DESIGN AND EVALUATION OF BLOCKCHAIN-BASED SECURITY PROTOCOLS

A thesis submitted for the degree of

DOCTOR OF PHILOSOPHY

at the

UNIVERSITY OF BIRMINGHAM
SCHOOL OF COMPUTER SCIENCE

by

Rujia Li

March 2022

DECLARATION

I, Rujia Li, hereby declare that both this thesis and the work presented in it are entirely the outcome of my own original research. Some of the work has been previously published in conference or journal, and this has been mentioned in the thesis. The contributions of others in the jointly-authored work are always explicitly indicated.

To my maternal grandfather, in loving memory.

ABSTRACT

Many security protocols rely on the assumption that the trusted third party (TTP) will behave “as it should”. However, this assumption is difficult to justify in the real world. A TTP may become malicious due to its hidden interests or having been compromised. It is publicly acknowledged that a failed TTP can easily destroy the entire security protocol. This thesis aims to provide results on how to use blockchain technologies to mitigate TTP challenges and thereby secure existing cryptographic protocols.

Firstly, we formally define a smart contract-based TTP (denoted as TTP-I) and give two security protocols based on such a type of TTP as concrete instances. In this approach, a smart contract can either complement a TTP’s actions or take over the entire functions of the existing TTP. This helps to obtain many security properties such as *transparency* and *accountability*. Smart contracts, however, are not adequate to replace TTP that is capable of maintaining secret information since all the states changed by TTP-I are in plaintext and publicly accessible.

To fill the gap, we propose another type of TTP (denoted as TTP-II) that enables confidential executions by combining smart contracts and Trusted Execution Environments (TEEs). To achieve this goal, we first investigate the state-of-the-art *TEE-aided confidential smart contracts* and then explore their core mechanisms. We further apply TTP-II to a traceable credential system and an accountable decryption system. These systems are proved secure and feasible. However, since blockchain systems suffer from scalability and performance issues, the development of blockchain-based cryptographic protocols is inevitably retarded.

At last, to make better blockchain systems, we provide two core mechanisms: a weak consensus algorithm and a delegatable payment protocol. The weak consensus algorithm allows parallel block generation, improving the performance and scalability of upper-layer blockchain systems. The delegatable payment protocol creates an offline payment channel, improving the payment speed. Both proposed algorithms have been practically implemented and systematically evaluated. Notably, the weak consensus algorithm has already been taken up by industries.

ACKNOWLEDGEMENTS

First of all, I would like to express my deep and sincere gratitude to my supervisors, Prof. David Galindo, Prof. Qi Wang and Prof. Mark Ryan, for their continuous support, inspiration, and encouragement during my four-year PhD career. In particular, David Galindo always looks at the bright side of life and encourages me to deal with any problem I encounter with great patience, which gets me through dark times. Qi Wang is a very kind mentor, and his immense knowledge, strong motivation, and enthusiasm are invaluable for my whole life. Mark Ryan's sense of responsibilities led me towards excellence. I feel so entirely lucky to have them as my mentor, advisor, and supervisor. Meanwhile, I am profoundly grateful to my enlightenment teacher, Prof. Rami Bahsoon, who introduced me to scientific research. Also, as a Research Students Monitoring Group (RSMG) member, he has offered me many useful suggestions and consistent supports over my MSc and PhD career. Especially, I would like to thank the examiners of my PhD viva, Prof. Tom Chothia and Prof. Siamak Shahandashti for their constructive and valuable comments.

Secondly, I would like to extend my thanks to my dear friends, Amy Li and Yifan Wu and Qin Wang. Amy Li and Yifan Wu helped me a lot with my PhD applications. I could not have made it without their support. Qin Wang is my precious colleague and friend who co-authored many papers during my PhD study. I cannot imagine my PhD life without his involvement. Meanwhile, many thanks go to the following people for their useful discussions, suggestions, and supports of my research during my PhD stage: Adi Shamir (Weizmann Institute of Science), Xinrui Zhang (NanKai), Kim-Kwang Raymond Choo (UTSA), Agnese Pacifico (Sapienza Università di Roma), Alphea Pagalaran, Jianyu Niu (UBC), Shuang Wu (NTNU), Jose Moreira (UoB), Wei Chen (UoB), Haiyang Chen (UoB), Jingyu Huang (UoB), Huanyuan Zhang (Oxford), Feng Liu (USTC), Geyang Wang (SUSTech), Yi Liu (SUSTech), Jamie Cui (Ant Group) and Ye Liu (Newcastle).

Thirdly, many sincere thanks also go to my colleagues from the industry: Xueqian Lu, Xunhe Cheng, Shanlin Lou, Jason Hu, Xiaoming Wang, who shaped my thinking on business and offered me a great opportunity of being a department leader. Despite being a temporary position, it still provided me with valuable experience for joining a digital economy start-up.

Finally, I greatly appreciate my family members (Ruping Gong, Hongmei Ji, etc.) for their persistent love, support and understanding. My sincere appreciation goes to my grandfather and brother for their financial support in my MSc stage. Without their help, I would never have had the opportunity to enter the United Kingdom for my MSc study. In addition, I would like to thank the Department of Computer Science at the University of Birmingham for providing me with a PhD opportunity and the Department of Computer Science and Engineering at Southern University of Science and Technology for funding my PhD research.

Rujia Li
March 2022

Contents

List of Figures

List of Tables

Acronyms

BFT Byzantine fault tolerant.

CA certificate authority.

CBE certificate-based encryption.

DDoS distributed denial-of-service.

EPC enclave page cache.

EUF-CMA existential unforgeability under chosen message attack.

IND-CCA indistinguishability under chosen-ciphertext attack.

IND-CCA2 adaptive indistinguishability under chosen-ciphertext attack.

IND-CPA indistinguishability under chosen-plaintext attack.

KC key curator.

MitM man-in-the-middle attack.

NC Nakamoto consensus.

PKE public key encryption.

PKG private key generator.

PKI public key infrastructure.

PPSC	privacy-preserving smart contracts.
PPSC-AD	auditor smart contract.
PPSC-KM	key management smart contract.
RBE	registration-based encryption.
SGX	Intel software guard extensions.
TEE	trusted execution environment.
TTP	trusted third party.

Chapter 1

Introduction

Security protocols [68, 146, 154], also called *cryptographic protocols*, typically consist of families of cryptographic primitives, which are defined as a series of security-related functions with message exchanges between multiple parties to achieve a specific security objective. Efficient cryptographic protocols generally describe how the cryptographic primitives are used, including the details of data structures, workflows, and representations. Cryptographic protocols are essential components of multiple interoperable versions of a program since they provide the most common techniques of providing information security [44].

From a high-level perspective, cryptographic protocols are designed to prevent malicious activities from deviating the system from its designated function [154], and therefore build a robust system to achieve security goals such as confidentiality, authentication, data integrity and non-repudiation [100]. However, these goals are difficult to achieve in some protocols without a trusted third party (TTP) or several trusted third parties. It has been proved that a fair exchange protocol cannot be completely fair without the help of a TTP [168]. Even if, in a simple example that Alice wants to send an encrypted message to Bob, without a TTP's assurance, it is difficult for Alice to confirm that Bob's identity is, in fact, the person for whom the information is intended.

With requirements for computer systems becoming increasingly complex, various TTP-based security protocols ranging from identity-based encryption (IBE) [28], public key encryption (PKE) [66] and signature scheme [178] have been proposed. Many of them are widely used in credential systems [183], timestamping systems [7], fair exchange systems [103, 104, 187] and attestation service [49, 188]. In these protocols or systems, a TTP offers a value-added communication service for users to aid them in achieving security properties. As a consequence, TTP plays an important role in providing ideal solutions to preserve confidentiality, integrity, and authenticity [173]. For example, in an anonymous credential system such as ABC4Trust [183], a TTP is introduced to help the credential issuer to reveal the identity and trace the credential, which enables the system to achieve conditional privacy; A TTP is adopted as a private key generator (PKG) to produce and distribute private keys in an IBE system [28], certificate-based encryption (CBE) system [83, 90] and certificateless encryption system [65]; In public key infrastructure (PKI) [150] system, the certificate authority (CA), inherently performing the function of a TTP, attests that a given public key indeed belongs to the user who claims it.

1.1 Challenges

“In this real-world view of security, a problem does not disappear because a designer assumes it away. The invocation or assumption in a security protocol design of a "trusted third party" (TTP) or a "trusted computing base" (TCB) controlled by a third party constitutes the introduction of a security hole into that design. ...., TTP assumptions cause most of the costs and risks in a security protocol.”

- Nick Szabo.

In TTP-based security protocols, a TTP must be entirely or partially trustworthy, requiring the TTP to hold an implicit assumption that it behaves “as it should”. However, this assumption is impractical in real-world settings, and applying a TTP to a protocol without compromising any claimed security goals is tricky. As Szabo pointed out, the TTP assumptions have become most of the costs and risks in protocol implementation [195]. Considering its threat model, the reasons that cause a TTP’s failure can be roughly categorized into two types: *internal evilness* and *external attacks*, indicating that a TTP may become malicious intentionally or may be compromised by an external adversary.

TTP may become malicious. A TTP may become malicious due to hidden interests. Abilities to fully control the operations of critical components put a TTP in a position where it can easily commit irregularities. A malicious TTP is sharply destructive for cryptographic protocols. It may deviate from the designated functions of losing targeted properties, which further causes the system to fail completely. This fact on the harmfulness of the malicious TTP has been publicly acknowledged [68, 128, 146]. Typical examples that support the above opinion are listed as follows.

- PKI system relies on CA to issue the certificates. However, a malicious CA may issue a bogus certificate for some domain names to launch impersonation or Man-in-the-Middle attacks [56], thus making the network communication between a client and a server become insecure. In fact, these vulnerabilities have been acknowledged and exploited in [56, 68].
- In the early-stage IBE system [28], PKG has to be completely trusted for managing users' private key. However, PKG may engage in malicious activities such as generating and re-distributing private keys for self-interest or, even worse, arbitrarily decrypting ciphertexts. The key escrow problem has been suggested to explain why IBE has not undergone rapid adoption as a standard [55].
- In a CBE system [83, 90, 142], CA, as a TTP, should issue certificates for message decryption. However, CA may maliciously revoke a valid certificate or arbitrarily deny a user's request for certificate issuance, which indirectly makes the user's decryption fail. Worse still, the revocation process privately happens in an isolated environment. Users have no ways to blame a CA because they lack valid evidence to prove the CA's malicious behaviour [208].

In the past decades, many different approaches have been proposed to prevent TTP's malicious activities. Among them, the distributed TTP and accountable TTP are two most featured approaches. The first approach focuses on distributing one single TTP's functionalities and responsibilities to multiple parties (e.g., sub-TTPs). The standard techniques include Byzantine agreement [20, 32], secure multi-party computation [19] and secret sharing [17]. A typical example is distributed PKGs [116] in IBE that combines the secret-sharing protocol and key generation protocol. In such a solution, the master secret is shared among multiple parties, thus achieving fairness and security. Unfortunately, this approach pays a high price at communication complexity, participation motivation and collusion issues. To be specific, a distributed solution inevitably entails extra communication. In the above case [116], a user has to interact with t sub-PKGs ($1 \leq t \leq n$, n is the total number of the sub-PKGs) for constructing her final private key. Beyond that, these participants lack the motivation to invest in higher security. Without offering external incentives/punishment methods to help sub-PKGs behave honestly, sub-PKGs may conspire to break the security promises [149]. These challenges make it difficult for distributed TTP to be widely adopted in reality.

The second approach tries to make a TTP's action accountable by showing evidence of misbehaviours. A typical example is the accountable authority IBE [131] that is based on the tracing algorithm, in which a malicious PKG runs the risk of being caught or sued if it discloses a decryption key associated with an identity over the Internet. However, this approach cannot guarantee that all TTPs' misbehaviours can be caught, since the tracing algorithm needs to take the maliciously issued (full or partial) decryption key as input. In the real world, finding such a malicious key from the Internet is difficult. Another route to achieve accountability is to employ one or more extra TTPs (monitors) to monitor functional TTP. Taking Accountable Key Infrastructure (AKI) [124] as an example, AKI leverages public log servers to enable public integrity validation for certificate information and further makes CA's (functional TTP's) actions accountable, thus creating deterrence against fraudulent CA activities. However, collusion problem and motivation-lacking issue among different TTPs (monitors) still remain.

TTP may be compromised by an adversary. Even if a TTP always behaves honestly and follows the rule as intended, there is a risk of being compromised by external adversaries. A TTP usually needs to store a piece of confidential information, normally privacy-sensitive, for the data owner to support their services. As in the IBE scheme discussed above, a master private key must be maintained by PKG to generate the user's private key. Another example is that, a CA is required to maintain a secret signing key in the PKI scheme to provide a signature for the certificate issuance and revocation. Either an intentional or unintentional leakage of private information will cause the TTP to be imitated immediately and, as a result, render all communications in the network insecure. Moreover, this leakage is easy to occur in real scenarios, as a TTP is actually an organization or a regulatory authority whose devices (e.g., servers, clients) are untrusted. The system environment (e.g., CPU, memory, disk) that stores the private information can easily be monitored or observed by adversaries. Many real-world attacks have supported this claim. For example, a huge amount of data stored in a database hosted by the Alibaba cloud was compromised recently, causing over 1 billion records of Chinese citizens to be leaked [224]. Another example is that over 1.2 million customers' SSL keys were leaked by GoDaddy since the attackers gained initial access permission to untrusted hosting platforms [200].

1.2 Motivation

With these considerations in mind, a TTP plays an essential role in security protocols but tends to suffer practical security and privacy issues. Thus, it is not surprising that achieving a fair TTP, or privacy-preserving TTP, has become one of the main research targets in the security area during the past decades. This thesis extends such a research direction: mitigating the TTP issue and further building security protocols by using innovative techniques: blockchain [108, 158, 213] systems and Trusted Execution Environments (TEEs) [61, 112, 133, 172].

The first goal of this thesis is to explore how to use blockchain to establish a secure and fair TTP, in which blockchain is a distributed and append-only ledger that maintains a continuously growing list of data records [85, 158]. In particular, a smart contract as the core application for a blockchain provides a transparent and automatic environment to execute pre-defined logic. We will explore how smart contracts can be employed to help the existing untrusted TTP.

