Harpocrates: Privacy-Preserving and Immutable Audit Log for Sensitive Data Operations

Mohit B. Thazhath

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Thang Hoang, Chair
Danfeng Yao
Kirk W. Cameron

May 9, 2022
Blacksburg, Virginia

Keywords: Blockchain; Immutable Logging; Record Anonymity; Zero-Knowledge Proofs

Harpocrates: Privacy-Preserving and Immutable Audit Log for Sensitive Data Operations

Mohit B. Thazhath

(ABSTRACT)

The immutability, validity and confidentiality of an audit log is crucial when operating over sensitive data to comply to standard data regulations (e.g., HIPAA). Despite its critical needs, state-of-the-art privacy-preserving audit log schemes (e.g., Ghostor (NSDI '20), Calypso (VLDB '19)) do not fully obtain a high level of privacy, integrity, and immutability simultaneously, in which certain information (e.g., user identities) is still leaked in the log.

In this work, we propose Harpocrates, a new privacy-preserving and immutable audit log scheme. Harpocrates permits data store, share, and access operations to be recorded in the audit log without leaking sensitive information (e.g., data identifier, user identity), while permitting the validity of data operations to be publicly verifiable. Harpocrates makes use of blockchain techniques to achieve immutability and avoid a single point of failure, while cryptographic zero-knowledge proofs are harnessed for confidentiality and public verifiability. We analyze the security of our proposed technique and prove that it achieves non-malleability and indistinguishability. We fully implemented Harpocrates and evaluated its performance on a real blockchain system (i.e., Hyperledger Fabric) deployed on a commodity platform (i.e., Amazon EC2). Experimental results demonstrated that Harpocrates is highly scalable and achieves practical performance.

Harpocrates: Privacy-Preserving and Immutable Audit Log for Sensitive Data Operations

Mohit B. Thazhath

(GENERAL AUDIENCE ABSTRACT)

Audit logs are an essential part of data storage systems as they allow to check if the system is working as intended. They are usually maintained on a server, a server with ill intentions can easily modify records of the log and make it appear that the system is working correctly. To store these records in an un-modifiable manner, prior works have leveraged special audit log storing mechanisms for e.g., blockchain due to its immutable nature. However, these works do not focus on the privacy of the records which is a crucial aspect for conforming to certain data regulations like HIPAA.

In this work, we propose Harpocrates, an immutable and privacy-preserving audit log platform that supports recording operations (share/access) on sensitive data. Harpocrates leverages blockchain to achieve immutability of the audit log. Harpocrates use specific cryptographic primitives to achieve public verifiability and confidentiality of the audit log. Real world deployment of Harpocrates shows that it is practical and achieves strong security guarantees.

Acknowledgments

I would like to thank Dr. Thang Hoang for his invaluable help throughout my graduate school journey. His active involvement in this work and patience to explain complex cryptographic concepts were reassuring during challenging times. I would also like to thank my committee members Dr. Danfeng Yao and Dr. Kirk Cameron for their feedback and comments on this work. Finally, I would like to thank my family and friends who were supportive and accommodative to my needs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Remote data storage systems have become predominant in the past decade due to the growth of cloud computing. Cloud Service Providers (CSPs) have dedicated resources to store data, and have developed a suite of services to support it such as AWS S3, Azure Blobs, GCP Buckets. The widespread use of cloud data storage has made it easy to share/access remote data.

Given that CSPs generally maintain a vast amount of user data, many of which can be highly sensitive (e.g., personal, health), it is vital to maintain an audit log that records all the data operations (e.g., sharing, access activities) to support system integrity inspection and critical security assurances such as intrusion detection and problem analysis [46]. Many data audit log schemes have been proposed to record fine-grained operations (e.g., read, write, share) over sensitive data (e.g., Electronic Health Records (EHRs) [15, 24], supply chains [22, 36, 38, 41]) in a tamper-resistant and immutable manner.

Despite their merits, achieving the immutability and integrity of the audit log may not be sufficient to ensure security against active threats. This is because the audit log contains all data operations (i.e., metadata), which can reveal significant sensitive information. By analyzing the data log, the adversary can obtain the needed information without having to access the actual data [47]. Such metadata leakage has become notorious in the related area such as communication surveillance as a former NSA General Counsel generally said, "*Metadata absolutely tells you everything about somebody's life*" [32]. Due to the sensitiveness

of metadata, it is therefore vital to ensure not only the integrity but also the confidentiality of audit logs as indicated in standard data regulations such as The Health Insurance Portability and Accountability Act of 1996 (HIPAA) [42].

To enable both privacy and integrity, several privacy-preserving data audit log schemes have been proposed. Preliminary constructions permit data operations to be logged in a single server with privacy and integrity guarantees using cryptographic tools such as digital signatures and symmetric encryption [26, 31]. Despite their merits, such centralized approaches suffer from a single point of failure, in which the corrupted server can compromise the validity and confidentiality of the audit log. To address single point of failure, several decentralized privacy-preserving audit log approaches have been proposed using blockchain techniques [7, 18, 21, 23, 27, 35, 44]. However, there are certain limitations to these techniques. For example, Droplet [35] offers confidentiality but not anonymity. Ghostor [18] records data sharing activities with anonymity, but the validity can only be verified privately by the data owner. This requires all the users to participate in the audit process to verify the validity of the data operations, thereby reducing the audit transparency. Calypso [21] achieves partial anonymity (i.e., leaks data owner identity) and private verifiability.

Research gaps. Given that existing data logging techniques lack a certain degree of anonymity, integrity, confidentiality and transparency, our objective is to design a new privacy-preserving audit log scheme that can offer all desirable security properties for standard data regulations compliance.

Contribution. In this paper, we propose Harpocrates, a new privacy-preserving and immutable audit log scheme for sensitive data operations. Harpocrates permits data operations (i.e., store, share, access) to be recorded in the audit log with validity, confidentiality, and public verifiability guarantees. Harpocrates makes use of blockchain technologies (e.g., distributed ledger) to achieve immutability and validity, while the confidentiality and public

verifiability are achieved using advanced cryptographic techniques such as zero-knowledge proofs [10].

Table 1.1: Comparison of Harpocrates with prior works.

Scheme	Audit Log Confidentiality	Audit Log Anonymity	Temporal Access	Validity Verifiability		
				Store Record	Share Record	Access Record
Calypso [21]	✓	Partial†	✗	Private	Private	Private
Ghostor [18]	✓	✓	✗	Private	Private	Private
Droplet [35]	✓	✗†	✗	Not Verifiable	Public	Not Verifiable
Harpocrates	✓	✓	✓	Public	Public	Public

† Calypso and Droplet does not hide data owner identity.

Harpocrates achieves the following properties.

- *Record Immutability*: Meaning all data operation records cannot be modified by anyone.
- *Full Anonymity*: The identity of the data and the user(s) remain hidden all the time. This security guarantee is stronger than existing works (e.g., [21, 35]).
- *Publicly Verifiable Validity*: In Harpocrates, the validity of the data operations (e.g., whether the share/access is performed by an authorized user) can be publicly verified by anyone. This improves audit transparency such that all the data users do not need to participate during the audit process.
- *Temporal Access Control*: Harpocrates supports temporal access control, which restricts the time for which the data can be accessed. This is useful when operating on highly sensitive data like EHRs.

Table 1.1 compares Harpocrates with other techniques. We analyze the security of our proposed technique and prove that it achieves standard security notions (non-malleability, indistinguishability). Finally, we implemented our technique and deployed it on Amazon EC2 to evaluate its efficiency. Experimental results showed that Harpocrates is highly scalable (§8) and practical. Therefore, it can be used for sensitive data storage applications that require privacy-preserving audit log such as EHRs [23, 44] or supply chains [7, 27].

Chapter 2

Literature Review

2.1 Prior Works

Centralized Audit Log. Traditional data sharing/access schemes record data operations directly on the server that stores the data [39, 45]. Such records are needed for audit purposes, where a third-party auditor determines whether any unauthorized access have been made in the system [40, 46]. However, with the introduction of regulations like HIPAA a *secure logging* mechanism [42] is needed. Secure logging primitives [31, 37] rely on a trusted author whose signature is considered the root of trust for integrity or requires analyzing the entirety of the audit log to detect tampering due to the privacy-preserving nature of the records. Alternatively, prior works [19, 26] use secure hardware to achieve immutable and privacy-preserving records. Since the audit log is maintained by the centralized server, it is vulnerable to a single point of failure, where the corrupted server can compromise the confidentiality and integrity of the audit log.

Decentralized Audit Log. Several works have proposed a decentralized audit log to address the single point of failure vulnerabilities. Decentralized audit log schemes were designed for medical data [15, 44], supply chain [7, 27], and financial applications [2, 8]. Some schemes permit tamper resistant and access control to data objects being logged [12, 30]. However, these approaches do not hide the user identities (e.g., sender/receiver). Several

works attempted to achieve either full anonymity [18] or partially anonymous [21]. There are several schemes that permit the validity of audit log to be verified publicly [35] or privately [18, 21]. Private validity verifiability can be detrimental during an audit because every user in the system will have to participate and prove the integrity of their records.

2.2 Use Cases

2.2.1 E-Health Data Sharing

Medical information sharing without violating patient rights has been a long studied topic. Laws such as The Health Insurance Portability and Accountability Act of 1996 (HIPAA) prohibit patient records from being disclosed without the patient's consent or knowledge. Compliance with HIPAA is a major concern for medical organizations as well as cloud data service providers. Over the years, many medical organizations have relied on cloud storage for their patient data. However, this has led to a number of privacy and security concerns through the loss of data control to the patients [25]. Present day research has employed blockchain as an access control mechanism for medical records which in turn give more control to the patients of their data. A lot of such works [13, 15, 23, 24, 44] do not address the anonymity concerns of patients and employ public blockchains for access control over medical data. This can be a concern for patients as it makes them vulnerable to data inference attacks. For example, a targeted marketing campaign could obtain information of all patients who share their records to cardiologists and serve advertisements related to heart diseases. Although blockchain systems have provided great amount of access control to the patients, their identities are still not protected. This calls for the need of an anonymized e-health data sharing and logging system.

2.2.2 Supply Chain

The supply chain industry is growing in complexity as multiple stakeholders want more transparency. A transparent supply chain helps hold malicious stakeholders accountable for their actions and helps establish trust in the supply chain. Many researchers have leveraged the use of blockchain to store and share supply chain related data [22, 36, 38, 41]. However, transparency in supply chain is not always possible and can be prohibited by law. The Federal Acquisition Supply Chain Security Act of 2018 (FASCSA) gives the power to federal agencies to issue removal orders to their suppliers [43]. In other words, supplier databases are modified to remove information related to orders made by federal agencies. Although done due to a matter of national security, this may cause issues for a supplier's bookkeeping and is not the best solution. Another approach to this problem is to anonymize entities in the supply chain and ensure that all order related records do not identify the participants. Therefore, there is a need to achieve transparency as well as maintain anonymity of all participants in the supply chain.

2.2.3 Research Data Procurement

Although most of the data used for research is publicly available, lot of data providers are concerned about the privacy of their data [20, 33]. This has lead to a rise in unethical means of data procurement in some research communities [11]. A few data procurement organizations provide no guarantees of unauthorized access of personal identifiable information [1] while others remove such information from data before publishing publicly. However, such techniques are vulnerable to re-identification attacks [16]. Therefore, there is a need for a system where users can share data securely while maintaining their anonymity and where researchers can verify the integrity of the data.

Chapter 3

Preliminaries

3.1 Notation

We denote $||$ as the concatenation operator. Let λ be a security parameter, negl be as a negligible function, i.e., $\lim_{n\to\infty}(\mathsf{negl}(n)) \times n^k \leftarrow 0 \ \forall \ k > 0 : (n,k) \in \mathbb{R}$. We denote $H : \{0,1\}^* \to \{0,1\}^\lambda$ as a collision-resistant hash function. $r \leftarrow \mathsf{PRF}(s)$ is a pseudorandom function that outputs a pseudorandom r given a seed s. Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an asymmetric public key encryption scheme, in which $(\mathsf{pk}_{\mathsf{adr}}, \mathsf{sk}_{\mathsf{adr}}) \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$ generates a public and private key pair given a security parameter λ; $c \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}_{\mathsf{adr}}, m)$ encrypts a plaintext m under public key $\mathsf{pk}_{\mathsf{adr}}$; $m \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{sk}_{\mathsf{adr}}, c)$ decrypts a ciphertext c with private key $\mathsf{sk}_{\mathsf{adr}}$. Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a digital signature scheme, in which $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$ generates a public and private signing key pair under security parameter λ; $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sig}}, m)$ produces a signature σ for message m under private key $\mathsf{sk}_{\mathsf{sig}}$; $\{0,1\} \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{sig}}, m, \sigma)$ verifies whether σ is a valid signature of m using public key $\mathsf{pk}_{\mathsf{sig}}$.

3.2 Cryptographic Building blocks

Commitment Scheme. A cryptographic commitment scheme allows to commit to a secret with the ability to reveal the committed value later. A commitment scheme is a tuple

of PPT algorithms $(\mathsf{Com}.\mathcal{G}, \mathsf{Com})$ defined as follows.

- $\mathsf{pp} \leftarrow \mathsf{Com}.\mathcal{G}(1^\lambda)$: Given a security parameter λ, it outputs public parameters pp.
- $\mathsf{cm}_r \leftarrow \mathsf{Com}_r(m, \mathsf{pp})$: Given a message m and a trapdoor r, it outputs a commitment cm_r.

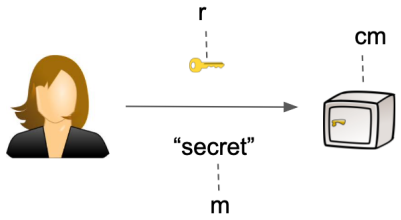


Figure 3.1: High level idea of a commitment scheme

A commitment scheme satisfies the following properties.

- *Binding:* For any PPT adversary $\mathcal{A}$, $\mathsf{pp} \leftarrow \mathsf{Com}.\mathcal{G}(1^\lambda)$, $(m_0, m_1, r_0, r_1) \leftarrow \mathcal{A}(\mathsf{pp})$ it holds that

$$\mathsf{Pr}\left[\mathsf{Com}_{r_0}(m_0, \mathsf{pp}) = \mathsf{Com}_{r_1}(m_1, \mathsf{pp}) \wedge m_0 \neq m_1\right] \leq \mathsf{negl}(\lambda)$$

  meaning that no adversary can reveal different openings to an already committed value.

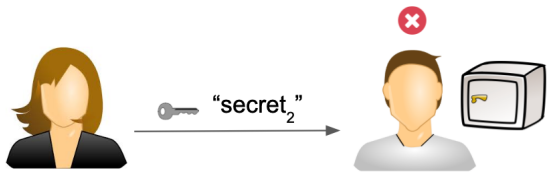


Figure 3.2: Binding property

- *Hiding:* For any PPT adversary $\mathcal{A}$, $\mathsf{pp} \leftarrow \mathsf{Com}.\mathcal{G}(1^\lambda)$, it holds that

$$\left|\mathsf{Pr}\left[b = b' \;\middle|\; \begin{array}{l}(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pp}), b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \{0,1\}^\lambda, \\ \mathsf{cm} \leftarrow \mathsf{Com}_{r_b}(m_b, \mathsf{pp}), b' \leftarrow \mathcal{A}(\mathsf{cm})\end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

meaning that no adversary can open the committed value until the committer reveals its opening.

