols. Although our implementation is prototyped on Intel SGX, we showed why LSTee is TEE-agnostic, and any other TEE could be deployed. Moreover, migration from one TEE to another is supported in LSTee, thus enabling stronger security and robustness guarantees, in case a TEE instance is compromised or unavailable. LSTee is also independent of the underlying secret sharing primitive, e.g., Shamir's secret sharing can be replaced with Tassa's hierarchical secret sharing schemes.

As future work, we plan to address scenarios where the data not only long-term secure storage of data but also privacy-preserving computations on those data are needed (e.g. studies on genomic data). Secure multi-party computation offers a viable solution to the above mentioned scenarios and LSTee can be used for a more efficient instantiation of these protocols. We leave adapting LSTee to secure multi-party computation as future work.

7 Conclusion

In this thesis, we investigated secure storage solutions that can be adopted in practice by companies, hospitals, and institutions to protect the confidentiality and the integrity of their data in the long term. We evolved state of the art secret sharing-based distributed storage systems from theory to a more practical state. In order to do that, we focused our attention on the needs and wants of the end users so that we could improve on secret sharing-based distributed storage systems accordingly. Besides confidentiality and integrity protection guarantees, we individuated three main needs in terms of secure storage and, respectively, three main roadblocks that, if removed, would render secret sharing-based distributed storage systems a much feasible and affordable solution for the storage protection of data in real world scenarios. In the following, we present these needs and explain how in this thesis we contributed to meet them.

First, there is the need for storage solutions whose access to the outsourced data mirrors the rules and policies to access regular documents and files that are already in place within the company, the hospital, the institution. More precisely, the data owner outsourcing the data to a secret sharing-based distributed storage system is likely not an isolated entity, but is rather part of a larger hierarchical organization and has specific rights and powers. This hierarchical structure should be maintained by the storage system, which, in addition, should allow to flexibly modify the access rules of the underlying secret sharing scheme to address promotions, and employees that leave or join. In Chapter 3, we proposed a hierarchical and dynamic secret sharing-based distributed storage systems that fulfills all the above requirements and that, being backward compatible with proactive secret sharing and verifiable secret sharing, fulfills as well the security requirements in terms of long-term confidentiality and integrity. We did this by providing verifiable algorithms to add shareholders, reset the access rules, and to renew the shares for Tassa's conjunctive and disjunctive secret sharing schemes. Furthermore, we ensured the possibility to perform computations over such outsourced data (e.g. statistics on electronic health records for research-related purposes) by providing algorithms to perform linear operation and multiplications for Tassa's schemes. The algorithms for Tassa's hierarchical secret sharing schemes that we presented are equivalent with the corresponding algorithms

for Shamir's secret sharing scheme (used in state of the art approaches), both in term of security and in computational complexity, as we demonstrated.

Second, there is the need for guidance and support in both setting up and maintaining the long-term storage. This is due to the lack of clarity with respect of the quality of service of available storage servers and the lack of cryptographic expertise on the data owners' side. More precisely, secret sharing-based distributed storage systems are in practice composed of storage servers (i.e. the data centers) of multiple storage service providers. Up to now, there is no standardized and transparent means to test the quality of service of the storage service providers and of the storage servers they offer in the marketplace. In Chapter 5, we designed a performance scoring mechanism run by a trust authority, which enforces the storage servers to behave honestly and to submit accurate performance ratings with respect to storage servers owned by competitor storage service providers. We formalized for the first time the rationality of the storage service providers and their preferences in terms of being assigned to a high or low aggregate score and treated this problem as an economic game. By leveraging this through incentives and penalties in terms of the final aggregate scores, our performance scoring mechanism outputs accurate scores that reflect the actual performance of the storage servers. Furthermore, in Chapter 4, we introduced AS^3, an adaptive social secret sharing scheme that combines the need for hierarchical rules to access the outsourced data and the need for guidance in setting up and maintaining the storage system. More precisely, given the aggregate scores of the storage servers, AS^3 supports data owners in choosing the best parameters of the underlying hierarchical secret sharing scheme. Also, throughout the life of the storage systems, it alerts data owners with warnings whenever the performance of the storage servers is not satisfying and there is the risk to either break confidentiality of the data or not being able to retrieve them.