Blockchain-based smart contracts lack confidentiality. The state information and the instruction code in contracts are completely transparent, and any state and its changes are publicly accessible. Without privacy, building advanced privacy-preserving protocols becomes a challenge. For this reason, the second motivation of this thesis is to combine TEEs and blockchain to build a fair and privacy-preserving TTP for aiding advanced protocols that are hard to achieve using the traditional cryptographic primitives.

Despite these many benefits, the blockchain system, as a peer-to-peer system, still has low performance and poor scalability problems [102]. For instance, the transactions per second (TPS) of Bitcoin is about 7 [134], while Ethereum can only handle an average of 15 TPS, which significantly restricts the development of upper-layer applications. To make the blockchain-based protocols more widely adopted, building a practical blockchain system with good performance and scalability becomes urgent and crucial. Thereupon, the third motivation of this thesis is to explore several innovative approaches to achieve a better blockchain system.

1.3 Main Contributions

The main contributions of this thesis are listed as follows.

- We introduce a contract-based TTP that achieves fairness (called TTP-I). Then, we apply such a type of TTP to rebuild existing security protocols, including CBE and registration-based encryption (RBE). The first protocol is proposed to provide a transparent certificate revocation mechanism for CBE [83, 90, 142], where a smart contract is involved as a TTP to assist CA in managing the certificate revocation(cf. [208]). The second protocol introduces a transparent RBE [86, 87], which transfers the right of key management from the centralized key curator (KC) to individual participants by on-chain registration.

- We investigate the state-of-the-art of *TEE-aided confidential smart contract* (TCSC) systems (cf. [138]). Based on that, we propose a hybrid TTP based on TCSC (called TTP-II). Furthermore, we explore the core mechanism of TTP-II, and separately apply it to a credential anonymity revocation system (cf. [135]) and an accountable decryption system (cf. [136]). The first cryptosystem proposes a novel anonymity revocation approach, where the revocation codes are running in TCSC, which offers confidentiality as well as auditability. The second cryptosystem uses TCSC as a decentralized key manager and a neutral auditor to make investigators' actions accountable during the execution of a warrant for accessing users' sensitive data.

- We propose two core blockchain algorithms to enhance blockchain scalability, performance, and usability: a weak consensus algorithm (cf. [205]) and a fast delegatable payment algorithm (cf. [137, 139]). The weak consensus algorithm provides good scalability by relaxing the strong consistency promise, providing a high-performance blockchain solution. The delegatable payment system allows users to pay the cryptocurrency instantly under the help of TEE, which increases the user's usability when using cryptocurrency. Notably, the proposed weak consensus algorithm has recently been adopted by a digital economics start-up [74]. Using our algorithm, they proposed an innovative NFT system [206] with the advanced properties of fast certification and low transaction fees.

1.4 Thesis Structure

Figure 1.1 depicts the logical organization of the research topics. Each chapter identifies the research problem mentioned in the previous chapter and then provides solutions for that problem. This thesis is organized as follows.

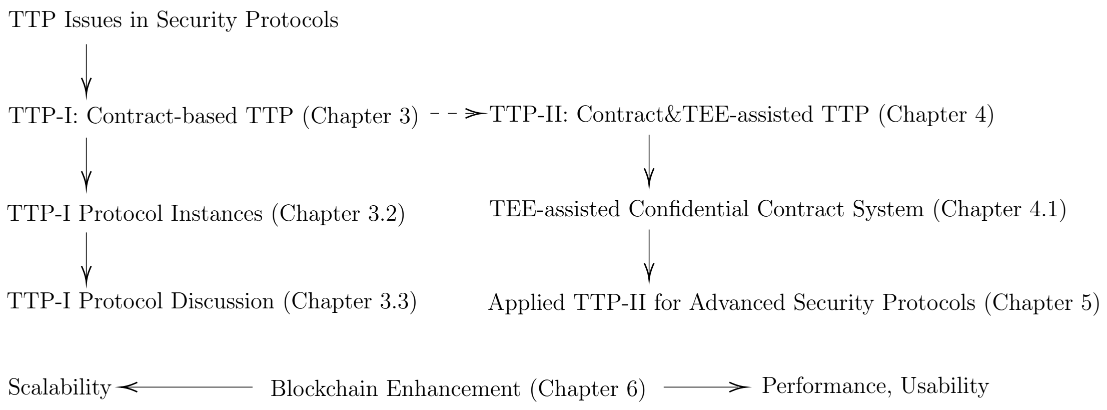

TTP Issues in Security Protocols
TTP-I: Contract-based TTP (Chapter 3)
TTP-II: Contract&TEE-assisted TTP (Chapter 4)
TTP-I Protocol Instances (Chapter 3.2)
TEE-assisted Confidential Contract System (Chapter 4.1)
TTP-I Protocol Discussion (Chapter 3.3)
Applied TTP-II for Advanced Security Protocols (Chapter 5)
Scalability
Blockchain Enhancement (Chapter 6)
Performance, Usability

Figure 1.1: Structure of the thesis and relationships between components

Chapter 2 presents the background for this thesis. Beginning with several related crypto preliminaries and assumptions, this chapter presents the concepts of the blockchain, smart contract and TEEs.

Chapter 3 introduces a smart contract-based TTP, called TTP-I. TTP-I regards the smart contract as an agent to achieve fairness. Next, two security protocols using TTP-I are given to illustrate the applicabilities and drawbacks.

Chapter 4 introduces a new type of contract, called the TEE-assisted smart contract (TCSC), aiming to address the privacy issues associated with smart contracts. It begins with a comprehensive investigation and evaluation of existing TCSC systems. Afterwards, a syntax of TCSC and a general construction of TCSC-based protocol (called TTP-II protocol) are defined, with emphasis on the difference between TTP-I protocol.

Chapter 5 gives two security protocols adopting TTP-II: an auditable credential anonymity revocation protocol and an accountable decryption protocol. These protocols employ TCSC as a high-level primitive, enjoying accountability and traceability properties. Both protocols provide the security analysis, full implementation, and comprehensive evaluation, followed by use case discussions.

Chapter 6 begins with the main challenges of current blockchain systems: *lack of scalability*, *low performance* and *poor usability*. Then, two algorithms are proposed to solve the above issues. The first algorithm allows running with parallel chains, providing an improved-scalability blockchain solution. The second algorithm exploits TEEs as decentralized "virtual agents" to prevent malicious delegation, which facilitates payments and improves users' usability.

Chapter 7 concludes this thesis and points out the future work.

1.5 List of Publications

The work covered in this thesis has been published in the following papers:

1. **Auditable Credential Anonymity Revocation Based on Privacy-Preserving Smart Contracts.** The 3rd International Workshop on Cryptocurrencies and Blockchain Technology on the 24th edition of the European Symposium on Research in Computer Security (ESORICS 19), Rujia Li, David Galindo, Qi Wang (Travel Grant Awarded).

2. **Poster: Transparent Certificate Revocation for CBE Based on Blockchain.** The 41st IEEE Symposium on Security and Privacy (IEEE S&P 20). Qin Wang*, Rujia Li*, Qi Wang, David Galindo.
3. **An Accountable Decryption System Based on Privacy-Preserving Smart Contracts.** The 23rd Information Security Conference (ISC 20). Rujia Li, Qin Wang, Feng Liu, Qi Wang, David Galindo.
4. **Poster: A Weak Consensus Algorithm and Its Application to High-Performance Blockchain.** The 28th Annual Network and Distributed System Security Symposium (NDSS 21). Qin Wang, Rujia Li, Qi Wang.
5. **A Weak Consensus Algorithm and Its Application to High-Performance Blockchain.** The 40th Annual IEEE Conference on Computer Communications (INFOCOM 21). Qin Wang*, Rujia Li*.
6. **Poster: An Offline Delegatable Cryptocurrency System.** The 28th Annual Network and Distributed System Security Symposium (NDSS 21). Rujia Li, Qin Wang, Xinrui Zhang, Qi Wang, David Galindo, Yang Xiang.
7. **An Offline Delegatable Cryptocurrency System.** The 3rd IEEE International Conference on Blockchain and Cryptocurrency (ICBC 21). Rujia Li*, Qin Wang*, Xinrui Zhang, Qi Wang, David Galindo, Yang Xiang.
8. **SoK: TEE-assisted Confidential Smart Contract.** The 22nd Privacy Enhancing Technologies Symposium (PETS 22). Rujia Li*, Qin Wang*, Qi Wang, David Galindo, Mark Ryan.
9. **Exploring Unfairness on Proof of Authority: Order Manipulation Attacks and Remedies.** The 17th ACM ASIA Conference on Computer and Communications Security (ASIACCS 22) Qin Wang*, Rujia Li*, Qi Wang, Shiping Chen, Yang Xiang.
10. **How Do Smart Contracts Benefit Security Protocols?** ArXiv.org E-Print Archive 22. https://arxiv.org/pdf/2202.08699.pdf. Rujia Li*, Qin Wang*, Qi Wang, David Galindo.

Chapter 2

Background

This chapter introduces cryptographic definitions (Section 2.1), the related background on the blockchain and smart contract (Section 2.2), and TEEs (Section 2.3). Note that our cryptographic definitions only capture the main preliminaries used in this thesis, and minor details such as security definitions of CBE and RBE are omitted.

2.1 Preliminaries and Assumptions

2.1.1 Preliminaries

Symmetric Encryption. A symmetric encryption scheme SE contains the following algorithms.

- $\mathsf{SE.KeyGen}(1^\lambda)$ The algorithm takes as input a security parameter λ and outputs a secret key sk.
- $\mathsf{SE.Enc}(sk, m)$ The algorithm takes as input sk and a message $m \in \mathcal{M}$, outputs a ciphertext ct.
- $\mathsf{SE.Dec}(sk, ct)$ The algorithm takes as input sk, the ciphertext ct, and outputs a message $m \in \mathcal{M}$.

Correctness. A symmetric encryption scheme SE is correct if for all $m \in \mathcal{M}$ and every secret key sk generated by $\mathsf{SE.KeyGen}(1^\lambda)$, it holds that,

$$\mathsf{SE.Dec}(sk, (\mathsf{SE.Enc}(sk, m))) = m.$$

A secure symmetric encryption scheme SE should provide data confidentiality. In particular, an adversary cannot learn which message is encrypted in a ciphertext. Formally, the security of SE is defined as follows.

Definition 1 (IND-CPA security of SE). *A symmetric encryption scheme* SE *achieves Indistinguishability under Chosen-Plaintext Attack (IND-CPA) if for all PPT adversaries, there exists a negligible function $negl(\lambda)$ such that*

$$\left| \Pr\left[\mathsf{G}^{\mathsf{IND-CPA}}_{\mathcal{A},\mathsf{SE}}(\lambda) = 1\right] - \frac{1}{2}\right| \leq negl(\lambda),$$

where $\mathsf{G}^{\mathsf{IND-CPA}}_{\mathcal{A},\mathsf{SE}}(\lambda)$ is defined as follows.

```
$\mathsf{G}^{\mathsf{IND-CPA}}_{\mathcal{A},\mathsf{SE}}(\lambda)$
1: $sk \leftarrow \mathsf{SE.KGen}(1^\lambda)$// $\mathcal{C}$ runs for a private $sk$
2: $b \xleftarrow{\$} \{0,1\}$// $\mathcal{C}$ chooses a random bit
3: $m_0, m_1 \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Enc}}_{sk}(\cdot)}(1^\lambda)$// $\mathcal{A}$ provides messages
4: $ct^\star \leftarrow \mathsf{SE.Enc}(sk, m_b)$// $\mathcal{C}$ replies with $ct^\star$
5: $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Enc}}_{sk}(\cdot)}(ct^\star)$//$\mathcal{A}$ finally outputs its guess $b'$
6: return $b = b'$
```

```
$\mathcal{O}^{\mathsf{Enc}}_{sk}(m)$
1: $ct \leftarrow \mathsf{SE.Enc}(sk, m)$
2: return $ct$
```

Signature Scheme. A signature scheme S [79] consists of the following algorithms.

- $\mathsf{S.KeyGen}(1^\lambda)$ The algorithm takes as input a security parameter λ and outputs a key pair (sk, vk) for signing and verification.
- $\mathsf{S.Sign}(sk, m)$ The algorithm takes as input sk and a message $m \in \mathcal{M}$, and outputs a signature σ.
- $\mathsf{S.Verify}(vk, \sigma, m)$ The algorithm takes as input vk, a signature σ, a message $m \in \mathcal{M}$, and outputs 1 or 0.

Correctness. A signature scheme S is correct if for all $m \in \mathcal{M}$ and every key pairs (vk, sk) generated by $\mathsf{S.KeyGen}(1^\lambda)$, it holds that,

$$\mathsf{S.Verify}(vk, (\mathsf{S.Sign}(sk, m)), m) = 1.$$

A signature scheme should provide authenticity. An adversary without a signing key cannot generate a valid signature. The security of the signature scheme S is formally defined as follows.

Definition 2 (EUF-CMA security of S)**.** *A signature scheme S is said to secure against Existentially Unforgeable under Chosen Message Attack (EUF-CMA) if for all PPT adversaries, there exists a negligible function $negl(\lambda)$ such that*

$$\Pr\left[\mathsf{G}_{\mathcal{A},\mathsf{S}}^{\mathsf{EUF\text{-}CMA}}(\lambda) = 1\right] \leq negl(\lambda),$$

where $\mathsf{G}_{\mathcal{A},\mathsf{S}}^{\mathsf{EUF\text{-}CMA}}(\lambda)$ is defined as follows.

```
$\mathsf{G}_{\mathcal{A},\mathsf{S}}^{\mathsf{EUF\text{-}CMA}}(\lambda)$
1 :  $(sk, vk) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$
2 :  $\mathcal{L} \leftarrow \{\}$// an empty set
3 :  $(m^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathcal{O}_{sk}^{\mathsf{Sign}}(\cdot)}(vk)$
4 :  return $\mathsf{S.Verify}(vk, \sigma^\star, m^\star) = 1 \wedge \sigma^\star \notin \mathcal{L}$
```

```
$\mathcal{O}_{sk}^{\mathsf{Sign}}(m)$
1 :  $\sigma \leftarrow \mathsf{S.Sign}(sk, m)$
2 :  $\mathcal{L} := \mathcal{L} \quad || \quad \sigma$
3 :  return $\sigma$
```

Public Key Encryption. A public key encryption scheme PKE [63] contains the following algorithms.

