Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Privacy-Preserving Smart Contracts for Industrial Services

Master's Thesis

Marko Pichler Trauber

pimarko@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Dr. David Kozhaya and Dr. Thomas Locher, ABB Corporate Research
Prof. Dr. Roger Wattenhofer

September 26, 2019

Acknowledgements

I would like to thank Professor Roger Wattenhofer and ABB Corporate Research Center Switzerland for giving me an opportunity to write this thesis in their collaboration.

I would like to thank my supervisor, David Kozhaya, for challenging and supporting me throughout the thesis. I am grateful for his patient and masterful guidance. Furthermore, I would like to thank Thomas Locher for all the valuable discussions we had and for supporting me with his immense knowledge while writing this thesis.

To my parents, Sanja and Petar. *This thesis would not have been possible without you. Thank you for your unconditional support every step of the way and the opportunity to follow my dreams. You are my heroes.*

To my family, Justyna and Liam. *I dedicate this thesis to my wife and son who always had a smile for me in time of need. Thank you, Justyna, for encouraging me all along, listening to all my ideas, going with me through ups and downs during my studies, always preparing warm food and countless nights you were taking care of Roo. Thank you for believing in me and giving me strength.*

Abstract

In this work, we propose a system for writing publicly verifiable privacy-preserving smart contracts in distributed ledger technologies (DLT)s. Our approach combines partially homomorphic encryption and zero-knowledge proofs in order to support a wide range of privacy-preserving operations. We showcase the effectiveness and the performance of our system by integrating it in Corda, a mainstream permissioned DLT.

Contents

List of Figures

Chapter 1

Introduction

Industrial service providers define the terms and conditions of the provided service in a *service level agreement* (SLA). Such agreement imposes a set of rules on the involved parties and governs their interaction. For example, an Internet Service Provider (ISP) may have a SLA with a customer. However, ensuring that all rules are indeed satisfied, as well as determining who is liable for violating SLA rules (if such violation exists) remains a very challenging task to realize in practice. To this end, and in order to realize the interaction in a distributed and decentralized way, we propose *distributed ledger technology* (DLT) as the backbone. DLTs remove the need for a *trusted third party* by shifting the trust to the honest majority, who follows specific protocols that prevent malicious entities from corrupting the system. Within a DLT, SLAs are implemented as *smart contracts* and embedded in the transactions, which are stored on a distributed ledger. In order to attest to a transaction's validity, many parties validate smart contracts by executing them on typically sensitive data. As data privacy is not preserved, an exploit of sensitive information may entail serious financial repercussions and reputational damage.

Contributions. We propose a new system for writing smart contracts that preserve the privacy of the distributed ledger data on which these smart contracts operate. Our solution provides privacy-preserving verification leveraging partially homomorphic encryption schemes and zero-knowledge proofs (ZKP) on (un-)signed floating-point numbers and strings. As we aim to support a wide range of secure operations, a fully homomorphic scheme, as presented in Gentry's work [1], would be an ideal candidate. However, subsequent work [2] has shown that current fully homomorphic schemes impose an impractically large computational overhead. Therefore, we combine a partially homomorphic encryption scheme with non-interactive ZKPs. Furthermore, we leverage an additional commitment scheme to overcome the limiting number of operations of the partially homomorphic encryption scheme. The main aspects that characterize our solution can be classified as follows:

Extended Set of Operations. We support privacy-preserving algebraic operations such as addition, subtraction, multiplication, and division on signed floating-point numbers. In contrast to order-preserving encryption [3], which suffers from severe security vulnerabilities as plaintext order is retained in the ciphertext space, we provide a secure way of performing comparisons on ciphertext tuples. Furthermore, leveraging the *ASCII* bijective mapping, our system offers both zero-knowledge set inclusion and ciphertext equality operation on textual data.

Private-key-less Secrecy. Our solution does not require private keys for ZKP generation or decryption of the values.

System Integration. We showcase a way to integrate our system in a DLT to achieve confidential transactions. Privacy-preserving smart contracts enable multiple parties to validate transactions, which in return significantly improves DLT's scalability in case of highly liquid assets in long transaction chains. In addition to improving scalability, we optimize for performance in terms of storage requirements on transacting parties by further expanding the transaction validation process.

Generality. Our system is usable in any scenario where one party has to prove a certain set of boolean expressions consisting of the aforementioned operators to another party without revealing any sensitive information, yet allowing the other party to perform restricted computations on the encrypted data in the intermediate computation steps.

In short, the main contributions of this thesis are:

- A privacy-preserving system with private-key-less secrecy supporting basic algebraic operations. In addition, we support secure comparisons on signed floating-points numbers and both secure set inclusion and equality operation on strings.
- A mechanism to integrate our system within an existing DLT, Corda. Our mechanism includes numerous optimizations to Corda itself enabling higher performance in scalability, storage, and throughput.
- An extensive evaluation of our system's performance and overhead, both standalone and when integrated within Corda.

Chapter 2

Related Work

Our work consists of two different research areas, distributed ledger technology and privacy-preserving verification. We aim to combine the knowledge gathered in these areas in order to provide confidential transactions for distributed industrial service applications.

Distributed Ledger Technology. Our initial selection process for choosing the most applicable DLT for the industrial use cases is based on the work by IBM Research in Zurich [4], which presents a general overview of permission-less and permissioned DLTs. This work discusses the problem of trust in the blockchain protocols and compares both different DLTs and their consensus mechanisms. We have identified Corda [5] and Hyperledger Fabric [6] as potential permissioned DLT candidates. We have considered and investigated other permissioned DLTs, such as Tendermint[1], Symbiont[2], and MultiChain[3]. The deficiencies of these platform include unclear consensus mechanism correctness, proprietary software, and lack of support for Byzantine network participants. Corda and Hyperledger Fabric operate in a permissioned network supporting smart contract code written in high-level programming language, such as Java. Previous work [7] has identified security and performance as one of the most influential DLT properties as well as analyzed the main trade-offs between them. The most interesting ones, confidentiality and auditability remain in our focus as we aim to provide transaction confidentiality without a loss of transparency. A survey on confidentiality- and privacy-preserving technologies for blockchains [8] compares the underlying technologies utilized to satisfy the security triad of confidentiality, integrity, and availability. Such technologies include stealth addresses [9] and ring signatures [10] to achieve anonymity of participants, and Pedersen commitments [11] and ZKPs to achieve transaction confidentiality. Corda's research survey [12] identifies the top ten obstacles for the adoption of distributed ledgers and argues that scalability and privacy represent the biggest barriers along the way. In ad-

[1] https://tendermint.com/
[2] https://symbiont.io/technology
[3] https://www.multichain.com/

dition, the survey states how the research has focused more on user anonymity in a distributed ledger rather than on their privacy, i.e., the confidentiality of their transactional data. Challenges in scaling DLTs, with respect to the underlying network, consensus protocols, storage requirements, and on-chain computations, have been explored [13]. Among the proposed techniques are delegation of trust, sidechains, and outsourcing transaction validation by leveraging cryptography. Our solution provides transaction confidentiality and improves both the scalability and storage requirements on network participants in DLTs such as Corda.

Confidential Computation. Partially homomorphic encryption has been utilized for confidential computation in numerous systems. CryptDB [14] is an example system providing encrypted query processing building on onion encryption layers. The inner most layer represents the least secure, deterministic encryption of a value, whereas the outer most layer is encrypted using a secure, randomized scheme. As operations are performed, onion layers are stripped down trading-off the security for functionality. These operations include searching, ordering, comparing, and adding values. SecureScala [15] has been proposed as a domain-specific language used to express programs without any cryptographic knowledge background. It leverages multiple encryption schemes to support a wide range of operations including additions, multiplications, equality, and ordering. The homomorphic addition property is satisfied using the Paillier encryption scheme [16]. The ElGamal encryption scheme [17] supports homomorphic multiplication property. On the other hand, equality operation is leveraging a symmetric encryption scheme such as randomized Advanced Encryption Standard (AES) [18] and the ordering is supported by order-preserving encryption [3]. MorphicLib [19] is a partially homomorphic library, which closely follows the solution provided in CryptDB. It is offering partially homomorphic functions, which support executing queries on encrypted data. In comparison, our approach offers a wider, more secure and more flexible set of operations by utilizing a combination of ZKPs and a partially homomorphic encryption scheme. This approach avoids utilizing multiple encryption schemes in an onion-layered fashion [14], [19], which requires both excessive key generation and management. Our approach is more secure, as it does not rely on a deterministic encryption scheme, which is vulnerable to chosen-plaintext attacks [20].

Confidential Transactions in DLTs. Previous studies have shown interest in achieving confidentiality of transactions in distributed ledgers. One of the first systems that started the research on privacy preservation in distributed ledgers was Zerocash [21]. Zerocash embeds the Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) into the transaction. No complex logic or scripting capabilities are provided and a major criticism to the system is the need for a trusted setup of initial zero-knowledge parameters. Zether [22], a confidential payment mechanism for account based models, specif-

ically compatible with Ethereum, operates on encrypted balances and exposes methods to perform a restricted set of operations through cryptographic proofs. Zether's smart contracts contain the efficient ZKP system Σ-Bullets to provide the transaction logic. Several platforms, such as Arpa [23] and Arbitrum [24] provide privacy-preserving smart contracts which are executed in their Turing complete Virtual Machines (VM). Arpa offers secure and verifiable computation by transforming high-level functions into arithmetic circuits, which are evaluated by parties using a Multi-party Computation (MPC) protocol. Raziel [25] is a system similar to Arpa's in that it provides both private and verifiable smart contracts. Smart contracts are written in Obliv-Java, a modified OPEN-JDK for secure computation. Enigma [26] is a decentralized computation platform providing a scripting language that performs MPC on the arithmetic circuit. Another interesting approach has been taken by the inventors of Hawk [27], where smart contracts have been split into public and private contracts. A public contract includes a proof that the private contract has been executed correctly. A drawback of this approach is the manager entity, which executes private contracts and, therefore, should be trusted. In the finance sector, a ZkLedger [28] system has been proposed to provide a solution based on ZKPs using Pedersen commitments for both confidential ledger information and auditing purposes. The system is providing audit proofs for operations such as ratios, percentages, sums and averages, while keeping the balances stored on the ledger hidden using the Pedersen commitment scheme. Offline computation and attestation to its validity on-chain has been investigated and proposed by Zexe [29]. By contrast, our solution does not rely on secure enclaves, VMs or arithmetic circuits, but rather enables non-experts in cryptography to write privacy-preserving smart contracts by integrating our portable system into their system infrastructure. Furthermore, in comparison to Zether [22], our system is applicable to UTXO models, supports secure operations on signed floating-point numbers and strings, and provides more generic secure operations such as comparisons on ciphertext tuples.

Chapter 3

Models

In this chapter, we provide an overview of the considered DLT model as well as the assumed adversary model.

3.1 System Model

DLT Model. The underlying DLT model is the unspent transaction output (UTXO) model operating on a permissioned network. In contrast to an account-based model, as used, e.g., in Ethereum[1], the UTXO model operates with states. States are contained in transactions and each transaction consumes and creates state(s). The sum of unspent states represents the owned assets. In a permissioned network, as opposed to a permission-less one, such as Bitcoin[2], all the participants' identities are known. We assume network participants to represent the nodes in the network, i.e., system's entities, whereas clients represent the owners of these entities. A client might be a single person or an entire organisation. Transaction life-cycle involves processes, which should be executed by each transaction participant. A process includes signing, verifying, and storing a transaction. All the processes are atomic, hence no partial process execution is allowed. The communication relies on secure and weakly synchronous channels between network participants. The secure channels are realized using the remote procedure call framework *gRPC*[3], which utilizes the *Advanced Message Queuing Protocol/1.0* (AMQP)[4] over Transport Layer Security (TLS) 1.3.

Corda. We have selected Corda as a UTXO model representative. The rationale behind our choice is backed up by a list of key advantages that Corda as DLT has to offer. Among the most important ones being the communication on a need-to-know basis, which offers a way to tackle the fundamental properties

[1] https://www.ethereum.org/
[2] https://bitcoin.org/bitcoin.pdf
[3] https://grpc.io/docs/guides/auth/
[4] www.amqp.org/

of DLTs - scalability and privacy. We are utilizing Corda as an underlying messaging system to showcase usage scenarios of our system for privacy-preserving industrial service applications. Since information in Corda is disseminated on a need-to-know basis, there is no global ledger. Instead, different parts of the ledger are stored with different nodes of the system. Under this mode of operation, Corda utilizes two consensus mechanisms to ensure uniqueness and validity of transactions. Double-spending is prevented in Corda using uniqueness consensus. This consensus is achieved by the notary service, a cluster of entities, that keep track of the states' consumption. If a state has been previously spent, the notary service will prohibit the state from being consumed in a different transaction. On the other hand, as Corda supports smart contracts that govern the transaction's validity, a validation consensus exists as well. All the participants, involved in a transaction, have to validate the transaction by executing the logic defined in the smart contracts. This process involves access to all the transaction details, hence potentially sensitive data. The clear disadvantage arising from a need-to-know information spread is a transaction dependency resolution. More specifically, a party that is new to a long transaction chain, has to "walk the chain" in order to verify each transaction it has not verified up to the issuance transaction. This introduces a computational bottleneck in case of highly liquid assets. An alternative approach to having participants validate transactions, is to delegate this job to the notary cluster. As opposed to having peers perform transaction resolution, the notary service attests to the validity of a transaction chain and hence removes the need of "walking the chain". Additionally, the sharding effect is enhanced as transactions are, at any point in time, provided only to the relevant participants. The downside of validating notaries is the clear violation of Corda's original notion of confidentiality as notary service obtains access to all the data stored across the transactions in the network. Corda's communication takes place in flows [5], which are small multi-party sub-protocols. These blocking protocols allow suspending and resuming communication between parties. A transaction itself goes through the following stages prior to being placed on the ledger:

- *Transaction builder.* Gathers input and output states.
- *Wire transaction.* Serializes the transaction object, i.e., transforms it into a series of bytes.
- *Signed transaction.* Appends the participants' signatures to a *Wire transaction.*
- *Ledger transaction.* Derived from *Wire transaction* and used during the transaction verification process. It includes de-serialising states.

3.2 Threat Model

Corda's threat model following the STRIDE framework[5] has been analysed in prior work.[6] We assume $n = 3f + 1$, where n corresponds to the number of network participants and f to the maximum number of malicious, Byzantine participants in the system. This assumption is important in order to achieve a consensus among network participants. As we are considering a permissioned DLT, where identities of network participants are known, the network is assumed to be resilient to *Sybil attacks*. In a *Sybil attack*, an adversary controls a majority of network participants, therefore having the largest influence, which may be used to corrupt the system. We assume a computationally bounded adversary, who may be passive or active. A passive adversary is assumed to be an honest, but curious entity. Such an entity may read the data in transit or on the ledger, but may not modify it. On the other hand, an active adversary is additionally assumed to be able to modify the data. We assume that an active adversary may not delay or drop the data in transit. We divide the threat model into process and communication threats. Under process threats we consider any action performed by a network participant, whereas communication corresponds to serialized transaction objects in transit.

Active Adversary. In the following, we present an overview of security threats that an active adversary, who may be a network participant, poses.

Communication Threats. We assume that an adversary may not delay or drop data in transit. On the other hand, it may read and modify it. Due to the utilization of secure channels, which provide confidentiality, integrity, and mutual authenticity, all the considered communication threats are mitigated.

Process Threats. Following Corda's threat model[7], the following summarizes and enhances possible attacks.

Spoofing. An adversary may impersonate an existing organisation to join the Corda Network.

Tampering. An adversary may gain access to the network participant's local storage and modify the data. Such an attack would not affect other network participants, but would compromise the transaction history of the specific participants. Furthermore, compromising the network map, an entity responsible

[5] https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)

[6] https://docs.corda.net/design/threat-model/corda-threat-model.html

[7] https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)

for storing the participants' identities, could lead to unauthorized addition or removal of network participants.

Repudiation. There are no threats if standard security measures are in place. More specifically, as Corda's transactions require digital signatures, an attacker is identifiable and any malicious action may be traced to its origin.

Information disclosure. An adversary may try to learn more information about the transaction than it should. Due to the assumed secure communication between network participants, the only attack strategy left to the adversary is to pose as a validating notary or "enter" a long transaction chain in order to observe the historic transaction data during the transaction resolution process. Furthermore, an adversary may gain access to the network participant's local storage and read the data. However, if standard security measures are in place, i.e., a secure authentication mechanism, gaining access to a network participant's local storage does not pose a threat.