Third, there is the need for efficient and affordable storage solutions that scale. More precisely, state of the art approaches need point-to-point information-theoretically secure channel between any two storage servers in order to periodically renew the shares without leaking any information about them. Establishing these channels is either not practical (i.e. using pre-distributed one-time pad key material) or very expensive (i.e. using quantum key distribution protocols). Thus, configurations reducing the amount of such channels while stile guaranteeing confidentiality are desirable to contain the costs of such distributed storage system infrastructure. Furthermore, the cryptographic primitives used to protect the integrity of the outsourced data over time are commitment schemes, which are computationally intensive so that they cannot be used for large files. In Chapter 4, we introduced the first third-party audit mechanism able to check the validity of the shares stored across the storage servers of a distributed storage system while preserving their confidentiality in an information-theoretic manner. Since the mechanism enables the audit of multiple

data at once to lower communication complexity and does not use intensive computations, it is a valuable solution for companies, hospitals, and institutions for outsourcing the integrity check of their data. In Chapter 6, we took a further step towards a more efficient storage solution with low communication complexity by presenting LSTee, a long-term secure distributed storage system based on a trusted execution environment. LSTee uses a trusted execution environment for the secure and correct computation of shares, both during the generation phase and during the periodical renewing phases, as well as for the reconstruction of the originally outsourced data. This enables to establish a distributed storage system with significantly less information-theoretically secure channel than state of the art approaches. Furthermore, integrity of the the outsourced data is provided through signature schemes and because the trusted execution environment never deviates from the protocol it is supposed to run and, thus, always computes valid shares. Moreover, LSTee is not tied to a specific trusted execution environment and can migrate to a new one in case this gets compromised.

Future Work

As future work, we plan to take further steps towards a practical deployment of long-term secure storage solutions based on secret sharing. More precisely, we plan to instantiate LSTee based on AS^3 so that it addresses hierarchical scenarios and distributes the shares to high-performing storage servers. We want to use trusted execution environments also to perform secure multi-party computations. For scenarios where the renewal of the shares is rarely performed, we want the trusted execution environment to take over also the third party audit mechanism, so that integrity can be verified more regularly. As for supporting data owners with accurate performance figures, we want to strengthen our performance scoring mechanism so that it can cope with rational storage service providers that may decide to submit inaccurate ratings with a certain probability only, so as to lower detection likelihood.

Bibliography

[1] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information System Security*, 13, 2009.

[2] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern. Distributed Computing Meets Game Theory: Robust Mechanisms for Rational Secret Sharing and Multiparty Computation. In *PODC*, pages 53–62. ACM, 2006.

[3] M. Agarwal and R. Mehr. Review of matrix decomposition techniques for signal processing applications. *Int. Journal of Engineering Research and Applications*, 2014.

[4] ARM. Security technology building a secure system using TrustZone technology (white paper). *ARM Limited*, 2009.

[5] G. Asharov and Y. Lindell. Utility Dependence in Correct and Fair Rational Secret Sharing. In *CRYPTO*, volume 5677 of *LNCS*, pages 559–576. Springer, 2009.

[6] M. Backes, A. Kate, and A. Patra. Computational verifiable secret sharing revisited. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security*, pages 590–609, 2011.

[7] B. Barak, Y. Lindell, and S. Vadhan. Lower bounds for non-black-box zero knowledge. *Journal of Computer and System Sciences*, 72(2):321–391, 2006.

[8] J. Baron, K. E. Defrawy, J. Lampkins, and R. Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015*, pages 23–41, 2015.

[9] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference*, pages 420–432, 1991.

[10] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *CRYPTO '92*, pages 390–420. Springer, 1993.

[11] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions. In *STOC '88*, pages 113–131. ACM, 1988.

[12] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC*, pages 1–10. ACM, 1988.

[13] F. Benhamouda, S. Krenn, V. Lyubashevsky, and K. Pietrzak. Efficient Zero-Knowledge Proofs for Commitments from Learning with Errors over Rings. In *ESORICS '15, Part I*, volume 9326 of *LNCS*, pages 305–325. Springer, 2015.