- $\mathsf{PKE.KeyGen}(1^\lambda)$ The algorithm takes as input a security parameter λ and generates a private key sk and a public key pk.
- $\mathsf{PKE.Enc}(pk, m)$ The algorithm takes as input a public key pk, a message $m \in \mathcal{M}$, and outputs a ciphertext ct.
- $\mathsf{PKE.Dec}(sk, ct)$ The algorithm takes as input a private key sk and a ciphertext ct, and outputs a message $m \in \mathcal{M}$.

Correctness. A public key encryption scheme PKE is correct if for all $m \in \mathcal{M}$ and all key pairs $(sk, pk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, it holds that,

$$\mathsf{PKE.Dec}(sk, (\mathsf{PKE.Enc}(pk, m))) = m,$$

A public-key encryption scheme provides confidentiality. An adversary can not learn which message is encrypted in a ciphertext, even if it equips with the decryption oracle before and after encryption. Formally, the security of PKE is defined as follows.

Definition 3 (IND-CCA2 security of PKE)**.** *A public key encryption scheme* PKE *is said to secure against Indistinguishability Security Under Adaptively Chosen Ciphertext Attack (IND-CCA2) if for all PPT adversaries, there exists a negligible function $negl(\lambda)$ such that*

$$\Big| \Pr\left[\mathsf{G}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{PKE}}(\lambda) = 1\right] - \frac{1}{2}\Big| \leq negl(\lambda),$$

where $\mathsf{G}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$ is defined as follows:

```
$\mathsf{G}^{\mathsf{IND\text{-}CCA2}}_{\mathcal{A},\mathsf{PKE}}(\lambda)$
1: $(sk, pk) \leftarrow \mathsf{PKE.KGen}(1^\lambda)$
2: $b \xleftarrow{\$} \{0,1\}$ // $\mathcal{C}$ chooses a random bit
3: $m_0, m_1 \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Dec1}}_{sk}(\cdot)}(1^\lambda)$ // $\mathcal{A}$ provides $(m_0, m_1)$
4: $ct^\star \leftarrow \mathsf{PKE.Enc}(sk, m_b)$ // $\mathcal{C}$ replies with $ct^\star$
5: $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Dec2}}_{sk}(\cdot)}(ct^\star, pk)$ // $\mathcal{A}$ outputs its guess $b'$
6: return $b = b'$
```

```
$\mathcal{O}^{\mathsf{Dec1}}_{sk}(ct)$
1: $m \leftarrow \mathsf{PKE.Dec}(sk, ct)$
2: return $m$
```

```
$\mathcal{O}^{\mathsf{Dec2}}_{sk}(ct)$
1: $if(ct == ct^\star)$ return $\perp$
2: $m \leftarrow \mathsf{PKE.Dec}(sk, ct)$
3: return $m$
```

Hash Functions. A hash function family H is a pair of algorithms $(\mathsf{Gen}, \{\mathsf{h}_k\}_k)$.

- $\mathsf{Gen}(1^\lambda)$ This algorithm takes as input a security parameter λ, and outputs a key k.

- $\mathsf{h}_k : \{0,1\}^{|m|} \rightarrow \{0,1\}^{|n|})$ This is an efficient algorithm parameterized by the key k that compresses $|m|$ length strings to $|n|$ length strings, where $|m| > |n|$.

Collision Resistant Hash Functions (CRHF) [18]. A family of functions H is said to be collision-resistant if for any probabilistic polynomial algorithm A, there exists a negligible function $negl(\lambda)$ such that

$$\Pr\left[k \leftarrow \mathsf{Gen}(1^\lambda), (x_1, x_2) \leftarrow \mathsf{A}(k, 1^\lambda) : x_1 \neq x_2 \wedge \mathsf{H}_k(x_1) = \mathsf{H}_k(x_2)\right] \leq negl(\lambda).$$

Certificate-based Encryption. A certificate-based encryption scheme CBE, first introduced by Gentry [90], consists of the following algorithms.

- $\mathsf{CBE.Gen}(1^\lambda, n)$ The algorithm takes as input a security parameter λ, the total number of time periods n, and outputs a certifier's master secret msk and public parameters pms that include the master public key mpk. The public parameters pms are implicit input for the rest of the algorithms.
- $\mathsf{CBE.Set}(1^\lambda)$ The algorithm takes as input a security parameter λ, and outputs a user's key pair (pk, sk). The algorithm is run by users.
- $\mathsf{CBE.Cert}(msk, i, user, pk)$ At the start of each time i, CA takes as input msk, a user's information $user$ and a user's public key pk, and outputs the user's certificate $Cert_i$.
- $\mathsf{CBE.Enc}(m, i, user, pk)$ The algorithm takes as input a message m, a user's information $user$, a user's public key pk at time period i, and returns a ciphertext ct.
- $\mathsf{CBE.Dec}(Cert_i, sk, ct)$ At time period i, the algorithm takes as input a certificate $Cert_i$, a user's private key sk, and the ciphertext ct, and then outputs a message m or a special symbol $\perp$ indicating a decryption failure.

Correctness. A certificate-based encryption scheme CBE is correct if at the time period $i \in n$, for all $m \in \mathcal{M}$, all user's key pairs (sk, pk) output by $\mathsf{CBE.Set}(1^\lambda)$, all msk output by $\mathsf{CBE.Gen}(1^\lambda, n)$ and all $Cert_i$ output by $\mathsf{CBE.Cert}(msk, i, user, pk)$, it holds that,

$$\mathsf{CBE.Dec}(Cert_i, sk, (\mathsf{CBE.Enc}(m, i, user, pk)) = m.$$

Registration-based Encryption. A registration-based encryption scheme RBE, first proposed by Garg in 2018 [86], is composed of the following algorithms.

- $\mathsf{RBE.Setup}(1^\lambda)$ The algorithm takes as input a security parameter λ and outputs a common random string crs. Here, crs can be sampled publicly using public randomness beacon.

- $\mathsf{RBE.KeyGen}(1^\lambda)$ The algorithm takes as input a security parameter λ, and outputs a user's key pair (sk, pk). Note that these keys are only public and secret keys, not the encryption or decryption keys.

- $\mathsf{RBE.Reg}^{[\mathsf{aux}]}(crs, pp, id, pk)$ The algorithm takes as input crs, current parameter pp, a registering identity id, a public key pk and outputs the updated public parameter pp'. The auxiliary information aux stores all the data on users' identifiers/corresponding public keys and the old parameters. It will be updated into aux' during the process of registration (in the setup stage, public parameters pp are initialized as $\perp$ and the auxiliary information aux is configured as $\varnothing$).

- $\mathsf{RBE.Enc}(crs, pp, id, m)$ The algorithm takes as input crs, a public parameter pp, a recipient identity id, a message m, and outputs a ciphertext ct.

- $\mathsf{RBE.Upd}^{[\mathsf{aux}]}(pp, id)$ The algorithm takes as input a user identity id, current public parameters pp and outputs newly updated public parameters u that can help id to decrypt its messages. The update of the public parameters is achieved by reading the auxiliary information aux.

- $\mathsf{RBE.Dec}(sk, u, ct)$ The algorithm takes as input a secret key sk, an update information u[1], a ciphertext ct, and outputs a message $m \in \{0,1\}^*$ or in $\{\perp, \mathsf{GetUpd}\}$. The symbol $\perp$ indicates a syntax error, while GetUpd indicates that u needs to be updated.

Correctness. A registration-based encryption scheme RBE is correct if for all $m \in \mathcal{M}$, all crs output by $\mathsf{RBE.Setup}(1^\lambda)$ all identities id, all user's key pairs (sk, pk)

[1]Here, u represents the latest public parameter (e.g., pp or pp'), it will be updated after some registrations of new users.

generated by $\mathsf{RBE.KeyGen}(1^\lambda)$, and u output by $\mathsf{RBE.Upd}^{[\mathsf{aux}]}(pp, id)$, it holds that,

$$\mathsf{CBE.Dec}(sk, u, (\mathsf{CBE.Enc}(crs, pp, id, m)) = m,$$

Fair Blind Signature. A fair blind signature scheme FBS, proposed by Stadler *et al.* [192], is defined by the following algorithms.

- $\mathsf{FBS.Setup}(1^\lambda)$ The algorithm takes as input a security parameter λ, and outputs public parameters pms, which are implicit input for the rest of the algorithms.
- $\mathsf{FBS.KeyGen}(1^\lambda)$ The algorithm takes as input a security parameter λ, and outputs a signing key pair (isk, ipk) and a private and public revocation key pair (rsk, rpk) for the issuer. Here, (rsk, rpk) can be independent of (sk, pk). Meanwhile, a user runs this algorithm to obtain a key pair (usk, upk).
- $\mathsf{FBS.Sign}(isk, rpk, m)$ The algorithm takes as input a signing private key sk, a revocation key key rpk, a message m, and outputs a blind signature σ.
- $\mathsf{FBS.Verify}(ipk, \sigma, m)$ The algorithm takes as input an issuer's public key ipk, a signature σ, a message m, and outputs the verification result $true$ or $false$.
- $\mathsf{FBS.TraceSig}(rsk, view)$ The algorithm takes as input a revocation key rsk, and a $view$ of the issuer during the target session, and outputs a signature identifier I_{sig}.
- $\mathsf{FBS.MatchSig}(I_{sig}, \sigma)$ The algorithm takes as input a signature identifier I_{sig} and signature σ, and outputs $true$ or $false$.
- $\mathsf{FBS.TraceSession}(rsk, \sigma)$ The algorithm takes as input a revocation key rsk, a target signature σ and outputs a session identifier I_{sid}.
- $\mathsf{FBS.MatchSession}(I_{sid}, view)$ The algorithm takes as input a session identifier I_{sid} and a $view$, and outputs $true$ or $false$.

Correctness. It is assumed that a fair blind signature scheme FBS is correct if the following conditions are satisfied.

- For all $m \in \mathcal{M}$, all signing key pairs (isk, ipk), and all revocation keys rpk

output by $\mathsf{FBS.KeyGen}(1^\lambda)$, it holds that,

$$\mathsf{FBS.Verify}(ipk, (\mathsf{FBS.Sign}(isk, rpk, m)), m) = true.$$

- For any *view*, as observed by an issuer, all revocation keys rsk output by $\mathsf{FBS.KeyGen}(1^\lambda)$, all signatures σ output by $\mathsf{FBS.Sign}(isk, rpk, m)$ in this *view*, it holds that,

$$\mathsf{FBS.MatchSig}((\mathsf{FBS.TraceSig}(rsk, view)), \sigma) = true.$$

- For any σ output by $\mathsf{FBS.Sign}(isk, rpk, m)$, as observed by the issuer, all revocation keys rsk output by $\mathsf{FBS.KeyGen}(1^\lambda)$, it holds that,

$$\mathsf{FBS.MatchSession}((\mathsf{FBS.TraceSession}(rsk, \sigma)), view) = true,$$

2.1.2 Assumptions

Decision Linear Assumption. The Decision Linear Assumption [27, 122] is based on the Linear Problem, which is defined as follows.

Definition 4 (Decision Linear Problem [122]). *Let $\mathbb{G}$ be a cyclic multiplicative group with a prime order p, and g_1, g_2, g_3 be generators of $\mathbb{G}$. Given the groups $g_1, g_2, g_3, g_1^a, g_2^b, g_3^c \in \mathbb{G}$, decide whether $a+b$ equals to c. If $a+b=c$, output true, or false otherwise. The advantage of an algorithm $\mathcal{A}$ in deciding the linear problem in $\mathbb{G}$ is*

$$adv_{\mathcal{A}}^{LP} = \left| \begin{array}{l} Pr[\mathcal{A}(g_1, g_2, g_3, g_1^a, g_2^b, g_3^{a+b}) = true: \\ \qquad\qquad\qquad g_1, g_2, g_3 \leftarrow \mathbb{G}, a, b \leftarrow \mathbb{Z}_p] \\ -Pr[\mathcal{A}(g_1, g_2, g_3, g_1^a, g_2^b, \eta) = true: \\ \qquad\qquad\qquad g_1, g_2, g_3, \eta \leftarrow \mathbb{G}, a, b \leftarrow \mathbb{Z}_p] \end{array} \right|,$$

with the probability taken over the uniform random choice of the parameters to $\mathcal{A}$ and over the coin tosses of $\mathcal{A}$.

Assumption 1 (Decision Linear Assumption). *No adversary $\mathcal{A}$ succeeds in decid-*

ing the Linear Problem in $\mathbb{G}$ with a non-negligible advantage.

Definition 5 (**Indistinguishability Obfuscation**). A uniform PPT algorithm Obf [15, 86] is called an indistinguishability obfuscator for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ (where each $\mathcal{C}_\lambda$ is a set indexed by a security parameter λ), if the following requirements are satisfied:

- *For all security parameters $\lambda \in \mathbb{N}$ and all circuits $\mathsf{C} \in \mathcal{C}_\lambda$, we obtain an obfuscation with the same function:*

$$\Pr_{\mathsf{Obf}}[\mathsf{C}' \equiv \mathsf{C} : \mathsf{C}' = \mathsf{Obf}(1^\lambda, \mathsf{C})] = 1.$$

- *For any PPT distinguisher D, there exists a negligible function $negl(\lambda)$ such that for all $\lambda \in \mathbb{N}$, for all pairs of functionally equivalent circuits $\mathsf{C}_1 \equiv \mathsf{C}_2$ from the same family $\mathsf{C}_1, \mathsf{C}_2 \in \mathcal{C}_\lambda$,*

$$\left|\Pr_{\mathsf{Obf}}[\mathsf{D}(1^\lambda, \mathsf{Obf}(1^\lambda, \mathsf{C}_1)) = 1] - \Pr_{\mathsf{Obf}}[\mathsf{D}(1^\lambda, \mathsf{Obf}(1^\lambda, \mathsf{C}_2)) = 1]\right| \leq negl(\lambda).$$

2.2 Blockchain and Smart Contracts

2.2.1 Blockchain

Blockchain, conceptualized by Nakamoto [158], was proposed as a distributed and append-only ledger in which all committed transactions are stored in a chain of data records (also named as *blocks*). According to the initial idea of Bitcoin [194], when blockchain maintainers reach an agreement on the newest block, transactions appearing in that time will be packaged in this block and further stored in the distributed network to maintain a continuously growing list. By providing a secure consensus solution to distribute information and allowing all participants to audit shared records, blockchain obtains many key characteristics such as decentralization and data transparency. Inspired by [84], we define a secure blockchain system as follows.