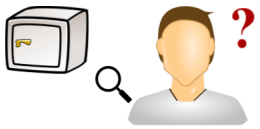


Figure 3.3: Hiding property

Merkle Tree. Merkle tree permits one to commit to a set of values with the ability to reveal an element in the committed set later (proof of membership). The tree is built as follows. Given $\vec{m} = (m_1, m_2, m_3, \ldots, m_n)$, It first initializes a binary tree T with n leaves, and set $H(m_i)$ to the i-th leaf node. The tree T is built in the bottom-up manner, in which each non-leaf node t of T is computed by computing the hash of its child nodes t_{l} and t_{r}, i.e., $t = H(t_{\mathsf{l}}||t_{\mathsf{r}})$. To prove $m_i \in \vec{m}$, it suffices to show the path path from rt to $H(m_i)$, where rt is root of T and path is the set of sibling nodes along the path from m_i to rt. This can be done by having the prover to reveal, path, so that the verifier can recompute rt for membership verification.

Given an updated set $\vec{m'}$, it is easy to update its commitment rt accordingly in logarithmic time. Let $m_i \in \vec{m'}$ be the updated value. The prover updates $i - th$ leaf with $H(m'_i)$ and recomputes non-leaf nodes along the path from $H(m'_i)$ to rt, and outputs the updated root rt'.

Argument of Knowledge. An argument of knowledge for an NP relation $\mathcal{R}$ is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, in which $\mathcal{P}$ convinces $\mathcal{V}$ that it knows a witness w for some statement in an NP language $x \in \mathcal{L}$ such that $(x, w) \in \mathcal{R}$.

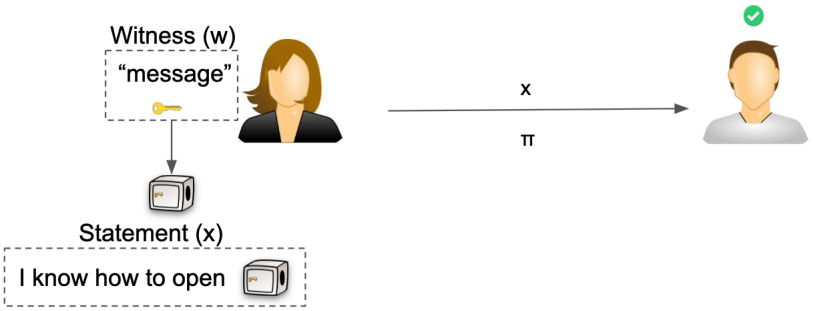


Figure 3.4: High level idea of an argument of knowledge scheme

Formally speaking, a zero-knowledge argument of knowledge is a tuple of PPT algorithms $(\mathsf{zkp}.\mathcal{G}, \mathsf{zkp}.\mathcal{P}, \mathsf{zkp}.\mathcal{V})$ as follows.

- $\mathsf{pp} \leftarrow \mathsf{zkp}.\mathcal{G}(1^\lambda)$: Given a security parameter λ, it outputs public parameter pp.
- $\pi \leftarrow \mathsf{zkp}.\mathcal{P}(x, w, \mathsf{pp})$. Given a statement x and a witness w, it generates a proof π indicating $(x, w) \in \mathcal{R}$.
- $\{0, 1\} \leftarrow \mathsf{zkp}.\mathcal{V}(x, \pi, \mathsf{pp})$. Given a statement x and a proof π, it outputs 1 if π is a valid and $(x, w) \in \mathcal{R}$, else it outputs 0.

A zero knowledge argument of knowledge satisfies the following properties

- *Completeness.* For any $(x, w) \in \mathcal{R}$, $\pi \leftarrow \mathsf{zkp}.\mathcal{P}(x, w, \mathsf{pp})$, $\mathsf{pp} \leftarrow \mathsf{zkp}.\mathcal{G}(1^\lambda)$, it holds that

$$\Pr\left[\mathsf{zkp}.\mathcal{V}(x, \pi, \mathsf{pp}) = 1\right] = 1$$

  meaning that a valid proof will always be verified.

- *Soundness.* For any PPT prover $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that, given the entire execution and randomness of $\mathcal{P}^*$, $\mathcal{E}$ can extract a witness w such that $\mathsf{pp} \leftarrow$

$\mathsf{zkp}.\mathcal{G}(1^\lambda)$, $w \leftarrow \mathcal{E}^{\mathcal{P}^*}(x, \pi^*, \mathsf{pp}), \pi^* \leftarrow \mathcal{P}^*(x, \mathsf{pp})$ and

$$\mathsf{Pr}\left[\mathsf{zkp}.\mathcal{V}(x, \pi, \mathsf{pp}) = 1 \wedge (x, w) \notin \mathcal{R}\right] \leq \mathsf{negl}(\lambda)$$

meaning that an adversarial prover cannot generate a valid proof without knowledge of the witness.

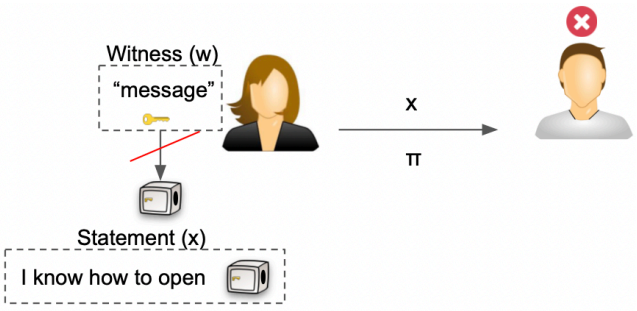


Figure 3.5: Soundness property

- *Zero knowledge.* There exists a simulator S such that for any PPT verifier $\mathcal{V}^*$, $(x, w) \in \mathcal{R}$, it holds that

$$\mathsf{Pr}\left[\begin{array}{c|c} \mathsf{zkp}.\mathcal{V}(x, \pi, \mathsf{pp}) = 1 & \begin{array}{r}\mathsf{pp} \leftarrow \mathsf{zkp}.\mathcal{G}(1^\lambda)\\ \pi \leftarrow \mathsf{zkp}.\mathcal{P}(x, w, \mathsf{pp})\end{array}\end{array}\right]$$

$$\overset{c}{\approx} \mathsf{Pr}\left[\begin{array}{c|c} \mathcal{V}^*(x, \pi, \mathsf{pp}) = 1 & \begin{array}{r}\mathsf{pp} \leftarrow S(1^\lambda)\\ \pi \leftarrow S(x, \mathsf{pp})\end{array}\end{array}\right]$$

meaning that an adversarial verifier cannot learn anything from the proof except its validity.

Figure 3.6: Hiding property

3.3 Blockchain

Blockchain is a distributed ledger that records all activities being performed between nodes in a peer-to-peer network. To perform an activity, the node creates and broadcasts a *transaction* to the network (e.g., via a gossip protocol), which contains basic information such as the sender address, receiver address, the message/data from sender to receiver, and a sender's digital signature of the transaction.

A node in the network will collect several transactions, verify their validity (e.g., correct signature, consistent format), and form a block to be included to the ledger. Upon achieving a consensus between several nodes in the network, the block will be verified and appended to the ledger. All the blocks in the ledger are linked together via cryptographic hash functions and each node in the network maintain a replica of the ledger.

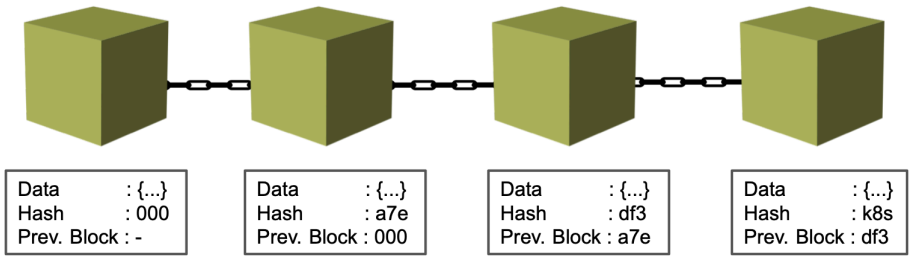


Figure 3.7: High level idea of a blockchain

Privacy-Preserving Blockchain. Since the ledger is public and maintained by multiple nodes in the network, it may create security concerns given that the information in

the transaction is sensitive. Privacy-preserving blockchain permits some information in the transaction (e.g., sender identity, receiver identity, message/data) to remain hidden against the blockchain nodes. We recall Zerocash [5], a privacy-preserving blockchain, which harnesses cryptographic commitment and zero-knowledge techniques (zk-SNARK) to prove the validity of transactions in the cryptocurrency context, without leaking any information (e.g., sender/receiver linkability, account balance) beyond the fact that the transaction is valid (e.g., no double-spending, no money created out of thin air). In Zerocash, there are two types of transactions: a mint transaction, which permits a participant to convert from their (non-private) basecoin currency (e.g., bitcoin) to the zerocoin currency; a pour transaction, which permits the participant to transact their zerocoin to the recipient without leaking the transaction amount and the sender/recipient information. Each pour transaction consists of a cryptographic proof indicating that the sender is the owner of the transacting coin, the transacting coin is never spent previously, and the input and output balance is preserved.

Chapter 4

Models

4.1 System Model

In our system, we consider three types of participants: (i) The data owner(s) (denoted $\mathcal{O}$) who owns some data; (ii) The data user(s) (denoted $\mathcal{U}$) who would like to access data of some data owner; (iii) The storage provider $\mathcal{S}$ who offers storage facilities for the data owners, as well as access facilities for the data users to access data shared by the owner. Our system permits data storage, sharing and access between these participants while, at the same time, *recording* all valid operations being performed on the data in a distributed audit log. Formally speaking, a distributed privacy preserving and immutable audit log (PIAL) scheme consists of PPT algorithms PIAL = $(\mathsf{Init}, \mathsf{Register}, \mathsf{Store}, \mathsf{AssignOwner}, \mathsf{Share}, \mathsf{Access})$ as follows –

- $(\mathcal{L}, \mathsf{pp}) \leftarrow \mathsf{Init}(1^\lambda, N)$: Given a security parameter λ, and the maximum number of supported operations N, it outputs an initial distributed audit log $\mathcal{L}$ and public parameters pp.
- $(\mathsf{pk}_{\mathsf{adr}}, \mathsf{sk}_{\mathsf{adr}}) \leftarrow \mathsf{Register}(1^\lambda)$: Given a security parameter λ, it outputs an address-key pair $(\mathsf{pk}_{\mathsf{adr}}, \mathsf{sk}_{\mathsf{adr}})$ as an identifier for the participant.
- $(\mathsf{tk}_{\mathsf{str}}, \mathsf{tx}_{\mathsf{str}}) \leftarrow \mathsf{Store}(D, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, \mathsf{pp})$: Given data D, and a data owner private identifier $\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}$, a server public identifier $\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}$, it outputs a store token $\mathsf{tk}_{\mathsf{str}}$ and a record $\mathsf{tx}_{\mathsf{str}}$ of the

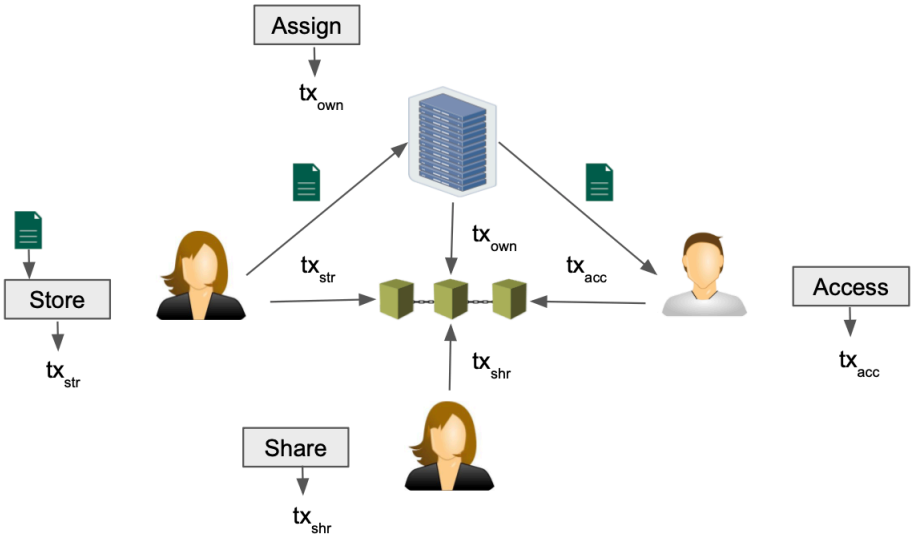


Figure 4.1: System Model

store operation.

- $(\mathsf{tk}_{\mathsf{own}}, \mathsf{tx}_{\mathsf{own}}) \leftarrow \mathsf{AssignOwner}(\mathsf{tk}_{\mathsf{str}}, \mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}, \mathsf{pp})$: Given a store request token $\mathsf{tk}_{\mathsf{str}}$, a server private identifier $\mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}$, and a data owner public identifier $\mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}$, it outputs an ownership token $\mathsf{tk}_{\mathsf{own}}$, which permits $\mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}$ to share their data stored on $\mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}$ later, and a record of ownership $\mathsf{tx}_{\mathsf{own}}$.

- $(\mathsf{tk}_{\mathsf{shr}}, \mathsf{tx}_{\mathsf{shr}}) \leftarrow \mathsf{Share}(\mathsf{tk}_{\mathsf{own}}, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}, \mathsf{ts}, \mathsf{pp})$: Given a data owner token $\mathsf{tk}_{\mathsf{own}}$, an owner private identifier $\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}$, a data user public identifier $\mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}$, a share expiry timestamp $\mathsf{ts} > 0$, it outputs a data share token $\mathsf{tk}_{\mathsf{shr}}$ that is only valid up to ts, and a record of share $\mathsf{tx}_{\mathsf{shr}}$.

- $(\mathsf{tk}_{\mathsf{acc}}, \mathsf{tx}_{\mathsf{acc}}) \leftarrow \mathsf{Access}(\mathsf{tk}_{\mathsf{shr}}, \mathsf{sk}^{\mathcal{U}}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, \mathsf{pp})$: Given a data share token $\mathsf{tk}_{\mathsf{shr}}$, a user identifier $\mathsf{sk}^{\mathcal{U}}_{\mathsf{adr}}$, and a server identifier $\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}$, it outputs a data access token $\mathsf{tk}_{\mathsf{acc}}$ (if $\mathsf{tk}_{\mathsf{shr}}$ is not expired) and a record of access $\mathsf{tx}_{\mathsf{acc}}$.

- $\{0, 1\} \leftarrow \mathsf{VerifyRecord}(\mathsf{pp}, \mathsf{tx}, \mathcal{L})$: Given a record entry tx, it outputs 1 if the record is valid with respect to the current state of the audit log $\mathcal{L}$, and 0 otherwise.

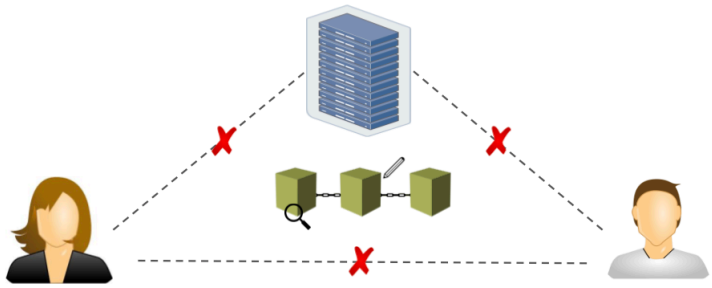


Figure 4.2: Threat Model

4.2 Threat and Security Models

Given that our system records all the data operations in a distributed audit log (realized using blockchain), we consider all the participants in the blockchain to be distrusted against each other in the sense that they can behave maliciously to the records created by other participants that are not related to them. Specifically, the adversary is curious about the record content, e.g., who is the owner of a particular data, whom the data is shared with or accessed by, what data is being shared or accessed. The adversary may also attempt to modify the record to compromise integrity or to gain access to unauthorized data.

To prevent these vulnerabilities, we aim to achieve integrity and confidentiality properties for records, which can be formally defined via record non-malleability and audit ledger indistinguishability as follows.

Definition 1 (Record Non-Malleability). Let Π be a candidate PIAL scheme. Consider the following experiment $\mathtt{NMGame}_{\Pi,\mathcal{A}}(\lambda)$ between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$.