Denial of service. An attacker may send a high volume of invalid transactions to a specific node. As Corda disseminates information on need-to-know basis, other network participants are not affected by such an attack. An adversary participating in a flow may fail to respond to an honest participant, who may suffer financial losses. Furthermore, an adversarial party may repeatedly and on purpose fail to verify the transaction, which prohibits the transaction to be placed on the ledger. This attack does not pose a threat as long as the maximum number f of such adversaries in the network does not reach $\frac{n}{3}$, i.e., as long as $f < \frac{n}{3}$.

Elevation of privilege. An adversary may execute malicious code on another network participant. This attack involves sending malicious transaction data, which a network participant executes while validating the smart contract code.

Passive Adversary. We consider a passive adversary which may read the data stored on the ledger and the data in transit. As transaction data may contain sensitive information, an access to this data would violate the transaction's confidentiality. Due to the utilization of secure communication channels, which provide confidentiality, integrity, and authenticity, a passive adversary does not pose a communication threat. However, a passive adversary poses an *information disclosure* process threat, i.e., it may read out more information than it should. In what follows, we elaborate on this process threat.

Transaction resolution information disclosure. A passive adversary can obtain all the transaction data associated with a long transaction chain even if it has not taken part in any transactions in that chain, except the last one. The passive

adversary gets access to all the transaction data, as it is, according to Corda's transaction protocol, required to perform the transaction verification of each transaction in the transaction chain. This time-consuming transaction resolution process may leak sensitive data that the adversary should not have an access to, as it has not participated in any of these transactions. This threat does not contradict the definition of a passive adversary, because the adversary simply follows the given transaction protocol. On the other hand, a validating notary would remove the need for the transaction resolution, i.e., it would prevent a passive adversary from gaining an access to other participants' sensitive data. But, a validating notary itself would gain access to all the data from all the transactions, hence, considering a passive adversary posing as a notary, results again in violation of transaction privacy.

In this thesis, we address the *information disclosure* threat, for both passive and active adversary, in a novel way by providing privacy-preserving transactions.

Chapter 4

Problem Statement

We aim to offer a system that satisfies Kerckhoff's principle of cryptography [30], which states that a system should be secure under the assumption that everything, but the private key material, is known to the public. Therefore, our system should allow any network participant to act as a notary service by placing enough public information in the transactions to attest to their validity, yet without revealing any sensitive information. In the following, we present the functionality and security requirements our solution should guarantee.

4.1 Functionality

Operations. Each secure operation should either attest to the validity of the claim made on a data or be used in a computation of a new value. Such computation must be validated by the other parties in a zero-knowledge fashion leveraging either homomorphism of the underlying encryption scheme or validation of ZKPs. We require a sufficient operation coverage for expressiveness of industrial smart contracts.

Notary Service. The parties representing the cluster of notaries should not be any external trusted third parties or belong only to a part of the permissioned network peers. Therefore, anyone willing to participate as a notary may be granted to do so without violating the confidentiality requirement, which we will define later in this section. If multiple parties are participating in a notary service, a consensus protocol must exist. We differentiate between crash-tolerant and fault-tolerant protocols currently supported in Corda's enterprise version. Given that malicious entities are expected in the system, a Practical Byzantine Fault Tolerance algorithm [31] may be considered. On the other hand, if our system should be only crash-tolerant, a computationally faster consensus protocol , such as Raft [32], is applicable.

Work Flow. The transaction initiator is the entity generating the proof associated with the state's smart contract. Issuance transaction involves the cryptographic material generation, where the private key is destroyed or kept securely off-line. Transfer transactions do not require re-generation of the cryptographic material, but rather operate under the public keys generated in the issuance transaction. Each entity in the system may validate proofs utilizing the encrypted data and ZKPs attesting to the transaction's validity.

Storage. Each party may locally store only the transactions in which it participates. In case of a long transaction chain, we want to avoid storage of a significant amount of information that a party should not have access to and may not wish to store.

Scalability. Peers should not perform transaction resolution, but rather leverage the notary's signature attesting to the validity of a transaction chain.

Availability. Transaction resolution time is the main indicator of the computational performance. Resolution time is defined as the time needed for successful transaction completion, which includes building, signing, validating, and placing the transaction on the ledger. System availability should not be violated by the imposed computational overhead for the privacy-preserving smart contracts. An acceptable upper bound depends on the smart contract's complexity, i.e., is use case specific. Furthermore, we assume eventual consistency, i.e., every created and valid transaction will be eventually placed on the ledger.

4.2 Security

Transaction Confidentiality. We identify confidential transactions as the main non-functional requirement of the system. The parties participating in a transaction should be the only ones with access to the raw underlying data, whereas encrypted data is provided for the other network participants. The validating parties should be able to perform data-agnostic, yet complete transaction validation.

Transaction Integrity. The transacting on- and off-ledger data should not be malleable and violate the integrity constraint. Such a violation would result in the inability to validate transactions.

Authenticity. The underlying communication must be realized using secure channels over TLS 1.3 with *gRPC* framework to ensure mutual authentication,

data confidentiality, and integrity.

Transaction Correctness. Each validated transaction should be accompanied with the corresponding ZKP attesting to its validity. Furthermore, any invalid ZKP generation must fail in the verification phase. Each validating party performs exactly the same verification and its output is deterministic.

Transaction Atomicity. A transaction should be placed on the ledger of all or none of the involved transaction participants. Therefore, no partial execution is allowed.

Non-repudiation of ZKP Generation. The participant that generated the ZKP is uniquely identified and its identity can not be disputed.

Non-contributory Key Agreement. We require an initial cryptographic key material generation, associated only with the issuance transaction, to be performed by the transaction initiator. We assume a non-contributory key agreement, which means that only the transaction initiator contributes to the key generation. In case of a computationally weaker entity, this may result in suboptimal pseudo-randomness used in the key generation.

Chapter 5

Methods

5.1 Overview

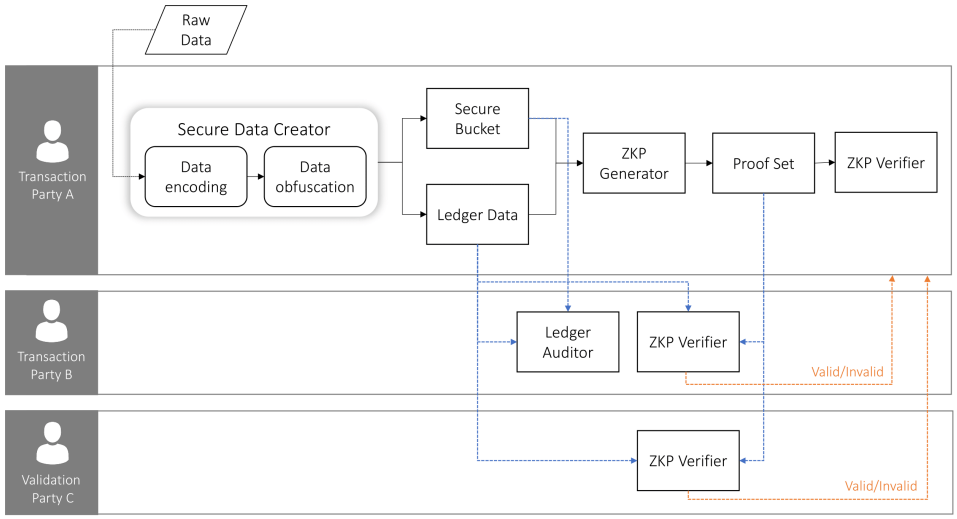

Figure 5.1: System level overview of components and flows.

We present a system, which when augmented to a DLT, enables peers within the DLT to perform privacy-preserving function validation. Our system provides privacy-preserving set of operations to implement service level agreements (SLA) and transforms the data on which peers transact to secured and private representation. Hence, the output of our system is a privacy-preserving contract and encrypted transaction data that can be posted on a distributed ledger. SLAs define the general terms and conditions governing a certain process between two or more participants. Generally, such participants need to perform actions (represented as transactions), which are only accepted if they satisfy SLA rules. For example, *company A* needs to perform maintenance action on a transformer owned by *party B* on a weekly basis and when wind speed is less than a cer-

tain threshold certain actions shall be performed. Transaction data as well as SLA data is visible to *party A* and *party B*, however, it should not be visible to any other parties that are delegated to perform only transaction validation. The privacy-preserving digital contract and encrypted transaction data enable any party on the distributed ledger to validate transactions, i.e., check if each of the encoded terms and conditions in the digital SLA truly holds for a given transaction, without revealing anything about the data used in the digital SLA and the transaction data. Privacy-preserving smart contract leverages additive homomorphic property of Paillier's encryption scheme, Fujisaki-Okamoto commitment scheme and numerous ZKPs which we define in the upcoming §5.3.2. We recall that our system is augmented to a DLT in order to provide higher security and privacy of data and contracts. In what follows, we list the set of components that make up our system.

1. *Secure data creator.* Creates a secure instance $\tilde{x}$ of the raw data input x by encoding, encrypting and generating the commitment to the data. The secret values used for the encryption and commitment, i.e., r_x and s_x are contained in $\tilde{x}$.

2. *DLT data.* Holds encrypted data open to the public, which is used for proof verification and homomorphic operations.

3. *Secure bucket.* Contains the secure instance $\tilde{x}$, which includes secret values r_x, s_x and the mantissa m_x. As the encoding base b and exponent e_x are publicly available, secure buckets provide an on-demand access to actual plaintext data when necessary, e.g., for auditing purposes.

4. *ZKP generator.* Generates the serializable *ProofSet* that can be used in the verification process. This proof is represented as a *HashMap*, where the privacy-preserving function is the key and the proof attesting to the function's validity is the value.

5. *ZKP verifier.* Uses the proof generated by the *ZKP generator* to attest to the validity of the claimed function execution on a specific input. The verification leverages homomorphic properties of Paillier's encryption scheme and goes through a step-by-step agnostic verification process of the supported operations which are presented in §5.3.2.

Figure 5.1 depicts the flow between our system's components, given a network consisting of transacting *parties A* and *B* and a validating *party C*. The main difference between the entities is as follows. *Party A* is the transaction initiator, hence performs the ZKP generation attesting to a validity of a transaction with the *party B*. *Party B* may verify the transaction's validity by running the ZKP verification, but may as well request the secure bucket which opens the data for auditing purposes. *Party C* is a validating party which may only attest to the

transaction's validity in a zero-knowledge fashion and may not gain access to the raw data.
In the following points, we present the interaction between the system's components based on Figure 5.1:

- In the first step *party A*, the initiator of some transaction over some data, feeds that data x to the local *secure data creator* component provided by our system.
- Upon the reception of such input, the *secure data creator* component generates a private secure representation $\tilde{x}$. This private secure representation consists of two main parts:
  - The public block: This includes encrypted data and public keys. The public block is written to the distributed ledger.
  - The private block: This includes information that makes it possible to break the cryptographic schemes. It has access to the plaintext data and the secret random numbers r_x, s_x. The private block is contained in a *secure bucket* component, which may be transferred over a secure channel.
- As a next step, *party A* feeds the public and private part to the local *ZKP generator*. The output of the *ZKP generator* is a serializable *ProofSet* containing the needed proof material that enables parties to check if submitted transactions are valid from the SLA's perspective without knowing the data values within the transaction being validated. The *ProofSet* is signed by the proof originator, thus providing authenticity and replay attack resilience over TLS communication.
- Finally, as opposed to transacting parties, validating parties (e.g., *party C* in Figure 5.1) have access only to the public ledger data and may validate transactions upon the receipt of a *ProofSet* by feeding it to the local *ZKP verifier* component.

Our system supports validating various algebraic operations in a privacy-preserving manner, i.e., without revealing any information about the operands. Below is a an overview of the supported operations. More detail about how this operations are realized is given in §5.3.2.

- *Range proof.* Used to prove that a number lies in a certain interval without revealing the number itself.
- *Addition, Subtraction, Multiplication, Division.* Prove that the computation has been performed correctly without revealing any of the operands including the result.

- *Set inclusion.* Proves that a value, which is either a number or a string, is contained in a set of values without revealing the value or the set of values.
- *Comparisons.* Enable comparisons of two values without revealing the operands or their ciphertext order.

The underlying communication between the network participants represented in Figure 5.1 is over TLS using *gRPC*. Therefore, transferring secure buckets and proof sets is assured to be confidential and authentic, while preserving the integrity. We may impose additional digital signatures on the proof sets and secure buckets to provide a non-repudiation property and combat the possibility of replay attacks. In such a scenario, a party would reject a proof in which the proof generator is not the same party as the one that transferred the proof.

5.2 Notation

We adapt the notation from Zether [22] to define the ZKPs as follows:

$$st : (\underbrace{a, b, c, ...}_{\text{public}}; \underbrace{x, y, z, ...}_{\text{private}}) : \underbrace{O_p(a, b, c, ..., x, y, z, ...)}_{\text{objective}} \tag{5.1}$$

Equation 5.1 states that given the public knowledge a, b, c known to all verifying parties, the proof generator party uses its private knowledge x, y, z to generate a proof p such that objective function O_p holds if p attests to the statement's st validity. Such statement st may be verified by evaluating p without the private knowledge. In the following we would like to introduce the notation used throughout this report.

- x : raw numerical/string value with encoding $x = m_x b^{e_x}$, where:
  - m_x : the mantissa of x.
  - b : the encoding base.
  - e_x : the encoding exponent.
- C_x : Paillier encryption of m_x.
- r_x : the randomness in C_x.
- $Comm_x$: Fujisaki-Okamoto commitment to m_x.
- s_x : the security parameter in $Comm_x$.
- $\tilde{x}$: the *secure data creator.* (Figure 5.1) instance of x.
- pk_{st} : Paillier's public key associated with the statement st.

5.3 Implementation

5.3.1 Background and Tools

In order to support a wide range of privacy-preserving operations defined in the upcoming §5.3.2, we utilize an encryption and a commitment scheme. The strength of our approach lies in the flexibility of choice for the encryption and commitment scheme. The following table summarizes all supported operations and tools used to implement them. The *Numbers* are considered to be real numbers.

Supported privacy-preserving operations.

	Supported operands	**Cryptosystem**
Addition, *Subtraction* *Multiplication*, *Division*	*Numbers*	*Paillier*
Interval (Range)	*Numbers*	*Fujisaki-Okamoto*
Set Inclusion	*Numbers*, *Strings*	*Paillier*
Comparisons	*Numbers*, *Strings*	*Paillier*, *Fujisaki-Okamoto*

Encryption Scheme. We utilize Paillier's partially homomorphic probabilistic encryption scheme [16]. Let g be an element of $\mathbb{Z}^*_{n^2}$, randomness $r_x \in \mathbb{Z}^*_n$ and value $x \in \mathbb{Z}_n$, then Paillier's encryption of x, C_x, is defined as:

$$C_x = g^x r_x^n \bmod n^2 \tag{5.2}$$

The encryption scheme has to satisfy the property of *additive homomorphism*, which states that an addition of plaintexts corresponds to a multiplication of their respective ciphertexts. More formally, given a decryption function D, the property of additive homomorphism is defined as:

$$\forall x, y \in \mathbb{R}_n, \exists C_x, C_y \in \mathbb{R}_{n^2} : D(C_x C_y \bmod n^2) = x + y \bmod n \tag{5.3}$$

Furthermore, the encryption scheme should be *non-deterministic*, i.e., multiple ciphertexts may encrypt the same plaintext. As r_x is chosen randomly for each encryption of value x, Paillier's additive homomorphic encryption satisfies non-deterministic property.

Commitment Scheme. A commitment scheme may have either a *computationally* or *unconditionally binding* and/or *hiding* property. Given a computationally unbounded adversary, a computationally binding/hiding property may be broken, whereas the unconditional one remains secure. The hiding property

states that a given commitment may be equally likely committing to any given plaintext, whereas the binding property states that a commitment may be opened in only one way. Opening a commitment reveals the secret material used for its creation. We pose the requirement of unconditionally binding and computationally hiding property on the encryption and commitment scheme. The main reason behind the choice of the commitment scheme is its applicability to the efficient Boudot's range proofs [33], which we incorporate in our ZKPs defined later in this chapter. Boudot's range proofs [33] utilize the Fujisaki-Okamoto commitment scheme to provide an efficient way of verifying that an integer lies within a specific interval. Let g, h be elements of $\mathbb{Z}_n^*$, security parameter $s_x \in \mathbb{Z}_n^*$, $x \in \mathbb{Z}_n$, then Fujisaki-Okamoto commitment to value x, $Comm_x$, is defined as:

$$Comm_x = g^x h^{s_x} \bmod n \tag{5.4}$$

Zero-Knowledge Proof - ZKP. We are utilizing zero-knowledge proof [34] of knowledge. ZKP provides a way for a participant, called *prover*, to prove to another participant, called *verifier*, that the *prover* knows certain information without revealing this information. More specifically, a ZKP should satisfy the following properties[1]:

- *Completeness.*: The *verifier* always accepts a valid ZKP.
- *Soundness.* The *verifier* always rejects an invalid ZKP.
- *Zero-Knowledge.* The only knowledge a ZKP reveals is that it is correct.