Assumption 2 (**Blockchain Assumption [84]**). *A blockchain is a robust public transaction ledger if it satisfies the following properties.*

- ***Persistence.*** *Once one honest player accepts a transaction (a transaction is stored more than k blocks deep*[2]*), other honest players (the number depending on a certain consensus algorithm) will eventually accept such a transaction.*
- ***Liveness.*** *As long as a transaction comes from an honest account holder, it will be accepted within time-bound δ by the honest blockchain players (the number depends on a certain consensus algorithm).*

Briefly speaking, the *persistence* assumption says a transaction that has been accepted by an honest player will be accepted (ended up at a depth of more than k blocks) in other honest players' local chains. Meanwhile, the *liveness* assumption states that all honest players will eventually agree on a decision or a value. The "eventually" indicates that it may take a delay time δ (δ is finite) to reach the agreement. By combining persistence and liveness, it ensures that the public ledger can only accept authentic transactions and will make them permanent.

2.2.2 Smart Contract

Smart contract was originally introduced by Szabo [194], and further first applied in blockchain systems by Ethereum [213]. Blockchain-based smart contracts adopt Turing-complete scripting languages to achieve complex functionalities [108] and execute through state transition/replication over consensus algorithms to realize final consistency. By its design, a smart contract includes parameters, multiple functions, and methods that can run on the blockchain when certain conditions or events are met. Specifically, the source code of a contract forming as part of a transaction is first sent to the blockchain. Once the transaction is included in a new block and confirmed by a majority of participants, the contract code becomes immutable and executable. When an outside user calls the contract, the state will be updated under the instruction of the preloaded source code. The neutrality of the execution environment among all blockchain nodes facilitates the same execution result of the program code. Smart contracts thus enable unfamiliar participants to exchange fairly, and thus provide a uniform approach to improve applications across a wide range of industries.

[2]We use the notation of Garay and Kiayias [84], e.g., k, δ to describe the properties.

Without loss of generality, we use Ethereum [213] as an example to explain the workflow and benefits of smart contracts. After implementing a smart contract, a user can compile it to obtain the *bytecode* and application binary interface (ABI). Next, the user sends a transaction including *bytecode* and ABI to the Ethereum miner. Once a miner receives the transaction, the *bytecode* will be included in the next block to ensure that a new contract has been created in the blockchain network. At the same time, the initial state will be set up for this contract.

Next, when a user wants to trigger the contract execution in the contract invocation stage, he needs to send a transaction to a blockchain node/player. Then, this node starts to run the operation code, locally updates the contract state, and broadcasts the transaction and block. Other nodes repeat the above steps. From a high-level perspective, smart contracts work as a state-machine replication. The states of contracts are replicated across different players in a distributed environment. The players participating in the system will automatically replicate the current state and transfer to a new state after a consensus round [22]. Thus, we define the smart contract as a distributed state machine, as shown in Definition 6.

Definition 6 ($\widehat{\mathcal{SC}}$)**.** *Smart contract is represented as a state machine by a tuple* $\langle \mathcal{S}, \mathcal{S}', \mathcal{T}, \mathbb{B} \rangle$*, which is defined as:*

$$f : \mathcal{S}' \xleftarrow{\mathbb{B}} \mathcal{S} \otimes \mathcal{T},$$

where the $\mathcal{S}$ *represents a set of states or views,* $\mathcal{S}'$ *is the new state set after the specified operations,* $\mathcal{T}$ *means the transaction space that can trigger the execution of a contract,* f *is the transition function describing the changes of states, the blockchain* $\mathbb{B}$ *provides a distributed computing network.*

A complete execution of a smart contract in blockchain systems consists of three procedures: *contract deploy*, *contract state transfer*, and *contract state read*. The predefined logic can be coded into a file *bytecode* for further deployment. Three sub-procedures are presented as follows.

- **Deploy.** $(\langle \mathit{opcode} \rangle, \langle \mathit{reqcode} \rangle, s) \leftarrow \langle \mathit{bytecode} \rangle \otimes \mathsf{Tx}$. The deployment is triggered by a transaction Tx where $\mathsf{Tx} \in \mathcal{T}$. It takes as input the binary code $\langle \mathit{bytecode} \rangle$, and outputs initial state s, where $s \in \mathcal{S}$. The contract is compiled into instruction codes $\langle opcode \rangle$ and $\langle reqcode \rangle$, where $\langle opcode \rangle$ specifies the operation set to be executed and $\langle reqcode \rangle$ defines the conditions depending on which the operation of $\langle opcode \rangle$ can be conducted.

- **Transfer.** $s' \overset{\mathbb{B}}{\leftarrow} \langle opcode \rangle \otimes \langle reqcode \rangle \otimes s \otimes \mathsf{Tx}'$. By sending a transaction Tx' with some inputs (optional), current state s is transited to a new state s' under the guidance of $\langle opcode \rangle$ and $\langle reqcode \rangle$, where $s' \in \mathcal{S}'$.
- **Read.** $s'' \overset{\mathbb{B}}{\leftarrow} \langle opcode \rangle \otimes \langle reqcode \rangle \otimes s'$. By sending a query request, the contract state s'' is returned by scanning the blockchain storage.

In this definition, all the state s, s', s'' are completely transparent. Also, any state changes in **Transfer** step are publicly accessible and publicly verifiable: (1) all instruction codes $\langle opcode \rangle$ and $\langle reqcode \rangle$ are visible to any observer; (2) the transactions Tx, Tx', and their executions on the instruction codes in a certain blockchain node will be verified by all other nodes.

2.3 Trusted Execution Environments

Trusted Execution Environment (TEE) [69] provides a protected area in the main processor that runs on a separation kernel to ensure confidentiality and integrity of inside data and computations. State-of-the-art implementations include Intel Software Guard Extensions (SGX) [61], ARM TrustZone [172], RISC-V Keystone [133], *etc.* For a TEE, three main TEE features are highlighted, including *runtime isolation*, *local/remote attestation* and *sealing technologies*. For simplicity, we use Intel SGX as an example to establish the concept of these features in the following section. It should be mentioned that TEE design used in our thesis can also be implemented on other trusted hardware platforms, as illustrated in Figure 2.1.

2.3.1 Runtime Isolation

SGX-enabled CPU protects the confidentiality and integrity of the internal computation by creating secure and isolated memory regions named *enclaves*. Sensitive data and intermediate computations run inside enclaves are protected against outside programs, including the operating system, hypervisor and hardware devices

- ARM
  - TrustZone [172]
  - OP-TEE [160]
  - Sanctuary [34]
  - Komodo [75]
- Intel
  - SGX [61]
  - Haven [16]
  - Graphene-SGX [198]
  - Scone [9]
- RISC-V
  - Sanctum [62]
  - MultiZone [162]
  - Keystone [133]
  - TIMBER-V [210]
- AMD
  - SEV [112]
  - SEV-ES [156]
  - SEV-SNP [113]

Figure 2.1: State-of-the-art implementations of existing TEEs/extensions

attached to the system bus. To be specific, Intel SGX reserves a memory area, called Processor Reserved Memory (PRM), to protect against all non-enclave memory access. PRM holds the Enclave Page Cache (EPC), which uses 4KB (kilobyte) pages to store enclave code and data. The allocation of the EPC page to each enclave is delegated to the outside untrusted software. When initial code and data are loaded, the system software requires the CPU to copy them from unprotected memory (outside the PRM) to the EPC page and assign the page to the enclave. Then, the system software requires the CPU to mark the enclave as initialized once all enclaves are loaded into EPC. At this point, the application code can be run inside the enclave, and all the loading methods defined above become disabled. Furthermore, to avoid leaking confidential data, a CPU running the enclave code is not allowed to directly interrupt the page when a page fault happens. Instead, it is required to perform an Asynchronous Enclave Exit (AEE) from enclave code to ring 3 code [115], and then serves the above functions.

2.3.2 Local/Remote Attestation

Attestation mechanism [151] is used to prove to a validator that an enclave has been correctly instantiated. When in that condition, the enclave can then proceed to establish a secure, authenticated connection for data transmission. TEE provides two kinds of attestation: *local attestation* and *remote attestation.* Local attestation is achieved by a specific measurement hash that is initialized when a TEE starts, while remote attestation depends on the measurement and attestation signature signed by the trusted hardware. Again, we employ Intel SGX as an instance to establish the attestation concept.

Local Attestation. In Intel SGX, local attestation is used to help an enclave to attest itself to another enclave that they are running on the same Trusted Computing Base (TCB) platform. When the target enclave is required to send the attestation report, it first uses EGETKEY instruction to derive a report key. Then, it uses EREPORT instruction to produce an attestation Report (REPORT) that binds a message supplied by the enclave with the enclave's measurement. The binding is accomplished by a Message Authentication Code (MAC) tag using the same report key that is shared with all enclaves initiated by the same platform.

Remote Attestation. Intel SGX enables an enclave to prove a correct loading of code and data to another enclave that resides in a remote platform. Remote attestation depends on local attestation report. After all provisioning steps have been completed, the Quoting Enclave first obtains the Provisioning Seal Key and uses it to decrypt the Attestation Key. Then, it invokes the instruction of EGETKEY to derive the Report keys for verifying received local attestation reports. If these reports are verified successfully, the Quoting Enclave will replace their MAC with an Attestation Signature generated by the Attestation Key. To have a clear understanding, we will give more details on the key management inside TEE. In the manufacturing process of the SGX-enabled processor, the manufacturer communicates with Intel's key generation facility and generates two secrets burned into e-fuses: *Provisioning Secret* and *Seal Secret* (deviate from Intel's official documents). The Provisioning Secret, burned into the e-fuses of each SGX-enabled processor, is generated at a key generation facility and stored in Intel's provisioning service. On the contrary, the Seal Secret is produced inside the processor chip, which is not known to Intel. Given the *Provisioning Secret*, the EGETKEY instruction derives the Provisioning key using the enclave's certificate-based identity and SGX implementation's SVN. This brings desirable security properties, such as quickly convincing the provisioning service that it is communicating to a trusted Provisioning Enclave (PE) in secure environments provided by an SGX-enabled processor. Once authentication is completed, the Intel provisioning service generates an Attestation Key (AK) and sends it back to PE. Afterwards, PE encrypts

AK using a Provisioning Seal key and then stores the encrypted key to the system software for Quoting Enclave.

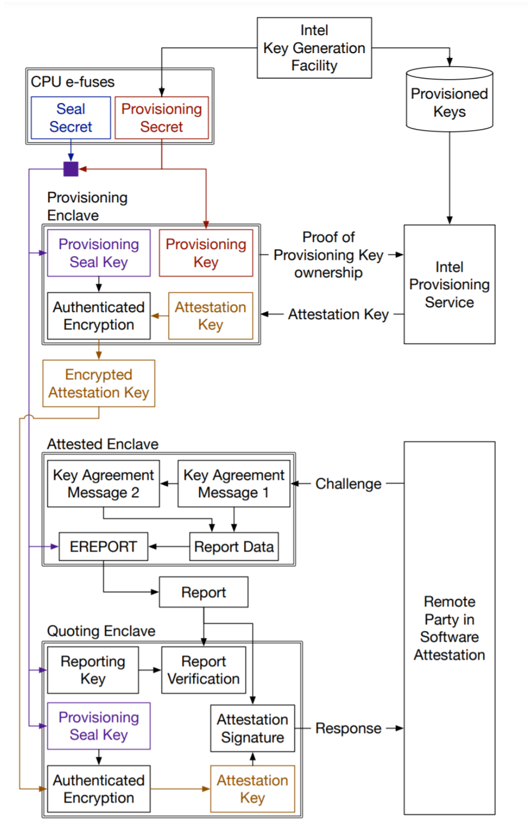


Figure 2.2: SGX key derivation procedure with attestation workflow, image source [61]

2.3.3 Sealing Technologies

Sealing [61] is a process of loading TEE internal secret state to persistent storage. Roughly speaking, using the sealing technologies, secrets are allowed to be encrypted and stored in the untrusted memory or disk. Further, such encrypted secrets can be retrieved once the enclave is torn down (either due to the host's power or the application itself). Sealing in SGX is achieved by using a private seal key [61], which covers two kinds of identities: *Enclave Identity* and *Signing Identity*. Enclave Identity is represented by the value of *MRENCLAVE*, which is a cryptographic hash of the enclave measurement. Any operation inside an enclave that changes measurement will yield a different key. Thus, it restricts the permission to sealed data: only the corresponding enclave can access sealed data. In contrast, Signing Identity, represented by *MRSIGNER*, is provided by an authority. It provides the same sealing key for different enclaves or different versions of the same enclave. Therefore, Signing Identity can be used to share sensitive data between multiple enclaves produced by the same development firm.

Chapter 3

TTP-I: Contract-based TTP

In this chapter, we introduce a smart contract-based TTP, called TTP-I. In this type of TTP, a smart contract is employed as a transparent and neutral agent. We first establish a universal framework for TTP-I protocols to capture the security properties (Section 3.1). Then, we provide two instances to show how TTP-I benefits existing cryptographic protocols. Specifically, our first case uses TTP-I to aid certificate authority (CA) for fairly revoking the certificate in the CBE [83, 90, 142] scheme. Our second case utilizes TTP-I to replace the role of key curator (KC) in the RBE [86, 87] scheme for transparent registration. Finally, we discuss the benefits and drawbacks of TTP-I based security protocols, with emphasis on their privacy and scalability issues (Section 3.3).