- **Initialization.** The challenger samples $\mathsf{pp} \leftarrow \mathsf{Init}(1^\lambda)$ and sends pp to $\mathcal{A}$. The challenger initializes a PIAL oracle $\mathcal{O}^{\mathsf{PIAL}}$ using pp. Let $\mathcal{L}$ be the audit log of $\mathcal{O}^{\mathsf{PIAL}}$.
- **Query.** At each time step, $\mathcal{A}$ adaptively specifies a query $q \in \{\mathsf{CreateAddress}, \mathsf{Store},$

$\mathsf{AssignOwner}$, Share, $\mathsf{Access}\}$ to $\mathcal{C}$. The challenger forwards q to $\mathcal{O}^{\mathsf{PIAL}}$ and receives a corresponding response r, which is then forwarded to $\mathcal{A}$. The challenger also provides the view of $\mathcal{L}$ to $\mathcal{A}$.

- **Challenge.** $\mathcal{A}$ outputs a log tx^*, let $\mathcal{T}$ be the set of logs with the same type of tx^* that $\mathcal{O}^{\mathsf{PIAL}}$ generates during the query phase.

  Case 1. If there exists a log $\mathsf{tx}_{\mathsf{own}} \in \mathcal{T}$ created in response to an $\mathsf{AssignOwner}$ query, $\mathcal{A}$ wins and outputs 1 if and only if all the following conditions hold – (i) $\mathsf{tx}^*_{\mathsf{own}} \neq \mathsf{tx}_{\mathsf{own}}$; (ii) $\mathsf{VerifyLog}(\mathsf{tx}^*_{\mathsf{own}}, \mathsf{pp}, \mathcal{L}') = 1$, where $\mathcal{L}'$ is the state of the audit log preceding $\mathsf{tx}_{\mathsf{own}}$; (iii) the serial number sn in $\mathsf{tx}^*_{\mathsf{own}}$ is the same as in $\mathsf{tx}_{\mathsf{own}}$. Otherwise, it outputs 0.

  Case 2. If there exists a log $\mathsf{tx} \in \mathcal{T}$ created in response to *any* other query (i.e, one of $\{\mathsf{Store}, \mathsf{Share}, \mathsf{Access}\}$), $\mathcal{A}$ wins and outputs 1 if and only if all the following conditions hold – (i) $m^* \neq m$; (ii) $\mathsf{VerifyLog}(\mathsf{tx}^*, \mathsf{pp}, \mathcal{L}') = 1$, where $\mathcal{L}'$ is the state of the audit log preceding tx; Otherwise, it outputs 0.

Π is said to achieve *record non-malleability* if $\mathsf{Pr}[\mathtt{NMGame}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$

Definition 2 (Audit Log Indistinguishability). Let Π be a candidate PIAL scheme. Consider the following experiment $\mathtt{IndGame}_{\Pi,\mathcal{A}}(\lambda)$ between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$.

- **Initialization.** The challenger first samples a random bit $b \xleftarrow{\$} \{0,1\}$, $\mathsf{pp} \leftarrow \mathsf{Com}.\mathcal{G}(1^\lambda)$ and initializes two PIAL oracles $\mathcal{O}_0^{\mathsf{PIAL}}$ and $\mathcal{O}_1^{\mathsf{PIAL}}$, where each oracle $\mathcal{O}_i^{\mathsf{PIAL}}$ has a separate audit log $\mathcal{L}_i$ as well as internal variables.

- **Query** At each time step, $\mathcal{A}$ adaptively specifies two queries Q, Q' of the same type (one of $\mathsf{Store}, \mathsf{AssignOwner}, \mathsf{Share}, \mathsf{Access}$). $\mathcal{C}$ forwards Q to $\mathcal{O}_0^{\mathsf{PIAL}}$, Q' to $\mathcal{O}_1^{\mathsf{PIAL}}$ and receives the corresponding answers R_0, R_1. $\mathcal{C}$ forwards (R_b, R_{1-b}) to $\mathcal{A}$. $\mathcal{C}$ also provides to $\mathcal{A}$ two

ledgers $\mathcal{L}_{\mathsf{left}} := \mathcal{L}_b$, $\mathcal{L}_{\mathsf{right}} := \mathcal{L}_{1-b}$, where $\mathcal{L}_b$ is the current audit log in $\mathcal{O}_b^{\mathsf{PIAL}}$.

- **Challenge.** $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. $\mathcal{A}$ wins if $b' = b$ and outputs 1. Otherwise, it outputs 0.

Π is said to achieve *audit log indistinguishability* if $\mathsf{Pr}[\mathtt{IndGame}_{\Pi,\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

Out-of-scope assumption. In our system, we do not hide the access pattern against the storage server. Specifically, the server sees which file is being accessed (but does not know who is accessing it). Hiding the file access pattern is challenging in the context of multi-user data access, which may require a costly primitive as multi-user ORAM [17]. We leave hiding the access pattern against the adversarial storage server as future work.

Chapter 5

Proposed Method

5.1 Overview

We start with the intuition of our construction. Harpocrates offers three data operations – store, share and access, wherein each operation is record in an immutable manner using blockchain. Specifically, to store a data, the data owner creates a "store" request, which serves both as a record and a request to the server indicating what data is to be stored. The corresponding server handles the store request by creating an "ownership" record to confirm that the data has been stored at a particular location, and provide a secret token for the owner to share their data later. While sharing data, the data owner creates a "share" record, which demonstrates their ownership and creates a token for the user to access their data on the server. Harpocrates permits *temporal* access control in the sense that the user can only access the data within a limited period of time. This is particularly useful for privacy-preserving applications such as medical data sharing and federal supply chains. To access data, the data user creates an "access" record, which indicates that his access is authorized by the data owner. If the record is valid, the storage server generates a link for the data user to retrieve the requested data off-chain.

In Harpocrates, all the data operations are recorded in a distributed audit log maintained by multiple parties. To ensure all the records are confidential (e.g., hides party identity or which data is being shared/accessed), while permitting verification of data operations, our

main idea is to use cryptographic commitment, encryption, and zero-knowledge techniques drawing inspiration from Zerocash [5] to create all aforementioned records. We present the detailed constructions for each phase of our scheme in the following section.

5.2 Detailed Construction

In detail, our scheme consists of two phases – i) initialization, where the audit log is created ii) utilization, where participants interact with our scheme to store/share/access data. Every time the participant performs an operation, they follow one of the protocols $\mathsf{Store}, \mathsf{AssignOwner}, \mathsf{Share}, \mathsf{Access}$ and record their operation.

5.2.1 Initialization

The initialization generates public parameters for NP-statements to be proven/verified in zero-knowledge and the commitment scheme. In Harpocrates, there are four types of data operations including store (denoted str), assign owner (denoted own), share (denoted shr), and access (denoted acc). Therefore, we initialize four empty Merkle trees with roots $(\mathsf{rt}_{\mathsf{str}}, \mathsf{rt}_{\mathsf{own}}, \mathsf{rt}_{\mathsf{shr}}, \mathsf{rt}_{\mathsf{acc}})$, each storing the commitments of data operations with the same type. For simplicity, we assume Harpocrates permits up to N number of operations per type.

Since Harpocrates operates in a distributed manner with multiple participants, each participant needs to create an address for communication and recording their data operations. The address consists of two components as $\mathsf{addr} = (\mathsf{pk}_{\mathsf{adr}}, \mathsf{sk}_{\mathsf{adr}})$, where $\mathsf{sk}_{\mathsf{adr}} \xleftarrow{\$} \{0,1\}^{\lambda}$ is the private key that is randomly sampled from a uniform distribution and $\mathsf{pk}_{\mathsf{adr}} = \mathsf{PRF}(\mathsf{sk}_{\mathsf{adr}})$ is the public address (known by all parties) generated from $\mathsf{sk}_{\mathsf{adr}}$. In Harpocrates, since $(\mathsf{pk}_{\mathsf{adr}}, \mathsf{sk}_{\mathsf{adr}})$ is never revealed, each party only needs to generate a single address to perform multiple

operations. Figure 5.1 presents the initialization in detail.

$(\mathcal{L}, \mathsf{pp}) \leftarrow \mathsf{Init}(1^\lambda, N)$:

1. $\mathsf{pp} \leftarrow \{\mathsf{zkp}.\mathcal{G}(1^\lambda), \mathsf{Com}.\mathcal{G}(1^\lambda)\}$
2. Initialize four empty Merkle trees containing N nodes with roots $(\mathsf{rt}_{\mathsf{str}}, \mathsf{rt}_{\mathsf{own}}, \mathsf{rt}_{\mathsf{shr}}, \mathsf{rt}_{\mathsf{acc}})$
3. $\mathcal{L} \leftarrow \{\mathsf{rt}_{\mathsf{str}}, \mathsf{rt}_{\mathsf{own}}, \mathsf{rt}_{\mathsf{shr}}, \mathsf{rt}_{\mathsf{acc}}\}$
4. **return** $(\mathcal{L}, \mathsf{pp})$

$(\mathsf{pk}_{\mathsf{adr}}, \mathsf{sk}_{\mathsf{adr}}) \leftarrow \mathsf{CreateAddress}(1^\lambda)$:

1. $\mathsf{sk}_{\mathsf{adr}} \xleftarrow{\$} \{0,1\}^\lambda$
2. $\mathsf{pk}_{\mathsf{adr}} \leftarrow \mathsf{PRF}(\mathsf{sk}_{\mathsf{adr}})$
3. **return** $(\mathsf{pk}_{\mathsf{adr}}, \mathsf{sk}_{\mathsf{adr}})$

Figure 5.1: Protocol initialization

5.2.2 Data Store Protocol

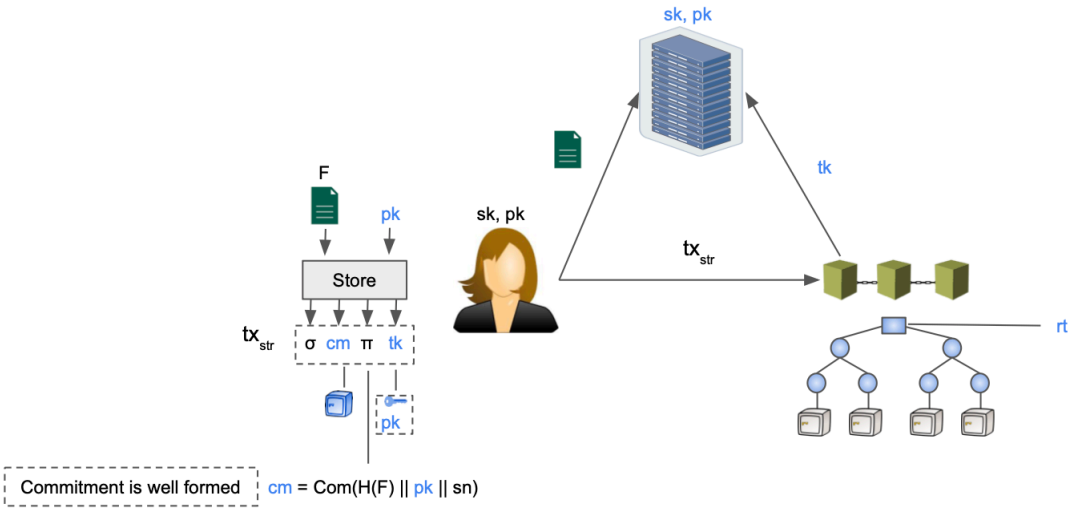


Figure 5.2: High level working of recording a store operation

Figure 5.3 presents our data store protocol, which we elaborate as follows. Given a data D, the data owner first creates a commitment that represents their store request. Specifically, the commitment is formed by the hash of the data $H(D)$, the designated server public address

$\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}$ and a random value ρ as $\mathsf{cm}_{\mathsf{str}} = \mathsf{Com}_r(H(D)||\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}||\rho)$ (Figure 5.3, line 3). The server address is needed in $\mathsf{cm}_{\mathsf{str}}$ to tie store request of D with a particular server. On the other hand, the random ρ is used to ensure that the request can only be handled *once* (see §5.2.3 for further explanation).

To ensure integrity, the data owner generates a one-time unique signing key pair $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}})$ to sign their store request[1] (line 4). To achieve non-repudiation and prevent malleability attacks, such keys must be tied with the data owner identity. Thus, the data owner creates a MAC tag h of the signing key $\mathsf{pk}_{\mathsf{sig}}$ under their address private key $\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}$ as $h = \mathsf{PRF}(\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}, h_{\mathsf{sig}})$, where $h_{\mathsf{sig}} = H(\mathsf{pk}_{\mathsf{sig}})$[2] (line 5). For creating a store record, the owner needs to show that the components of $(\mathsf{cm}_{\mathsf{str}}, h)$ have been formed correctly under their secrets $(\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r, \rho, v, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}, h_{\mathsf{sig}})$. Specifically, the data owner creates a proof (using zkSNARK circuits, more about this in §7) for the following statements in zero-knowledge (line 7).

NP-statement for Data Store. Given a store instance $x = (\mathsf{cm}_{\mathsf{str}}, h)$, the witness $w = (\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r, \rho, v, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}, h_{\mathsf{sig}})$ is a valid witness for x if the following holds:

- The commitment $\mathsf{cm}_{\mathsf{str}}$ is well-formed: $\mathsf{cm}_{\mathsf{str}} = \mathsf{Com}_r(\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}||\rho||v, \mathsf{pp})$.
- The MAC is computed correctly by the address secret key $\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}$: $h = \mathsf{PRF}(\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}||h_{\mathsf{sig}})$ where $h_{\mathsf{sig}} = H(\mathsf{pk}_{\mathsf{sig}})$.

The data owner then creates a store token $\mathsf{tk}_{\mathsf{str}} = \mathcal{E}.\mathsf{Enc}(\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r||\rho||v||\mathsf{cm}_{\mathsf{str}})$ containing the openings of the commitment and encrypts it under the server address public key (line 8). This token will be used by the storage server to respond the owner's request appropriately. The data owner forms a store record $\mathsf{tx}_{\mathsf{str}}$, which contains the commitment $\mathsf{cm}_{\mathsf{str}}$, the MAC

[1] The key is only used one-time to ensure unlinkability between multiple requests from the same data owner.

[2] Since $\mathsf{pk}_{\mathsf{sig}}$ is generally a group element, which may be large, we "compress" it with the hash function to reduce the circuit size of proving PRF evaluation. In fact, $\mathsf{pk}_{\mathsf{sig}}$ is only required to be unique and used once.

tag h, the encrypted token $\mathsf{tk}_{\mathsf{str}}$, proof π_{str}, and the signature verification key $\mathsf{pk}_{\mathsf{sig}}$. We store the token in the record for two main reasons - i) tokens need not be stored permanently by the participants ii) During an audit, participants do not need to be trusted to provide the right token for the transaction being audited. Finally, the data owner signs the record with $\mathsf{sk}_{\mathsf{sig}}$, and appends it to the distributed audit log while sending the data D to the storage provider. The server checks if the decrypted store token $\mathsf{tk}_{\mathsf{str}}$ is consistent with the commitment and the record is valid (i.e., valid signature and valid proof). If so, the storage server stores the data in an appropriate location and creates a record to give the ownership to the data owner, which will be described in the next section.