[14] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theor. Comput. Sci.*, 560(P1):7–11, 2014.

[15] I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, and A. Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. *IACR Cryptology ePrint Archive*, (2017/1153), 2017.

[16] C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[17] C. Blundo, A. Cresti, A. D. Santis, and U. Vaccaro. Fully dynamic secret sharing schemes. In *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference*, pages 110–125, 1993.

[18] W. M. Bolstad. *Introduction to Bayesian Statistics*. John Wiley & Sons, Inc, 2004.

[19] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference*, pages 213–229, 2001.

[20] K. D. Bowers, A. Juels, and A. Oprea. HAIL: A High-Availability and Integrity Layer for Cloud Storage. In *CCS '09*, pages 187–198. ACM, 2009.

[21] F. Brasser, S. Capkun, A. Dmitrienko, T. Frassetto, K. Kostiainen, U. Müller, and A. Sadeghi. DR.SGX: hardening SGX enclaves against cache attacks with data location randomization. *CoRR*, abs/1709.09917, 2017.

[22] F. Brasser, T. Frassetto, K. Riedhammer, A.-R. Sadeghi, T. Schneider, and C. Weinert. Voiceguard: Secure and private speech processing. *Interspeech*, pages 1303–1307, 2018.

[23] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi. Software grand exposure: SGX cache attacks are practical. In *WOOT*. USENIX Association, 2017.

[24] J. Braun, J. Buchmann, D. Demirel, M. Geihs, M. Fujiwara, S. Moriai, M. Sasaki, and A. Waseda. Lincos: A storage system providing long-term integrity, authenticity, and confidentiality. In *Asia Conference on Computer and Communications Security (AsiaCCS'17)*, pages 461–468. ACM, 2017.

[25] J. Braun, J. Buchmann, C. Mullan, and A. Wiesmaier. Long Term Confidentiality: a Survey. *Designs, Codes and Cryptography*, 71(3):459–478, 2014.

[26] J. Brendel and D. Demirel. Efficient proactive secret sharing. In *Privacy, Security and Trust (PST'16)*, pages 543–550. IEEE, 2016.

[27] E. F. Brickell. Some ideal secret sharing schemes. In *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of Cryptographic Techniques*, pages 468–475, 1989.

[28] S. Buchegger and J.-Y. Le Boudec. A robust reputation system for peer-to-peer and mobile ad-hoc networks. In *P2PEcon 2004*, number LCA-CONF-2004-009, 2004.

[29] J. Buchmann, A. May, and U. Vollmer. Perspectives for Cryptographic Long-Term Security. *Communications of the ACM*, 49(9):50–55, 2006.

[30] N. Burow, S. A. Carr, S. Brunthaler, M. Payer, J. Nash, P. Larsen, and M. Franz. Control-flow integrity: Precision, security, and performance. *CoRR*, 2016.

[31] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *ACM Conference on Computer and Communications Cecurity (CCS'02)*, pages 88–97. ACM, 2002.

[32] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *STOC'88*.

[33] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai. Sgxpectre attacks: Leaking enclave secrets via speculative execution. *arXiv preprint arXiv:1802.09085*, 2018.

[34] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS 1985*.

[35] V. Costan, I. A. Lebedev, and S. Devadas. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium*, 2016.

[36] R. Cramer, I. Damgård, and U. M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques*, pages 316–334, 2000.

[37] R. Curtmola, O. Khan, R. C. Burns, and G. Ateniese. MR-PDP: Multiple-Replica Provable Data Possession. In *ICDCS '08*, pages 411–420, 2008.

[38] D-Wave. Announcing the d-wave 2x quantum computer. `https://www.dwavesys.com/blog/2015/08/announcing-d-wave-2x-quantum-computer/`, 2015.

[39] I. Damgård. On Σ-Protocols. Lecture on Cryptologic Protocol Theory; Faculty of Science, University of Aarhus, 2010.

[40] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference*, pages 572–590, 2007.