3.1 General Construction

Smart contracts run on decentralized blockchain nodes, which fits the role of TTP well. A TTP can be defined as an event-driven customized contract. A TTP's operations and logic can be coded in the form of executable functions. These functions are then compiled and distributed to the blockchain network to receive a global consensus in the deployment stage. This guarantees that all the blockchain nodes share the same TTP's logic and an initial configuration. When other parties in security protocols want to obtain auxiliary data for encryption and decryption, they need to send a transaction or a message call to the blockchain. Once a blockchain node receives a transaction to trigger the execution, all nodes will run the same function as agreed in the deployment stage to obtain the latest state. This automatically distributed execution ensures that all blockchain nodes, including other protocol parties, can equally observe the contract-based TTP's operations.

A TTP-I protocol consists of two main types of participants: *protocol users* and a *smart contract*. The protocol users include both the message sender and the message receiver, while the smart contract is used as a "bulletin board" to aid or replace a TTP in maintaining auxiliary information for the sender's encryption and the receiver's decryption. A generic construction is shown as follows.

System Setup $pms \leftarrow \mathsf{Setup}(1^\lambda)$. The algorithm takes as input a security parameter λ, and outputs system parameters pms.

Key Generation $(sk, pk), (sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. The algorithm takes as input λ, and outputs a key pair (sk, pk), and a key pair $(sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}})$ for transaction signing and transaction verification.

Then, a smart contract is deployed, outputting a contract identity $\widehat{c}$, an initial state $\overline{s}$, the operational code $\langle opcode \rangle$, and the execution condition $\langle reqcode \rangle$. The logic of a TPP is coded into $\langle opcode \rangle$, and the execution condition of the logic is coded into $\langle reqcode \rangle$. This step is finished by calling $\widehat{\mathcal{SC}}$.**Deploy** described in Definition 6. Next, a message sender encrypts a message using the receiver's identity/key and auxiliary data with the assistance of the deployed smart contract. This assistance is represented as storing or changing auxiliary data in the contract by ways of sending transactions.

Transaction Generation[1] $\mathsf{Tx} \leftarrow \mathsf{Sign}(sk_{\mathsf{Tx}}, metadata, auxdata)$. The algorithm takes as input a private signing key sk_{Tx}, a transaction $metadata$, and auxiliary data $auxdata$ used for encryption and decryption, and outputs a transaction Tx.

TTP-I Operation $s \overset{\mathbb{B}}{\leftarrow} \mathsf{TtpOperate}(\widehat{c}, \overline{s}, \mathsf{Tx})$. The algorithm takes as input $\widehat{c}$, current state $\overline{s}$ and a transaction Tx with auxiliary data $auxdata$, and outputs the transferred state s. This algorithm is finished by calling the algorithm $\widehat{\mathcal{SC}}$.**Transfer** described in Definition 6.

[1]We use blue texts to emphasize blockchain-related operations.

Encryption $ct \leftarrow \mathsf{Enc}(pk, auxdata, m)$. The algorithm takes as input a user's public key pk, auxiliary data $auxdata$ and a message m, and outputs a ciphertext ct. This algorithm is completed in the users' local clients.

State Read $s' \overset{\mathbb{B}}{\leftarrow} \mathsf{Read}(\widehat{c})$. The algorithm takes as input a contract identity $\widehat{c}$, and outputs a newly transferred state s'. Here, the s' is necessary for the following decryption algorithm. This algorithm is finished by calling the algorithm $\widehat{\mathcal{SC}}$.**Read** described in Definition 6.

Decryption $m/\perp \leftarrow \mathsf{Dec}(sk, s', ct)$. The algorithm takes as input a user's private key sk, a new contract state s', a ciphertext ct, and outputs a message m or the special symbol $\perp$ indicating decryption failure. This algorithm is completed in the users' local client.

Inspection $true/false \overset{\mathbb{B}}{\leftarrow} \mathsf{Inspect}(\mathsf{Tx}, s')$. This algorithm takes as input a transaction Tx, s' and returns the legality of the **State Transfer** operation. In particular, the transactions that trigger the execution of a contract in the **State Transfer** operation can be used as evidence to indicate the users' or TTP's misbehaviours. By tracing the transaction sender, the inspector will know when the auxiliary information is changed and who changes the auxiliary information for encryption or decryption. If the transaction sender's invocation does not match the established rules, he will be caught and blamed, significantly reducing the probability of these parties committing malicious behaviours.

3.2 TTP-I Protocol Instances

To demonstrate TTP-I's feasibility and practicality, this section presents two protocols using TTP-I, namely, a transparent certificate revocation protocol for CBE scheme, and a transparent registration protocol for RBE scheme.

3.2.1 Transparent Certificate Revocation for CBE

In a CBE model, an up-to-date certificate must be obtained from a certificate authority (CA) since it is used as a partial decryption key. A full definition can be found in Section 2.1.1. However, such a mechanism that heavily relies on

CA, presents several concerns: (1) CA may arbitrarily revoke a valid certificate and repudiate its actions, indirectly causing decryption to fail (see Table 3.1). For example, Alice sends a ciphertext to Bob, but a malicious CA has already revoked the certificate without Bob's permission. There is no way for Bob to obtain an up-to-date certificate, and as a result, he cannot decrypt the ciphertext; (2) Users cannot blame the CA due to the absence of valid evidence of its malicious behaviours; (3) There is a lack of incentive for the CA to behave honestly.

Table 3.1: Definition of CA's valid revocation and illegal revocation

Description	Legal revocation	Illegal revocation
A user sends a revocation request, and a CA revokes his certificate.	✓	
A user does not send a revocation request, and a CA revokes his certificate.		✗
A user's certificate has expired, and a CA revokes such a certificate.	✓	
A user's certificate is valid, and a CA revokes such a certificate.		✗

Our scheme utilizes a customized smart contract as a transparent agent to manage revocations, which takes over part of CA's functions, making the revocation operation accountable. In particular, a public list $\mathcal{L}$ (including an invalid set $\mathcal{IS}$ and a valid set $\mathcal{VS}$) in the smart contract stores the information on the authorized address, certificate expiry date, certificate state, etc (see Table 6.2). Any users who want to revoke the certificate need to send a revocation request to update the list $\mathcal{L}$. After receiving the request, the smart contract verifies the users' eligibility. Only the authorized address owner has permission to update the list $\mathcal{L}$; if a transaction sender's address matches an entry in the authorized address list, the contract logic of updating the list $\mathcal{L}$ will be executed. Otherwise, the execution is aborted. Meanwhile, the contract is also required to automatically check the expiry state and then to update $\mathcal{L}$. Next, CA periodically fetches all the items in list $\mathcal{L}$ and stops issuing the certificate for the revoked or expired ones.

Our solution focuses on the certificate revocation procedure. Here, we provide a generic CBE construction and then emphasize the enhancement of this certificate

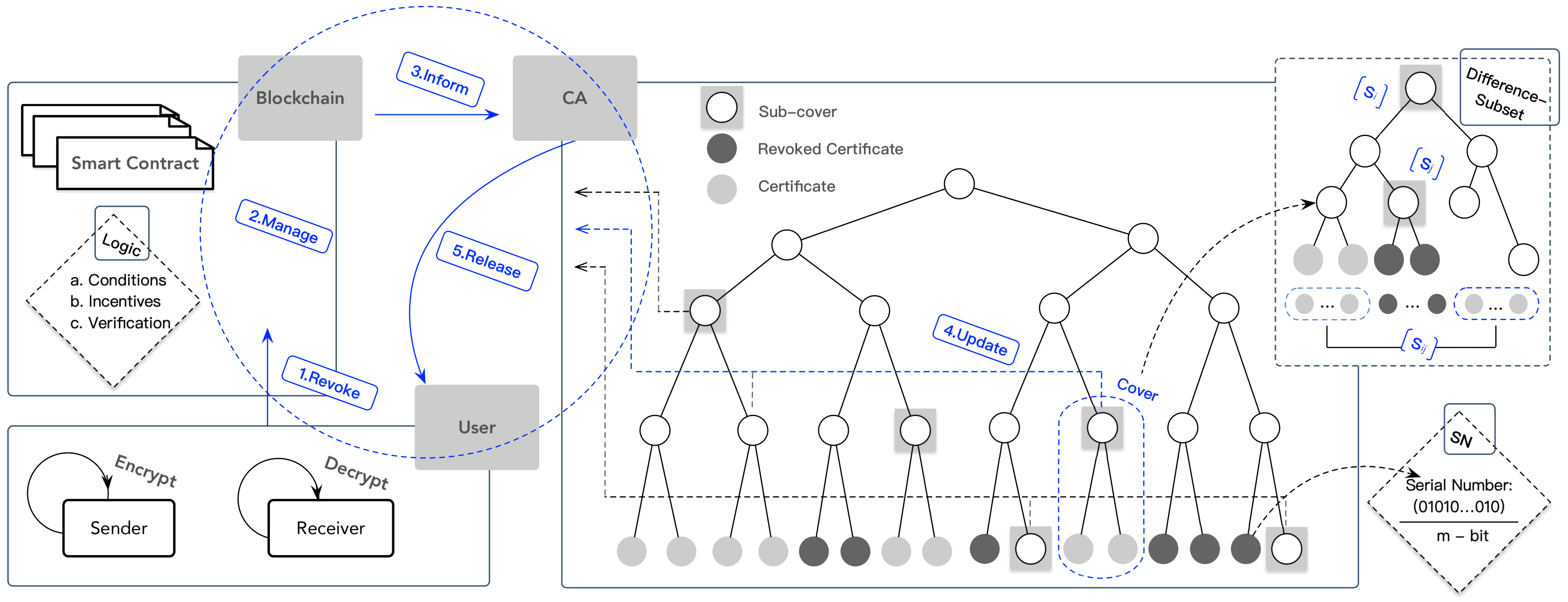


Figure 3.1: Certificate revocation diagram for CBE using TTP-I

revocation algorithm. Our instantiation follows the design of Gentry's scheme [90], and detailed steps are described below.

System Setup $pms \leftarrow \mathsf{Setup}(1^\lambda, n)$. The algorithm takes as input, a security parameter λ, a total number of time periods n (optionally), and outputs system parameters pms. $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of some large prime order q.

$$\begin{aligned}
\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2 \quad & \text{//}\hat{e} \text{ is a bilinear map pairing} \\
P \in \mathbb{G}_1 \quad & \text{// select a generator from a group } \mathbb{G}_1 \\
H_1 : \{0,1\}^\star \rightarrow \mathbb{G}_1 \quad & \text{// map an arbitrary string to an element in } \mathbb{G}_1 \\
H_2 : \mathbb{G}_1 \rightarrow \{0,1\}^n \quad & \text{// map an arbitrary string to an element with a fixed length} \\
s_C \in \mathbb{Z}/q\mathbb{Z} \quad & \text{// master secret key} \\
Q = s_C P \quad & \text{// master public key} \\
pms = (\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, H_1, H_2) \quad & \text{//public parameters}
\end{aligned}$$

Key Generation $(s_B, p_B), (sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. The algorithm takes as input a security parameter λ, and outputs a user's (e.g., Bob's) key pair (s_B, p_B) for encryption and decryption, and a signature key pair $(sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}})$ for signing and verifying transactions.

$$\begin{aligned}
s_B \in \mathbb{Z}/q\mathbb{Z} \quad & \text{// Bob's private key} \\
p_B = s_B P \quad & \text{// Bob's public key}
\end{aligned}$$

Then, a smart contract is deployed, outputting a contract identity $\widehat{c}$, an initial state s, the operational code $\langle opcode \rangle$, and the execution condition $\langle reqcode \rangle$. $\langle opcode \rangle$ defines some functionalities and interfaces to update the list $\mathcal{L}$, while $\langle reqcode \rangle$ stores the revocation conditions that need to satisfy for updating list $\mathcal{L}$. These conditions contain the certificate expiry date and the eligibility of users' revocations. This step is finished by calling $\widehat{\mathcal{SC}}$.**Deploy** (see Definition 6).

Transaction Generation $\mathsf{Tx} \leftarrow \mathsf{Sign}(sk_{\mathsf{Tx}}, metadata, auxdata)$. The algorithm signs a transaction *metadata* and auxiliary data *auxdata* with a signing key sk_{Tx} to obtain a transaction Tx. Here, *auxdata* represents a request mapping to the certificate state. For example, $auxdata = [Bob : revoked]$ means that Bob's certificate needs to be revoked. This algorithm is run by a user (e.g., Bob) who wants to revoke his certificate in case his private key is lost.

Table 3.2: A public list $\mathcal{L}$ of users' certificate state

	Authorized Address	**User ID**	**State**	**Expiry Date**	**Current Date**	**Certificate**
$\mathcal{IS}$	0x0a...670	Bob	revoked	Dec 25, 2023	Dec 01, 2021	0000...
	0xe7...d30	Tom	revoked	Jan 03, 2025	Dec 01, 2021	0000...
	0x60...b6e	Kate	expired	Jun 21, 2020	Dec 01, 2021	0000...
	0x67...ea2	Dave	expired	Nov 30, 2020	Dec 01, 2021	0000...
	0x45...912	Alice	expired	Dec 07, 2020	Dec 01, 2021	0000...
$\mathcal{VS}$	0xbb...709	Rujia	valid	May 09, 2023	Dec 01, 2021	achgm...
	0xc9...a98	Tomas	valid	Aug 17, 2022	Dec 01, 2021	mdemx...

TTP-I Operation It consists of three algorithms.

1. *Revocation Launch* $s' \overset{\mathbb{B}}{\leftarrow} \mathsf{TtpOperate}(\widehat{c}, s, \mathsf{Tx})$ The algorithm takes as input a contract identity $\widehat{c}$, current state s, a transaction Tx, and outputs the transferred state s'. Here, s' refers to the newly updated users' certificate state (from "valid" to "revoked"). It consists of two sub-algorithms, which are finished by calling $\widehat{\mathcal{SC}}$.**Transfer** (see Definition 6).

    - *Revocation Qualification Check* Once the contract $\widehat{c}$ receives the certificate revocation request, it checks the revocation qualification. Only the authorized user has the ability to update the revocation state in $\mathcal{L}$; successful execution of this algorithm indicates that the transaction's address is authorized.

    - *Revocation List Update* The contract $\widehat{c}$ updates the public list $\mathcal{L}$. The revoked certificates and the expired certificates will be added to the invalid set $\mathcal{IS}$. Next, CA fetches all the data in the $\mathcal{L}$.