$\underline{(\mathsf{tk}_{\mathsf{str}}, \mathsf{tx}_{\mathsf{str}}) \leftarrow \mathsf{Store}(D, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, \mathsf{pp})}$:

1. $v \leftarrow H(D)$
2. $\rho \xleftarrow{\$} \{0,1\}^{\lambda}$
3. $\mathsf{cm}_{\mathsf{str}} \leftarrow \mathsf{Com}_r(\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}} || \rho || v, \mathsf{pp})$
4. $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$
5. $h_{\mathsf{sig}} \leftarrow H(\mathsf{pk}_{\mathsf{sig}}), \qquad h \leftarrow \mathsf{PRF}(\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}} || h_{\mathsf{sig}})$
6. $x = (\mathsf{cm}_{\mathsf{str}}, h), \qquad w = (\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r, \rho, v, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}, h_{\mathsf{sig}})$
7. $\pi_{\mathsf{str}} \leftarrow \mathsf{zkp}.\mathcal{P}(x, w, \mathsf{pp})$
8. $\mathsf{tk}_{\mathsf{str}} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r || \rho || v || \mathsf{cm}_{\mathsf{str}})$
9. $m = x || h_{\mathsf{sig}} || \pi_{\mathsf{str}} || \mathsf{tk}_{\mathsf{str}}$
10. $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sig}}, m)$
11. $\mathsf{tx}_{\mathsf{str}} = (\mathsf{pk}_{\mathsf{sig}}, \sigma, \mathsf{cm}_{\mathsf{str}}, h, \pi_{\mathsf{str}}, \mathsf{tk}_{\mathsf{str}})$
12. **return** $(\mathsf{tk}_{\mathsf{str}}, \mathsf{tx}_{\mathsf{str}})$

Figure 5.3: Data store protocol.

5.2.3 Assign Ownership Protocol

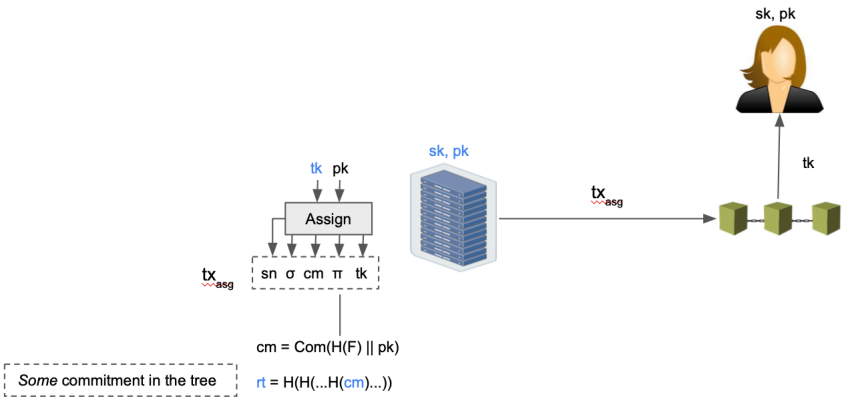


Figure 5.4: High level working of recording an assign operation

The storage server performs an assign ownership operation on receiving a $\mathsf{tx}_{\mathsf{str}}$. The server decrypts $\mathsf{tk}_{\mathsf{str}}$ to identify which data needs to be assigned. A serial number (sn) is generated using the seed ρ (Figure 5.5, line 1) in the store record. The serial number uses a collision-resistant PRF function to ensure that only one assignment is created for the corresponding store operation. Without this, the storage server can use a single $\mathsf{tx}_{\mathsf{str}}$ to maliciously perform assign operations to multiple data owners. Subsequently, the server generates a commitment $\mathsf{cm}_{\mathsf{own}} \leftarrow \mathsf{Com}_{r'}(\mathsf{pk}^{\mathcal{O}}_{\mathsf{addr}}||v', \mathsf{pp})$ (line 2) to represent ownership assignment. To successfully create an assign record, the storage server must generate a proof of ownership π_{own} (line 7). Verification of the proof shows that the storage server indeed received a store request for the corresponding data. The proof is generated by the following NP statements:

NP-statement for Data Owner Assignment. Given an assign owner instance $x = (\mathsf{rt}_{\mathsf{str}}, \mathsf{sn}, \mathsf{cm}_{\mathsf{own}}, h)$, the witness $w = (\mathsf{path}_{\mathsf{str}}, \mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}, \rho, r, v, \mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r', v', \mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}, h_{\mathsf{sig}})$ is a valid witness for x if the following holds:

- The commitment $\mathsf{cm}_{\mathsf{str}}$ appears in the Merkle tree with root $\mathsf{rt}_{\mathsf{str}}$: $\mathsf{rt}_{\mathsf{str}} = H(\ldots(H(H(\mathsf{cm}_{\mathsf{str}})||\mathsf{path}^{d}_{\mathsf{str}})||\mathsf{path}^{d-1}_{\mathsf{str}})||\ldots||\mathsf{path}^{1}_{\mathsf{str}}))$ where d is the height of the merkle tree

and $\mathsf{path}^i_{\mathsf{str}}$ represents a node at height i in the merkle tree along the path from $\mathsf{cm}_{\mathsf{str}}$ to $\mathsf{rt}_{\mathsf{str}}$.

- The commitments $\mathsf{cm}_{\mathsf{str}}$ and $\mathsf{cm}_{\mathsf{own}}$ are well-formed: $\mathsf{cm}_{\mathsf{str}} = \mathsf{Com}_r(\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}||\rho||v, \mathsf{pp})$ and $\mathsf{cm}_{\mathsf{own}} = \mathsf{Com}_{r'}(\mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}||v', \mathsf{pp})$.
- The serial number sn is computed correctly: $\mathsf{sn} = \mathsf{PRF}(\mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}||\rho)$.
- Hashes of the data committed to $\mathsf{cm}_{\mathsf{str}}$ and $\mathsf{cm}_{\mathsf{own}}$ are the same: $v = v'$.
- The address secret key $\mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}$ ties h_{sig} to h: $h = \mathsf{PRF}(\mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}||h_{\mathsf{sig}})$.

We utilize merkle trees to prove set membership i.e, the statement to prove that $\mathsf{cm}_{\mathsf{str}} \in \mathsf{rt}_{\mathsf{str}}$. The tree itself is stored locally by the nodes of the distributed audit log and only the root is added to the record. This is because we want to capture the state of the tree when the particular record was created. The server then creates an encrypted ownership token $\mathsf{tk}_{\mathsf{own}}$ (line 8). In §5.2.4, we shall see how this token is used to share data with a data user. Finally, the server creates a record $\mathsf{tx}_{\mathsf{own}}$, signs it using a one time signing key and appends it to the distributed audit log. The server also sends the corresponding assignment token $\mathsf{tk}_{\mathsf{own}}$ to the data owner off-chain.

$(\mathsf{tk}_{\mathsf{own}}, \mathsf{tx}_{\mathsf{own}}) \leftarrow \mathsf{AssignOwner}(\mathsf{tk}_{\mathsf{str}}, \mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}, \mathsf{pp})$:

1. $\mathsf{sn} \leftarrow \mathsf{PRF}(\mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}||\rho)$
2. $\mathsf{cm}_{\mathsf{own}} \leftarrow \mathsf{Com}_{r'}(\mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}||v', \mathsf{pp})$ ▷ $v' = v$
3. $\mathsf{path}_{\mathsf{str}}$ = Merkle path from $\mathsf{cm}_{\mathsf{str}}$ to $\mathsf{rt}_{\mathsf{str}}$ ▷ Obtain $\mathsf{rt}_{\mathsf{str}}$ from $\mathcal{L}$
4. $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
5. $h_{\mathsf{sig}} \leftarrow H(\mathsf{pk}_{\mathsf{sig}}), \quad h \leftarrow \mathsf{PRF}(\mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}||h_{\mathsf{sig}})$
6. $x = (\mathsf{rt}_{\mathsf{str}}, \mathsf{sn}, \mathsf{cm}_{\mathsf{own}}, h), \quad w = (\mathsf{path}_{\mathsf{str}}, \mathsf{sk}^{\mathcal{S}}_{\mathsf{adr}}, \rho, r, v, \mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r', v', \mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}, h_{\mathsf{sig}})$

7. $\pi_{\mathsf{own}} \leftarrow \mathsf{zkp}.\mathcal{P}(x, w, \mathsf{pp})$

8. $\mathsf{tk}_{\mathsf{own}} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}, r'||v'||\mathsf{cm}_{\mathsf{own}})$

9. $m = x||h_{\mathsf{sig}}||\pi_{\mathsf{own}}||\mathsf{tk}_{\mathsf{own}}$

10. $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sig}}, m)$

11. $\mathsf{tx}_{\mathsf{own}} = (\mathsf{rt}_{\mathsf{str}}, \mathsf{pk}_{\mathsf{sig}}, \sigma, \mathsf{sn}, \mathsf{cm}_{\mathsf{own}}, h, \pi_{\mathsf{own}}, \mathsf{tk}_{\mathsf{own}})$

12. **return** $(\mathsf{tk}_{\mathsf{own}}, \mathsf{tx}_{\mathsf{own}})$

Figure 5.5: Assign owner protocol

5.2.4 Sharing Protocol

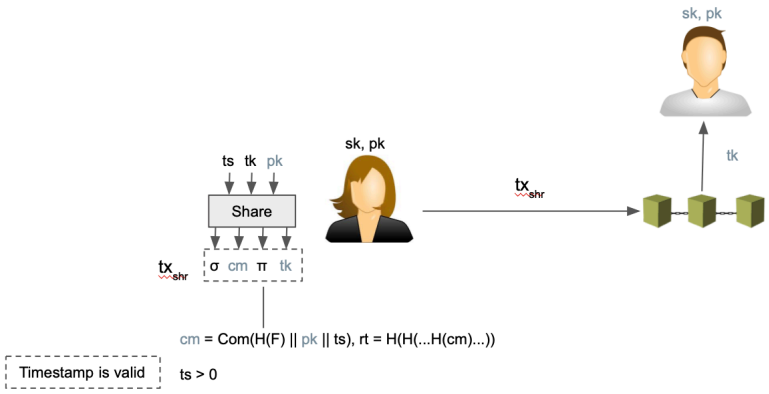


Figure 5.6: High level working of recording a share operation

Given an ownership token $\mathsf{tk}_{\mathsf{own}}$, a share expiry timestamp ts, data user's public identity $\mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}$ and private identity of the data owner $\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}$ a share operation generates a share token $\mathsf{tk}_{\mathsf{shr}}$ to $\mathcal{U}$ and a record $\mathsf{tx}_{\mathsf{shr}}$. A commitment $\mathsf{cm}_{\mathsf{shr}}$ is created to hide the data user, share expiry timestamp and the hash of the data being shared (Figure 5.7, line 1). Intuitively, the data owner generates a proof, π_{shr}, of the data they wish to share (line 6). To produce a valid proof π_{shr}, the data owner proves the following NP statements –

NP-statement for Data Sharing. Given a share instance $x = (\mathsf{rt}_{\mathsf{own}}, \mathsf{cm}_{\mathsf{shr}}, h)$, the witness $w = (\mathsf{path}_{\mathsf{own}}, \mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}, r, v, \mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}, r', v', h_{\mathsf{sig}}, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}})$ is a valid witness for x if the following

holds:

- The commitment $\mathsf{cm}_{\mathsf{own}}$ appears in the Merkle tree with root $\mathsf{rt}_{\mathsf{own}}$: $\mathsf{rt}_{\mathsf{own}} = H(\ldots(H(H(\mathsf{cm}_{\mathsf{own}})||\mathsf{path}^{d}_{\mathsf{own}})||\mathsf{path}^{d-1}_{\mathsf{own}})||\ldots||\mathsf{path}^{1}_{\mathsf{own}}))$.
- The commitments $\mathsf{cm}_{\mathsf{own}}$ and $\mathsf{cm}_{\mathsf{shr}}$ are well-formed, i.e., $\mathsf{cm}_{\mathsf{own}} = \mathsf{Com}_{r}(\mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}||v, \mathsf{pp})$, and $\mathsf{cm}_{\mathsf{shr}} = \mathsf{Com}_{r'}(\mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}||\mathsf{ts}||v', \mathsf{pp})$.
- Hashes of the data committed to $\mathsf{cm}_{\mathsf{str}}$ and $\mathsf{cm}_{\mathsf{own}}$ are the same: $v = v'$.
- The expiration timestamp committed to $\mathsf{cm}_{\mathsf{shr}}$ is valid: $\mathsf{ts} > 0$.
- The address secret key $\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}$ ties h_{sig} to h: $h = \mathsf{PRF}(\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}||h_{\mathsf{sig}})$.

Verification of this proof means that the data owner owns the corresponding data being shared and is sharing it for a valid period of time. Once the proof is generated, all the generated data is bundled into a record $\mathsf{tx}_{\mathsf{shr}}$ and signed using $\mathsf{sk}^{\mathcal{O}}_{\mathsf{sig}}$ (line 9). The record is added to the distributed audit log for verification and the token $\mathsf{tk}_{\mathsf{shr}}$ is sent to the data user. We note that the data owner can share their data multiple times. This can be done because there is nothing preventing the data owner to re-use their token $\mathsf{tk}_{\mathsf{own}}$ more than once to show their ownership. This is unlike the case in AssignOwner where the storage provider reveals the opening of $\mathsf{cm}_{\mathsf{str}}$ in $\mathsf{tk}_{\mathsf{str}}$ through sn.

$(\mathsf{tk}_{\mathsf{shr}}, \mathsf{tx}_{\mathsf{shr}}) \leftarrow \mathsf{Share}(\mathsf{tk}_{\mathsf{own}}, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}, \mathsf{ts}, \mathsf{pp})$:

1. $\mathsf{cm}_{\mathsf{shr}} \leftarrow \mathsf{Com}_{r'}(\mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}||\mathsf{ts}||v', \mathsf{pp})$ ▷ $v' = v$
2. $\mathsf{path}_{\mathsf{own}} \leftarrow$ path from $\mathsf{cm}_{\mathsf{own}}$ to $\mathsf{rt}_{\mathsf{own}}$ in Merkle tree ▷ Obtain $\mathsf{rt}_{\mathsf{own}}$ from $\mathcal{L}$
3. $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$
4. $h_{\mathsf{sig}} \leftarrow H(\mathsf{pk}_{\mathsf{sig}}), \quad h \leftarrow \mathsf{PRF}(\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}||h_{\mathsf{sig}})$
5. $x = (\mathsf{rt}_{\mathsf{own}}, \mathsf{cm}_{\mathsf{shr}}, h), \quad w = (\mathsf{path}_{\mathsf{own}}, \mathsf{pk}^{\mathcal{O}}_{\mathsf{adr}}, r, v, \mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}, r', v', h_{\mathsf{sig}}, \mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}})$

6. $\pi_{\mathsf{shr}} \leftarrow \mathsf{zkp}.\mathcal{P}(x, w, \mathsf{pp})$

7. $\mathsf{tk}_{\mathsf{shr}} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}, r'||v'||\mathsf{ts}||\mathsf{cm}_{\mathsf{shr}})$

8. $m = x||h_{\mathsf{sig}}||\pi_{\mathsf{shr}}||\mathsf{tk}_{\mathsf{shr}}$

9. $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sig}}, m)$

10. $\mathsf{tx}_{\mathsf{shr}} = (\mathsf{rt}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{sig}}, \sigma, \mathsf{cm}_{\mathsf{shr}}, h, \pi_{\mathsf{shr}}, \mathsf{tk}_{\mathsf{shr}})$

11. **return** $(\mathsf{tk}_{\mathsf{shr}}, \mathsf{tx}_{\mathsf{shr}})$

Figure 5.7: Data share protocol

5.2.5 Access Protocol

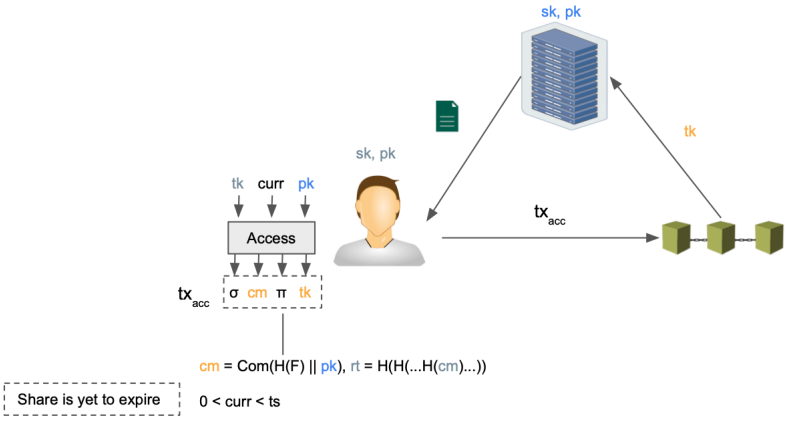


Figure 5.8: High level working of recording an access operation

The data user uses the share token obtained from the data owner to perform an access operation. A commitment representing an access $\mathsf{cm}_{\mathsf{acc}} \leftarrow \mathsf{Com}_{r'}(\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}||v', \mathsf{pp})$ is generated using the storage provider's public key and the hash of the data that needs to be accessed (Figure 5.9, line 1). The data user then creates a proof π_{acc} by proving the following statements –