We differentiate between *interactive* and *non-interactive* ZKPs. In an interactive ZKP, a protocol between the *prover* and the *verifier* consists of multiple interaction rounds. The *verifier* poses a challenge to the prover, such that upon prover's response, the *verifier* is assured of prover's secret knowledge. In order to obtain a non-interactive ZKP, we apply Fiat-Shamir [35] heuristic. However, as Fiat-Shamir heuristic utilizes hash functions, our supported operations are considered secure under a random oracle model (ROM), i.e., we assume the existence of an ideal hash function.

Mechanism. Due to the randomization property of Paillier's encryption scheme, we may operate under the same public key encrypting the values and leveraging the randomness r_x as a secret key known only to the one that encrypted the value. In fact, as proven in Paillier's original paper (*Lemma 3* [16]), any pair of a value x and randomness r_x mapped to the Paillier's encryption is bijective. Due to the randomization and the bijection property of Paillier's encryption scheme, we use it as a commitment. We elaborate further about this in what follows.

[1] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4032740/

Lemma 5.1. *Paillier's encryption scheme offers an unconditionally binding and computationally hiding property.*

Proof. In order to prove the unconditionally binding property, we refer to *Lemma 3* from Paillier's original paper [16], which proves the bijection between the plaintext and ciphertext. We recall that the encryption function takes a tuple (x, r_x) from the plaintext space and generates a ciphertext $C_x = g^x r_x^n \bmod n^2$. According to the aforementioned *Lemma 3* [16], ciphertext C_x is uniquely determined by the tuple (x, r_x), hence the unconditionally binding property holds.
In order to prove the computationally hiding property, we refer to the underlying semantic security assumption of Paillier's encryption scheme, which is based on the *Decisional Composite Residuosity Assumption* (DCRA) [16]. According to *Conjecture 2* [16], DCRA [16] is intractable, i.e., it is considered to be a computationally hard problem. Considering a computationally unbounded adversary, the hiding property may be broken, hence a relaxation in form of a computationally hiding property under DCRA holds. □

According to Lemma 5.1, we may utilize Paillier's encryption scheme as a commitment, where a commitment is defined as the encryption C_x. Opening a commitment involves exposing the tuple (x, r_x). Furthermore, as we want to keep the raw data x hidden, i.e., hide the underlying tuple (x, r_x), and allow verifiable computation on the encrypted value C_x, we utilize ZKP of knowledge of randomness r_x, explained in detail in §5.3.2. Using Paillier's encryption scheme as a commitment allows us to omit the private keys needed for the decryption as revealing the secret randomness and the original encoded plaintext value replaces the decryption. As generating ZKP does not require decryption of the underlying values, they may operate without the private keys. The needed knowledge for ZKP generation is the secret randomness r_x, which we transmit in a secure fashion using the concept of secure buckets, explained in §5.1.

Floating-point numbers. Supporting signed floating-point values may leak additional information about the operands in questions in case the exponents are large positive numbers. In the following, we elaborate about the security vulnerability in detail. As we encode numbers using publicly known base and exponent, while keeping the mantissas private, the knowledge of the base and the exponent may infer additional information of the underlying raw value. Due to the fact that mantissas are natural numbers, the knowledge of the exponent and the base may provide a lower bound on the underlying raw value. This is considered a security vulnerability. As we operate under a malleable encryption scheme, we provide enforcement of low maximum exponent values, which reduce the gained additional information on the lower bound of the raw values, hence providing more security.

Libraries. We have adapted an implemented Paillier library [36] to fit our purposes, more specifically the flow used for the encryption and serialization of the values. Some of the adaptations we have performed include an encryption with a known randomness as well as an encryption with the stored randomness, as we are leveraging Paillier's encryption scheme in form of a commitment scheme. Paillier's encryption scheme is used for the privacy-preserving basic arithmetic operations, set inclusions and comparisons. Furthermore, we are building upon range proofs [33] implemented in ING's library [37] that we adapted for proofs serialization. We extend the library's functionality by providing range proofs on both negative as well as floating point numbers. Range proofs are used both in privacy-preserving comparisons and interval operations.

5.3.2 Supported Operations

In the following, we present a detailed overview of the supported operations.

Ciphertext Equality

We have implemented a ciphertext equality ZKP that builds on the protocol for proving the knowledge of the *n-th* power [38]. This ZKP verifies that two ciphertexts, C_x and C_y, hide the same value. More formally, we are proving the following statement:

$$\begin{aligned} Proof_{Eq_{cipher}} : \{(C_x, C_y, g, n, u, b, e_x, e_y, pk_{st}; x, y, m_x, m_y, v, r_x, r_y) : \\ C_x = g^{m_x} r_x^n \bmod n^2 \wedge C_y = g^{m_y} r_y^n \bmod n^2 \wedge m_x = m_y \\ \wedge\ u = C_x (C_y)^{-1} = g^{m_x - m_y} (r_x r_y^{-1})^n = (r_x r_y^{-1})^n = v^n \bmod n^2\} \end{aligned} \tag{5.5}$$

As our privacy-preserving operations operate with floating-point numbers, the real values x and y are encoded. As a result, base, b, and exponents, e_x and e_y, are public, whereas the mantissas, m_x and m_y, are private and encrypted. We prove that two ciphertexts C_x and C_y encrypt the same value by proving the knowledge of randomness $v = r_x (r_y)^{-1}$, i.e., the quotient of the randomness from the ciphertexts involved in the computation.

Proof of correctness. In order to prove the correctness of *ciphertext equality* operation, we rely on Lemma 5.1, the additive homomorphism of the underlying encryption scheme, and *Lemma 3* [38].

Given two ciphertexts, C_x and C_y, that claim to encrypt the same value, i.e., $x = y$, the additive homomorphic property of the underlying encryption scheme ensures that the following holds:

$$\forall x, y \in \mathbb{R}_n, \exists C_x, C_y \in \mathbb{R}_{n^2} :\ x = y \iff C_x C_y^{-1} = g^{x-y} (r_x r_y^{-1})^n = (r_x r_y^{-1})^n \tag{5.6}$$

What remains is for the *prover* to provide a ZKP that attests to the knowledge of $r_x r_y^{-1}$ without revealing r_x, r_y, and $r_x r_y$. The protocol in [38] provides such a proof, however it does so for the interactive case where parties communicate, namely the *verifier* challenges the *prover*. In order to minimize the computational and communication overhead needed, our approach relies on a non-interactive version of the protocol in [38]. Our non-interactive version is realized using the Fiat-Shamir heuristic [35]. Applying this heuristic to any secure protocol ensures that the security of this protocol remains intact assuming a *random oracle model* [35].

Addition and Subtraction

Due to the additive homomorphic property of our encryption scheme, we may provide a proof of valid computation by leveraging the multiplication and division of two ciphertexts resulting in an encryption of zero value. More formally, privacy-preserving addition of values x and y is defined by the following statement:

$$\begin{aligned} Proof_{Add} : \{(C_x, C_y, C_z, b, e_x, e_y, e_z, pk_s; x, y, z, m_x, m_y, m_z, r_x, r_y, r_z) : \\ C_x = g^{m_x} r_x^n \bmod n^2 \wedge C_y = g^{m_y} r_y^n \bmod n^2 \wedge C_z = g^{m_z} r_z^n \bmod n^2 \\ \wedge\ C_x C_y = g^{m_x + m_y} (r_x r_y)^n \bmod n^2 = g^{m_z} {r'_z}^n \bmod n^2 \\ \wedge\ C_x C_y {C_z}^{-1} = ((r'_z) {r_z}^{-1})^n \bmod n^2 \wedge x + y = z \bmod n\} \end{aligned} \tag{5.7}$$

Prior to multiplying the ciphertexts, we perform an encoding and encryption step. As a result, we obtain ciphertexts C_x, C_y, and C_z, where the mantissas m_x, m_y, and m_z are encrypted. Subtraction is based on the additive inverse of the encoded, i.e., encrypted number. Division is using modular multiplicative inverse of Paillier randomness, which is required to verify the ciphertext equality. We support addition and subtraction on signed and floating-point numbers. As Paillier's encryption scheme satisfies the property of additive homomorphism, the following holds:

$$\forall a, g, r, n \in \mathbb{R}_n : \prod_{i=1}^{k} C_{a_i} = g^{\sum_{i=1}^{k} a_i} (\prod_{i=1}^{k} r_i)^n \bmod n^2 = g^{s_a} r_{s_a}^n \bmod n^2 = C_{s_a} \tag{5.8}$$

The above equation states that proving k correct plaintext additions corresponds to proving k correct ciphertext multiplications, where the proof itself consists of proving the knowledge of r_{s_a}, where s_a represents the plaintext sum and r_{s_a} the product of the plaintexts' randomness. In order to complete the proof, the proof generator's commited encryption to value C'_{s_a}, has to be verified. We utilize division of C'_{s_a} by the verifier's computed ciphertext C_{s_a} and the $Proof_{Eq_{cipher}}$ to provide a proof of ciphertext equality. As this proof operates on

a single secret knowledge r_{s_a}, it leads to a constant addition/subtraction ZKP generation/verification time and proof size.

Proof of correctness. In order to prove the correctness of the *addition* and *subtraction* operation, we utilize the additive homomorphism of the underlying encryption scheme and the proof of *ciphertext equality*, which has been proven correct in §5.3.2. We want to prove the following statement:

$$\begin{gathered} \textit{Given } C_x, C_y, C_z \in \mathbb{R}_{n^2}, \exists x, y, z \in \mathbb{R}_n, \text{ such that} \\ C_x = g^{m_x} r_x^n \bmod n^2, C_y = g^{m_y} r_y^n \bmod n^2, C_z = g^{m_z} r_z^n \bmod n^2 : \\ x + y \bmod n = C_x C_y \bmod n^2 = C_z \end{gathered} \tag{5.9}$$

Values r_x and r_y are rescaled by the *prover* under the following conditions:

$$\begin{cases} r_x = r_x^{b^{e_x - e_y}} \bmod n & \text{if } e_x > e_y \\ r_y = r_y^{b^{e_y - e_x}} \bmod n & \text{otherwise} \end{cases}$$

In order to prove Equation 5.9, the *prover* generates a *ciphertext equality* proof that attests to the validity of the addition operation. In order to do so, the *prover* performs a homomorphic addition on ciphertexts C_x and C_y and generates a proof that the result of this operation is equal to the ciphertext C_z, i.e.:

$$\begin{gathered} \textit{Given } C_x, C_y, C_z \in \mathbb{R}_{n^2}, \exists {C'}_z \in \mathbb{R}_{n^2}, \text{ such that } C_x C_y = {C'}_z \bmod n^2 : \\ C_z {C'}_z{}^{-1} = g^{z-z} (r_z {r'}_z{}^{-1})^n = (r_z {r'}_z{}^{-1})^n \end{gathered} \tag{5.10}$$

Equation 5.10 has been proven in §5.3.2 to attest that C_z and ${C'}_z$ encrypt the same value, which concludes the proof for the addition operation.

The *subtraction* operation utilizes the additive and multiplicative inverse properties shown below:

$$\forall x, y, z \in \mathbb{R}_n, \exists C_x, C_y, C_z \in \mathbb{R}_{n^2} : \; x - y = z \bmod n \iff C_x C_y^{-1} = C_z \bmod n^2 \tag{5.11}$$

Equation 5.11 is analogous to Equation 5.9 and hence the proof follows as for the addition operation.

Multiplication and Division

Our aim is to hide both of the operands for multiplication and/or division operation and to provide a proof attesting to the computation's validity. As Paillier's encryption scheme supports homomorphic multiplication only if one of the operands is in plaintext, we decided to implement a ZKP, where both of the operands remain encrypted, together with the computation result. Concretely,

given three ciphertexts C_x, C_y, and C_z we would like to prove that the relationship between their respective plaintexts x, y, and z is $xy = z \bmod n$. As we are operating with floating-point numbers, the first step is to encode the values x, y, and z. Upon encoding step, the mantissas m_x, m_y and m_z should be rescaled to a common publicly known exponent and their encryptions, C_x, C_y, and C_z, generated. We adapt the protocol based on the previous work by Damgard and Jurik [38], where we are proving the following statement:

$$\begin{aligned} Proof_{Mul} : \{(C_x, C_y, C_z, b, e_x, e_y, e_z, pk_s; x, y, z, m_x, m_y, m_z, r_x, r_y, r_z) : \\ C_x = g^{m_x} r_x^n \bmod n^2 \wedge C_y = g^{m_y} r_y^n \bmod n^2 \wedge C_z = g^{m_z} r_z^n \bmod n^2 \\ \wedge\ xy = z \bmod n\} \end{aligned} \tag{5.12}$$

To support division, the proof generator has two choices at its disposal. Either the generator encrypts the inverse of the multiplicand, which makes it a dividend or the equation is rearranged to a multiplication, hence there is no need to encrypt an additional number representing the inverse. The generator rearranges the equation by the order of inputs to the ZKP generator. The *verifier* verifies in the exact same order. Encrypting an inverse of a number may be the way to go in case the number is constantly used as a dividend, whereas in a more dynamic environment encrypting an additional number, which in addition needs to be proven to represent an inverse of the original one, poses a significant effort and hence rearranging the equation is the proposed way to go.

Multiplication with multiple operands involves intermediate results which may need to be verifed. In contrast, the addition operation relies on the encryption scheme's additive homomorphic property, where the final value contains the accumulation of intermediate computation steps needed for verification (as shown in §5.3.2). In case the intermediate results of a multiplication operation are needed, a linear increase in the number of operands in both time and space is imposed. Such scenario may be observed in case the operands are being used in other statements and cannot be accumulated into a one-time operation.

Proof of correctness. The result of multiplication on floating-point numbers, in the plaintext is the following:

$$\forall x, y, z \in \mathbb{R}_n : z = xy = m_x b^{e_x} m_y b^{e_y} = (m_x m_y) b^{e_x + e_y} \bmod n \tag{5.13}$$

In order to prove the correctness of *multiplication* and *division* operation, we refer to *Lemma 5* [38]. We adapt the multiplication protocol of [38] by applying Fiat-Shamir heuristic [35] in order to obtain a non-interactive protocol. Therefore, the non-interactive version of the protocol remains secure under a *random oracle model* [35]. To prove equation 5.13, we need to prove

$$\forall x, y, z \in \mathbb{R}_n \text{ such that } z = xy : m_z = m_x m_y \bmod n \wedge b^{e_z} = b^{e_x + e_y} \bmod n \tag{5.14}$$

Lemma 5 [38] proves the multiplication of mantissas, i.e., $m_z = m_x m_y \bmod n$. What remains to prove is the correctness of the exponent part of the encoding, i.e.:

$$\forall b, e_x, e_y, e_z \in \mathbb{R}_n : b^{e_x} b^{e_y} = b^{e_z} \bmod n \tag{5.15}$$

Proving the Equation 5.15 does not involve ZKPs: all information (i.e., b^{e_x}, b^{e_y}, and b^{e_z}) is public knowledge and can be verified by anyone. We now prove that having this information as public does not affect the security of our mechanism. We do so by reducing the problem to the *factorization* problem, which has been proven hard under specific assumptions.

A factorization problem is defined as follows:

$$Given\ c \in \mathbb{R}, \exists a, b \in \mathbb{R} : ab = c, \tag{5.16}$$

where x and y values are unknown.

The security of our protocol relies on the hardness of the following problem:

$$Given\ b^{e_x}, b^{e_y}, b^{e_z} \in \mathbb{R}, \exists y, z \in \mathbb{R} : m_x m_y b^{e_x} b^{e_y} = m_z b^{e_z}, \tag{5.17}$$

$$But\ m_x m_y b^{e_x} b^{e_y} = m_z b^{e_z} \iff m_x m_y \frac{1}{m_z} = \frac{b^{e_z}}{b^{e_x} b^{e_y}}, \tag{5.18}$$

Hence we reduce our problem to the factorization problem. We conclude that our protocol is considered secure under the assumptions for which factorization is assumed to be a hard problem.