2. *Certificate Issue* $Cert_i \leftarrow \mathsf{Cert}(s_C, i, user, pk)$. CA takes as input a certifier's master secret key s_C, user's information *user*, a public key *pk* obtained from

valid set $\mathcal{VS}$, and outputs a certificate $Cert_i$. Here, we emphasize that CA must follow the items in set $\mathcal{VS}$. The detailed algorithm is shown as follows.

$$\begin{aligned} H_5 : \{0,1\}^\star &\rightarrow \mathbb{G}_1 \\ T_i &= H_5(Q, i) \\ P_k &= H_1(b_1 \dots b_k) \\ Cert_i &= s_C T_i + x P_k \end{aligned}$$

To be specific, CA updates certificates' status through a binary tree[2]. CA arranges for most 2^m clients as leaves in a m-level binary tree. A unique m-bit serial number (SN, $b_1 \dots b_m$) is embedded in each client's long-lived certificate. SN provides both identities and positions in the tree. $b_1 \dots b_k$ are ancestors of $b_1 \dots b_m$, where $b_1 \dots b_m$ are leaves in the m-level binary tree. The revocation is represented by the deletion of a leaf's sub-cover nodes [159]. In our example, as shown in Figure 3.2, there are 2^3 clients and Alice's SN is 001, namely, $(b_0 = 0, b_1 = 0, b_2 = 1)$. If the certificate revokes, Alice's SN becomes $00*$, $(b_0 = 0, b_1 = 0, b_2 = *)$, and the corresponding certificate becomes invalid. This algorithm runs in CA's local machine without the involvement of blockchain.

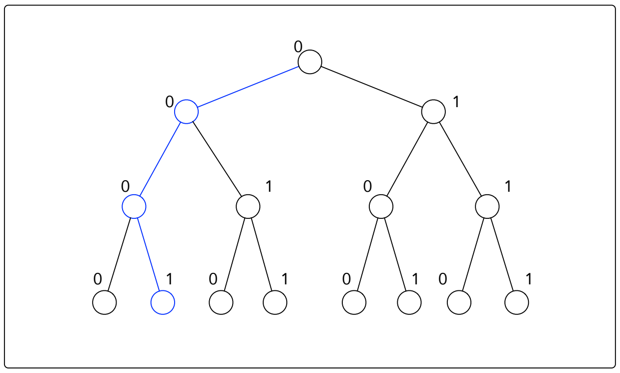


Figure 3.2: m-bit serial number is labelled as m-level binary tree

3. *Certificate Upload* CA uploads newly issued $Cert_i$ to the public list $\mathcal{L}$. If

[2]This binary tree is maintained by CA off-chain, while the corresponding certificate state is stored on-chain.

the certificate has been revoked, the CA uploads $****...$ for this certificate. This algorithm is achieved by calling $\widehat{\mathcal{SC}}$.**Transfer** (see Definition 6).

Encryption $ct \leftarrow \mathsf{Enc}(Q, BobInfo, m)$. At the time period i, the algorithm takes as input Bob's information $BobInfo$, CA's public key Q, a message m, and outputs a ciphertext ct.

$$\begin{aligned} r &\in \mathbb{Z}/q\mathbb{Z} \\ P'_B &= H_1(BobInfo) \in G_1 \\ T_i &= H_5(Q, i) \\ g &= \hat{e}(Q, T_i)\hat{e}(s_B P, P'_B) \\ V &= m \otimes H_2(g^r) // \quad \text{XOR} \\ ct = [rP, rP_1, ..., rP_m, V], \quad \text{where} P &\in \mathbb{G}_1, P_1 \in \mathbb{G}_1, ...P_m \in \mathbb{G}_1 \end{aligned}$$

Note that, at this stage, the message sender has already verified Bob's long-lived certificate and therefore knows $BobInfo$ and his serial number.

State Read $Cert_i \overset{\mathbb{B}}{\leftarrow} \mathsf{Read}(\widehat{c})$. The algorithm takes as input a contract identity $\widehat{c}$, and outputs a reconfirmation certificate $Cert_i$ (if it exists). This algorithm is finished by calling the algorithm $\widehat{\mathcal{SC}}$.**Read** described in Definition 6.

Decryption $m/\perp \leftarrow \mathsf{Dec}(s_B, Cert_i, ct)$. At the time period i, the algorithm takes as input a secret key s_B, a reconfirmation certificate $Cert_i$ (if it exists), and outputs a message m or a special symbol $\perp$ indicating decryption failure.

$$m = V \otimes H_2(\frac{\hat{e}(rP, Cert_i + s_B P'_B)}{\hat{e}(xP, rP_k)})$$

Inspection $true/false \overset{\mathbb{B}}{\leftarrow} \mathsf{Inspect}(\mathsf{Tx}, \mathcal{L})$. This algorithm takes as input Tx, the list $\mathcal{L}$ and returns the legality of the **Transfer** operation. $true$ indicates that the certificate is revoked under Bob's intention. Here, we emphasize the consistency of the CA's operation. CA and the contract $\widehat{c}$ share the same list $\mathcal{L}$, and CA must follow the items in $\mathcal{IS}$ to revoke the certificate. Otherwise, CA's illegal revocation will be identified instantly, and the cryptocurrency-based rewards will be confiscated.

Here, the **Correctness** of our construction is easy to check as we have

$$
\begin{aligned}
m &= V \otimes H_2(\frac{\hat{e}(rP, Cert_i + s_B P'_B)}{\hat{e}(xP, rP_k)}) \\
&= V \otimes H_2(\frac{\hat{e}(rP, s_C T_i + xP_k + s_B P'_B)}{\hat{e}(xP, rP_k)}) \\
&= m \otimes H_2(g^r) \otimes H_2(\frac{\hat{e}(rP, s_C T_i + xP_k + s_B P'_B)}{\hat{e}(xP, rP_k)}) \\
&= m \otimes H_2(g^r) \otimes H_2((\frac{\hat{e}(P, s_C T_i + xP_k + s_B P'_B)}{\hat{e}(xP, P_k)})^r) \\
&= m \otimes H_2(g^r) \otimes H_2((\frac{\hat{e}(P, s_C T_i)\hat{e}(P, xP_k)\hat{e}(P, s_B P'_B)}{\hat{e}(xP, P_k)})^r) \\
&= m \otimes H_2((\hat{e}(s_C P, T_i)\hat{e}(s_B P, P'_B))^r) \otimes H_2((\frac{\hat{e}(P, s_C T_i)\hat{e}(P, xP_k)\hat{e}(P, s_B P'_B)}{\hat{e}(xP, P_k)})^r) \\
&= m \otimes H_2((\hat{e}(s_C P, T_i)\hat{e}(s_B P, P'_B))^r) \otimes H_2((\hat{e}(P, s_C T_i)\hat{e}(P, s_B P'_B))^r) \\
&= m \otimes H_2((\hat{e}(P, T_i)\hat{e}(P, P'_B))^{r s_C s_B}) \otimes H_2((\hat{e}(P, T_i)\hat{e}(P, P'_B))^{r s_C s_B}) \\
&= m
\end{aligned}
$$

3.2.2 Transparent Registration-based Encryption

Identity-based encryption (IBE) was first proposed by Shamir [186]. IBE has gained massive popularity in cryptography research since the early 2000s [26, 29, 43, 184], and it has thus become a central primitive in public-key cryptography. Aimed at simplifying public key certificate management in Public Key Infrastructure solutions, IBE adopts a receiver's actual identifier as the public key for message encryption instead of a standard public key, which is a human-unmemorable bit string. However, a high price is paid in exchange for the convenience of using human-readable and memorable identifiers as public keys, namely the so-called *key escrow* problem, is introduced. Indeed, a trusted party called Private Key

Generator (PKG) is now in charge of computing the decryption keys associated with each identifier. On the one hand, PKG needs to be trusted to not arbitrarily generate or utilize a user's secret key without their consent. On the other hand, centralized key generation increases the risk of a single point of failure.

Table 3.3: Comparison of various public-key encryption schemes

Scheme	**Setup**	**Key Gen**	**Encryption**	**Decryption**
IBE [26, 186]	$\mathsf{Setup}(1^\lambda)$ $\downarrow$ pp, msk	$\mathsf{Gen}_{\mathsf{pkg}}(pp, msk, id)$ $\downarrow$ sk_u	$\mathsf{Enc}_{\mathsf{u}}(pp, id, m)$ $\downarrow$ ct	$\mathsf{Dec}_{\mathsf{u}}(sk_u, pp, ct)$ $\downarrow$ m
CLE [65]	$\mathsf{Setup}(1^\lambda)$ $\downarrow$ $pp(mpk), msk$	$u \leftarrow \mathsf{Gen}_{\mathsf{pkg}}(msk, id)$ $sk_u \leftarrow \mathsf{Gen}_{\mathsf{u}}(pp, id)$	$\mathsf{Enc}_{\mathsf{u}}(pp, id, pk, m)$ $\downarrow$ ct	$\mathsf{Dec}_{\mathsf{u}}((sk_u, u), pp, ct)$ $\downarrow$ m
CBE [83, 90]	$\mathsf{Setup}(1^\lambda)$ $\downarrow$ $pp(mpk), msk$	$sk_u, pk_u \leftarrow \mathsf{Gen}_{\mathsf{u}}(pp)$ $u \leftarrow \mathsf{Gen}_{\mathsf{pkg}}(msk, id, pk_u)$	$\mathsf{Enc}_{\mathsf{u}}(pp, id, pk, m)$ $\downarrow$ ct	$\mathsf{Dec}_{\mathsf{u}}(sk_u, u, ct)$ $\downarrow$ m
RBE [86, 87]	$\mathsf{Setup}(1^\lambda)$ $\downarrow$ $pp(crs)$	$sk_u \leftarrow \mathsf{Gen}_{\mathsf{u}}(1^\lambda)$ $u \leftarrow \mathsf{Upd}_{\mathsf{pkg}}(pp, id)$	$\mathsf{Enc}_{\mathsf{u}}(pp, id, pk, m)$ $\downarrow$ ct	$\mathsf{Dec}_{\mathsf{u}}(sk_u, u, ct)$ $\downarrow$ m

• msk: PKG's master private key; pk_u: user's public key; sk_u: user's private key.
• pp: public parameters; u: auxiliary data for decryption.

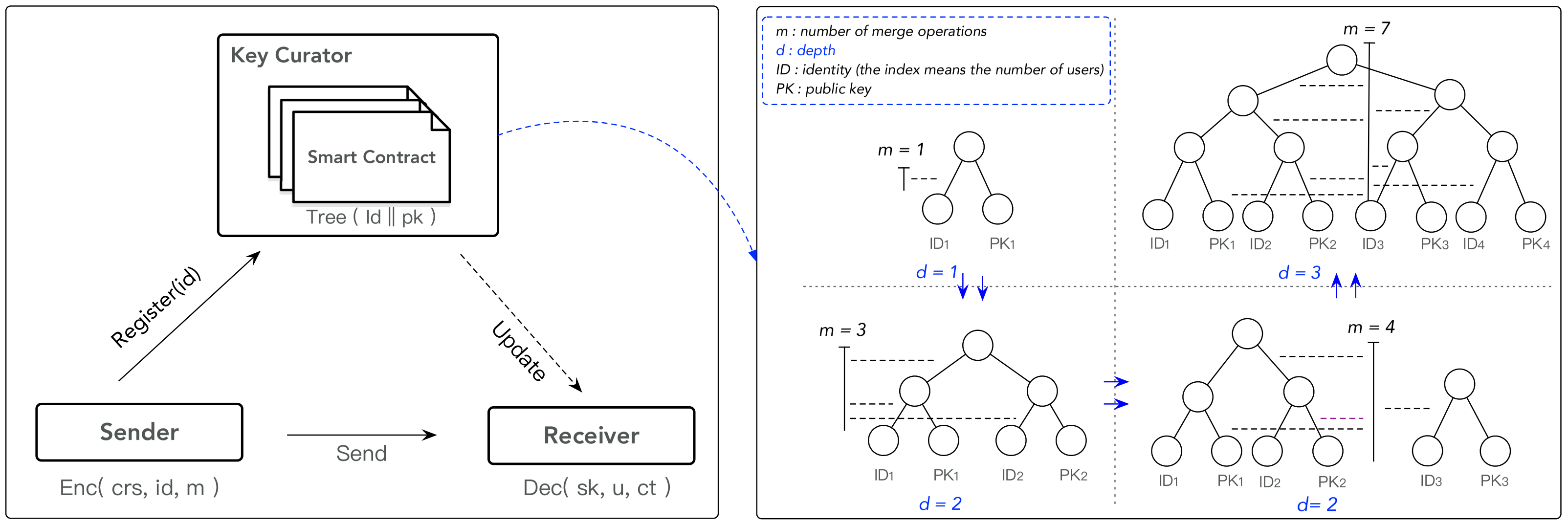
Registration-based encryption (RBE) [86], proposed by Garg recently, decouples the key production process from the PKG altogether by replacing the PKG with a public key accumulator called Key Curator (KC). Every user in an RBE system generates its own public-secret key pair and sends the public key to the KC for registration. The KC is merely responsible for compressing all the registered user identity-key pairs into a short reference string. When a sender wants to send encrypted data to a receiver, he only requires the compressed public parameters along with the target identifier, whereas the receiver requires the public parameter along with the secret key associated with the registered public key. Importantly, the public parameter is not frequently changed due to KC's multiple trees structure, which makes the decryption smooth. As shown in Table 3.3, we make a comparison between RBE scheme and other schemes, to emphasize RBE's advantages.

However, RBE still places a significant amount of trust in key curator (KC), whose actions are isolated from the outside world and are not accountable. A dishonest KC may hide a trapdoor that enables the secret creation of a key pair for a yet unregistered identity or even register multiple keys for already-registered users.

Even if KC is completely honest without these aforementioned misbehaviours, RBE systems are still far from practical since their identity authentication in the registration stage is based on certification authorities in public-key infrastructure. Such a procedure is dense and complex, and these drawbacks severely hamper the development of RBE-based applications.