[41] L. Davi, A. Dmitrienko, S. Nürnberger, and A. Sadeghi. Gadge me if you can: secure and efficient ad-hoc instruction-level randomization for x86 and ARM. In *AsiaCCS*, pages 299–310. ACM, 2013.

[42] Y. Desmedt and S. Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, Technical Report ISSE TR-97-01, George Mason University, 1997.

[43] M. C. Doganay, T. B. Pedersen, Y. Saygin, E. Savaş, and A. Levi. Distributed privacy preserving k-means clustering with additive secret sharing. In *Proceedings of the 2008 international workshop on Privacy and anonymity in information society, PAIS*, 2008.

[44] A. K. Ekert. Quantum cryptography based on bell's theorem. *Physical review letters*, 67(6):661, 1991.

[45] M. Erwin. The latest rules on how long NSA can keep americans' encrypted data look too familiar. `https://www.justsecurity.org/19308/`, 2015.

[46] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, 1987*, pages 427–437, 1987.

[47] M. Fitzi, J. A. Garay, S. Gollakota, C. P. Rangan, and K. Srinathan. Round-optimal and efficient verifiable secret sharing. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, pages 329–342, 2006.

[48] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology – CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, 1984.

[49] M. D. Garris, J. L. Blue, G. T. Candela, P. J. Grother, S. Janet, and C. L. Wilson. NIST form-based handprint recognition system (release 2.0). *Interagency/Internal Report (NISTIR) - 5959*, 1997.

[50] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The round complexity of verifiable secret sharing and secure multicast. In J. S. Vitter, P. G. Spirakis, and M. Yannakakis, editors, *STOC*, pages 580–589. ACM, 2001.

[51] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fact-track multiparty computations with applications to threshold cryptography. In *PODC'98*.

[52] H. Ghodosi, J. Pieprzyk, and R. Safavi-Naini. Secret sharing in multilevel and compartmented groups. In *Information Security and Privacy, Third Australasian Conference, ACISP'98*, pages 367–378, 1998.

[53] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*, pages 218–229. ACM, 1987.

[54] S. D. Gordon and J. Katz. Rational Secret Sharing, Revisited. In *SCN*, volume 4116 of *LNCS*, pages 229–241. Springer, 2006.

[55] C. Gritti, W. Susilo, and T. Plantard. Efficient Dynamic Provable Data Possession with Public Verifiability and Data Privacy. In *ACISP '15*, volume 9144 of *LNCS*, pages 395–412. Springer, 2015.

[56] C. Gritti, W. Susilo, T. Plantard, and R. Chen. Improvements on efficient dynamic provable data possession scheme with public verifiability and data privacy. Cryptology ePrint Archive, Report 2015/645, 2015. `http://eprint.iacr.org/`.

[57] V. H. Gupta and K. Gopinath. An extended verifiable secret redistribution protocol for archival systems. In *Availability, Reliability and Security (ARES'06)*, pages 8–pp. IEEE, 2006.

[58] V. H. Gupta and K. Gopinath. G_{its}^2 VSR: An Information Theoretical Secure Verifiable Secret Redistribution Protocol for Long-term Archival Storage. In *SISW*, pages 22–33, 2007.

[59] J. Y. Halpern and V. Teague. Rational Secret Sharing and Multiparty Computation: Extended Abstract. In *STOC*, pages 623–632. ACM, 2004.

[60] C.-W. Hang and M. P. Singh. Trustworthy service selection and composition. *ACM Trans. Auton. Adapt. Syst.*, 6(1):5:1–5:17, Feb. 2011.

[61] J. Heather and D. Lundin. The append-only web bulletin board. In *FAST 2008*.

[62] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Secret Sharing Or: How to Cope With Perpetual Leakage. In *CRYPTO*, pages 339–352. 1995.

[63] M. Hogan, F. Liu, A. Sokol, and J. Tong. NIST Cloud Computing Standards Roadmap (NIST-SP 500-291). *NIST Special Publication*, 35, 2011.

[64] Intel. Intel Software Guard Extensions (Intel SGX). `https://software.intel.com/en-us/sgx/details`.

[65] A. Josang and R. Ismail. The beta reputation system. In *Proceedings of the 15th bled electronic commerce conference*, volume 5, pages 2502–2511, 2002.