The *division* operation follows the analogous protocol, where the following holds:

$$\forall x, y, z \in \mathbb{R}_n : z = xy^{-1} = m_x b^{e_x} (m_y b^{e_y})^{-1} = (m_x m_y^{-1}) b^{e_x - e_y} \bmod n \tag{5.19}$$

Range Proofs

We provide range proofs on mantissas of the encoding and support signed *BigInteger*, *Long* interval boundaries. In case an interval boundary is a negative number, its encoding results in a larger mantissa than the encoding of its additive inverse, hence a modulus subtraction from the mantissa is done prior to the proof creation. In addition, our range proofs include a proof of equality between the underlying encryption and commitment scheme discussed in §5.3.2. More specifically, in order to prove that value x is contained in an interval $[l_x, u_x]$, we prove the following statement:

$$\begin{aligned} Proof_{Range} : \{(C_x, Comm_x, g_1, g_2, h, n, l_x, u_x, b, e_x; x, m_x, r_x, s_x) : \\ C_x = g_1^{m_x} r_x^n \bmod n^2 \wedge Comm_x = g_2^{m_x} h^{s_x} \bmod n \wedge x \in [l_x, u_x]\} \end{aligned} \tag{5.20}$$

The intervals boundaries l_x and u_x are encoded and rescaled, such that their exponents, e_u and e_z are equal to e_x, i.e., the exponent of x. An example usage scenario may be the following. A party would like to hide his/her age, yet be able to prove to a third party that the encrypted age lies within publicly known interval.

Proof of correctness. We want to prove that the following statement holds:

$$\begin{aligned} &Given\ C_x \in \mathbb{R}_{n^2}, Comm_x, l_x, u_x \in \mathbb{R}_n : m_x \in [l_x, u_x] \wedge \\ &C_x = g_1^{m_x} r_x^n \bmod n^2, Comm_x = g_2^{m_x} h^{s_x} \bmod n \end{aligned} \tag{5.21}$$

The *prover* rescales publicly known interval boundaries l_x, u_x as follows:

$$l_x = \begin{cases} l_x^{b^{e_x}} & \text{if } e_{l_x} > e_x \\ l_x & \text{otherwise} \end{cases}$$

$$u_x = \begin{cases} u_x^{b^{e_x}} & \text{if } e_{u_x} > e_x \\ u_x & \text{otherwise} \end{cases}$$

In order to prove Equation 5.21, we first prove the following statement:

$$Given\ C_x \in \mathbb{R}_{n^2}, Comm_x, l_x, u_x \in \mathbb{R}_n : m_x \in [l_x, u_x] \tag{5.22}$$

For the proof of Equation 5.22, we refer to Boudot's *range proof with tolerance* [33].

In order to complete the proof of Equation 5.21, we need to prove the statement below, which says that the Fujisaki-Okamoto commitment ($Comm_x$) hides the same value, m_x, as the underlying Paillier encryption (C_x).

$$Given\ C_x \in \mathbb{R}_{n^2}, Comm_x \in \mathbb{R}_n : C_x = g_1^{m_x} r_x^n \bmod n^2, Comm_x = g_2^{m_x} h^{s_x} \bmod n \tag{5.23}$$

In order to prove Equation 5.23, we refer to the proof $Proof_{Eq_{PF}}$, specified in §5.3.2. Therefore, the completeness of the range proof protocol depends on the correctness of the *Paillier and Fujisaki-Okamoto Equality* protocol in §5.3.2.

Paillier and Fujisaki-Okamoto Equality

As the implemented range proofs, presented in §5.3.2, work with Fujisaki-Okamoto commitment scheme and our ZKPs work with a randomized partially homomorphic scheme, chosen to be Paillier, we need to provide a ZKP stating that the Fujisaki-Okamoto commitment and Paillier encryption both hide the same value. In order to do so, we adapt the prior work [39], which is in essence an adapted

non-interactive sigma protocol. Concretely, we are proving the following statement:

$$Proof_{EqPF} : \{(C_x, Comm_x, g_1, g_2, h, n, b, e_x; x, m_x, r_x, s_x) : \\ C_x = g_1^{m_x} r_x^n \bmod n^2 \wedge Comm_x = g_2^{m_x} h^{s_x} \bmod n\} \tag{5.24}$$

$Proof_{EqPF}$ binds the encryption and the commitment scheme.

Proof of correctness. We want to prove the following statement:

$$Given\ C_x \in \mathbb{R}_{n^2}, Comm_x \in \mathbb{R}_n : C_x = g_1^{m_x} r_x^n \bmod n^2, Comm_x = g_2^{m_x} h^{s_x} \bmod n \tag{5.25}$$

In our implementation of *Paillier and Fujisaki-Okamoto Equality* protocol, we adapt the non-interactive *ZKP of equality* protocol [39]. As our system does not operate using *Pedersen commitment scheme* [11], our adaptation of the protocol involves utilizing only the *Paillier and Fujisaki-Okamoto Equality*, thus removing the *Pedersen commitment scheme* from the proof. In order to prove Equation 5.25, we refer to the original protocol idea and its correctness proof [40].

The protocol is considered secure under *random oracle model* as Fiat-Shamir heuristic [35] is applied.

Set Inclusion

We have implemented Paillier's ZKP of set membership [41] and extended its usage to work with all numerals and strings values using an additional encoding step transforming the strings into their respective *ASCII* values serving as input to Paillier's encryption scheme. In addition, due to the fact that the proof size and both generation and verification time grow linearly with the number of set elements, we propose a slight adaptation. The proposed idea utilizes the concept of k-anonymity [42], where we prove the set inclusion for the subset of the original set M, i.e., for $k < |M|$. As such, we pose a trade-off between the computational effort and data privacy, or in that regard, anonymity. The implementation of the set inclusion proof is additionally used for comparison between different proof embedding techniques in §7.2.3. Formally, set inclusion is proving the following statement:

$$Proof_{Incl} : \{(C_m, g, n, u; M, m, r) : C_m = g^m r^n \bmod n^2 \\ \wedge\ u = \frac{C_m}{g^{m_i}} \bmod n^2 \wedge M = \{m_1, m_2, ..., m_{|M|}\} \wedge m \in M\} \tag{5.26}$$

Values u are accessible to the verifier, whereas M remains hidden. An example usage scenario may be a security audit. *Party A*, uniquely identified by a *string*

value $UUID_A$, may wish to prove to a third party that he/she is on a list of allowed participants without disclosing $UUID_A$ or other participants' identities.

Proof of correctness. We want to prove the following statement:

$$Given\ u, C_m \in \mathbb{R}_{n^2},\ \text{such that}\ u = \frac{C_m}{g^{m_i}}, C_m = g^m r^n \bmod n^2, \qquad (5.27)$$
$$M = \{m_1, m_2, ..., m_{|M|}\} : m \in M\}$$

In order to support floating-point numbers and string values, we perform an initial encoding step. In case of string values, given a set of strings S, we perform the following encoding step utilizing the $ASCII$ bijective mapping:

$$\forall s \in S, v \in \mathbb{N} : s \mapsto v \qquad (5.28)$$

In case of floating-point numbers, we perform the standard encoding into a private mantissa and publicly known base and exponent. The encoding step is performed on both the set values M and the value m, for which we claim that $m \in M$ holds.

In order to prove Equation 5.27, we refer to the underlying set inclusion proof [41]. As we implement a non-interactive protocol, security under a *random oracle model* is assumed. Furthermore, the set inclusion proof reveals the values $u_i = {}^{C_m}/_{g^{m_i}} \bmod n^2$ to the *verifier*, thus hiding the underlying values m_i.

By Lemma 5.1, where we prove the *computationally hiding* property of the encryption scheme, the values of set M remain hidden to the verifier. Namely, the values of m_i and m are not revealed by exposing u_i to the *verifier*. Upon proof verification, the *verifier* may only attest that a certain value m is in the set M without learning its position in M or the value m itself.

As the time to perform set inclusion operation grows linearly with the set size, we extend the protocol by utilizing the concept of *k-anonymity* [42]. We prove the set inclusion for the subset of the original set M, i.e., for $k < |M|$. Computational time of the set inclusion operation decreases proportionally to the decrease in value k. On the other hand, value k dictates the size of the anonymity set, hence poses a security vulnerability in case of low value k.

Ciphertext Comparisons

Comparisons on ciphertexts are of crucial importance in smart contracts validation. In order to compare two numbers in their corresponding ciphertext space, we propose, to the best of our knowledge, a novel four-stage protocol 1 involving:

- Encoding, encryption and commitment to a number z acting as a summand while leveraging encryption scheme's additive homomorphic property.

- ZKP of equality upon summation with z.
- ZKP of equality of Paillier encryption of z and Fujisaki-Okamoto commitment to z.
- Positivity proof of z.

The proposed ZK comparison proof is utilizing the fact that we may switch between encoding scales of the ciphertext by rescaling it as shown in Protocol 1. It does not require an access to the underlying encoded values, but rather to the publicly known encoding base and exponent. This property allows us to generate proofs ahead of time mimicking the results that will be needed for the *verifier* to validate a proof.

Protocol 1 Secure Ciphertext Comparison

Inputs. Let $op \in \{>, <\}$ denote the comparison operator. Proof generator holds input $\tilde{x}$, $\tilde{y}$, whereas proof *verifier* holds C_x, C_y encrypted under pk_s.

Goal. Generate and validate the proof such that:

$$\begin{aligned} Proof_{Comp_{cipher}} : \{(C_x, C_y, b, e_x, e_y, pk_s; x, y, m_x, m_y, r_x, r_y) : \\ C_x = g^{m_x} r_x^n \bmod n^2 \wedge C_y = g^{m_y} r_y^n \bmod n^2 \wedge x \ op \ y\} \end{aligned} \tag{5.29}$$

The protocol:

1. **Proof generation.**
   (a) $\exists z \in \mathbb{R}_n$, such that:

$$m_z = \begin{cases} (m_x - m_y) \bmod n & \text{if } op \text{ } is \text{ } > \\ (m_y - m_x) \bmod n & \text{if } op \text{ } is \text{ } < \end{cases}$$

   i. Let $r_{c_a} = r_{c_{\{x,y,z\}}}$. Rescale r_{c_a}, such that:

$$r_{c_a} = \begin{cases} r_{c_a}^{b^{e_a}} \bmod n & \text{if } e_z > 0 \\ \begin{cases} r_x^{b^{e_x - e_y}} \bmod n & \text{if } e_x > e_y \\ r_y^{b^{e_y - e_x}} \bmod n & \text{otherwise} \end{cases} & \text{otherwise} \end{cases}$$

   ii. Calculate v as:

$$v = \begin{cases} r_x (r_y r_z)^{-1} \bmod n & \text{if } op \text{ } is \text{ } > \\ r_x r_z r_y^{-1} \bmod n & \text{if } op \text{ } is \text{ } < \end{cases}$$

   (b) Generate secure number $\tilde{z}$ with the raw encoded value m_z, scaled exponent $e_z \geq 0$ and state's public key pk_s.

(c) Generate ciphertext equality proof $Proof_{Eq_{cipher}}$ attesting that v is the randomness used in the encryption C_0 of zero. C_0 is the result of homomorphic subtraction of two ciphertexts C_x, C_y performed by the *verifier* and prepared ahead of time by the generator.

(d) Generate a commitment to m_z denoted by $Comm_z$. The commitment is used to generate a positivity proof $Proof_{z>0}$. Let $l_z = 0$ denoted the lower and $u_z = n$ denoted the upper interval bound. Encode l_z and u_z under the same context as z. Prepare the range proof interval I by scaling as follows:

$$l_z = \begin{cases} l_z^{b^{e_z}} & \text{if } e_{l_z} > e_z \\ l_z & \text{otherwise} \end{cases}$$

$$u_z = \begin{cases} u_z^{b^{e_z}} & \text{if } e_{u_z} > e_z \\ u_z & \text{otherwise} \end{cases}$$

(e) Generate the equality proof $Proof_{Eq_{PF}}$ of Paillier encryption and Fujisaki-Okamoto commitment hiding the same underlying value.

(f) Generated $proofSet \leftarrow \{C_z, Comm_z, Proof_{z>0}, Proof_{Eq_{cipher}}, Proof_{Eq_{PF}}, I\}$.

2. **Proof verification.**

(a) Verify $Proof_{Eq_{PF}}$, i.e., prove that Paillier encryption and Fujisaki-Okamoto commitment hide the same value.

(b) Verify $Proof_{z>0}$, i.e., C_z encrypts a positive value.

(c) Let ς denote the result of verifier's homomorphic addition as follows:

$$\varsigma = \begin{cases} (C_y + C_z) \bmod n^2 & \text{if } op \; is \; > \\ (C_x + C_z) \bmod n^2 & \text{if } op \; is \; < \end{cases}$$

Let $\overline{u}$ denote the result of verifier's homomorphic subtraction as follows:

$$\overline{u} = \begin{cases} (C_x - \varsigma) \bmod n^2 & \text{if } op \; is \; > \\ (\varsigma - C_y) \bmod n^2 & \text{if } op \; is \; < \end{cases}$$

(d) Verify $Proof_{Eq_{cipher}}$, i.e., $\overline{u}$ is an encryption of zero value with randomness v unknown to the proof verifier, but proven to be known by the proof generator.

We compare our approach to order-preserving encryption (OPE). In the following points we present our analysis:

- OPE is a deterministic encryption scheme, hence is not secure under indistinguishability against chosen-plaintext attack (IND-CPA)[2], as an encryption of a specific value always results in the same ciphertext. Our approach utilizes a randomized encryption scheme, hence is IND-CPA secure.
- OPE preserves the order of plaintexts in ciphertext space, which leaks relative distance between plaintexts. Such a property may be exploited to approximate the plaintext. In addition, due to OPE's determinism, additional information relative to the frequency of plaintexts can be leaked. As we do not pose any order on ciphertexts, and our encryption scheme is randomized, our approach cannot be exploited to approximate the plaintexts' position.
- Existing OPE poses a security vulnerability due to its determinism. An onion-layer encryption scheme has been proposed in order to tackle this vulnerability [14]. This scheme encapsulates value x in encryption layers, where the innermost encryption layer corresponds to OPE encryption used for comparison and the outermost layer to a randomized encryption, such as Advanced Encryption Standard (AES), without any functionality. Onion-layer encryption involves distributing AES's symmetric keys for stripping an onion layer prior to using OPE and requires an excessive key generation and management.
- Our results have shown that the proposed ciphertext comparison proof is approximately 2 times faster than OPE in addition to offering higher security.

Proof of correctness. We want to prove the following statement:

$$\begin{aligned} &Given\ op \in \{>, <\}, C_x, C_y \in \mathbb{R}_{n^2},\ \text{such that } C_x = g^{m_x} r_x^n \bmod n^2, \\ &C_y = g^{m_y} r_y^n \bmod n^2 : x\ op\ y\} \end{aligned} \tag{5.30}$$

We recall that the *verifier* is given access to the values C_x, C_y, C_z, and $Comm_z$, where $z = m_z b^{e_z}$ is defined as follows.

$$\exists z \in \mathbb{R}_n,\ \text{such that } : m_z = \begin{cases} (m_x - m_y) \bmod n & \text{if } op\ is\ > \\ (m_y - m_x) \bmod n & \text{if } op\ is\ < \end{cases} \tag{5.31}$$

We generate the encryption C_z by using the randomness below:

$$r_z = r_z^{b^z} \bmod n \ \text{ if } e_z > 0 \tag{5.32}$$

[2] https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf

We assume for the rest that we want to prove that $x < y$ (similar logic can be applied to $x > y$). Given the encryption C_z, we prove as a first step that $x + z = y$ is used in the following statement:

$$\begin{gathered} Given\ C_x, C_y, C_z \in \mathbb{R}_{n^2}, \exists x, y, z \in \mathbb{R}_n,\ \text{such that} \\ C_x = g^{m_x} r_x^n \bmod n^2, C_y = g^{m_y} r_y^n \bmod n^2, C_z = g^{m_z} r_z^n \bmod n^2 : \\ x + z \bmod n = C_x C_z \bmod n^2 = C_y \end{gathered} \tag{5.33}$$

Furthermore, as described in our protocol the *verifier* holds the proofs $Proof_{z>0}$, $Proof_{Eq_{cipher}}$, $Proof_{Eq_{PF}}$. $Proof_{z>0}$ attests to the claim that the underlying value z is a strictly positive number, whereas $Proof_{Eq_{PF}}$ provides the proof that the encryption and commitment scheme hide the same value z. Additionally, as the *verifier* performs the homomorphic addition operation on values C_x and C_y, proof $Proof_{Eq_{cipher}}$ attest to fact that the result of *verifier's* computation is indeed the correct one.