NP-statement for Data Access. Given an access instance $x = (\mathsf{rt}_{\mathsf{shr}}, \mathsf{cm}_{\mathsf{acc}}, h)$, the witness $w = (\mathsf{path}_{\mathsf{shr}}, \mathsf{pk}^{\mathcal{U}}_{\mathsf{adr}}, r, v, \mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r', v', \mathsf{ts}, \mathsf{curr}, h_{\mathsf{sig}}, \mathsf{sk}^{\mathcal{U}}_{\mathsf{adr}})$ is a valid witness for x if the

following holds:

- The commitment $\mathsf{cm}_{\mathsf{shr}}$ appears in the Merkle tree with root $\mathsf{rt}_{\mathsf{shr}}$: $\mathsf{rt}_{\mathsf{shr}} = H(\ldots(H(H(\mathsf{cm}_{\mathsf{shr}})||\mathsf{path}_{\mathsf{shr}}^{d})||\mathsf{path}_{\mathsf{shr}}^{d-1})||\ldots||\mathsf{path}_{\mathsf{shr}}^{1}))$.
- The commitments $\mathsf{cm}_{\mathsf{shr}}$ and $\mathsf{cm}_{\mathsf{acc}}$ are well-formed, i.e., $\mathsf{cm}_{\mathsf{shr}} = \mathsf{Com}_{r}(\mathsf{pk}_{\mathsf{adr}}^{\mathcal{O}}||\mathsf{ts}||v, \mathsf{pp})$, and $\mathsf{cm}_{\mathsf{acc}} = \mathsf{Com}_{r'}(\mathsf{pk}_{\mathsf{adr}}^{\mathcal{U}}||v', \mathsf{pp})$.
- Hashes of the data committed to $\mathsf{cm}_{\mathsf{shr}}$ and $\mathsf{cm}_{\mathsf{acc}}$ are the same: $v = v'$.
- The timestamp committed to $\mathsf{cm}_{\mathsf{shr}}$ is not expired: $\mathsf{ts} - \mathsf{curr} > 0$.
- The address secret key $\mathsf{sk}_{\mathsf{adr}}^{\mathcal{U}}$ ties h_{sig} to h: $h = \mathsf{PRF}(\mathsf{sk}_{\mathsf{adr}}^{\mathcal{U}}||h_{\mathsf{sig}})$.

A valid proof for the above statements can only be generated if the data user was given the share token by the data owner and the share expiry timestamp is not expired. We note that the data user can access the data as many times as they wish within the given time period after which they will not be able to generate a valid π_{acc}. Finally, the data user creates an access record (line 10) and appends it to the distributed audit log. The data user also sends the access token to the storage server to obtain the data shared with them. The server responds with the data only if the record (i.e $\mathsf{tx}_{\mathsf{acc}}$) corresponding to the access token has been validated by the distributed ledger.

$(\mathsf{tk}_{\mathsf{acc}}, \mathsf{tx}_{\mathsf{acc}}) \leftarrow \mathsf{Access}(\mathsf{tk}_{\mathsf{shr}}, \mathsf{sk}_{\mathsf{adr}}^{\mathcal{U}}, \mathsf{pk}_{\mathsf{adr}}^{\mathcal{S}}, \mathsf{pp})$:

1. $\mathsf{cm}_{\mathsf{acc}} \leftarrow \mathsf{Com}_{r'}(\mathsf{pk}_{\mathsf{adr}}^{\mathcal{S}}||\mathsf{ts} - \mathsf{curr}||v', \mathsf{pp})$ ▷ $v' = v$
2. $\mathsf{path}_{\mathsf{shr}} \leftarrow$ path from $\mathsf{cm}_{\mathsf{shr}}$ to $\mathsf{rt}_{\mathsf{shr}}$ in Merkle tree ▷ Obtain $\mathsf{rt}_{\mathsf{shr}}$ from $\mathcal{L}$
3. $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$
4. $h_{\mathsf{sig}} \leftarrow H(\mathsf{pk}_{\mathsf{sig}}), \quad h \leftarrow \mathsf{PRF}(\mathsf{sk}_{\mathsf{adr}}^{\mathcal{U}}||h_{\mathsf{sig}})$
5. $x = (\mathsf{rt}_{\mathsf{shr}}, \mathsf{cm}_{\mathsf{acc}}, h), \quad w = (\mathsf{path}_{\mathsf{shr}}, \mathsf{pk}_{\mathsf{adr}}^{\mathcal{U}}, r, v, \mathsf{pk}_{\mathsf{adr}}^{\mathcal{S}}, r', v', \mathsf{ts}, \mathsf{curr}, h_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{adr}}^{\mathcal{U}})$

6. $\pi_{\mathsf{acc}} \leftarrow \mathsf{zkp}.\mathcal{P}(x, w, \mathsf{pp})$

7. $\mathsf{tk}_{\mathsf{acc}} \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{pk}^{\mathcal{S}}_{\mathsf{adr}}, r'||v'||\mathsf{curr}||\mathsf{cm}_{\mathsf{acc}})$

8. $m = x||h_{\mathsf{sig}}||\pi_{\mathsf{acc}}||\mathsf{tk}_{\mathsf{acc}}$

9. $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sig}}, m)$

10. $\mathsf{tx}_{\mathsf{acc}} = (\mathsf{rt}_{\mathsf{shr}}, \mathsf{pk}_{\mathsf{sig}}, \sigma, \mathsf{cm}_{\mathsf{acc}}, \mathsf{cm}_{\mathsf{shr}}, \mathsf{ts}, h, \pi_{\mathsf{acc}}, \mathsf{tk}_{\mathsf{acc}})$

11. **return** $(\mathsf{tk}_{\mathsf{acc}}, \mathsf{tx}_{\mathsf{acc}})$

Figure 5.9: Data access protocol

5.2.6 Record Verification

The VerifyRecord algorithm is run on the distributed audit log in order to validate a record received by the nodes of the network. Mainly, the verification of the zero knowledge proof and the signature have to be done in this step. If either of the verification procedures fails, the record is considered invalid and is discarded. If the record verification succeeds, the audit log nodes update their local copies of the corresponding commitment tree. For e.g, if a data user performs an access operation and generates a record $\mathsf{tx}_{\mathsf{acc}}$, the commitment $\mathsf{cm}_{\mathsf{acc}}$ is appended to the global tree $\mathsf{rt}_{\mathsf{acc}}$ containing all access commitments.

$\{0,1\} \leftarrow \mathsf{VerifyRecord}(\mathsf{tx}, \mathcal{L}, \mathsf{pp})$:

1. If $\mathsf{tx} = (\mathsf{pk}_{\mathsf{sig}}, \sigma, \mathsf{cm}_{\mathsf{str}}, h, \pi_{\mathsf{str}}, \mathsf{tk}_{\mathsf{str}})$ ▷ $\mathsf{tx} = \mathsf{tx}_{\mathsf{str}}$
2. $x = (\mathsf{cm}_{\mathsf{str}}, h)$, $m = x||h_{\mathsf{sig}}||\pi_{\mathsf{str}}||\mathsf{tk}_{\mathsf{str}}$
3. Else if $\mathsf{tx} = (\mathsf{rt}_{\mathsf{own}}, \mathsf{pk}_{\mathsf{sig}}, \sigma, \mathsf{sn}, \mathsf{cm}_{\mathsf{str}}, h, \pi_{\mathsf{own}}, \mathsf{tk}_{\mathsf{own}})$ ▷ $\mathsf{tx} = \mathsf{tx}_{\mathsf{own}}$
4. If $\mathsf{sn} \in \mathcal{L}$, abort. ▷ Owner has been previously assigned
5. $x = (\mathsf{rt}_{\mathsf{str}}, \mathsf{sn}, \mathsf{cm}_{\mathsf{own}}, h)$, $m = x||h_{\mathsf{sig}}||\pi_{\mathsf{own}}||\mathsf{tk}_{\mathsf{own}}$
6. Else if $\mathsf{tx} = (\mathsf{rt}_{\mathsf{shr}}, \mathsf{pk}_{\mathsf{sig}}, \sigma, \mathsf{cm}_{\mathsf{shr}}, h, \pi_{\mathsf{shr}}, \mathsf{tk}_{\mathsf{shr}})$ ▷ $\mathsf{tx} = \mathsf{tx}_{\mathsf{shr}}$
7. $x = (\mathsf{rt}_{\mathsf{own}}, \mathsf{cm}_{\mathsf{shr}}, h)$, $m = x||h_{\mathsf{sig}}||\pi_{\mathsf{shr}}||\mathsf{tk}_{\mathsf{shr}}$
8. Else if $\mathsf{tx} = (\mathsf{rt}_{\mathsf{acc}}, \mathsf{pk}_{\mathsf{sig}}, \sigma, \mathsf{cm}_{\mathsf{acc}}, h, \pi_{\mathsf{acc}}, \mathsf{tk}_{\mathsf{acc}})$ ▷ $\mathsf{tx} = \mathsf{tx}_{\mathsf{acc}}$
9. $x = (\mathsf{rt}_{\mathsf{shr}}, \mathsf{cm}_{\mathsf{acc}}, h)$, $m = x||h_{\mathsf{sig}}||\pi_{\mathsf{acc}}||\mathsf{tk}_{\mathsf{acc}}$
10. **return** $\mathsf{zkp}.\mathcal{V}(\mathsf{pp}, x, \pi) \wedge \Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{sig}}, m, \sigma)$

Figure 5.10: Transaction Verification

Chapter 6

Security Analysis

Theorem 1. Harpocrates *achieves record non-malleability by Definition 1.*

Proof Overview. We show Harpocrates achieves record non-malleability by defining an adversary who intercepts a record sent to the distributed audit log by an honest user. The adversary then modifies the record and submits it to the audit log for verification.

However, by highlighting properties of cryptographic primitives used in the system, we show that the adversary's modified record will never be verified by the distributed audit log. Hence, Harpocrates achieves record non-malleability

Proof. **Case 1.** Let $Q_{\mathsf{Register}} = \{1^\lambda, 1^\lambda, \dots\}$ be the queries by $\mathcal{A}$ to Register and $R_{\mathsf{Register}} = \{(\mathsf{pk}^{\mathcal{S},1}_{\mathsf{adr}}, \mathsf{sk}^{\mathcal{S},1}_{\mathsf{adr}}), (\mathsf{pk}^{\mathcal{S},2}_{\mathsf{adr}}, \mathsf{sk}^{\mathcal{S},2}_{\mathsf{adr}}), \dots\}$ be the response from $\mathcal{C}$. $Q_{\mathsf{AssignOwner}} = \{(\mathsf{pp}, \mathsf{tk}^1_{\mathsf{str}}, \mathsf{sk}^{\mathcal{S},1}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{O},1}_{\mathsf{adr}}), (\mathsf{pp}, \mathsf{tk}^2_{\mathsf{str}}, \mathsf{sk}^{\mathcal{S},2}_{\mathsf{adr}}, \mathsf{pk}^{\mathcal{O},2}_{\mathsf{adr}}), \dots\}$ be the queries created by $\mathcal{A}$ and $R_{\mathsf{AssignOwner}} = \{(\mathsf{tk}^1_{\mathsf{own}}, \mathsf{tx}^1_{\mathsf{own}}), (\mathsf{tk}^2_{\mathsf{own}}, \mathsf{tx}^2_{\mathsf{own}}), \dots\}$ be the response by $\mathcal{C}$. $\mathcal{A}$ wins the Log-Entry Non-Malleability game when it outputs a log $\mathsf{tx}^*_{\mathsf{own}}$ such that (i) $m^* \neq m$, where m is the message in $\mathsf{tx}_{\mathsf{own}} \in R_{\mathsf{AssignOwner}}$; (ii) $\mathsf{VerifyLog}(\mathsf{pp}, \mathsf{tx}^*_{\mathsf{own}}, \mathcal{L}') = 1$, where $\mathcal{L}'$ is the state of the audit log preceding $\mathsf{tx}_{\mathsf{own}}$; (iii) sn^* revealed in $\mathsf{tx}^*_{\mathsf{own}}$ is same as in $\mathsf{tx}_{\mathsf{own}}$.

We assume that $\mathcal{A}$ successfully creates an ownership record $\mathsf{tx}^*_{\mathsf{own}} = (\mathsf{rt}^*_{\mathsf{str}}, \mathsf{pk}^*_{\mathsf{sig}}, \sigma^*, \mathsf{sn}^*, \mathsf{cm}^*_{\mathsf{own}}, h^*, \pi^*_{\mathsf{own}}, \mathsf{tk}^*_{\mathsf{own}})$ with proof instance $x^* = (\mathsf{rt}^*_{\mathsf{str}}, \mathsf{sn}^*, \mathsf{cm}^*_{\mathsf{own}}, h^*)$ and $m^* = x^* || h^*_{\mathsf{sig}} || \pi^*_{\mathsf{own}} || \mathsf{tk}^*_{\mathsf{own}}$.

Each of the above event captures all the possible modifications $\mathcal{A}$ makes to $\mathsf{tx}_{\mathsf{str}}$ to obtain $\mathsf{tx}^*_{\mathsf{str}}$ such that $m^* \neq m$. E_1 captures when $m^* \neq m$ but $\mathsf{pk}^*_{\mathsf{sig}} = \mathsf{pk}_{\mathsf{sig}}$. E_2 captures when $m^* \neq m$ but $h^*_{\mathsf{sig}} = h_{\mathsf{sig}}$. E_3 captures when $m^* \neq m$ but $h^* = h$. E_3 captures when $m^* \neq m$. Note that $\mathsf{sn}^* = \mathsf{sn}$ applies to all event as it is the condition of the game.

Let E_1 be the event that $\mathcal{A}$ wins and there exists $\mathsf{pk}_{\mathsf{sig}} \in R_{\mathsf{AssignOwner}}$ such that $\mathsf{pk}^*_{\mathsf{sig}} = \mathsf{pk}_{\mathsf{sig}}$.

$E_2 : \mathcal{A}$ wins, E_1 does not occur, and there exist $\mathsf{pk}_{\mathsf{sig}} \in R_{\mathsf{AssignOwner}}$ such that $h^*_{\mathsf{sig}} = h_{\mathsf{sig}}$.

$E_3 : \mathcal{A}$ wins, (E_1, E_2) do not occur and $h^* = \mathsf{PRF}(\mathsf{sk}^{\mathcal{S}^*}_{\mathsf{adr}} || h^*_{\mathsf{sig}})$ for some $\mathsf{sk}^{\mathcal{S}^*}_{\mathsf{adr}} \in R_{\mathsf{Register}}$.

$E_4 : \mathcal{A}$ wins, (E_1, E_2, E_3) do not occur and $h^* \neq \mathsf{PRF}(\mathsf{sk}^{\mathcal{S}^*}_{\mathsf{adr}} || h^*_{\mathsf{sig}})$ for *all* $\mathsf{sk}^{\mathcal{S}^*}_{\mathsf{adr}} \in R_{\mathsf{Register}}$.

Now we present the intuition to show that each event occurs with negligible probability.