To provide a solution to problems of strong centralization, untrustworthy KC, and identity authentication issues found in original RBE systems [86, 87], we propose to reboot the RBE approach by coupling it with blockchain technology. Blockchain has already been proposed as a vehicle to realize decentralized key management in PKI [149, 174], but it has not yet been applied, in the RBE setting, to the best of our knowledge. Our transparent RBE construction leverages smart contracts to automate the logic of the KC in a transparent manner, enabling publicly upgradeable proofs on-chain that render the KC's actions accountable. The solution transfers the right of key management from KC to users. Smart contracts guarantee strong availability, the persistence of its state, and the correct execution of predefined protocols, while the RBE scheme provides a secure, compact, and user-friendly cryptosystem. For these reasons, building hybrid protocols that take advantage of both sides seems an appealing solution.

Our scheme consists of two main entities: *user* and *smart contract*. A smart contract is employed as a public bulletin board that maintains the relationship between the identity and the corresponding public key, and based on that, it returns some public parameters as the encryption key. The *user* is composed of two roles: a message sender and a message receiver. The workflow is as follows. The sender encrypts the message using the receiver's identity and public parameters and then sends the ciphertext to the receiver. The receiver then decrypts the ciphertext using the private key and public parameters updated from the smart contract. In particular, a smart contract is used as a KC to compress all users' identity-key pairs into a short public parameter with auxiliary information. The identity-key pair is organized as a Merkle tree, and every leaf of the tree is an identity or its public key. At a high level, after n users have registered, the smart contract holds auxiliary information: $\eta \quad (\eta \leq \mathsf{log}(n))$ full binary Merkle trees $Tree_1, \ldots, Tree_\eta$ with number of leaves $2^{id_1} \ldots 2^{id_\eta}$ and corresponding depths $id_1 >$



a. TypeX Protocol

b. Key Accumulation in Smart Contract

Figure 3.3: Overview of transparent RBE scheme

$id_2 > id_3 \ldots > id_\eta$. Meanwhile, the public parameter of each user holds the form $pp = ((hk_1, \ldots, hk_\lambda), (rt_1, d_1), \ldots, (rt_\eta, d_\eta))$ where $rt_i \in \{0,1\}^\lambda$ represents the root of $Tree_i$, λ represents the length of identities, and each hk_i is sampled from $\mathsf{HGen}(1^\lambda, 0)$, where $\mathsf{HGen}(1^\lambda, 0)$ is a hash key generation algorithm of somewhere statistically binding hash functions [86]. We show the detailed protocols as follows.

System Setup $crs \leftarrow \mathsf{Setup}(1^\lambda)$. The algorithm takes as input a security parameter λ, and outputs a common random string crs. Here, crs can be sampled publicly using some public randomness beacons.

Key Generation $(sk, pk), (sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. This algorithm takes as input a security parameter λ and outputs a user's key pair (sk, pk), and a key pair $(sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}})$ for signing and verifying a transaction. As discussed in Section 2.1.1, sk and pk are only secret/public keys, not encryption or decryption keys.

Then, a smart contract is deployed, outputting a contract identity $\widehat{c}$, an initial state s, the operational code $\langle opcode \rangle$, and the execution condition $\langle reqcode \rangle$. $\langle opcode \rangle$ defines registration logic, a public parameter list $\mathcal{L}$. $\langle reqcode \rangle$ stores the registration conditions (preventing cases of registering multiple keys for a registered user or registering multiple users for a certain key). Also, KC setups an empty auxiliary information $aux = \varnothing$ and an public parameter $pp_0 = \perp$. This step is finished by calling $\widehat{\mathcal{SC}}$.**Deploy** (see Definition 6).

Transaction Generation $\mathsf{Tx} \leftarrow \mathsf{Sign}(sk_{\mathsf{Tx}}, metadata, auxdata)$. A user sends his identity id and public key pk to KC. Then, KC signs a transaction $metadata$ and auxiliary information $auxdata$ using private signing key sk_{Tx} to obtain a transaction Tx. This procedure is represented as a user's registration. The auxiliary information $auxdata$ refers to the registration information, including a user's identity id and public key pk, which is stored locally in KC.

TTP-I Operation (Registration) $pp_{n+1} \overset{\mathbb{B}}{\leftarrow} \mathsf{TtpOperate}^{[aux]}(\widehat{c}, s, \mathsf{Tx})$. This algorithm takes as input public parameter pp_n (extracted from the current contract state s), a registering identity id, a public key pk (supposedly for the identity id) and outputs a new public parameter pp_{n+1}. This algorithm contains three sub-algorithms, which are finished by calling $\widehat{\mathcal{SC}}$.**Transfer** (see Definition 6).

1. *Identity Verification.* Once the identity registration request arrives, the con-

tract $\widehat{c}$ checks the identity based on the predefined logic. The registration function of the contract $\widehat{c}$ stops if the identity has been registered or the key has been registered.

2. *Tree Generation.* The smart contract $\widehat{c}$ parses the parameter and then creates a new tree.

   * Parse $aux := ((Tree_1, \ldots, Tree_\eta), (id_1, \ldots, id_\eta))$ where these trees have corresponding depths $d_1 > d_2 \ldots > d_\eta$, and $(id_1, \ldots, id_\eta)$ is the order by which the current identities have registered. In initialization phase, $aux := \varnothing$.

   * Parse pp_n as a sequence $((hk_1, \ldots, hk_\lambda), (rt_1, d_1), \ldots, (rt_\eta, d_\eta))$, and the $rt_i \in \{0,1\}^\lambda$ represents the root of $Tree_i$, and d_i is the depth of $Tree_i$. In initialization phase, $pp_0 := \perp$.

   * Create a new tree $Tree_{\eta+1}$ with leaves id and pk. Then, set its root as $rt_{\eta+1} := \mathsf{Hash}(hk_1, id||pk)$ and thus its depth would be $d_{\eta+1} = 1$.

3. *Tree Compression.* The smart contract $\widehat{c}$ starts to merge multiple Merkle hash trees through the sub-algorithm $\widehat{\mathcal{SC}}$.**Transfer**. Let $\mathcal{T} = \{Tree_1, ..., Tree_{\eta+1}\}$. While there are two different trees $Tree_L, Tree_R \in \mathcal{T}$ of the same depth d, same size $s = 2^d$ (as our trees are full binary trees), the algorithm keeps doing the following steps.

   * Let $Tree$ be a new tree of depth $d+1$ that contains $Tree_L$ as its left subtree, $Tree_R$ as the right subtree, and $rt = \mathsf{Hash}(hk_{d+1}, rt_L||rt_R)$ as the root.

   * Remove both of $Tree_L$,$Tree_R$ from $\mathcal{T}$, and add $Tree$ to $\mathcal{T}$ instead.

   * Let $\mathcal{T} := (Tree_1, \ldots, Tree_\zeta)$ be the final set of trees with depths $d'_1 > \ldots > d'_\zeta$ and roots $rt'_1, \ldots, rt'_\zeta$. Set the pp_{n+1} and aux as follows:

     - $pp_{n+1} := ((hk_1, \ldots, hk_\lambda), (rt'_1, d'_1), \ldots, (rt'_\zeta, d'_\zeta))$,

     - $aux := (\mathcal{T}, (id_1, \ldots, id_n, id_{n+1} = id))$.

Here, we emphasize the identity-key pair compression algorithm. Intuitively, when a new user joins the system, the tree root is updated, and the public parameters of registered users also need to be updated. To minimize the effect of registration by new users on previously registered users, our solution, following the idea of the original RBE [86], adopts multiple Merkle hash trees such that any individual user is affected only a limited number of times. In particular, the trees with the same depth are continuously merged into a new one in the on-chain calculation (see Figure 5.1); the tree that holds the identity only needs to be updated at most $\mathcal{O}(\mathsf{log}n)$ times, where n represents the total number of registered users. A registered user does not have to query the smart contract for public parameters each time of decryption. Alternatively, the user needs to query the smart contract only when the current public parameters are changed.

Encryption $ct \leftarrow \mathsf{Enc}(pp, id, m)$. The algorithm takes as input the public parameter pp obtained from the contract $\widehat{c}$, an identity id, a message m, and outputs a ciphertext ct. In particular, it parses $pp := ((hk_1, \ldots, hk_\lambda), (rt_1, d_1), \ldots, (rt_\eta, d_\eta))$. Then, it generates programs $P_1, \ldots, P_\eta$ where P_i works as follows,

Hardwired values: $rt_i, d_i, (hk_1, \ldots, hk_{d_i}), m, id, r$ (randomness); Note that hk_i corresponds to the level i in a Tree.
Input: pth

- Parse $\mathsf{pth} := [(h_0^0, h_0^1), (h_1^0, h_1^1, b_1) \ldots, (h_{d_i-1}^0, h_{d_i-1}^1, b_{d_i-1}), rt]$.
- If $rt_i \neq rt$, then output $\perp$.
- If $id \neq h_0^0$, then output $\perp$.
- If $rt = \mathsf{Hash}(hk_{d_i}, h_{d_i-1}^0 || h_{d_i-1}^1)$ and $h_j^{b_j} = \mathsf{Hash}(hk_j, h_{j-1}^0 || h_{j-1}^1)$ for all $j \in [d_i - 1]$, then output $\mathsf{PKE.Enc}(h_0^1, m; r)$ by using h_0^1 as the public key and r as the randomness, otherwise output $\perp$.

Next, the algorithm outputs $ct := (pp, \mathsf{Obf}(P_1), \ldots \mathsf{Obf}(P_\eta))$ where Obf specifies the IO obfuscation (see Definition 5). The encryption is performed by obfuscation of the program P using the public key and some auxiliary information. In particular, the program P outputs the encryption of m only if the path (called *Merkle opening*) is from leaves (id, pk) to the root rt in the Merkle tree. When there are multiple trees $Tree_1, \ldots Tree_\eta$ held by the smart contract, the ciphertext includes η obfuscations, one for each $Tree_i$ $(i \leq \eta)$.

Update $u \overset{\mathbb{B}}{\leftarrow} \mathsf{Update}^{[\mathsf{aux}]}(\widehat{c})$. The algorithm takes as input a contract identity $\widehat{c}$, and outputs an updated Merkle opening u. Letting $aux := (Tree_1, ..., Tree_\zeta)$ and i be an index of the tree which holds id, and thus u is the whole Merkle opening of the path that leads to id in $Tree_i$.

Decryption $m/\perp \leftarrow \mathsf{Dec}(sk, u, ct)$. The algorithm takes as input the secret key sk, an updated Merkle opening u, a ciphertext ct, and outputs the message $m \in \{0,1\}^*$ or in $\{\perp, \mathsf{GetUpd}\}$. The special symbol $\perp$ indicates a syntax error, while GetUpd indicates that the latest received information needs to be updated by re-executing the algorithms of Update. In particular, it parses $ct = (u, \overline{P}_1, \ldots, \overline{P}_\eta) = (u, \mathsf{Obf}(P_1), \ldots \mathsf{Obf}(P_\eta))$, and then executes $m_i = \mathsf{PKE.Dec}(sk, \overline{P}_i(u))$ for every program $\overline{P}_i$, and finally outputs the message satisfying $m_i \neq \{\perp, \mathsf{GetUpd}\}$.

Inspection $true/false \overset{\mathbb{B}}{\leftarrow} \mathsf{Inspect}(\mathsf{Tx}, \mathcal{L})$. This algorithm takes as input the transaction Tx, and returns the legality of the **Registration** operation. $true$ indicates the registration is under the user's intention. In particular, if a user finds that his identity id or the key pk is muti-registered in the contract, he knows the malicious registration.

3.3 TTP-I Protocol Discussion

In this section, we first show how TTP-I benefits CBE and RBE. Then, we discuss the challenges of using a blockchain-based contract as a TTP.

3.3.1 TTP-I Roles and Benefits

The main difference between using a traditional third party and TTP-I is that *TTP-I shifts the requirement for trustworthiness from any particular party to the majority of ledger maintainers.* The conflict of interest between different ledger maintainers ensures that it is difficult for any malicious entity to corrupt TTP-I, and further guarantees that TTP-I's operations can be carried out as intended. In a nutshell, as a customized smart contract driven by blockchain, TTP-I helps security protocols in two ways: *assistance with the existing TTP* and *replacement of the existing TTP.*

Table 3.4: Comparison of existing CBE schemes and our CBE scheme

Schemes	**Existing CBE Schemes**	**Our CBE Scheme**
Roles	A TTP is used as CA.	• A TTP is used as CA. • A smart contract is used as an agent to aid CA.
Duties	CA manages the revocation requests and certificate issuance and revocation.	• CA manages certificate issuance and revocation. • A contract manages the revocation request.

Assistance with the Existing TTP. We use the contract-based CBE protocol to explain this concept. As discussed before, traditional CBE schemes rely on CA (see Table 3.4). However, an illegal certificate revocation may disable the decryption capabilities of the corresponding certificate's owner. In particular, CA may arbitrarily revoke a valid certificate[3], repudiate its actions, and thus indirectly cause decryption to fail. For example, Alice sends a ciphertext to Bob, but an evil CA has already revoked Bob's certificate without his permission. There is no way for Bob to obtain an up-to-date certificate. As a result, he cannot decrypt the ciphertext. Worse still, Bob cannot blame CA due to the absence of valid evidence on his malicious revocation. Meanwhile, the absence of external incentives for CA decreases their willingness to behave honestly.

We observe that smart contracts can support the existing TTP by adding functions of validity checking, on-chain storage, and cryptocurrency-based rewards/punishments.

TTP-I based CBE protocol mitigates the problems of malicious revocation (for the definition of CA's valid revocation and illegal revocation, see Table 3.1), absent evidence and poor incentives. Firstly, the transparency property lets the revocation conditions become publicly visible and verifiable. Secondly, CA, in practice, may become malicious, and then it denies his misbehaviour on illegal revocation

[3]Reasons for CA's malicious activities are diverse, e.g., hidden interests, and details are outside the scope of this work

(a valid certificate is maliciously revoked), or a user illegally revokes his certificate and then frames CA's honesty. In our solution, the smart contract is required to receive revocation requests from users and then provide valid ones to CA through transactions, which can be used as evidence to detect illegal revocations, making CA and users' actions undeniable and accountable. Through tracing the transaction sender, the tracer will know the initiator of revocation operations. Thirdly, if necessary, our scheme can automatically provide cryptocurrency-based rewards/punishments under predefined policies in the smart contract for CA's actions, which motivates CA to behave honestly.