We now prove that the *verifier* may only attest to the following claims:

- There exists a number z, encrypted and given as C_z, such that:
  - $C_x C_z \bmod n^2 = C_y$.
  - C_z is an encryption of a strictly positive number.
  - C_z is encrypting the same value z hidden under the Fujisaki-Okamoto commitment used for the positivity proof.
- C_x and C_y encrypt x and y such that $x < y$.

In order to prove Equation 5.30, we proceed as follows.

Equation 5.33 holds directly as a result of the proof provided by the *ciphertext equality* (Equation 5.9).

Furthemore, we have to prove that the commitment $Comm_z$ hides a strictly positive value z. More formally, we are proving the following statement:

$$Given\ C_z \in \mathbb{R}_{n^2}, Comm_z, l_z, u_z \in \mathbb{R}_n : m_z \in [l_z, u_z], \tag{5.34}$$

where the lower interval bound $l_z = 0$ and the upper interval bound $u_z = n$ are rescaled as follows:

$$l_z = \begin{cases} l_z^{b^{e_z}} & \text{if } e_{l_z} > e_z \\ l_z & \text{otherwise} \end{cases}$$

$$u_z = \begin{cases} u_z^{b^{e_z}} & \text{if } e_{u_z} > e_z \\ u_z & \text{otherwise} \end{cases}$$

The Equation 5.34 holds by the proof provided for Equation 5.22.

In the final step, we have to provide a proof that the encryption C_z, used for the homomorphic addition and the ciphertext equality proof, and the commitment $Comm_z$, used for proving the positivity of value z, both hide the same value z. More formally, we want to prove that the following statement holds:

$$Given\ C_z \in \mathbb{R}_{n^2}, Comm_z \in \mathbb{R}_n : C_z = g_1^{m_z} r_z^n \bmod n^2, Comm_z = g_2^{m_z} h^{s_z} \bmod n \tag{5.35}$$

The Equation 5.35 holds by the proof provided for Equation 5.25, thus concludes the proof of correctness for secure ciphertext comparison.

As we apply Fiat-Shamir heuristic [35] to provide a non-interactive protocol, our protocol is secure under a *random oracle model*.

Ciphertext-Plaintext Comparisons

As it would defy the purpose of the privacy-preserving computation, an equality proof on ciphertext-plaintext tuple is not supported. Instead, the aforementioned concept of secure buckets (§5.1) for, e.g., audit proof reveals the underlying values without an access to the private key material. The main aim of the protocol is to create a valid encoding of the interval boundaries and transform a comparison with a plaintext into a range proof. We are proving the following statement:

$$\begin{aligned} Proof_{Comp_{plain}} : \{(C_x, y, b, e_x, pk_s; x, m_x, r_x) : C_x = g^{m_x} r_x^n \bmod n^2 \\ \wedge\ x\ op\ y\} \end{aligned} \tag{5.36}$$

Transforming a comparison to a range proof is performed by an addition, in case of *greater than* operation, and subtraction, in case of *less than* operation, of a known positive number to the known value y resulting in the lower and higher interval bound, respectively. This steps involves the encoding procedure as in the ciphertext comparison proof. Upon proof generation, the proof set consists of the following elements:

$$proofSet \leftarrow \{C_x, Comm_x, y, Proof_{x>y}, Proof_{Eq_{PF}}, I\}$$

Proof of correctness. We want to prove the following statement:

$$Given\ op \in \{>, <\}, C_x \in \mathbb{R}_{n^2}, y \in \mathbb{R}_n \text{ such that } C_x = g^{m_x} r_x^n \bmod n^2 : x\ op\ y\} \tag{5.37}$$

We recall that the *verifier* is given access to the values C_x, $Comm_x$, and y. Furthermore, the *verifier* holds the proofs $Proof_{x>y}$ and $Proof_{Eq_{PF}}$. $Proof_{x>y}$ attests to the claim that the underlying value x is strictly greater than the value y, whereas $Proof_{Eq_{PF}}$ provides the proof that the encryption and commitment scheme hide the same value x.

We prove that the *verifier* may only attest to the following claim:

- C_x is an encryption of a value x, for which holds that $xopy$.

In order to prove Equation 5.37, we proceed as follows.

In the first step, we obtain a value y' defined as:

$$\exists y' \in \mathbb{R}_n, \epsilon > 0 \text{ such that } : y' = \begin{cases} (y+\epsilon) \bmod n & \text{if } op \text{ } is \text{ } > \\ (y-\epsilon) \bmod n & \text{if } op \text{ } is \text{ } < \end{cases} \tag{5.38}$$

The value y' is used as an interval boundary, from which the proof proceeds as specified in §5.3.2, i.e., we prove the following statement:

$$Given \ C_x \in \mathbb{R}_{n^2}, Comm_x, l_x, u_x \in \mathbb{R}_n : \begin{cases} m_x \in [m_{y'}, u_x] & \text{if } op \text{ } is \text{ } > \\ m_x \in [l_x, m_{y'}] & \text{if } op \text{ } is \text{ } <, \end{cases} \tag{5.39}$$

where u_x is the highest possible and l_x the lowest possible value of m_x. Equation 5.39 holds by the proof provided for Equation 5.21. Furthemore, the proof provided by Equation 5.25, which proves that C_x and $Comm_x$ hide the same value x, concludes the correctness proof for secure ciphertext-plaintext comparison.

Note. As described in our protocol, the *verifier* knows that encrypted value C_x encrypts a value which is less or greater than a publicly known plaintext value y. In addition, the value hidden by C_x and $Comm_x$ is proven to be the same by validating $Proof_{Eq_{PF}}$. The strength of this approach lies in the fact that it does not reveal order on all ciphertexts, as we are operating under a randomized probabilistic scheme. On the other hand, this operation has to be used cautiously not to reveal too much information. If we perform a plaintext comparison on ciphertext C_x, such that we state that the value x should be greater than one and less than three, then we have revealed x. As such, the plaintext comparison should be primarily used with at least one open-ended interval, i.e., it should only state that the value is either greater or less than a plaintext value. It is proof generator's task to decide if ciphertext comparison may be more suitable for a specific scenario. An example where we can use the ciphertext-plaintext comparison could be an agreement where we need to prove that a violation, which we would like to hide, has happened after the publicly known date of signing the agreement.

5.4 Integration in Corda

Figure 5.2 depicts embedding privacy-preserving smart contracts in Corda[3]. *Party A* generates a ZKP while building the transaction which involves consuming an old reference state and creating a new state. Each state contains a ZKP attesting

[3] https://docs.corda.net/api-flows.html

to the validity of its creation. There are two approaches we consider for embedding the ZKP within the state. We discuss these approaches in detail in §7.2.3 and §7.2.3. Together with the ZKP, a state contains the publicly available ledger data, which corresponds to the encrypted transaction data, e.g., data collected from different sensors. In addition, ledger data contains the public key material used for data encoding and encryption. The contract associated with the state is a collection of privacy-preserving ZKP verification operations which correspond to the ZKP generation functions that led to the ZKPs created by the transaction initiator.

As we may see in Figure 5.2, and as explained in §3.1, Corda's transaction life-cycle starts by building, signing, and verifying a transaction. The proposed concept of validating notaries, as explained in §3.1, now comes into play. As we gather counter-party's signature, the counter-party may or may not verify the transaction. In case the party does not verify the transaction, the notary is verifying the transaction and providing a signature which attests to the transaction's validity. This is the case of validating notary. In case the notary does not validate, the counter-party will run the privacy-preserving smart contract verification. This might require performing time-consuming transaction dependency resolution in case the counter-party has not been involved in previous transactions involving the consumed state in current transaction. As Corda's verification logic is decoupled from the business logic used to generate the smart contracts, a claim that smart contract's function privacy can be preserved, in addition to the data itself, is valid. Furthermore, the aforementioned concept of secure buckets (§5.1) is implemented utilizing Corda's flow over *gRPC* to send the binding randomness and raw data material to the counter-party.

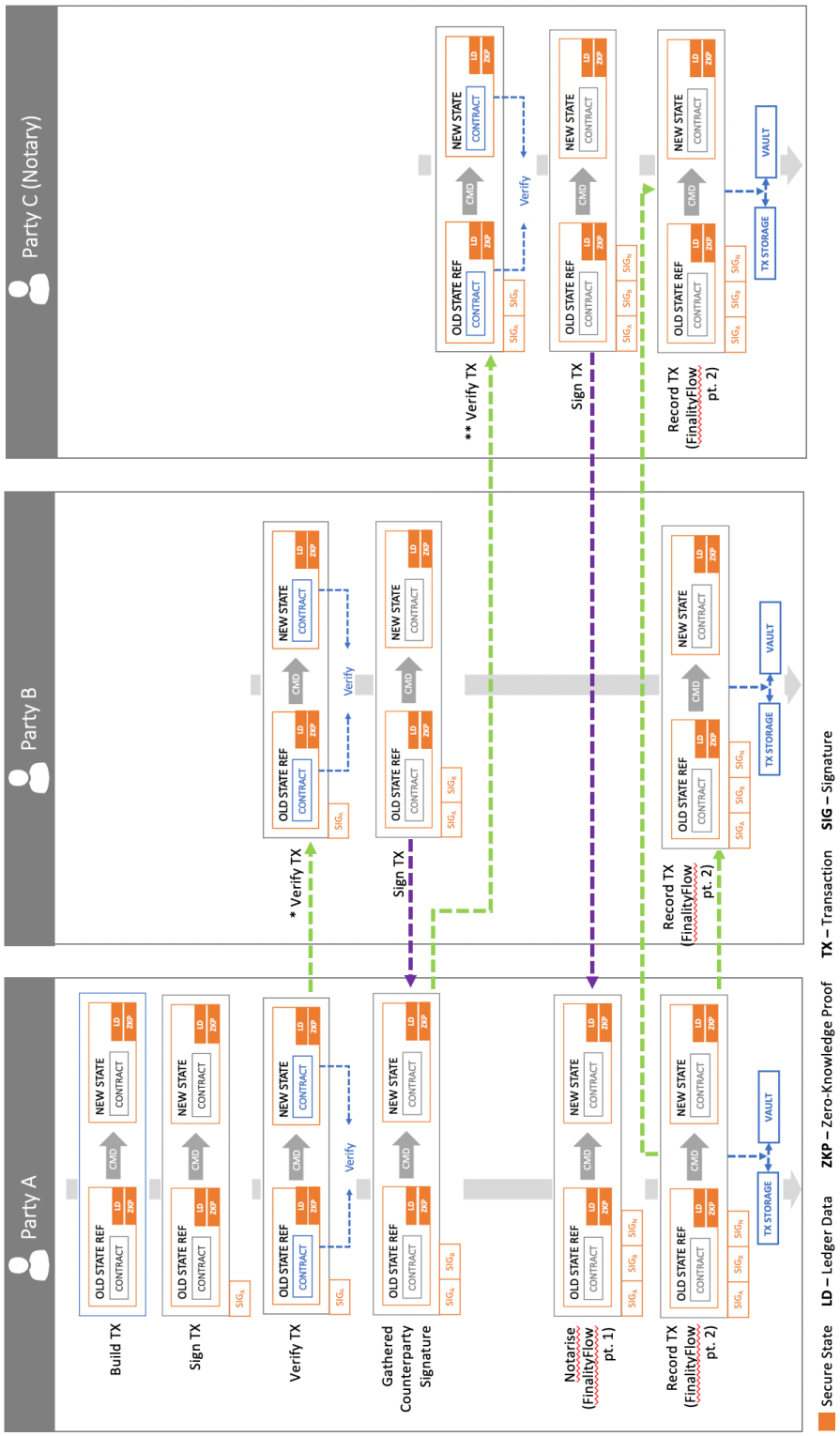

Figure 5.2: Privacy-preserving smart contracts integration in Corda

.

Chapter 6

Security Analysis

Our system mitigates the *information disclosure* process threat, for both active and passive adversary, specified in §3.2, by enabling privacy-preserving smart contracts. However, embedding our system within Corda increases the overall system's complexity, thus increasing the attack surface. In this section we would like to address the assumed attacker model, investigate possible attacks, and provide attack mitigation strategies. As specified in §3.2, we assume a computationally bounded adversary, who may be passive or active.

Active Adversary We have decided to investigate possible attacks with respect to the security triad consisting of confidentiality, integrity, and availability. Concretely, we have identified an active adversary, who may be a network participant, to pose the following threats.

Private key disclosure. Collusion with another network participant who has not destroyed the private key material and an adversary not destroying the private key material represent the most severe vulnerability an active adversary may exploit. This exploit breaks the transaction chain confidentiality if the attackers can get a hold of all the transactions associated with the key material. The problem is that the private key can be used to circumvent the protection layer. A malicious participant may have anticipated that storing the private key and using it as leverage at a later point in time might be beneficial. A mitigation strategy requires our privacy-preserving system not to return or provide access to the private keys upon key material generation. Instead, only public key material may be presented to the issuance transaction initiator.

Insufficiently specific smart contracts. Insufficient smart contract verification logic coverage offers an ability to cheat. More specifically, generating gibberish values which pass verification process may succeed if the smart contract logic is not sufficiently restrictive. Contract specification must cover possible corner cases and limit the possibility of an exploit. In case the contract was not specific enough, the mitigation of this vulnerability is the secure bucket concept, which

is introduced and explained in §5.1. Each transaction counter-party is provided with the off-chain secure bucket, which attests to the data validity without private keys.

System stalling. Performing invalid verification or generating bogus ZKPs may harm the system's availability and cause it to stall. We require the aforementioned fault-tolerant consensus protocol to be in place in case Byzantine entities are expected in the system. Invalid verification may only cause a problem if the upper bound imposed on the number of allowed Byzantine entities in the system is violated. Due to the non-repudiation requirement, in case of bogus ZKP generation, the participant may be identified.

Ciphertext malleability. An active man-in-the-middle (MITM) attack while transferring secure buckets and transactions may harm both the transaction confidentiality and, if modified, the transaction integrity. Due to ciphertext malleability of the underlying encryption scheme, which is defined in §5.3.1, an adversary may alter the encrypted data potentially resulting in an encryption of different plaintext. In addition, public key modification would prevent the generation or validation of a ZKP, and it would break the confidentiality as the attacker would hold the private key. Such an attack requires breaking the underlying communication relying on TLS 1.3, which is considered unfeasible for the assumed computationally bounded adversary.

A ZKP replay attack. A ZKP replay attack may not be performed as the party verifying the proof verifies that the party that generated the proof is the same as its current transaction counter-party.

Denial of service on the validating notary. As the validating notary needs access to all the transactions, it has to sign the issuance transaction as well. Therefore, a malicious party may perform numerous self-issuance transactions with time-consuming ZKP verification justified by the fact that it may improve its UTXO model's parallelism. More concretely, the more states a party has, the more concurrent transactions it may perform compared to the account model. A possible mitigation may include rate limiting and verification timeouts for self-issuance transactions.

Passive Adversary A passive adversary, who may be a network participant, may only read the data stored on the ledger and the data in transit, but does not perform any modifications or other actions. As communication relies on secure channel, a passive adversary is not able to read the transaction data between the parties. On the other hand, as we allow all the network participants to perform transaction validation, all the transaction data is made available to every network participants. In §3.2 we referred to this threat as *information disclosure* process

threat. This threat is mitigated, because transactions contained on the ledger are either encrypted or do not reveal any sensitive information. As discussed in §5.1, the only plaintext transaction data consists of public keys, encoding constants and ZKPs.

In conclusion, integrating our system in Corda has mitigated the *information disclosure* threat associated with both passive and active adversary by encrypting all the ledger data. We faced two main challenges towards transaction confidentiality. The first challenge was to allow participants to perform complete transaction verification, while keeping the data encrypted. This challenge was tackled using ZKPs and homomorphic encryption. The other challenge was to provide transaction auditability, i.e., conducting a transaction audit without revealing the private key used for data encryption. This challenge has been solved using the secure buckets, as presented in §5.

Chapter 7

Performance Evaluation

7.1 Experiment Setup

The experimented results are obtained on a 3.20GHz Intel(R) Core(TM) i5-4570 cluster machine with four CPUs. In the following we elaborate on the experiment setup between the two kinds of experiments we conduct.

Standalone System Performance. An experiment measuring standalone system performance involves the following:

- *Setup.* Key material generation.
- *Operation.*
  - *Secure data type creation.* Involves the following stages:
    - Data encoding
    - Data encryption
    - Commitment
    - Homomorphic operation
  - *ZKP generation.*
- *Verification.* Verifies the operation by ZKP verification and homomorphic operations.