[66] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision support systems*, 43(2):618–644, 2007.

[67] R. Jurca and B. Faltings. An Incentive Compatible Reputation Mechanism. In *CEC*, pages 285–292. IEEE, 2003.

[68] R. Jurca, F. Garcin, A. Talwar, and B. Faltings. Reporting Incentives and Biases in Online Review Forums. *TWEB*, 4(2):5, 2010.

[69] E. Käsper, V. Nikov, and S. Nikova. Strongly multiplicative hierarchical threshold secret sharing. In *ICITS 2007*.

[70] J. Katz, C. Koo, and R. Kumaresan. Improving the round complexity of VSS in point-to-point networks. *Inf. Comput.*, 2009.

[71] G. Kol and M. Naor. Cryptography and Game Theory: Designing Protocols for Exchanging Information. In *TCC*, volume 4948 of *LNCS*, pages 320–339. Springer, 2008.

[72] S. C. Kothari. Generalized linear threshold scheme. In *Advances in Cryptology*, pages 231–241. Springer, 1985.

[73] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz. Sok: Automated software diversity. In *IEEE Symposium on Security and Privacy*, pages 276–291. IEEE Computer Society, 2014.

[74] S. L. Lauritzen. *Graphical models*, volume 17. Clarendon Press, 1996.

[75] H. W. Lauw, E.-P. Lim, and K. Wang. Bias and controversy: Beyond the statistical deviation. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 625–630, New York, NY, USA, 2006. ACM.

[76] X. Liu, A. Datta, K. Rzadca, and E.-P. Lim. Stereotrust: A group based personalized trust model. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 7–16, New York, NY, USA, 2009. ACM.

[77] X. Liu, G. Tredan, and A. Datta. A generic trust framework for large-scale open systems using machine learning. *Computational Intelligence*, 30(4):700–721, 2014.

[78] A. Lysyanskaya and N. Triandopoulos. Rationality and Adversarial Behavior in Multi-party Computation. In *CRYPTO*, volume 4117 of *LNCS*, pages 180–197. Springer, 2006.

[79] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[80] N. Miller, P. Resnick, and R. Zeckhauser. Eliciting Informative Feedback: The Peer-Prediction Method. *Management Science*, 51(9):1359–1373, 2005.

[81] M. E. G. Moe, B. E. Helvik, and S. J. Knapskog. *Comparison of the Beta and the Hidden Markov Models of Trust in Dynamic Environments*, pages 283–297. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[82] A. W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 397–405. Morgan Kaufmann Publishers Inc., 2000.

[83] M. Nielsen, K. Krukow, and V. Sassone. A bayesian model for event-based trust. *Electronic Notes in Theoretical Computer Science*, 172:499–521, 2007.

[84] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part i. *Theoretical Computer Science*, 13(1):85 – 108, 1981.

[85] C. NIST. The digital signature standard. *Communications of the ACM*, 35(7):36–40, 1992.

[86] M. Nojoumian and T. C. Lethbridge. A new approach for the trust calculation in social networks. In *E-business and Telecommunication Networks: 3rd International Conference on E-Business*, volume 9 of *CCIS*, pages 64–77. Springer, 2008.

[87] M. Nojoumian and D. R. Stinson. Brief announcement: secret sharing based on the social behaviors of players. In A. W. Richa and R. Guerraoui, editors, *ACM PODC*, pages 239–240. ACM, 2010.

[88] M. Nojoumian and D. R. Stinson. Social secret sharing in cloud computing using a new trust function. In N. Cuppens-Boulahia, P. Fong, J. García-Alfaro, S. Marsh, and J. Steghöfer, editors, *PST*, pages 161–167. IEEE Computer Society, 2012.

[89] M. Nojoumian and D. R. Stinson. Socio-Rational Secret Sharing as a New Direction in Rational Cryptography. In *GameSec*, volume 7638 of *LNCS*, pages 18–37. Springer, 2012.

[90] M. Nojoumian, D. R. Stinson, and M. Grainger. Unconditionally secure social secret sharing scheme. *IET Information Security*, 4(4):202–211, 2010.