E_1 occurs iff $\mathcal{A}$ can forge a new signature σ^* in tx^* for the message m^* that contain a new serial number sn^*, under the same signing and verification key, which only happens with negligible probability due to the SUF-CMA property of signature scheme.

E_2 only occurs iff $\mathcal{A}$ can provide a new $\mathsf{pk}^*_{\mathsf{sig}}$ in tx^* such that $H(\mathsf{pk}^*_{\mathsf{sig}}) = H(\mathsf{pk}_{\mathsf{sig}})$, which only happens with negligible probability due to the collision-resistant property of H

E_3 occurs iff $\mathcal{A}$ can distinguish PRF from a random function, which happens with negligible probability.

E_4 occurs iff $\mathcal{A}$ can find a collision for the PRF used to generate sn, which happens only with negligible probability.

- We show that $\mathsf{Pr}[E_1] = \mathsf{negl}(\lambda)$. Let σ^* be the signature in $\mathsf{tx}^*_{\mathsf{own}}$ and σ be signature in $\mathsf{tx}_{\mathsf{own}}$ that has $\mathsf{pk}_{\mathsf{sig}}$. Because $\mathsf{pk}_{\mathsf{sig}} = \mathsf{pk}^*_{\mathsf{sig}}$, both σ^* and σ are the signatures created from the same public key. Since $\mathcal{A}$ wins, by definition, there exists $\mathsf{tx}_{\mathsf{own}} \in R_{\mathsf{AssignOwner}}$ such that $m^* \neq m$ and $\mathsf{sn}^* = \mathsf{sn}$. To satisfy case this, $\mathcal{A}$ needs to produce σ^* such

that $\Sigma.\mathsf{Verify}(\mathsf{pk}^*_{\mathsf{sig}}, m^*, \sigma^*) = 1$, $m^* \neq m$ and $\mathsf{pk}_{\mathsf{sig}} = \mathsf{pk}^*_{\mathsf{sig}}$. However, this happens with negligible probability as it breaks the SUF-CMA property of the underlying signature scheme. Also, as $\mathsf{sn}^* = \mathsf{sn}$, this only happens with negligible probability given that the serial number $\mathsf{sn} = \mathsf{PRF}(\mathsf{sk}^{\mathcal{O}}_{\mathsf{adr}}||\rho)$ is generated by a PRF with a random seed ρ.

- We show that $\mathsf{Pr}[E_2] = \mathsf{negl}(\lambda)$. Let $h^*_{\mathsf{sig}} \leftarrow H(\mathsf{pk}^*_{\mathsf{sig}})$ be the hash of the signature verification key $\mathsf{pk}^*_{\mathsf{sig}}$ in $\mathsf{tx}^*_{\mathsf{own}}$ and $h_{\mathsf{sig}} = H(\mathsf{pk}_{\mathsf{sig}})$ be the hash of the signature verification key $\mathsf{pk}_{\mathsf{sig}}$ in $\mathsf{tx}_{\mathsf{own}}$. Since $\mathcal{A}$ wins, there is a $m^* \neq m$ such that $h^*_{\mathsf{sig}} = h_{\mathsf{sig}}$. Now when $\mathsf{pk}^*_{\mathsf{sig}} = \mathsf{pk}_{\mathsf{sig}}$, it would be handled in E_1. When $\mathsf{pk}^*_{\mathsf{sig}} \neq \mathsf{pk}_{\mathsf{sig}}$, due to the collision-resistant property of the hash function, $h^*_{\mathsf{sig}} = h_{\mathsf{sig}}$ cannot hold true.

- We show that $\mathsf{Pr}[E_3] = \mathsf{negl}(\lambda)$ by contradiction. Assume $\epsilon = \mathsf{Pr}[E_3]$ is non negligible. Now we construct an attacker $\mathcal{B}$ that distinguishes with a non negligible probability a pseudorandom function PRF from a random function RAND by crafting $\mathsf{tx}^{\dagger}_{\mathsf{own}}$. $\mathcal{B}$ follows the same conditions that $\mathcal{A}$ follows to craft tx^*. To do so, $\mathcal{B}$ selects a random $(\mathsf{pk}^{\mathcal{S}^{\dagger}}_{\mathsf{adr}}, \mathsf{sk}^{\mathcal{S}^{\dagger}}_{\mathsf{adr}}) \in R_{\mathsf{Register}}$. Then, $\mathcal{B}$ crafts $h^{\dagger} = \mathcal{O}^{\mathsf{PRF}}(\mathsf{sk}^{\mathcal{S}^{\dagger}}_{\mathsf{adr}}||h^{\dagger}_{\mathsf{sig}})$ or $h^{\dagger} = \mathcal{O}^{\mathsf{RAND}}(\mathsf{sk}^{\mathcal{S}^{\dagger}}_{\mathsf{adr}}||h^{\dagger}_{\mathsf{sig}})$ using either of the two oracles $\mathcal{O}^{\mathsf{PRF}}$ or $\mathcal{O}^{\mathsf{RAND}}$. $\mathcal{B}$ then invokes $\mathcal{A}$ which outputs $\mathsf{tx}^*_{\mathsf{own}}$. Now, according to conditions of E_3, $\mathcal{B}$ outputs 1 (i.e, wins) iff $h^{\dagger} = h^*$ where h^* is a part of $\mathsf{tx}^*_{\mathsf{own}}$ crafted by $\mathcal{A}$. $\mathcal{B}$ *aborts and wins* if it outputs $h^{\dagger} = h^*$ *before* $\mathcal{A}$ outputs $\mathsf{tx}^*_{\mathsf{own}}$.

  Now, we calculate the probability that $\mathcal{B}$ wins. Note that $\mathcal{B}$ wins when it aborts *or* if it outputs 1 and does not abort. Recall that $\mathcal{B}$ aborts iff it outputs $h^{\dagger} = h^*$ before $\mathcal{A}$ outputs a h^*. Then, when $\mathcal{A}$ finally outputs h^*, it would imply that $\mathsf{pk}^*_{\mathsf{sig}} = \mathsf{pk}^{\dagger}_{\mathsf{sig}}$ and $h^*_{\mathsf{sig}} = h^{\dagger}_{\mathsf{sig}}$ which does not hold in E_3 (when E_3 occurs, E_1 and E_2 don't occur). Hence, $\mathcal{B}$ aborts with negligible probability. Therefore, to calculate the probability with which $\mathcal{B}$ wins, it is sufficient to calculate the probability that $\mathcal{B}$ outputs 1 and does not abort.

  Since $\mathcal{B}$ uses either of the two oracles, we have the following cases – i) $h^{\dagger}$ is created using $\mathcal{O}^{\mathsf{RAND}}$. If $h^{\dagger}$ is of one bit length, the probability that $\mathcal{B}$ wins and does not abort is $\frac{1}{2}$.

Since the output is of length $|h^\dagger|$, the probability that $\mathcal{B}$ wins and does not abort is $(\frac{1}{2})^{|h^\dagger|}$, which is negligible.

ii) $h^\dagger$ is crafted using $\mathcal{O}^{\mathsf{PRF}}$. Since $\mathcal{A}$ performs the experiment independent of $\mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}^\dagger}$, probability that $\mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}^*} = \mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}^\dagger}$ will be $\frac{1}{|R_{\mathsf{Register}}|}$. Therefore, the probability that $\mathcal{B}$ wins and $\mathcal{B}$ does not aborts is at least $\frac{\epsilon}{|R_{\mathsf{Register}}|}$. Since ϵ is non negligible (by assumption), $\frac{\epsilon}{|R_{\mathsf{Register}}|}$ is also non negligible.

Now to calculate the probability that $\mathcal{B}$ distinguishes between a pseudorandom and a random function, we find the difference in probabilities when $\mathcal{B}$ wins in i) and ii). Since in ii), $\mathcal{B}$ wins with a non negligible probability, the difference is non negligible.

We successfully crafted an attacker $\mathcal{B}$ that can distinguish between a PRF and a random function with an overwhelming probability. However, this is a contradiction as a pseudorandom function is indistinguishable from a random function.

- We show that $\mathsf{Pr}[E_4] = \mathsf{negl}(\lambda)$ by contradiction. We craft an algorithm $\mathcal{B}$ which finds collisions in PRF used to compute sn with an overwhelming probability. $\mathcal{B}$ first runs $\mathcal{A}$ to obtain $\mathsf{tx}_{\mathsf{own}}^*$. Then, it runs a zero-knowledge extractor $\mathcal{E}$ to obtain a witness w. If w is invalid, $\mathcal{A}$ aborts and returns 0. If w is valid, find $\mathsf{tx} \in R_{\mathsf{AssignOwner}}$ with $\mathsf{sn}^* = \mathsf{sn}$. If $\mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}^*} \neq \mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}}$ and $\mathsf{sn}^* = \mathsf{sn}$, $\mathcal{B}$ found a collision for PRF.

  Now we show that $\mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}^*} \neq \mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}}$ occurs with overwhelming probability. Since w is valid, $h^* = \mathsf{PRF}(\mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}^*} || h_{\mathsf{sig}}^*)$. Now, if $\mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}^*} = \mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}}$, it contradicts conditions of E_4. Therefore, if E_4 occurs, $\mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}^*} \neq \mathsf{sk}_{\mathsf{adr}}^{\mathcal{S}}$ with overwhelming probability. This implies that $\mathcal{B}$ always finds a $\mathsf{tx} \in R_{\mathsf{AssignOwner}}$ such that $\mathsf{sn}^* = \mathsf{sn}$ with an overwhelming probability.

  We successfully created an algorithm $\mathcal{B}$ that finds a collision for PRF used to create sn with an overwhelming probability. However, since the PRF used to create sn has to be collision-resistant, this is a contradiction.

Case 2. We assume that $\mathcal{A}$ creates store operation record, however, the same proof applies if $\mathcal{A}$ creates share or assign operation records as well. Let $Q_{\mathsf{Register}} = \{1^\lambda, 1^\lambda, \dots\}$ be the queries by $\mathcal{A}$ to $\mathsf{Register}$ and $R_{\mathsf{Register}} = \{(\mathsf{pk}_{\mathsf{adr}}^{\mathcal{O},1}, \mathsf{sk}_{\mathsf{adr}}^{\mathcal{O},1}), (\mathsf{pk}_{\mathsf{adr}}^{\mathcal{O},2}, \mathsf{sk}_{\mathsf{adr}}^{\mathcal{O},2}), \dots\}$ be the response from $\mathcal{C}$. $Q_{\mathsf{Store}} = \{(\mathsf{pp}, D_1, \mathsf{sk}_{\mathsf{adr}}^{\mathcal{O},1}, \mathsf{pk}_{\mathsf{adr}}^{\mathcal{S},1}), (\mathsf{pp}, D_2, \mathsf{sk}_{\mathsf{adr}}^{\mathcal{O},2}, \mathsf{pk}_{\mathsf{adr}}^{\mathcal{S},2}), \dots\}$ be the queries created by $\mathcal{A}$ and $R_{\mathsf{Store}} = \{(\mathsf{tk}_{\mathsf{str}}^1, \mathsf{tx}_{\mathsf{str}}^1), (\mathsf{tk}_{\mathsf{str}}^2, \mathsf{tx}_{\mathsf{str}}^2), \dots\}$ be the response by $\mathcal{C}$. $\mathcal{A}$ wins $\mathtt{NMGame}_\Pi$ when it outputs a record $\mathsf{tx}_{\mathsf{str}}^*$ such that (i) $m^* \neq m$, where m is a part of $\mathsf{tx}_{\mathsf{str}} \in R$; (ii) $\mathsf{VerifyRecord}(\mathsf{pp}, \mathsf{tx}_{\mathsf{str}}^*, \mathcal{L}') = 1$, where $\mathcal{L}'$ is the state of the audit log preceding $\mathsf{tx}_{\mathsf{str}}$.

We assume that $\mathcal{A}$ successfully creates a store record $\mathsf{tx}_{\mathsf{str}}^* = (\mathsf{pk}_{\mathsf{sig}}^*, \sigma^*, \mathsf{cm}_{\mathsf{str}}^*, h^*, \pi_{\mathsf{str}}^*, \mathsf{tk}_{\mathsf{str}}^*)$ with proof instance $x^* = (\mathsf{cm}_{\mathsf{str}}^*, h^*)$ and $m^* = x^* || \pi_{\mathsf{str}}^* || \mathsf{tk}_{\mathsf{str}}^*$.

Let E_1 be the event that $\mathcal{A}$ wins and there exists $\mathsf{pk}_{\mathsf{sig}} \in R_{\mathsf{Store}}$ such that $\mathsf{pk}_{\mathsf{sig}}^* = \mathsf{pk}_{\mathsf{sig}}$.

$E_2 : \mathcal{A}$ wins, E_1 does not occur, and there exist $\mathsf{pk}_{\mathsf{sig}} \in R_{\mathsf{Store}}$ such that $h_{\mathsf{sig}}^* = h_{\mathsf{sig}}$.

$E_3 : \mathcal{A}$ wins, (E_1, E_2) do not occur and $h^* = \mathsf{PRF}(\mathsf{sk}_{\mathsf{adr}}^{\mathcal{O}^*}, h_{\mathsf{sig}}^*)$ for some $\mathsf{sk}_{\mathsf{adr}}^{\mathcal{O}^*} \in R_{\mathsf{Register}}$.

Each of the above event captures all the possible modifications $\mathcal{A}$ makes to $\mathsf{tx}_{\mathsf{str}}$ to obtain $\mathsf{tx}_{\mathsf{str}}^*$ such that $m^* \neq m$. E_1 captures when $m^* \neq m$ but $\mathsf{pk}_{\mathsf{sig}}^* = \mathsf{pk}_{\mathsf{sig}}$. E_2 captures when $m^* \neq m$ but $h_{\mathsf{sig}}^* = h_{\mathsf{sig}}$. E_3 captures when $m^* \neq m$ but $h^* = h$. Note that we don't have E_4, like in **Case 1**, which captures $m^* \neq m$ and $\mathsf{sn}^* = \mathsf{sn}$ since the response to a query to one of the functions $\{\mathsf{Store}, \mathsf{Share}, \mathsf{Access}\}$ does not include sn.

Now we show that each event occurs with a neglible probability.

- We show that $\mathsf{Pr}[E_1] = \mathsf{negl}(\lambda)$. Let σ^* be the signature in tx^* and σ be signature in tx that has $\mathsf{pk}_{\mathsf{sig}}$. Because $\mathsf{pk}_{\mathsf{sig}} = \mathsf{pk}_{\mathsf{sig}}^*$, both σ^* and σ are the signatures created from the same public key. Since $\mathcal{A}$ wins, by definition, there exists $\mathsf{tx}_{\mathsf{str}} \in R_{\mathsf{Store}}$ such that $m^* \neq m$. To craft $m^* \neq m$, $\mathcal{A}$ needs to produce σ^* such that $\Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{sig}}^*, m^*, \sigma^*) = 1$, $m^* \neq m$ and $\mathsf{pk}_{\mathsf{sig}} = \mathsf{pk}_{\mathsf{sig}}^*$. However, this happens with negligible probability as it breaks the SUF-CMA

property of the underlying signature scheme.

- We show that $\mathsf{Pr}[E_2] = \mathsf{negl}(\lambda)$. The proof outline follows the same as that shown for E_2 in **Case 1**. Basically, we show that the $\mathcal{A}$ will have to find a collision of a hash function to win this event. However, this only occurs with negligible probability.

- We show that $\mathsf{Pr}[E_3] = \mathsf{negl}(\lambda)$. Similar to how we prove E_3 occurs with negligible probability in **Case 1**, we construct an attacker $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine. The attacker is constructed such that it distinguishes between a PRF function and a random function with an overwhelming probability. However, due to the indistinguishability property of the PRF function, this happens with a negligible probability.

□

Theorem 2. Harpocrates *achieves audit log indistinguishability by Definition 2*

Proof Overview. We show Harpocrates achieves audit log indistinguishability by defining an adversary who can issue records of any type. The record is sent to a challenger who does the verification before appending to the audit log.