System's Performance in Corda. The experiments are run on the setup *mock network* within Corda. Such a network consists of the network participants identified by their certificates issued by the Doorman CA[1] and the notary service consisting of either the network participants or acting as a separate entity. Figure 5.2 presents all network participants and flow logic we consider in performance evaluation of privacy-preserving operations. We consider two transacting

[1] https://docs.corda.net/permissioning.html

participants, *party A* and *party B*, as well as the notary, *party C*, which is a separate entity.

We consider two different scenarios regarding the notary service. In what follows, we present and elaborate on these scenarios.

- *Validating notary.* In case of a validating notary, transaction verification is performed by the notary and not by the transaction counter-party, i.e., *party B* in Figure 5.2.
- *Non-validating notary.* In case of a non-validating notary, the counter-party, *party B*, performs transaction verification by executing the smart contract. Therefore, the notary is not validating the transaction.

An experiment within Corda utilizes transactions between parties and involves the complete life cycle of a transaction, which includes the following stages:

- *Generation.* Transaction creation and, in case of privacy-preserving operations, ZKP generation.
- *Verification.* Smart contract execution, which, in case of privacy-preserving operations, performs ZKP verification.
- *Signature.* Signing the transaction.
- *Counter-party signature collection.* In case of a non-validating notary, it involves transaction verification by the counter-party, i.e., *party B* in Figure 5.2.
- *Finality flow.* Includes transaction notarisation and, in case of a validating notary, transaction verification.

In order to achieve statistical significance, we have decided to repeat each experiment a hundred times. Therefore, in the upcoming figures each data point corresponds to an average and standard deviation across hundred experiment runs. As the results have shown negligible variance across experiment runs, we have decided to exclude the warm-up and cool-down phase, hence reporting the results across all hundred runs.

We are interested in the performance comparison of plain and secure operations in both the standalone system and the system embedded in Corda.

7.2 Results

The experimental results are divided into the following sections:

- Standalone system performance
- System's performance in Corda
- Realistic industrial use case scenarios

We are interested in comparison between plain, unsecured operations and their respective secure counterparts provided by our system. The comparison is based on the computation time as well as storage requirements. Corda's underlying communication involves transaction serialization, therefore as we increase the transaction size, the serialization process becomes more apparent in terms of computation time.

7.2.1 Standalone System Performance

Prior to running the experiments, we expect to observe significant slow-down for the secure operation due to computationally involved ZKPs. As certain privacy-preserving operations, such as, e.g., a ciphertext comparison, utilize other ZKPs, we expect that all the utilized proofs, in a ciphertext comparison, have a lower computation time as well as smaller proof size. We expect the equality of ciphertexts, addition, subtraction, multiplication and division to be the least expensive operations, whereas plaintext and ciphertext comparison including the set membership and range proofs are expected to be the most expensive ones. As computation time and proof size of set membership operation linearly depends on the size of the set in question, we decide to fix the set size to one hundred elements for the reported performance.

The results in Figure 7.1 show that floating- point numbers (denoted by prefix FP in Figure 7.1) have an impact on computational effort. The explanation is that rescaling of both the mantissa and secure randomness of the encrypted floating-point number prior to ZKP generation involves costly exponentiation, as shown in Protocol 1. It is important to notice that secure operations are tens of thousands times slower than their non-private counterparts. Even with a considerable slow-down in a standalone comparison, our aim is to achieve a reasonable slow-down (in the order of one magnitude) within a DLT embedding our system. The rationale behind the idea is that a careful embedding will not result in such a considerable slow-down. The setup phase is performed only once in the issuance transaction and it suffers from non-negligible variance. The reason is the generation of Paillier public and private key pairs which involve generation of random probable safe primes p and q requiring a probabilistic testing. In order to tackle the variance exhibited during the setup phase, we report an average time needed for the key material generation across all operations. The upside to the variance in the setup time is the fact that the setup is not a reoccurring event, as generated public keys may be used for numerous proof generations and verifications. Proof sizes depicted in Figure 7.2 support our expectations in

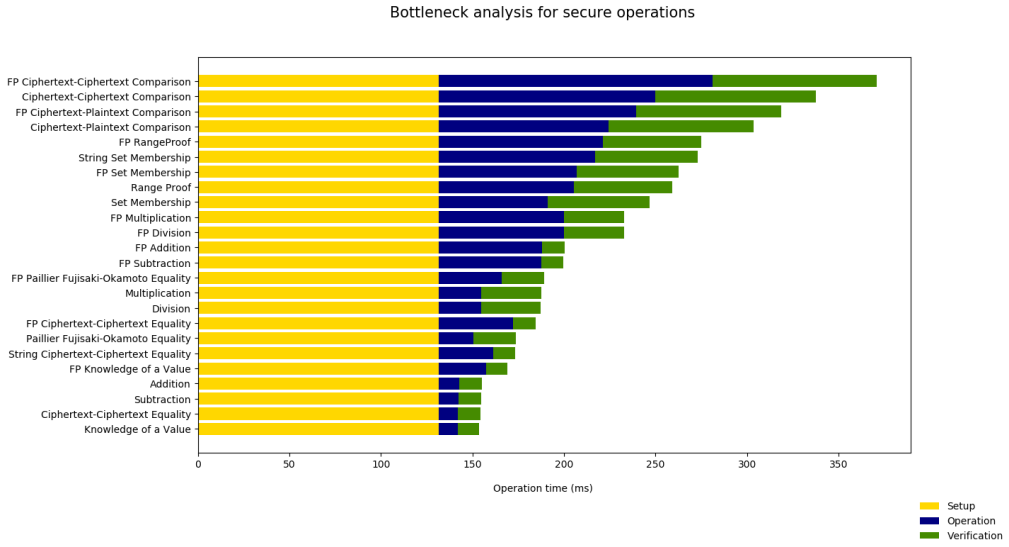

Figure 7.1: Bottleneck analysis for standalone system's performance ordered by increasing computation time.

terms of operator grouping and their order. Computation time (Figure 7.1) and proof size (Figure 7.2) results exhibit the expected behaviour in case of proofs which consist of other proofs, such as, e.g., plaintext and ciphertext comparisons. The largest and most computationally expensive privacy-preserving operation is the ciphertext comparison with a size of approximately 12kB. The results of unsecured operations have been omitted from Figure 7.1 as they are hardly visible due to their much lower computation time. Therefore, we report these results in Appendix Table A.1. The *average time* in Table A.1 is the time needed for a complete privacy-preserving operation. The *slow-down factor* in Table A.1 is calculated as quotient of computation time for privacy-preserving operation (excluding the time needed for the setup phase) and the computation time for its plaintext counterpart operation.

7.2.2 System's Performance in Corda

Due to the significant overhead posed by Corda's transaction life cycle, we expect the slow-down in performance of the privacy-preserving computation as opposed to the plain computation to be significantly less than in the standalone system comparison. In each of the transaction stages we expect to observe an increase in computation time for the secure operations as each stage either contains an overhead of proof creation or verification. As we propose the concept of validating notary and hence omit the need for the transacting counter-party to resolve the transaction chain, we expect to observe a computation time decrease in the counter-party signature collection stage. The rationale is that the cumulative cost

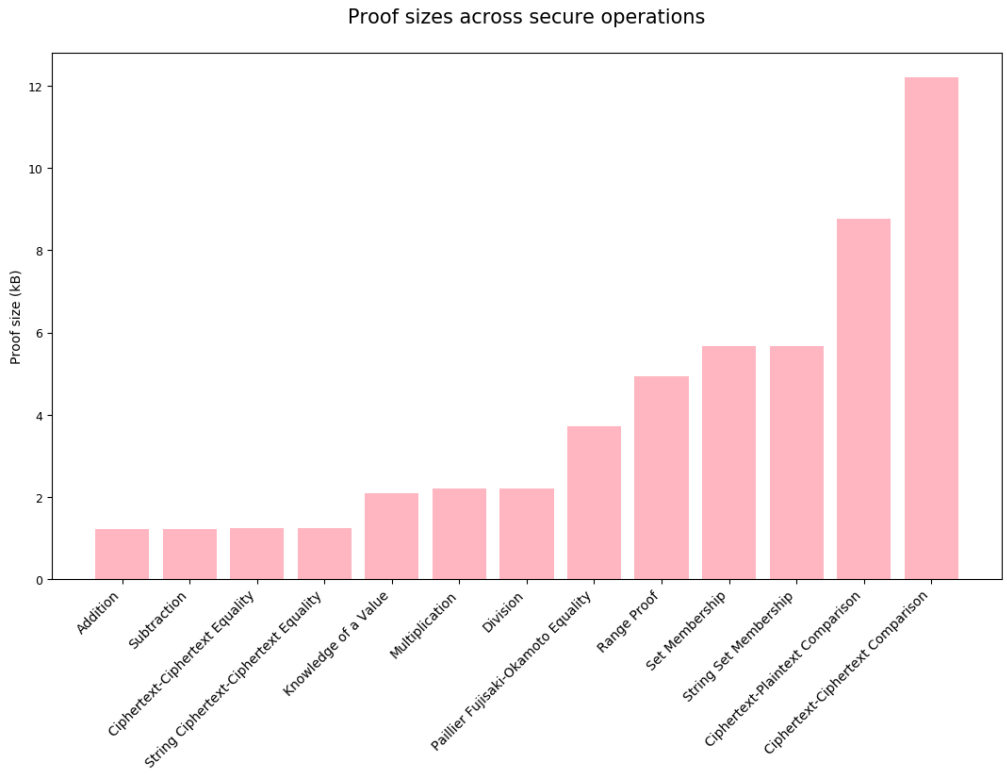

Figure 7.2: Sizes of individual proofs for the supported operations.

of plain computations performed while resolving a long transaction chain for the non-validating notary reaches a point where a privacy-preserving computation for a single transaction is less expensive. In addition to the cost effectiveness, the main contribution of the privacy-preserving computation is the achieved confidentiality which offers higher validation scalability, where more parties may validate transactions without revealing any information.

Figure 7.3 presents a bottleneck analysis of all privacy-preserving operations implemented in Corda for issuance and transfer transactions together with the respective unsecured operations. In order to provide a fair comparison, as we are proposing the usage of validating notary with privacy-preserving computation, all the plaintext operations are also run using the validating notary. In addition, we have omitted any counter-parties in the transactions, as we look into their performance later on in the realistic industrial use case scenarios. As the finality flow involves the transaction verification, we observe that the finality flow time includes the verification time in every operation. Issuance transactions do not involve transaction validation and do not hold any proofs as they may be created at any point in time and kept local. Therefore, the transaction generation time, which involves the effort of initial key material creation is the main bottleneck.

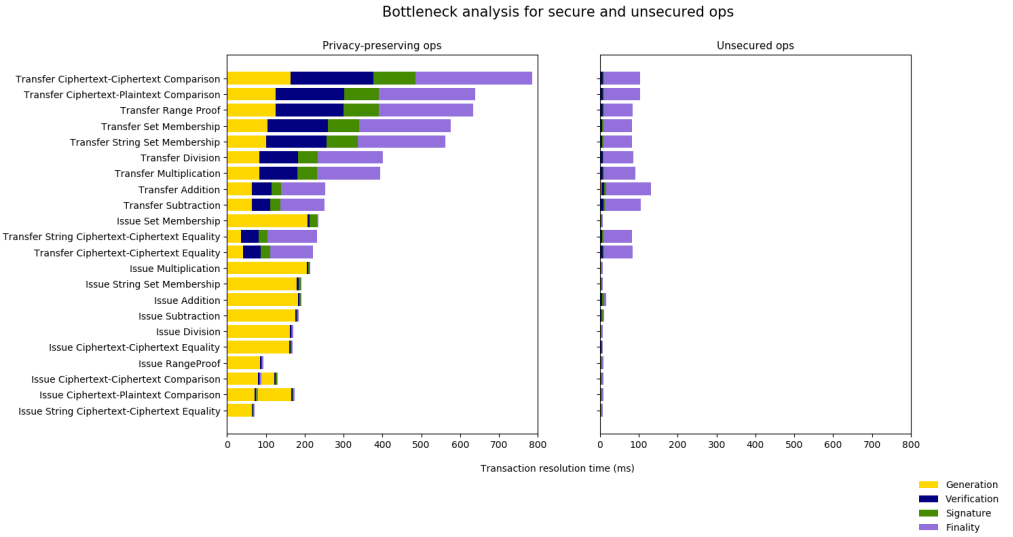

Figure 7.3: Bottleneck analysis after integrating our system in Corda.

It is important to mention that the generation of the key material suffers from non-negligible variance as the public modulus is obtained via multiplication of two secret, secure random probable prime numbers. The same variance around the key material generation has been observed in the standalone system performance. Due to the fact that public keys are associated with the state(s) in the transactions, the transfer transaction does not involve re-creation of the key material, but rather propagates the public part of the key material generated in the issuance transaction. Therefore, we observe a decrease in the generation time for the transfer transaction.

In the upcoming §7.2.3, we reflect on the performance of the system integrated in Corda on realistic use case scenarios, where we combine presented privacy-preserving operations in order to build smart contracts.

7.2.3 Realistic Industrial Use Cases

We are interested in comparison between validating notary with secure, privacy-preserving computation and the non-validating notary with plaintext computation. Thus, we compare the performance using our system with Corda's notion of confidentiality leveraging need-to-know communication basis under two suitable models - validating and non-validating notaries. We investigate different proof embedding strategies and highlight their strengths and weaknesses based on a specific use case. Specific for the industrial service setting, our system offers a privacy-preserving immutable maintenance log. The investigated industrial setting does not involve fungible assets, but rather data containers that should

satisfy certain contract requirements. In case of fungible assets and need of audit proofs, the transacting party may use our system's secure buckets. The secure buckets are as well applied to the case where prior input data is needed for proof generation.

General Use Case Description

Each use case contains four main entity groups:

- Internet Service Provider (ISP)
- ISP Proxy
- Contractor
- Customer

The ISP is responsible for providing the agreed Internet access to the customer specified by the Service Level Agreement (SLA). Each ISP may delegate certain tasks to an external collaborator, i.e., contractor. Once the SLA has been signed by the customer, ISP sends an ISP Proxy device, which is set up by the customer on-premise. The ISP Proxy device is responsible for data gathering, event-based transaction initialization and, in case of privacy-preserving operations, proof generation. More specifically, ISP proxy is an access point connected to the router running *OpenWRT*[2], where a packet sniffer such as *tcpdump*[3] is installed.

Corrective maintenance

The first use case presents a scenario of interaction between the ISP, represented by the ISP Proxy, and the customer. The data logged on the ledger represents SLA violations and corresponding corrections. More concretely, in case of an agreement violation, a violation state should be issued in a transaction notifying all the involved participants about it. Furthemore, there should be an immutable log of the corresponding violation resolution. Such an application provides the non-repudiation property and may be used to automatically track the participants' SLA compliance. ISP would like to have the agreement with the contractor validated. Such agreement may include the maximum response time, i.e., the time between violation and correction state issuance. In addition, our use case includes a scenario where a customer rents the apartment to a party which wishes to continue the SLA under the same terms and conditions as the previous customer. Such a scenario spares significant effort on both the customer and the ISP side as it, e.g., does not require a new ISP Proxy device. We are

[2] https://openwrt.org/
[3] https://www.tcpdump.org/

investigating two different models operating under different levels of confidentiality. In one model we have a non-validating notary without privacy-preserving computation and a validating notary with privacy-preserving computation in the other model. More formally, smart contract logic may be summarized as follows. Let b_{st_i} denote the boolean representing the validity of a statement st_i. Smart contract SC is defined as a union of statements st_i, i.e., $SC = \bigcup_{i=1}^{n} st_i$. Let u_i represent the upload and d_i the download speed measured at state i. Let u_{sla} and d_{sla} represent the minimum upload and download speed, respectively, as defined by SLA's terms and conditions. In addition, let t_i represent the timestamp at which the state measurement has occurred, i.e., has been placed in a transaction and let δ be the maximum allowed response time in case of a violation, defined in the agreement between the ISP and the contractor.

- Verify that the maintenance correction was performed in time:

$$b_{st_1} = \begin{cases} (t_{i+1} - t_i) \leq \delta & \text{if } (u_i < u_{sla} \vee d_i < d_{sla}) \\ True & \text{otherwise} \end{cases} \tag{7.1}$$

- Verify that the order of states holds, i.e., a violation state must be followed by a correction state and vice versa:

$$b_{st_2} = \begin{cases} (u_i \geq u_{sla} \wedge d_i \geq d_{sla}) & \text{if } (u_{i+1} < u_{sla} \vee d_{i+1} < d_{sla}) \\ (u_i < u_{sla} \vee d_i < d_{sla}) & \text{otherwise} \end{cases} \tag{7.2}$$

Such a smart contract makes use of the various ZKPs described in §5.3.2.