[91] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[92] N. Pakniat, Z. Eslami, and M. Nojoumian. Ideal Social Secret Sharing Using Birkhoff Interpolation Method. *IACR Cryptology ePrint Archive*, 2014:515, 2014.

[93] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO'91*.

[94] V. Ramasubramanian and K. Paliwal. A generalized optimization of the kd tree for fast nearest-neighbour search. In *TENCON'89. Fourth IEEE Region 10 International Conference*, pages 565–568. IEEE, 1989.

[95] S. Ries. Extending bayesian trust models regarding context-dependence and user friendly representation. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA*, pages 1294–1301, 2009.

[96] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[97] V. Scarani, H. Bechmann-Pasquinucci, N. J. Cerf, M. Dušek, N. Lütkenhaus, and M. Peev. The security of practical quantum key distribution. *Reviews of modern physics*, 81(3):1301, 2009.

[98] L. Schabhüser, D. Demirel, and J. A. Buchmann. An unconditionally hiding auditing procedure for computations over distributed data. In *2016 IEEE Conference on Communications and Network Security, CNS 2016*, pages 552–560, 2016.

[99] D. A. Schultz, B. Liskov, and M. Liskov. Mobile proactive secret sharing. In *ACM Symposium on Principles of Distributed Computing (PODC'08)*, page 458. ACM, 2008.

[100] D. A. Schultz, B. Liskov, and M. Liskov. MPSS: mobile proactive secret sharing. *ACM Trans. Inf. Syst. Secur.*, 2010.

[101] T. S. J. Schwarz and E. L. Miller. Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage. In *ICDCS 2006*, page 12, 2006.

[102] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[103] M. Shih, S. Lee, T. Kim, and M. Peinado. T-SGX: eradicating controlled-channel attacks against enclave programs. In *NDSS*. The Internet Society, 2017.

[104] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comp.*, 26(5):1484–1509, 1997.

[105] G. J. Simmons. How to (really) share a secret. In *Proceedings on Advances in cryptology*, pages 390–448. Springer-Verlag New York, Inc., 1990.

[106] J. Tang, H. Gao, H. Liu, and A. Das Sarma. etrust: Understanding trust evolution in an online world. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 253–261, New York, NY, USA, 2012. ACM.

[107] T. Tassa. Hierarchical threshold secret sharing. *J. Cryptology*, 20(2):237–264, 2007.

[108] L. Trailovic and L. Y. Pao. Variance estimation and ranking of gaussian mixture distributions in target tracking applications. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 2, pages 2195–2201. IEEE, 2002.

[109] C. Tsai, D. E. Porter, and M. Vij. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *USENIX Annual Technical Conference (USENIX ATC)*. USENIX, 2017.

[110] J. Van Bulck, F. Piessens, and R. Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018.

[111] C. Wang, Q. Wang, K. Ren, and W. Lou. Ensuring Data Storage Security in Cloud Computing. In *IWQoS '09*, Los Alamitos, CA, USA, 2009. IEEE.

[112] T. Watson et al. A programmable two-qubit quantum processor in silicon. *Nature*, 2018.

[113] T. M. Wong, C. Wang, and J. M. Wing. Verifiable secret redistribution for archive systems. In *Security in Storage Workshop*, pages 94–105. IEEE, 2002.

[114] L. Zhou, F. B. Schneider, and R. Van Renesse. Apss: Proactive secret sharing in asynchronous systems. *ACM transactions on information and system security (TISSEC)*, 8(3):259–286, 2005.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Darmstadt, March 2019 _____

Wissenschaftlicher Werdegang

Februar 2015 - heute

Wissenschaftliche Mitarbeiterin in der Arbeitsgruppe von Prof. Dr. Johannes Buchmann, Fachbereich Informatik, Fachgebiet Theoretische Informatik – Kryptographie und Computeralgebra an der Technischen Universität Darmstadt.

September 2012 - Oktober 2014

Studium im Studiengang Mathematics (Master of Science) an der Università degli Studi di Trento (Italien).

September 2009 - September 2012

Studium im Studiengang Mathematics (Bachelor of Science) an der Università degli Studi di Trento (Italien).