We define a set of "games" where the challenger modifies one part of the record and show that the modification provides no added advantage to the adversary to distinguish between the records issued by him. This means that the audit log is indistinguishable. Finally, we show that the modifications we suggest is computationally equivalent to the original experiment, hence proving audit log indistinguishability of the original experiment.

Proof. We deduce the proof by showing that the queries (Q, Q') answered by the challenger $\mathcal{C}$ is independent of bit b. $\mathcal{C}$ creates a simulation $\texttt{IndGame}_{\mathsf{Sim}}$ of $\texttt{IndGame}_{\Pi}$ by performing the following modifications to the original functions before answering the queries (Q, Q') –

- To answer $\mathsf{AssignOwner}$ queries, $\mathcal{C}$ modifies $\mathsf{AssignOwner}$ to create sn and h randomly. $\mathsf{cm}_{\mathsf{own}} \leftarrow \mathsf{Com}_{r_2}(k_2, \mathsf{pp})$ is computed using random inputs k_2, r_2. The encrypted token $\mathsf{tk}^*_{\mathsf{own}} \leftarrow \mathsf{Enc}(e_1, p_1)$ is generated using a random plaintext p_1 and key e_1. Finally, the proof $\pi^*_{\mathsf{own}} \leftarrow S(x, \mathsf{pp})$ is computed using a simulator.

- To answer Share queries, $\mathcal{C}$ replaces h with a random value. The commitment is created as $\mathsf{cm}^*_{\mathsf{shr}} \leftarrow \mathsf{Com}_{r_3}(k_3, \mathsf{pp})$ where k_3, r_3 are random inputs. $\mathsf{tk}^*_{\mathsf{shr}}$ is created using a random plaintext p_2 and key e_2. Lastly, π^*_{shr} is generated similar to how π^*_{own} is generated.

- To answer Access queries, $\mathcal{C}$ replaces h with a random value. $\mathcal{C}$ creates $\mathsf{cm}^*_{\mathsf{acc}} \leftarrow \mathsf{Com}_{r_4}(k_4, \mathsf{pp})$ where k_4, r_4 are random inputs and creates $\mathsf{tk}^*_{\mathsf{acc}}$ using a random plaintext p_3 and key e_3. Finally, π^*_{acc} is generated in a similar fashion as π^*_{own}.

In all the above cases, the simulated functions compute Q and Q' independent of bit b -

$$\mathsf{Pr}[b = b'] = \frac{1}{2}$$

Therefore, $\mathcal{A}$ has negligible advantage in $\mathtt{IndGame}_{\mathsf{Sim}}$. Now, to show that $\mathtt{IndGame}_{\mathsf{Sim}}$ is indistinguishable from $\mathtt{IndGame}_{\Pi}$, we define a sequence of hybrid games as follows –

- G_0 – This is the original protocol, where the challenger $\mathcal{C}$ executes protocols in §5 to answer $\mathcal{A}$ queries.

- G_1 – G_1 is the same as G_0, except that to answer queries from $\mathcal{A}$, $\mathcal{C}$ uses a zero-knowledge simulator to generate a proof $\pi^* = S(\mathsf{pp}, x)$ with an arbitrary witness. Due to the zero-knowledge property of the underlying proof system, we say that $\mathtt{IndGame}_1$ is indistinguishable from $\mathtt{IndGame}_0$.

- G_2 – G_2 is same as G_1 except that $\mathcal{C}$ generates the encrypted tokens differently to answer

queries from $\mathcal{A}$. In particular, $\mathcal{C}$ generates $\mathsf{tk}^*_{\mathsf{str}}, \mathsf{tk}^*_{\mathsf{own}}, \mathsf{tk}^*_{\mathsf{shr}}, \mathsf{tk}^*_{\mathsf{acc}}$ (depending on the query) using $\mathsf{Enc}(e, p)$ where p is a random plaintext and e is a random public encryption key. Due to the IND-CCA property of the underlying encryption scheme, $\mathcal{A}$ cannot distinguish between a ciphertext generated from a random plaintext or a particular plaintext. Also, due to the IK-CCA property, $\mathcal{A}$ cannot distinguish between a ciphertext generated from a random public key or a particular public key. Due to the IK-CCA and IND-CCA property of the underlying encryption scheme, G_2 is indistinguishable from G_1.

- G_3 – G_3 is the same as G_2, however, $\mathcal{C}$ substitutes PRF generated values with random strings to answer queries from $\mathcal{A}$. Due to the indistinguishability property of the PRF function from a random function, this modification gives no added advantage to $\mathcal{A}$. Therefore, G_3 is indistinguishable from G_2.

- G_4 – G_4, equivalent to simulation $\mathtt{IndGame}_{\mathsf{Sim}}$, is identical to G_3 except that $\mathcal{C}$ computes $\mathsf{cm}^*_{\mathsf{str}}, \mathsf{cm}^*_{\mathsf{own}}, \mathsf{cm}^*_{\mathsf{shr}}, \mathsf{cm}^*_{\mathsf{acc}}$ (depending on the query) as $\mathsf{Com}_r(k, \mathsf{pp})$ where r and k are random inputs. Due to the hiding property of the underlying commitment scheme, G_4 is indistinguishable from G_3. Note that all modifications are independent of bit b.

We can see that $\mathtt{IndGame}_{\Pi} = G_0 \overset{c}{\approx} G_1 \overset{c}{\approx} G_2 \overset{c}{\approx} G_3 \overset{c}{\approx} G_4 = \mathtt{IndGame}_{\mathsf{Sim}}$ and this completes the indistinguishability proof. □

Chapter 7

Implementation

We fully implemented our proposed techniques in Rust consisting of around 3,100 lines of code. Our implementation used Hyperledger Fabric [4] to build up a generic blockchain platform to deploy the distributed audit log to record all data operations. For the signature scheme used to sign data operation records, we implemented Schnorr signature scheme [34]. For the encryption scheme used to encrypt tokens, we implemented ECIES with AES-GCM as the underlying encryption scheme [28]. We implemented four Merkle trees, each of which records and verifies the existence of a data operation (i.e., store, assign, share, access) being performed in our system. For the commitment scheme, we used Pedersen commitment [29]. We used Poseidon as the Merkle tree hash function because it can efficiently compute hashes of 2 elements at once [14]. We implemented MiMC as the primary hash function as well as PRF due to its low multiplicative complexity [3]. Finally, our implementation used Bulletproofs [10] as the back-end zero-knowledge proof due to its small proof size and transparent setup.

Proving Gadgets. In our implementation, we first build up the following gadgets to implement our audit log functionalities.

- **Pedersen Opening Gadget.** We constructed gadget PedG based on proof of exponentiation in [9] to prove and verify the opening of a Pedersen commitment cm. Specifically, given a commitment cm, the prover creates a proof π demonstrating its knowledge of (v, r)

such that $\mathsf{cm} = \mathsf{Com}_r(v, \mathsf{pp})$.

- **Membership Proof Gadget.** We implemented a membership proof gadget MemG, which permits to prove and verify the membership of a committed value cm in a set of commitments C using Merkle proof. Specifically, given a Merkle tree of C with root rt, the prover creates a proof π demonstrating its knowledge of cm such that $\mathsf{cm} \in C$.

- **MiMC Hash Pre-image Gadget.** We implemented the hash pre-image proof gadget HshG in [3] to prove and verify knowledge of the pre-image of a MiMC hash digest. Specifically, given a public hash h, the prover creates a proof π to demonstrate its knowledge of w such that $h = \mathsf{MiMC}(w)$.

- **Range Proof Gadget.** We used a bounded range gadget RgeG described in [10] to ensure that a given value lies within a certain range of values. Given a value b and a range $[a, c]$, the prover uses this gadget to create a proof π to demonstrate its knowledge of b such that $a \leq b \leq c$. We used this to check the validity of timestamps.

Now we look at how we used these gadgets to prove the statements in data operation records.

Store. We implemented a circuit with one PedG and one HshG gadget to prove/verify NP statements needed to create a store record.

AssignOwner. To generate a valid assign record, we implemented a circuit with one MemG, two PedG and two HshG.

Share. We implemented a circuit with one RgeG, one HshG, one MemG and two PedG gadgets to create a valid share operation record.

Access. To prove and verify NP statements of an access operation, we implemented a circuit with RgeG, one HshG, one MemG and two PedG gadgets.

Chapter 8

Experiments

We evaluate the performance of Harpocrates using a micro benchmark and a macro benchmark. The micro benchmark is used to observe the overhead for creating records of each data operation using Harpocrates. The macro benchmark is used to evaluate how well our system performs when implemented on a blockchain platform.

8.1 Configuration

Hardware. We use up to 50 AWS EC2 `t3.medium` instances to deploy our private blockchain network based on Hyperledger Fabric with 50 nodes. Each instance has 4GB RAM with 2-core 3.1 GHz CPU (Intel Xeon Platinum 8000) and a dedicated network bandwidth of 4.3 Gbps. All the nodes were deployed within the `us-east-1` region.

Parameter Setting. We use standard parameters for underlying cryptographic building blocks being used in our scheme to achieve 128-bit security. Specifically, we use Ed25519 curve of order $2^{255} - 19$ for zero-knowledge proofs based on Bulletproofs. We use standard Poseidon Hash parameters for the Merkle tree including width $t = 6$, full rounds $R_F = 8$ and partial rounds $R_p = 130$ [14]. We use MiMC hash function with rate 512 and capacity 513. The Schnorr signature scheme used produces a 512-bit signature. The ECIES encryption scheme uses a 256-bit public and secret key. Both the encryption and signature scenes use

Table 8.1: Performance of each data operation

Data Operation	Circuit Complexity		Delay (s)		Memory (MB)		Bandwidth (KB)
	# const.	# mult.	Prove	Verify	Prove	Verify	
Store	2,389	644	1.8	1.5	40.1	39.3	1.1
Assign	87,315	37,640	23.2	8.6	165.6	94.7	1.4
Share	86,067	37,014	23.0	8.5	159.3	89.8	1.4
Access	86,068	37,014	23.1	8.5	159.4	89.8	1.4

the same Ed25519 curve as used by Bulletproofs.

We use default parameters of Hyperledger Fabric. Specifically, we set block size to 10, and the number of transaction confirmations is 1.

Evaluation Metrics. We measure the performance of our techniques based on the circuit complexity, end-to-end delay, memory usage, and bandwidth overhead.

8.2 Results

8.2.1 Micro-Benchmark

Table 8.1 presents the processing overhead of our techniques to record each data operation in terms of circuit complexity, processing latency, memory usage and bandwidth overhead.

Circuit Complexity. Recording Store operations incur least circuit complexity, compared with other operations such as Assign, Share and Access. This is because the Store operation only invokes one opening gadget and one hash pre-image gadget (see §7), while other operations invokes a set membership gadget along with other gadgets. The constraint system for checking set membership is more complicated compared to that of the other gadgets. Specifically, it requires proving and verifying (in zero-knowledge) $d = 64$ Poseidon hash pre-images, each of which incurs $3 \times t \times R_F + 3 \times R_p = 534$ number of multiplication

constraints [14]. Assign operation incurs a little higher circuit complexity than Share and Access because of an extra hash gadget (for sn) used to verify an Assign operation. Share has one number of constraints over Access as it requires an additional constraint to calculate difference between the current timestamp, curr and the share expiry timestamp, ts.

Processing Latency. The processing delay for each data operation depends on the its circuit complexity, especially the number of multiplication gates. A Store operation incurs a small latency compared with other operations because it does not require checking set membership, which attributes to a large circuit complexity. In our benchmark, all the reported delays include the time to load the Common Reference String (CRS) component that is needed for the backend zero-knowledge proof (i.e., Bulletproofs), which takes around 1.5 seconds in total. Without CRS, the proving and verification time for Store operation is 0.35 seconds and 85 milliseconds, respectively, which are about 60 times lesser than Assign, Share and Access. This matches the difference in multiplicative complexity between the circuits.

Memory Usage. Similar to the processing delay, the memory usage also depends on the circuit complexity to prove each data operation. The CRS component takes approximately 32 MB. Proving incurs more memory usage than verification, due to the space needed to store the witness.

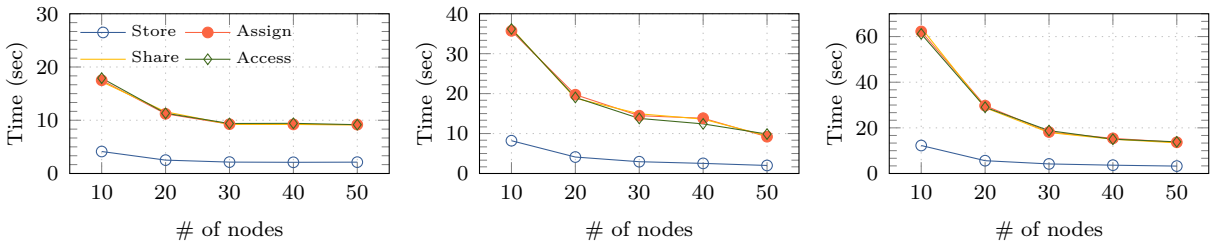
Bandwidth. Since Bulletproof only incurs a polylograthmic proof size, the bandwidth overhead for each data operation is small. Specifically, the proof size produced by Bulletproofs is $2\lceil \log_2(m) \rceil + 13$ group elements where m is the number of multiplication gates. In data store operation, $m = 644$ so that its proof size contains $2\lceil \log_2(644) \rceil + 13 = 33$ group elements, which results in a total of $32 \cdot 33 = 1,056$ bytes (B). For Assign, Share and Access operations, $m \approx 37,000$ and thus, their proof size is 1,440 B.

8.2.2 Macro-Benchmark

We now measure the actual performance of our scheme when integrated with a blockchain platform. As discussed, we used Hyperledger Fabric to deploy 50 blockchain nodes on Amazon EC2. We develop Hyperledger Fabric chaincodes for verifying each data operation. The user first creates a transaction using one of the algorithms described in §7 based on the data operation the user wishes to perform. The transaction, which serves as a record for the data operation, is then submitted to the blockchain network. The nodes in the network calls the corresponding chaincode to verify the transaction. Finally, the transaction gets verified and is added to the blockchain.

Overall delay. Figure 8.1 presents the latency to verify a data operation with different numbers of concurrent operations and blockchain nodes. Specifically, given 25 concurrent data operations, it takes around 17 seconds to verify a record of Assign, Share, and Access operations with 10 blockchain nodes (Figure 8.1a). We can see that the verification delay reduces when increasing the number of blockchain nodes. In particular, it reduces to around 11 seconds for 20 nodes, and 9 seconds for 30 nodes, and after which, we observe no further improvement. This is because, at 30 nodes, every node only verifies one operation at a time. One may notice that verifying an ownership assignment should only take 8.5 seconds (from micro-benchmark), rather than 9 seconds as shown in Figure 8.1 (when $n \geq k$). This is due to the delay caused by Hyperledger Fabric implementation of about 0.5 seconds. The same trend is observed for data store operation, where it takes 4.13 seconds to verify a record with 10 nodes, and reduces to 2.51 seconds and 2 seconds with 20 nodes and 30 nodes, respectively.

As can be seen in Figure 8.1b and Figure 8.1c, increasing the number of concurrent data operations increase the workload of blockchain nodes, thereby impacting the overall verification



(a) $k = 25$ **(b)** $k = 50$ **(c)** $k = 75$

Figure 8.1: Verification latency of each data operation

latency. We observe that when doubling the number of concurrent operations, the latency does not doubly increase, but around 2 - 1.5 times. This is due to the implementation of Hyperledger Fabric, which attempts to exploit all the CPU resources (e.g., 2 cores in our setting) to verify all transactions submitted to it. The speedup is not observed when $n = 10$ because the load per node is 5 (when $k = 50$) or 7.5 (when $k = 75$). The 5/7.5 verifications happen at the same time across 2 CPU cores. This requires the CPUs to constantly switch between tasks which incurs a large delay.