Figure 7.4 compares the performance of privacy-preserving and plain smart contracts for a validating and non-validating notary. We measure the performance with regard to two aspects - confidentiality and computational effort.

The performance metric for the computational effort is the transaction resolution time as a function of chain length. At approximately 160 transactions in the chain we reach the point from which it is beneficial for the system to operate with a privacy-preserving validating notary. The reason for such a behaviour can be seen in Figure 7.5. In case of a non-validating notary, as a customer's corrective maintenance chain grows, changing the customer operating under the same SLA agreement requires a long transaction resolution process as depicted in the counter-party signature collection stage. This stage involves validating the entire history of previous customer transactions and hence violates the notion of confidentiality. Up until the point of customer change, Corda's need-to-know communication for the non-validating has kept the transactions private to the participants. This raises the question of who the notary is. Given privacy-preserving validating notaries, such question may be answered by letting each system participant contribute to the validation process without learning any sensitive information. Introducing a trusted third party or an entity belonging only

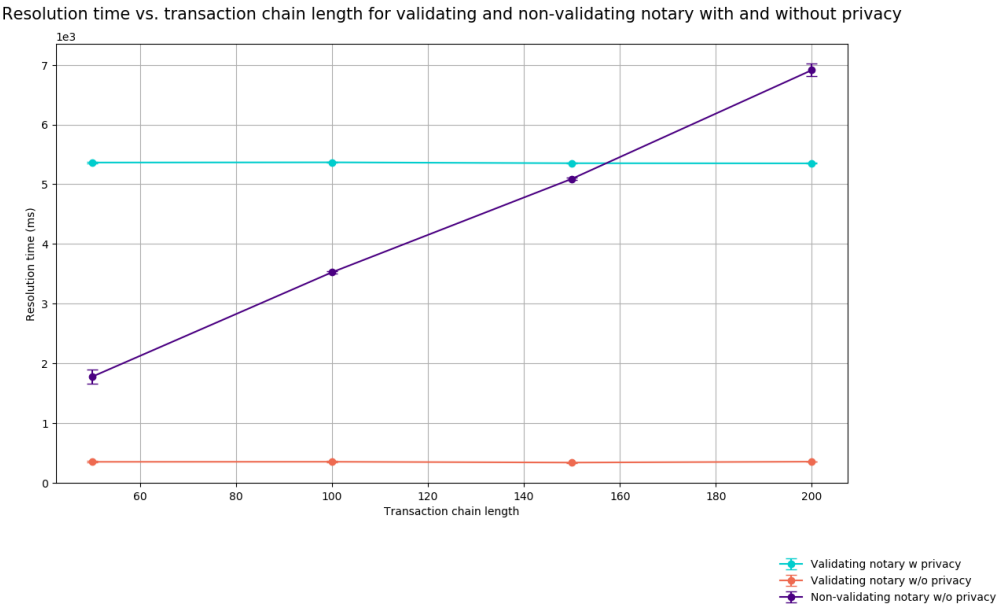

Figure 7.4: Corda's dependency resolution time for validating and non-validating notaries with and without privacy.

to some participants as a notary service requires trust enforcement. In an industrial setting, a certain computational overhead can be accepted, in trade-off for confidentiality. Our results show that there is a long-term computational effort pay off for using privacy-preserving validating notaries. The computational performance comparison of a standalone transaction with a known transaction history shows that privacy-preserving smart contracts cause a slow-down by a factor of 16. On the other hand, the transaction size with privacy-preserving contracts increases by approximately a factor of 10, resulting in $80kB$. As a result, in each transaction life-cycle stage, the privacy-preserving mode takes more time than the plain one. A closer look into Figure 7.5 and Figure 7.3 lets us draw the following conclusions. Our smart contract consists of five ciphertext comparisons including homomorphic addition. Each ciphertext comparison takes approximately 200 ms to verify in Corda (Figure 7.3), therefore the expected verification time matches the approximately observed 1000 ms (Figure 7.3). As expected, the finality stage takes at least as much as the verification time in case of a validating notary as the notary performs the validation itself. The additional overhead contains the communication between involved parties and transaction storage. The concept of non-validating notary with privacy has shown to suffer from high transaction resolution time in case of a long transaction chain and is therefore omitted from the Figure 7.4. Nevertheless, in case an asset does not change hands, the notary service is computationally limited and privacy is crucial, a non-validating privacy-preserving notary would be a feasible solution.

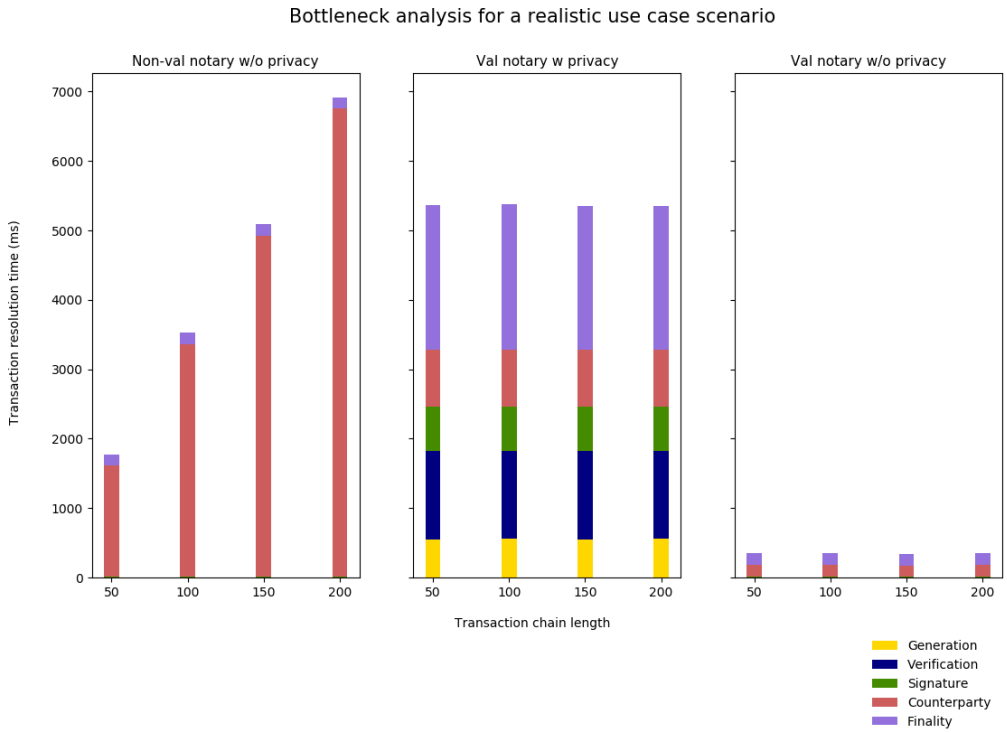

Figure 7.5: Bottleneck analysis for dependency resolution for validating and non-validating notaries with and without privacy.

URL blacklisting

Our next use case depicts a browsing scenario, where each visit to a blacklisted URL site triggers a transaction. Such a transaction chain represents an immutable blacklisted browsing log which may be used for various applications. Such applications may include agnostic firewall analysis or tracing the cause leading to an identity theft. We enforce the set of blacklisted URLs to remain private to the ISP and ISP proxy, whereas all the system participants may attest to the fact whether certain blacklisted site is contained in the set. Our aim is to investigate performance of state and reference proof embedding strategies.

Let S represent the set of blacklisted sites and s_i the visited site, then the plain smart contract is defined as:

$$b_{st_1} = \begin{cases} True & \text{if } s_i \in S \\ False & \text{otherwise} \end{cases} \tag{7.3}$$

As we would like to hide both the visited URLs and the original set of blacklisted URLs, our approach utilizes our system's k-anonymity set inclusion operation. We analyse performance as a function of proof size, due to aforementioned

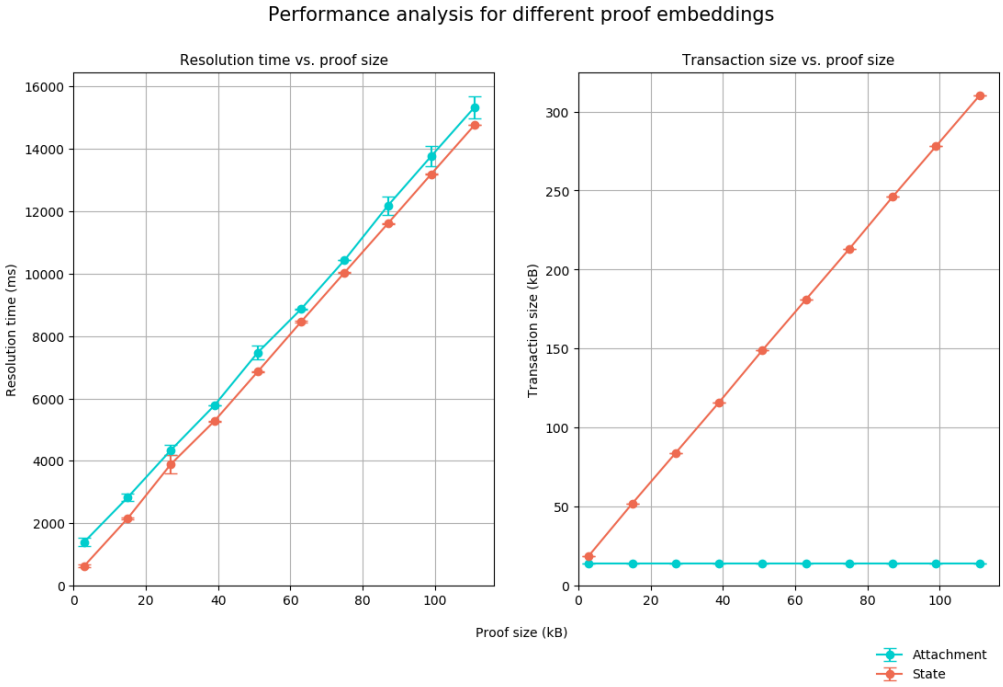

Figure 7.6: Performance for proof embedding strategies with increasing verification time.

linear increase in time and size with respect to the set size. Figure 7.6 presents two different proof embedding strategies, where we compare the transaction resolution time and storage requirements of both approaches.

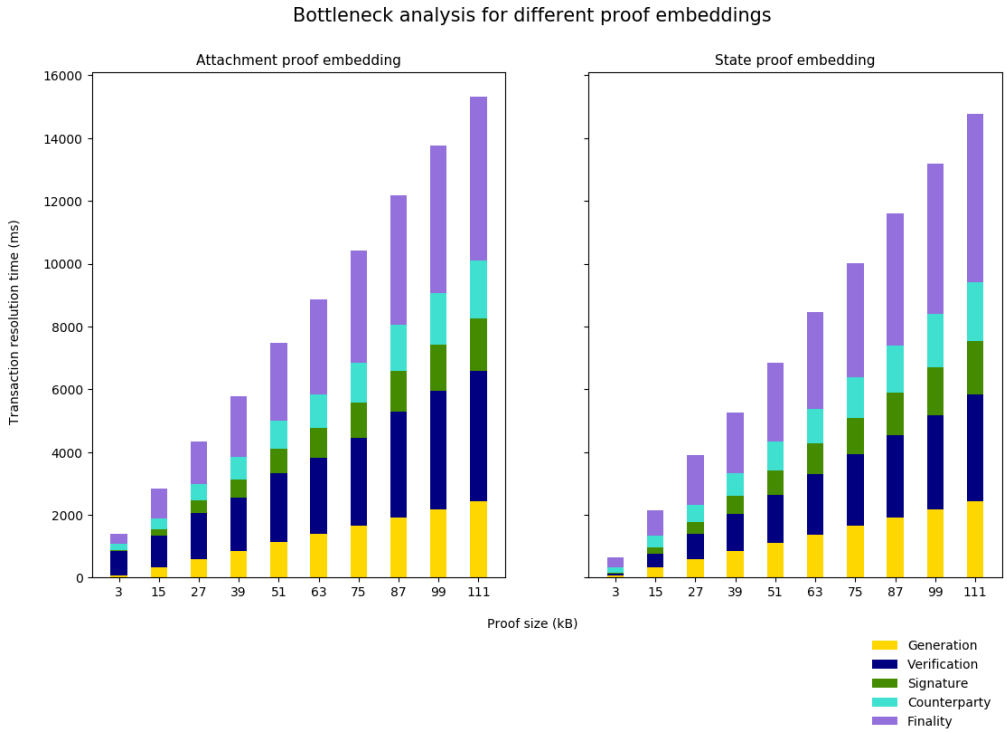

Figure 7.7: Bottleneck analysis for proof embedding strategies with increasing verification time.

State proof embedding State proof embedding places the *ProofSet* generated by the transaction initiator in the state itself, which is encapsulated in the transaction. Therefore, transaction size grows proportionally to the proof size (Figure 7.6). We expect to observe high overhead in the initial stage where the transaction is being serialized into a *WireTransaction* as well as in the stages where the transaction is being sent across the wire, i.e., in the counter-party and finality stage. Furthermore, the contract complexity increases proportionally to the proof size, hence resulting in higher generation and verification time (Figure 7.7). As transaction participants in a UTXO model store the unspent states in their vault, state proof embedding leads to a significant storage increase ultimately resulting in data explosion. As consumed and created states carry proofs, transfer transaction size s_{tx} is:

$$s_{tx} = 2s_p + s_{btx},$$

where s_p corresponds to the contract specific proof size and s_{btx} to the base transaction size, which includes signatures and the anonymous participants' identities.

Attachment proof embedding Our aim is to leverage an off-chain *ProofSet* embedding by placing a *SecureHash* (SHA-256) reference in the state pointing to the *ProofSet* attachment. This approach keeps the transaction size constant regardless of the proof size or contract complexity (Figure 7.6). The attachments are stored on an external storage and are sent over *gRPC* on-demand to the participating transaction parties.

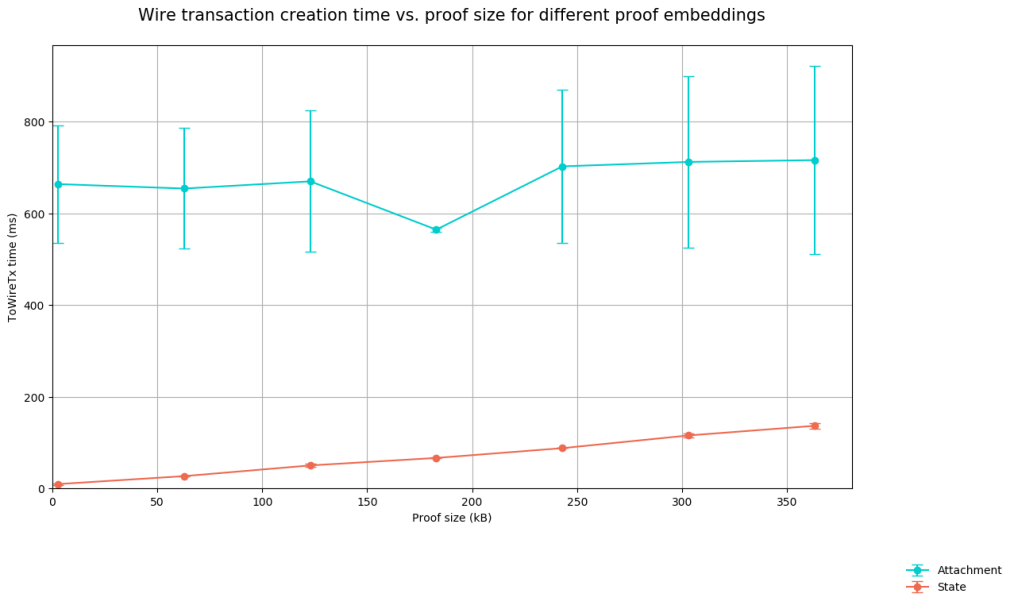

Figure 7.8: Wire transaction creation time for large proof embedding strategies.

Contrary to our expectations, Figure 7.6 shows that state proof embedding outperforms the reference in terms of the resolution time. On the other hand, the transaction size is showing the expected behaviour. Further analysis of Corda's transaction life-cycle offers valuable insights. Figure 7.8 demonstrates the time needed to transform a transaction from the *transaction builder* to a *WireTransaction*. Attachment proof embedding suffers from relatively high variance, but does not correlate with the transaction size (i.e., the proof size). The reason for such behaviour is the fact that the attachment is turned into a *ClassLoader* instance during the initial stage. In our experiment setup, this time-consuming task has to be performed only once prior to *WireTransaction* creation. The proof state embedding approach in Figure 7.6 reveals the expected proportional increase in time with respect to the transaction size as the serialization takes place. Given this knowledge, we further investigate at which transaction size may an attachment proof embedding be the favorable option. In order to do so, we increase

the proof size further, reaching the limits of plausible smart contract complexity. As we are interested in the effect of the transaction size on the performance, the verification time is set to a constant. It depicts a scenario where the proof size increases faster than the underlying verification consuming the proofs.

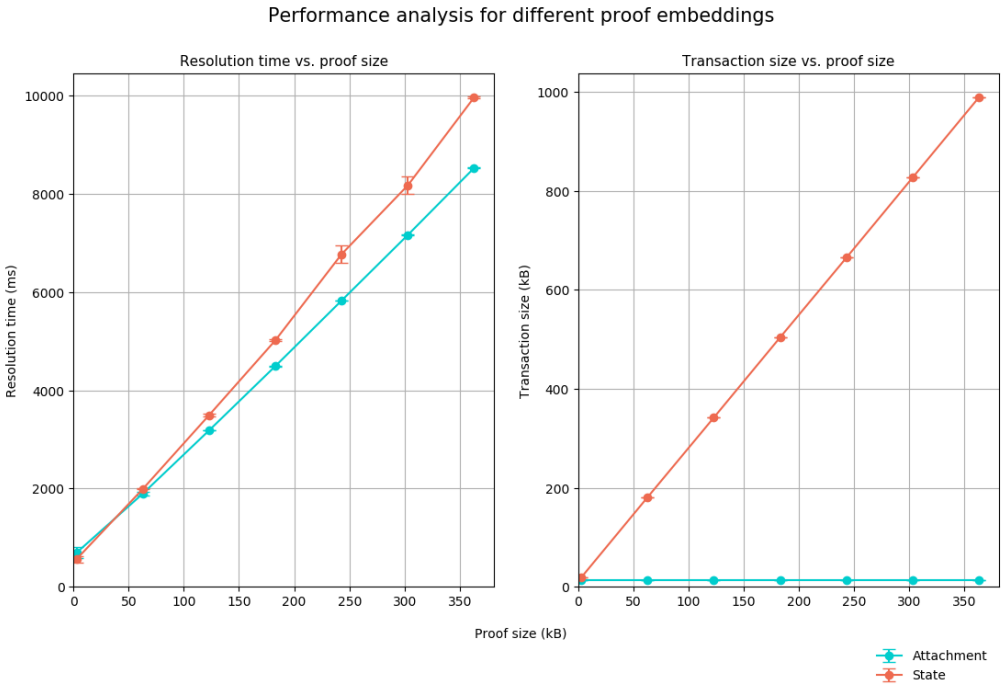

Figure 7.9: Performance for large proof embedding strategies with constant verification time.

Figure 7.9 supports our hypothesis that the attachment proof embedding is beneficial for large transaction sizes, i.e., proof sizes.

Further bottleneck analysis depicted in Figure 7.10 shows the initially expected behaviour, where the stages involving multi-party communication and serialization suffer from the overhead introduced by a large transaction size. As Figure 7.2 demonstrated, in order to reach a proof size of approximately $300kB$, at least 25 privacy-preserving ciphertext comparisons should be contained in the smart contract, which may be reasonable to expect. In conclusion, regardless of the slight differences in the resolution time performance, the benefits of a constant transaction size are of crucial importance and hence, reference proof embedding is a favorable long-term option.

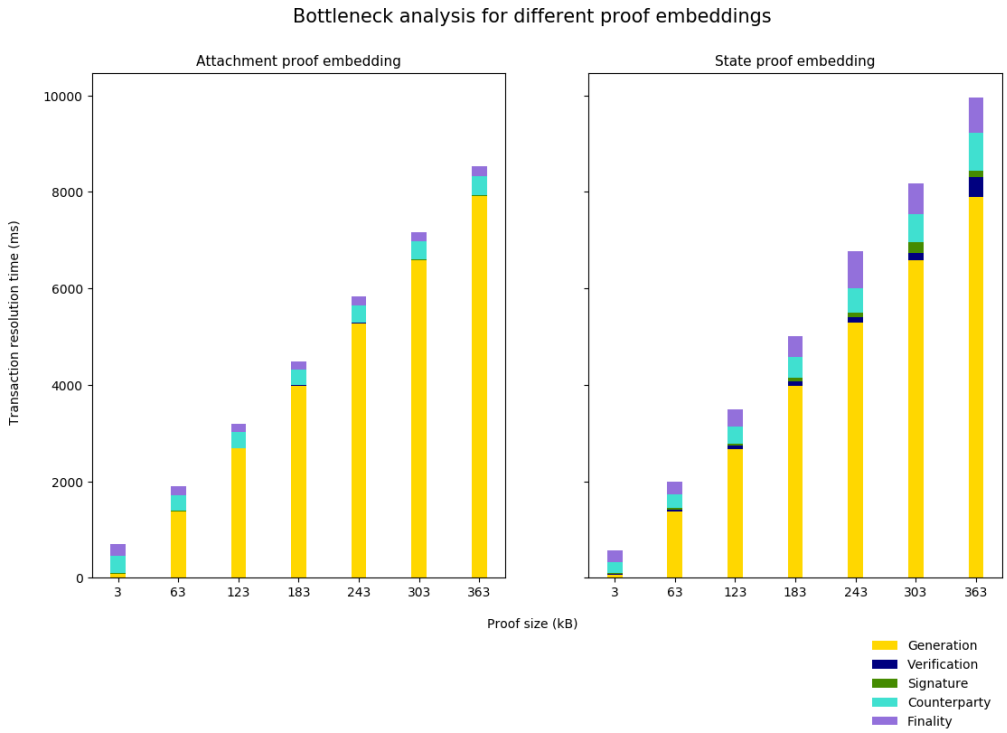

Figure 7.10: Bottleneck analysis for large proof embedding strategies with constant verification time.

Chapter 8

Summary

This thesis investigated current distributed ledger technologies and their applicability to industrial service applications. In that regard, we have identified confidentiality as the main shortcoming. As a result, we proposed a solution in form of a system that enables writing privacy-preserving smart contracts.

The proposed system consists of wide range of operations on both numbers and textual data which may be verified in a privacy-preserving manner. Such operations include, to the best of our knowledge, a novel approach towards secure comparisons in which neither the order nor the ciphertext's frequency is revealed. The method underlying our approach combines partially homomorphic encryption schemes and zero-knowledge proofs. The extensive evaluation of our system's performance has shown that supporting floating-point numbers as well as textual data introduces an additional encoding overhead to the privacy-preserving operations. Our system operates without private keys and utilizes the non-determinism of the encryption scheme. Therefore, the re-generation of key material is not required during a life-cycle in which the public key material is being used.

Beyond the design and implementation of the system itself, this thesis showed how to integrate that system into Corda, a permissioned DLT. This integration included performance optimizations to expand Corda's concept of validating notaries. We remove the need for validation by peers and delegate this job to the notaries themselves. This delegation results in significant reduction in transaction resolution time for highly liquid assets in a long transaction chain. We also evaluated our system's performance. The results have shown that adding a privacy-preserving layer to the transactions does not result in an impractical computational slow-down and as such may be applicable for industrial applications. Furthermore, we investigate different proof embedding approaches and show that embedding a ZKP reference in a transaction results in a constant transaction size and does not have a significant impact on computational performance.

In conclusion, integration of our system into Corda mitigated the *information disclosure* threat by providing privacy-preserving transactions.

Future Work. In following, we highlight the most interesting future system improvements.

Anonymity. The currently supported public key randomization may be replaced by a more robust *ring signature scheme* [10]. Network participants would form a group of entities, where each participant holds its own public/private key pair. This key pair may be used to generate a *ring signature* over a certain transaction. A signature from a specific party would not trace back to its original identity, but would prove it belongs to a specific group of entities.

Single Commitment Scheme. As the current approach operates with an encryption and a commitment scheme to support a wide range of operations, we require an additional ZKP stating that both schemes hide the same value. As range proofs operate with commitment schemes and we do not utilize private keys from the encryption scheme, a future improvement may include operating under a single commitment scheme while supporting the same range of operations. The challenge is to provide the ZKPs for all the basic arithmetic operations and set inclusion utilizing only a commitment scheme as opposed to the encryption scheme. Moreover, the underlying ZKPs may be further optimised in their size resulting in lower serialization overhead when being transferred.

Bibliography

[1] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, Stanford University, 2009, crypto.stanford.edu/craig.

[2] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic Evaluation of the AES Circuit," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 850–867.

[3] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-Preserving Symmetric Encryption," in *Advances in Cryptology - EUROCRYPT 2009*, A. Joux, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 224–241.

[4] C. Cachin and M. Vukolic, "Blockchain Consensus Protocols in the Wild," *CoRR*, vol. abs/1707.01873, 2017. [Online]. Available: http://arxiv.org/abs/1707.01873

[5] M. Hearn, "Corda: A Distributed Ledger," Online, available from: https://www.corda.net/content/corda-technical-whitepaper.pdf [July 2019].

[6] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," *CoRR*, vol. abs/1801.10228, 2018. [Online]. Available: http://arxiv.org/abs/1801.10228

[7] N. Kannengiesser, S. Lins, T. Dehling, and A. Sunyaev, "What Does Not Fit Can be Made to Fit! Trade-Offs in Distributed Ledger Technology Designs," 01 2019.

[8] "Survey of Confidentiality and Privacy Preserving Technologies for Blockchains," Online, available from: https://www.r3.com/wp-content/uploads/2018/04/Survey_Confidentiality_Privacy_R3.pdf [July 2019].

[9] N. T. Courtois. and R. Mercer., "Stealth Address and Key Management Techniques in Blockchain Systems," in *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, INSTICC. SciTePress, 2017, pp. 559–566.

[10] R. L. Rivest, A. Shamir, and Y. Tauman, "How to Leak a Secret," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565.

[11] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *Advances in Cryptology — CRYPTO '91*, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140.

[12] S. Meiklejohn, "Top Ten Obstacles along Distributed Ledgers Path to Adoption," *IEEE Security Privacy*, vol. 16, no. 4, pp. 13–19, July 2018.

[13] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On Scaling Decentralized Blockchains (A Position Paper)," 02 2016.

[14] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting Confidentiality with Encrypted Query Processing," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 85–100. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043566

[15] M. Hauck, S. Savvides, P. Eugster, M. Mezini, and G. Salvaneschi, "SecureScala: Scala Embedding of Secure Computations," in *Proceedings of the 2016 7th ACM SIGPLAN Symposium on Scala*, ser. SCALA 2016. New York, NY, USA: ACM, 2016, pp. 75–84. [Online]. Available: http://doi.acm.org/10.1145/2998392.2998403

[16] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology - EUROCRYPT 1999, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic*, April 1999.

[17] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," in *Advances in Cryptology*, G. R. Blakley and D. Chaum, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 10–18.

[18] "Advanced Encryption Standard: Federal Inf. Process. Stds." Online, available from: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf [September 2019].

[19] E. A. da Silva, "Practical use of Partially Homomorphic Cryptography," Master's thesis, Instituto Superior Tecnico, Faculdade de Direito, Escola Naval, Portugal, 2016.

[20] J. Katz and Y. Lindell, *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[21] E. Ben-sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," pp. 459–474, 05 2014.

[22] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards Privacy in a Smart Contract World," Cryptology ePrint Archive, Report 2019/191, February 2019, https://eprint.iacr.org/2019/191.

[23] D. Zhang, A. Su, F. Xu, and J. Chen, "ARPA Whitepaper," *CoRR*, vol. abs/1812.05820, 2018. [Online]. Available: http://arxiv.org/abs/1812.05820

[24] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *Proceedings 27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1353–1370. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner

[25] D. C. Sánchez, "Raziel: Private and Verifiable Smart Contracts on Blockchains," Cryptology ePrint Archive, Report 2017/878, 2017, https://eprint.iacr.org/2017/878.

[26] G. Zyskind, O. Nathan, and A. Pentland, "Enigma: Decentralized Computation Platform with Guaranteed Privacy," *CoRR*, vol. abs/1506.03471, 2015. [Online]. Available: http://arxiv.org/abs/1506.03471

[27] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 839–858.

[28] N. Narula, W. Vasquez, and M. Virza, "zkLedger: Privacy-Preserving Auditing for Distributed Ledgers," in *Proceedings 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 65–80. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/narula

[29] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling Decentralized Private Computation," Cryptology ePrint Archive, Report 2018/962, 2018, https://eprint.iacr.org/2018/962.

[30] A. Kerckhoffs, "La Cryptographie Militaire ," 1883.

[31] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, ser.

OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186. [Online]. Available: http://dl.acm.org/citation.cfm?id=296806.296824

[32] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," 2014.

[33] F. Boudot, "Efficient Proofs that a Commited Number Lies in an Interval," in *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium*, May 2000.

[34] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, Feb. 1989. [Online]. Available: http://dx.doi.org/10.1137/0218012

[35] A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Proceedings on Advances in cryptology—CRYPTO '86*. London, UK, UK: Springer-Verlag, 1987, pp. 186–194. [Online]. Available: http://dl.acm.org/citation.cfm?id=36664.36676

[36] "A Java Library for Paillier Partially Homomorphic Encryption," Online, available from: https://github.com/n1analytics/javallier/tree/master/src/main/java/com/n1analytics/paillier [July 2019].

[37] "ING Zero-Knowledge Range Proofs," Online, available from: https://github.com/ing-bank/zkproofs/tree/master/examples/java/zkrp/src/main/java/com/ing/blockchain/zk/ [July 2019].

[38] I. Damgård, M. Jurik, and J. Nielsen, "A Generalization of Paillier's Public-Key System with Applications to Electronic Voting," *International Journal of Information Security*, vol. 9, pp. 371–385, 04 2003.

[39] H. Duan *et al.*, "Aggregating Crowd Wisdom via Blockchain: A Private, Correct and Robust Realization," in *IEEE International Conference on Pervasive Computing and Communications, Kyoto, Japan*, March 2019.

[40] J. Camenisch and V. Shoup, "Practical Verifiable Encryption and Decryption of Discrete Logarithms," in *Advances in Cryptology - CRYPTO 2003*, D. Boneh, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 126–144.

[41] T. F. Dahlin, "Paillier Zero-Knowledge Proof," The Daylighting Society, Tech. Rep., December 2016.

[42] P. Samarati and L. Sweeney, "Protecting Privacy when Disclosing Information: k-Anonymity and its Enforcement through Generalization and Suppression," in *Technical Report SRI-CSL-98-04*. Computer

Science Laboratory, SRI International, 1998. [Online]. Available: http://www.csl.sri.com/papers/sritr-98-04/

Appendix A

Standalone System Performance

Operation	**Average time (ms)**	**Slow-down**
Ciphertext-Ciphertext Equality	22.45 ± 0.08	70903.06
FP Ciphertext-Ciphertext Equality	52.63 ± 0.38	105513.99
String Ciphertext-Ciphertext Equality	41.67 ± 0.22	70308.63
Subtraction	23.11 ± 0.26	8722.06
FP Subtraction	68.06 ± 0.34	24280.23
Addition	23.51 ± 0.15	9683.39
FP Addition	68.57 ± 0.95	28726.68
Paillier Fujisaki-Okamoto Equality	41.8 ± 0.16	-*
FP Paillier Fujisaki-Okamoto Equality	57.39 ± 0.56	-
Division	55.53 ± 0.07	25962.15
FP Division	100.94 ± 0.4	34040.75
Multiplication	55.85 ± 0.41	22020.65
FP Multiplication	101.24 ± 0.44	28234.02
Set Membership	115.17 ± 0.33	89178.87
FP Set Membership	130.92 ± 0.82	40311.22
FP RangeProof	143.06 ± 0.69	120596.31
Range Proof	127.55 ± 0.49	122943.88
String Set Membership	141.38 ± 4.89	6437.24
Ciphertext-Plaintext Comparison	168.25 ± 0.32	581892.44
FP Ciphertext-Plaintext Comparison	183.29 ± 0.82	394490.42
Ciphertext-Ciphertext Comparison	205.87 ± 0.38	689203.85
FP Ciphertext-Ciphertext Comparison	238.15 ± 1.28	646940.64

Table A.1: Standalone system performance results with comparison to unsecured operations.

*No unsecured operation.