Figure 8.2 presents the end-to-end processing delay for each data operation in our system with $n = 30$ blockchain nodes, starting from when the operation is initialized to the time when its record appears in the blockchain. The end to end delay captures the proving time, time for forming a record, verification time and consensus agreement time. In this experiment, we consider that there are upto 100 users who perform the same data operation at the same time.

Memory Usage. We report the distribution of memory usage across the blockchain nodes in our system when verifying the log of each data operation. Figure 8.3 presents (peak) memory used by every node during the verification with different numbers of concurrent data operations and blockchain nodes. As shown in Figure 8.3a, with 25 concurrent data requests and 10 nodes, the inter-quartile range of memory usage to verify a Store/Assign/Share/Ac-

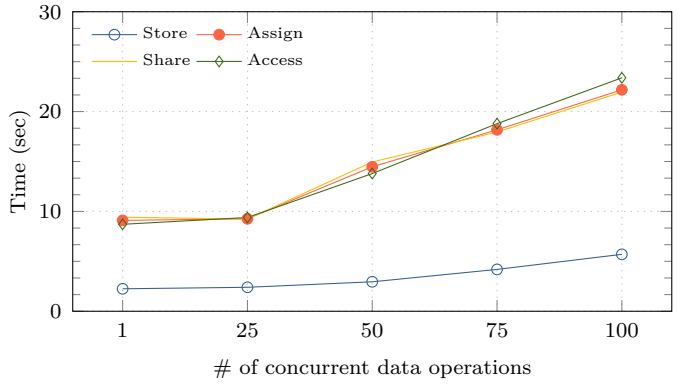
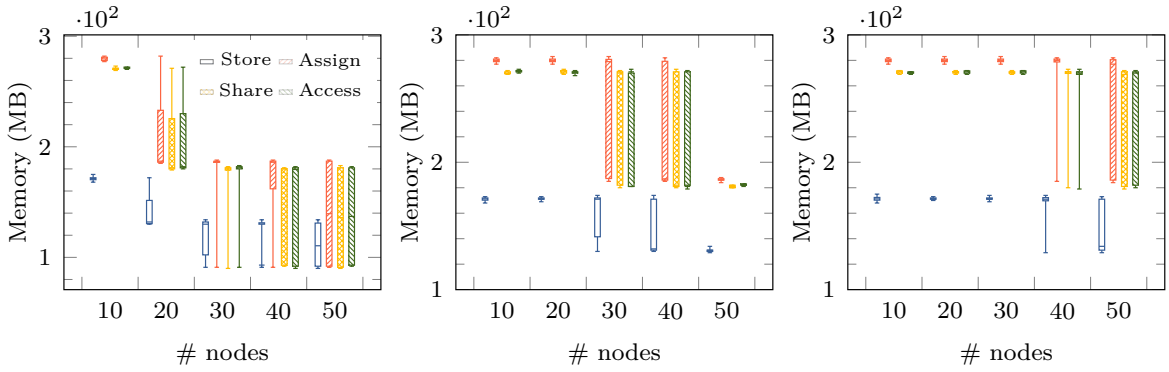


Figure 8.2: End-to-end delay per data operation with 30 blockchain nodes



(a) $k = 25$ **(b)** $k = 50$ **(c)** $k = 75$

Figure 8.3: Memory usage distribution across blockchain nodes

cess operation is about 171/279/271/271 MB, respectively. Similar to latency, we can see that the memory used by nodes to verify data operations reduces when increasing the number of blockchain nodes. For example, the inter-quartile range of memory usage when $n = 20$ and $k = 25$ for a Store operation is 130-151 MB with a median of 132 MB. This is because majority of the nodes verifies 1 Store operation and some handle 2 as seen by the maximum value of 172 MB. When $n = 50$ and $k = 25$, half the nodes verifies one Store operation and the other half are idle, therefore, we observe an inter-quartile range of 92-131 MB with a median of 110 MB. The lower quartile of 92 MB shows the memory needed to run a Hyperledger Fabric node. The upper quartile of 131 MB shows the memory needed to verify

one store request. We can see that there is a 92 MB memory usage difference between micro-benchmark and macro-benchmark, which entirely attributes to the memory needed to deploy a Hyperledger Fabric node as can be confirmed by the lower quartile.

As shown in Figure 8.3b and Figure 8.3c, increasing k increases the memory usage by the blockchain nodes as observed by the tighter inter-quartile range. However, the maximum memory used, regardless of k, is about 174/282/272/271 MB for Store/Assign/Share/Access operations which is equivalent to verifying 2 operations at a time. This is because even though Hyperledger Fabric populates all pending transactions to the CPU, the peak memory usage will be equivalent to verifying 2 transactions at a time since every node only has 2 CPUs.

Chapter 9

Conclusion

We proposed Harpocrates, a privacy-preserving and immutable audit log scheme. Harpocrates achieved immutability and resistance to single point of failure by leveraging blockchain, and achieved public verifiability of record validity using zero-knowledge proofs. We fully implemented Harpocrates and evaluated its performance by using Hyperledger Fabric and Amazon EC2.

As part of future work, we plan to allow access revocation through our system as the current system only permits time based loss of access. Also, since our system does not hide access patterns against the storage provider, there could be a potential side channel attacks if the storage provider is malicious. Analyzing attacks in such a scenario can help develop a stronger scheme and has been left as future work. Another aspect left as future work is the possibility to replace merkle tree with a more efficient data structure to prove set membership. Proving set membership using merkle trees induces the most overhead in our system. RSA accumulators have been shown to be good candidates [6] and may be a good replacement for merkle trees to prove set membership.

Bibliography

[1] Geneyouin - terms and conditions, Oct 2019. URL https://www.geneyouin.ca/terms-conditions/.

[2] Hamed Al-Shaibani, Noureddine Lasla, and Mohamed Abdallah. Consortium blockchain-based decentralized stock exchange platform. *IEEE Access*, 8, 2020. doi: 10.1109/ACCESS.2020.3005663.

[3] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *IACR ASIACRYPT*, 2016. ISBN 978-3-662-53887-6.

[4] Elli Androulaki et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *ACM EuroSys*, 2018. ISBN 9781450355841. doi: 10.1145/3190508.3190538.

[5] Eli Ben Sasson et al. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE S&P*, 2014. doi: 10.1109/SP.2014.36.

[6] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. In *Springer FC*, 2021. ISBN 978-3-662-64321-1. doi: 10.1007/978-3-662-64322-8_19.

[7] Federico Matteo Benčić, Pavle Skočir, and Ivana Podnar Žarko. Dl-tags: Dlt and smart tags for decentralized, privacy-preserving, and verifiable supply chain management. *IEEE Access*, 2019.

[8] Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. Findel: Secure derivative contracts for ethereum. In *Springer FC*, 2017. ISBN 978-3-319-70278-0.

[9] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *Springer CRYPTO*, 2019. ISBN 978-3-030-26948-7.

[10] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE S&P*, 2018. doi: 10.1109/SP.2018.00020.

[11] Emil Chiauzzi and Paul Wicks. Digital trespass: Ethical and terms-of-use violations by researchers accessing data from an online patient community. *J Med Internet Res*, 21(2), Feb 2019. ISSN 1438-8871. doi: 10.2196/11985. URL https://www.ncbi.nlm.nih.gov/pubmed/30789346. PMC6403524.

[12] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. A blockchain based approach for the definition of auditable access control systems. *Computers & Security*, 2019. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2019.03.016.

[13] Kai Fan, Shangyang Wang, Yanhui Ren, Hui Li, and Yintang Yang. Medblock: Efficient and secure medical data sharing via blockchain. *Journal of Medical Systems*, 42(8), Jun 2018. ISSN 1573-689X. doi: 10.1007/s10916-018-0993-7.

[14] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for Zero-Knowledge proof systems. In *USENIX Security*, August 2021. ISBN 978-1-939133-24-3.

[15] Hao Guo, Wanxin Li, Mark Nejad, and Chien-Chung Shen. Access control for electronic

health records with hybrid blockchain-edge architecture. In *IEEE Blockchain*, 2019. doi: 10.1109/Blockchain.2019.00015.

[16] Jane Henriksen-Bulmer and Sheridan Jeary. Re-identification attacks—a systematic literature review. *International Journal of Information Management*, 36(6, Part B), 2016. ISSN 0268-4012. doi: https://doi.org/10.1016/j.ijinfomgt.2016.08.002. URL https://www.sciencedirect.com/science/article/pii/S0268401215301262.

[17] Thang Hoang, Ceyhun D. Ozkaptan, Attila A. Yavuz, Jorge Guajardo, and Tam Nguyen. S^3oram: A computation-efficient and constant client bandwidth blowup oram with shamir secret sharing. In *ACM CCS*, 2017. ISBN 9781450349468.

[18] Yuncong Hu, Sam Kumar, and Raluca Ada Popa. Ghostor: Toward a secure Data-Sharing system from decentralized trust. In *USENIX NSDI*, feb 2020.

[19] Vishal Karande, Erick Bauman, Zhiqiang Lin, and Latifur Khan. Sgx-record: Securing system logs with sgx. In *ACM Asia CCS*, 2017. ISBN 9781450349444. doi: 10.1145/3052973.3053034.

[20] Katherine K. Kim, Pamela Sankar, Machelle D. Wilson, and Sarah C. Haynes. Factors affecting willingness to share electronic health data among california consumers. *BMC Medical Ethics*, 18(1), Apr 2017. ISSN 1472-6939. doi: 10.1186/s12910-017-0185-x. URL https://doi.org/10.1186/s12910-017-0185-x.

[21] Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Linus Gasser, Philipp Jovanovic, Ewa Syta, and Bryan Ford. Calypso: Private data management for decentralized ledgers. *Proc. VLDB Endow.*, 14(4), December 2020. ISSN 2150-8097. doi: 10.14778/3436905.3436917.

[22] Wang Fat Lau, Dennis Y. W. Liu, and Man Ho Au. Blockchain-based supply chain

system for traceability, regulation and anti-counterfeiting. In *IEEE Blockchain*, 2021. doi: 10.1109/Blockchain53845.2021.00022.

[23] Jingwei Liu, Xiaolu Li, Lin Ye, Hongli Zhang, Xiaojiang Du, and Mohsen Guizani. Bpds: A blockchain based privacy-preserving data sharing for electronic medical records. In *IEEE GLOBECOM*, 2018. doi: 10.1109/GLOCOM.2018.8647713.

[24] Xiaoguang Liu, Ziqing Wang, Chunhua Jin, Fagen Li, and Gaoping Li. A blockchain-based medical data sharing and protection scheme. *IEEE Access*, 7, 2019. doi: 10.1109/ACCESS.2019.2937685.

[25] Theo Lynn, Lisa van der Werff, and Grace Fox. *Understanding Trust and Cloud Computing: An Integrated Framework for Assurance and Accountability in the Cloud*. Springer International Publishing, Cham, 2021. ISBN 978-3-030-54660-1. doi: 10.1007/978-3-030-54660-1_1. URL https://doi.org/10.1007/978-3-030-54660-1_1.

[26] Di Ma and Gene Tsudik. A new approach to secure logging. *ACM Trans. Storage*, 5 (1), mar 2009. ISSN 1553-3077. doi: 10.1145/1502777.1502779.

[27] Mourad El Maouchi, Oğuzhan Ersoy, and Zekeriya Erkin. Decouples: A decentralized, unlinkable and privacy-preserving traceability system for the supply chain. In *ACM SAC*, 2019. ISBN 9781450359337.

[28] Mihir Bellare Michel Abdalla and Phillip Rogaway. Dhaes: An encryption scheme based on the diffie-hellman problem. IACR Cryptology ePrint Archive, Report 1999/007, 1999.

[29] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1992. ISBN 978-3-540-46766-3.

[30] Otto Julio Ahlert Pinno, Andre Ricardo Abed Gregio, and Luis C. E. De Bona. Con-

trolchain: Blockchain as a central enabler for access control authorizations in the iot. In *IEEE GLOBECOM*, 2017. doi: 10.1109/GLOCOM.2017.8254521.

[31] Indrajit Ray, Kirill Belyaev, Mikhail Strizhov, Dieudonne Mulamba, and Mariappan Rajaram. Secure logging as a service—delegating record management to the cloud. *IEEE Systems Journal*, 2013.

[32] Alan Rusbridger. The Snowden leaks and the public. In *The New York Review of Books—NYR Daily, 2013*.

[33] Francesco Sardanelli, Marco Alì, Myriam G. Hunink, Nehmat Houssami, Luca M. Sconfienza, and Giovanni Di Leo. To share or not to share? expected pros and cons of data sharing in radiological research. *European Radiology*, 28(6), Jun 2018. ISSN 1432-1084. doi: 10.1007/s00330-017-5165-5. URL https://doi.org/10.1007/s00330-017-5165-5.

[34] C. P. Schnorr. Efficient identification and signatures for smart cards. In *Springer CRYPTO*, 1990. ISBN 978-0-387-34805-6.

[35] Hossein Shafagh, Lukas Burkhalter, Sylvia Ratnasamy, and Anwar Hithnawi. Droplet: Decentralized authorization and access control for encrypted data streams. In *USENIX Security*, August 2020. ISBN 978-1-939133-17-5.

[36] Affaf Shahid et al. Blockchain-based agri-food supply chain: A complete solution. *IEEE Access*, 8, 2020. doi: 10.1109/ACCESS.2020.2986257.

[37] Y. Shen, T.C. Lam, J.-C. Liu, and W. Zhao. On the confidential auditing of distributed computing systems. In *IEEE ICDCS*, pages 600–607, 2004.

[38] Qun Song, Yuhao Chen, Yan Zhong, Kun Lan, Simon Fong, and Rui Tang. A supply-

chain system framework based on internet of things using blockchain technology. *ACM Trans. Internet Technol.*, 21(1), jan 2021. ISSN 1533-5399. doi: 10.1145/3409798.

[39] Smitha Sundareswaran, Anna Squicciarini, and Dan Lin. Ensuring distributed accountability for data sharing in the cloud. *IEEE TDSC*, 9(4), 2012. doi: 10.1109/TDSC.2012.26.

[40] Hui Tian et al. Public audit for operation behavior logs with error locating in cloud storage. *Soft Computing*, 23(11), Jun 2019. ISSN 1433-7479. doi: 10.1007/s00500-018-3038-8.

[41] Kentaroh Toyoda, P. Takis Mathiopoulos, Iwao Sasase, and Tomoaki Ohtsuki. A novel blockchain-based product ownership management system (poms) for anti-counterfeits in the post supply chain. *IEEE Access*, 5, 2017. doi: 10.1109/ACCESS.2017.2720760.

[42] United States Department of Health. The Health Insurance Portability and Accountability Act of 1996.

[43] United States Federal Register. The Federal Acquisition Supply Chain Security Act of 2018 (FASCSA).

[44] Qi Xia, Emmanuel Boateng Sifah, Kwame Omono Asamoah, Jianbin Gao, Xiaojiang Du, and Mohsen Guizani. Medshare: Trust-less medical data sharing among cloud service providers via blockchain. *IEEE Access*, 5, 2017. doi: 10.1109/ACCESS.2017.2730843.

[45] Zhen Yang, Wenyu Wang, and Yongfeng Huang. Ensuring reliable logging for data accountability in untrusted cloud storage. In *IEEE ICC*, 2017. doi: 10.1109/ICC.2017.7997109.

[46] Jiawei Yuan and Shucheng Yu. Public integrity auditing for dynamic data sharing with multiuser modification. *IEEE TIFS*, 10(8), 2015. doi: 10.1109/TIFS.2015.2423264.

[47] Rui Zhou, Mohammad Hamdaqa, Haipeng Cai, and Abdelwahab Hamou-Lhadj. Mobilogleak: A preliminary study on data leakage caused by poor logging practices. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 577–581. IEEE, 2020.