Replacement of the Existing TTP. In the second case, RBE allows users to generate their public and secret keys. However, RBE still places a significant amount of trust in KC, whose actions are not accountable. A dishonest KC may hide a trapdoor that facilitates the secret creation of a key pair for yet unregistered identity or even registers multiple keys for already registered users. The issues of strong centralization, untrusted KC, and identity authentication threaten the wide adoption of existing RBE schemes. In our case, smart contracts are used to automate the logic of KC transparently, enabling publicly upgradeable proofs on-chain that render KC's actions accountable. Equivalently, smart contacts take over the tasks of KC, replacing this centralized role with a distributed blockchain. The solution thereby transfers the right of key management from KC to users.

> We find that smart contracts can replace the traditional TTP by acting as an agent that takes over various tasks. It brings transparency and trust to centralized TTPs.

Our TTP-I based RBE scheme delivers a number of advanced properties. Firstly, the scheme provides transparency to every participant, making KC's behaviours accountable. Users' registration and parameters can be publicly accessible. The only secret parameter is the user's private key generated by him/herself. This separation between the key pair and public parameter effectively limits the ability of KC to misbehave, and the permanent on-chain records make KC's actions traceable and accountable. Secondly, our scheme eliminates the reliance on a central authority and moves related operations of KC on-chain, where the original TTP is replaced by the smart contract with automatic state transitions. This replacement protects KC from external attacks, such as MitM [88] between the message senders and the message receivers. Furthermore, the smart contract ensures KC's high availability since it is preserved by a group of maintainers.

We have compared our designs with existing studies [86, 95] with respect to the properties of *transparency*, *verifiability*, *succinctness*, and *high availability* (see Table 4.5). The KC in the original version of RBE does not hold a master secret key and thus cannot arbitrarily decrypt the users' ciphertexts. However, KC works in a non-transparent way, and all the registration operations are carried out locally. The lack of transparency leads to the system's lack of verifiability, where KC is vulnerable to internal cheating. These drawbacks hinder the wide adoption of RBE. Verifiable RBE [95] equips the RBE with provable short proofs to give evidence of the existing registration. Nevertheless, integrating more crypto primitives with the original protocol makes it extremely complicated and inefficient. Also, the high availability of KC in these studies can not be guaranteed. Our transparent solution resolves the above issues by utilizing external blockchain services. To the best of our knowledge, it is the first work that uses the contract to solve the untrusted KC issue for making the RBE scheme more practical to be adopted.

Table 3.5: Comparison of various RBE solutions

	Transparency	Verifiability	Succinctness	High availability
Original RBE [86]	✗	✗	✓	✗
Verifiable RBE [95]	✗	✓	✗	✗
Transparent RBE	✓	✓	✓	✓

3.3.1.1 Properties Summary

TTP-I brings security and accountability to existing protocols. The properties of *state-change-transparency* and *state-consistency* of smart contracts ensure the non-equivocal data of a TTP. In particular, a smart contract shares the same auxiliary data (state) for message sender's encryption and message receiver's decryption, which lets the system avoid split-world attacks [166] where a malicious TTP may provide a different state view to the message sender and message receiver. In TTP-I protocols, after a transaction's Tx invocation, a unique and deterministic state will be given and shared with the message sender, message receiver, and other observers. Meanwhile, TTP-I provides a high available service for all participants (e.g., the message sender, message receiver). It runs on a decentralized network, which can operate continuously without failing for a designated period of time. Even if one blockchain maintainer goes down, the other maintainers still work. This high availability guarantees that the message sender and receiver can always get the auxiliary information for encryption and decryption when they need it.

Also, as discussed before, to trigger a contract execution, an external message is required. This evidence is represented as a transaction sent from a particular address and confirmed by blockchain maintainers, where the transaction must be signed. This transparent execution and the unforgeability of signature guarantee *non-repudiation* and *non-frameability*, further achieving accountability. A user cannot deny having executed a certain function in a smart contract. This indirectly indicates that the transaction that triggered the execution of smart contracts cannot be tampered with. Moreover, a user cannot be framed by producing evidence of his "honest behaviour". If an honest user does not invoke the function in a contract, he will never be wrongfully accused.

3.3.2 TTP-I Challenges and Issues

In this section, we discuss the main challenges of using TTP-I in security protocols.

Firstly, TTP-I lacks confidentiality and privacy. The state information and the instruction code in TTP-I are completely transparent, and any state and its changes are publicly accessible and observable; all users' transaction data and contract variables are visible to any external observer. Without privacy, building advanced security protocols that rely on sensitive data becomes a challenge [12, 182]. A TTP typically requires maintaining secret information for further operations (as discussed in Section 1.1). The complete transparency of TTP-I constrains its wide adoption. For instance, TTP-I cannot be directly employed as the role of PKG in

IBE schemes, where all the TTP-I's states are transparent and PKG requires hiding the master private key from the public. In other words, TTP-I runs contrary to the intention of the existing TTP.

Secondly, using TTP-I for a security protocol pays a high price on gas consumption. For example, under the initial design of Ethereum, the operations related to calculation or storage require the contract caller to pay a gas fee to the blockchain miners for providing computational resources. Furthermore, Ethereum sets a ceiling for operations that can be included in each contract. For example, in our implementation of transparent RBE, each smart contract can only tolerate about 30 *merge* operations for key accumulation in the Merkle Tree management due to the bottleneck of gas limits (upper bound reaches $12,134,453$ gas). Towards this setting, each contract can involve 14 registered users (equivalently, 28 tree leaves at the bottom and a total 29 *merge* operations), which is impractical for large-scale applications.

Thirdly, TTP-I runs based on distributed smart contracts, which requires each blockchain node to perform computations and communicate with other peers to validate results, reach a consensus and update its final state. This mechanism makes TTP-I suffer from low-performance issues; it can take several seconds to thousands of seconds to finish a full-cycle TTP-I operation, depending on the particular consensus algorithm and network scale. Worse still, neither increasing the transaction load nor increasing the number of blockchain nodes improve performance significantly, which means that TTP-I has scalability issues.

Last but not least, TTP-I protocol, essentially, is a combination of smart contract and cryptographic schemes. Such a combination inevitably introduces new exploitable attack vectors to undermine security. On the one hand, for the contract codes, there is no effective method to make them avoid all vulnerabilities [143]. Any unintentional bugs caused by inappropriate design may make the whole protocol fail. On the other hand, some critical bugs that may exist in the contract execution environment (e.g., Ethereum Virtual Machine [213]) also pose security risks to TTP-I protocols.

Chapter 4

TTP-II: Smart Contract & TEE-assisted TTP

As aforementioned, privacy is of critical concern in TTP-I based security protocols. This chapter introduces a fair and privacy-preserving TTP (we will refer to as TTP-II). To achieve the goal of creating such a TTP, we first investigate state-of-the-art technologies on the implementation of existing TEE-assisted confidential smart contract (TCSC) systems (Section 4.1). Then, we provide a systematization and a unified framework to evaluate them (Section 4.2). Based on the common features, we further present a formal treatment of TCSC (Section 4.3). Finally, we explore how to employ TCSC to build secure protocols with fairness and confidentiality (Section 4.4).

4.1 TEE-assisted Confidential Smart Contract

Blockchain-based smart contracts enable a fair and secure exchange between unfamiliar participants and present a uniform approach for improving applications across a range of industries. However, they lack *confidentiality*. The state information and the instruction code are completely transparent [93, 221, 225], and any state and its changes are publicly accessible; all users' transaction data and contract variables are visible to any observer. Without privacy, building advanced decentralized applications (DApps [176]) that rely on users' sensitive data becomes a challenge [12, 182, 199, 220]. For instance, smart contracts in Ethereum [213] cannot be directly applied to Vickrey auction [23, 82] or e-voting systems [58, 59], where the bid and vote are required to be hidden from the public. Moreover, DApps without privacy protections might be prohibited by the European Union because they go against the General Data Protection Regulation (GDPR) [201]. Thus, the complete transparency of smart contracts constrains their wide adoption. Recently, researchers have explored many cryptographic solutions to solve these issues, including utilizing techniques of zero-knowledge proof (ZKP) [12, 36, 37, 51, 111, 126], homomorphic encryption (HE) [190] and secure multiparty computation (MPC) [226]). However, these approaches are time-consuming and complicated, hindering their adoption by applications requiring compute-intensive tasks.

Moving complex computations into secure hardware offers applications with privacy as well as good performance. The use of TEEs [34, 69, 112, 123, 133] becomes thus a general-purpose solution for confidential smart contracts. The TEE is a new feature provided by recent commodity CPUs. It has the ability to provide secure environments for running contract code in isolation while guaranteeing execution integrity and state confidentiality. As a promising alternative technology, TEE has been adopted by various smart contract platforms, especially by companies working on consortium blockchain platforms, such as Alibaba CONFIDE [215], Visa's LucidiTEE [189] and China's official blockchain CHANG'AN CHAIN [76, 117].

4.1.1 System Workflow

This section first provides a high-level description of TCSC. Then, a detailed example is provided to demonstrate the protocol roles and contract working mechanism.

4.1.1.1 A Lightning Tour

Numerous TCSC systems have been proposed, as shown in Table 4.4. From these examples, we observe that these systems have a common workflow: Establishing a confidential smart contract mainly requires four steps, namely *invocation*, *computation*, *consensus* and *response* (see Figure 4.1). Note that we assume that TCSC has been successfully deployed and the contract state has been initialized.

Invocation. In current blockchain systems such as Ethereum [213], once a contract is deployed successfully, the initial state and operational code are replicated among distributed nodes. Since each transaction requires computational resources for executions, an external message call with sufficient gas must be launched. This type of external message call is represented as a transaction (denoted as Tx) sent from a user. TCSC, as a special type of smart contract, inherits the state-triggering mechanism. A major difference between confidential contracts and original protocols lies in whether a transaction has to carry a ciphertext (denoted as c_u).

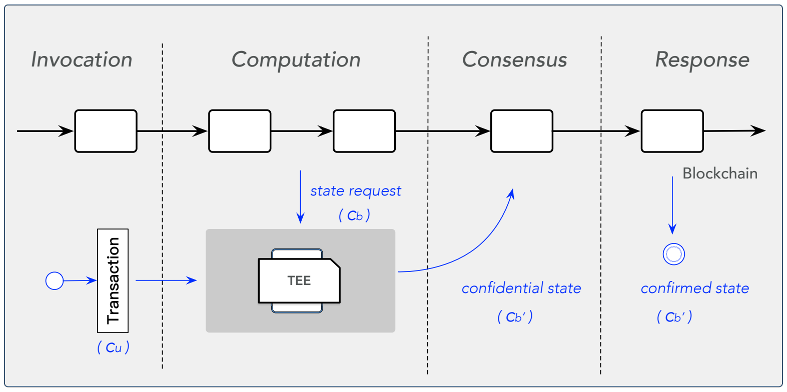


Figure 4.1: TEE-assisted confidential smart contract workflow

Computation. Once receiving an invocation request (Tx with an encrypted argument of c_u) from a user, TEE decrypts the ciphertext c_u and loads the contract source code and current encrypted contract state (denoted as c_b) from the blockchain network. Here, the contract code can be in plaintext or encrypted, but the states and inputs must be encrypted (see Table 4.6). Then, TEE decrypts the state c_b using a TEE private key sk, executes the contract logic, and outputs an execution result (denoted as m_b). Afterwards, TEE encrypts m_b with a specific user's public key and a service key to obtain a ciphertext (denoted as c_b'). Next, TEE sends (denoted as c_b') to the blockchain network. This computation algorithm can run in a TEE or multiple TEEs. If it runs in multiple TEEs, the service key must be shared among multiple TEEs. As for the concept of service key and key management, we refer to Section 4.1.2 for more details.

Consensus. After obtaining the encrypted state c_b', the consensus algorithm starts to reach an agreement over all distributed nodes. Since a block cannot contain the entire view of the blockchain at one time, each miner must re-execute the consensus progress. This is usually triggered by the block synchronization mechanism. To be specific, once enough blockchain miners receive the block and re-execute transactions, the transferred state c'_b will eventually reach the final agreement. Meanwhile, the transaction Tx becomes immutable. Note that, in all the consensus procedures, c'_b is kept encrypted, and the agreement runs on encrypted data.

Response. Finally, the blockchain returns the encrypted state c'_b and corresponding transaction Tx to the corresponding user, and this user decrypts the ciphertext c'_b to obtain the final state. Note that, in the real-world settings, c'_b may be formed as two parts: *encrypted state* for blockchain and *encrypted data* for users (see our example in Section 4.1.1.2). Here, we emphasize that no matter what the format of c'_b is, only the user who owns the private key can access the plaintext.

4.1.1.2 E-voting Example using Confidential Smart Contract

This section provides a modified secret e-voting example borrowed from Oasislabs[1]. Again, we take Intel SGX [61] as the TEE instance, and we note that the TEE design used in our example can also be implemented on other trusted hardware platforms, such as RISC-V Keystone [133].

In this example, three voters $voter1$, $voter2$, and $voter3$ want to vote for their favourite candidates (see Table 4.1). A high-level overview is that: $voter1$, $voter2$ and $voter3$ call the contract inside TEE with encrypted inputs by sending transactions with an encrypted argument c^1_u, c^2_u and c^3_u. Next, TEE decrypts the arguments c^1_u, c^2_u and c^3_u, and decrypts the current encrypted blockchain state c_b (to obtain metadata for voting). Afterward, TEE confidentially executes the voting logic and correspondingly returns $m'_{blockchain}$ and m'_t. Then, TEE encrypts $m'_{blockchain}$ as $c'_{blockchain}$, m'_t as c'_t, and sends $c'_{blockchain}$ and c'_t to the blockchain. Eventually, the blockchain reaches a consensus on $c'_{blockchain}$ and c'_t. After that, the teller fetches c'_t and decrypts it to obtain the voting result $voteresult$.

What properties does TCSC-based voting protocol own? A TCSC can be well qualified for the role of *decentralized vote manager* in an e-voting sys-

[1]https://github.com/oasislabs/secret-ballot/blob/master/contracts/SecretBallot.sol

Table 4.1: Data workflow of TCSC-based voting system

Stage/Role	**Voter**	**Teller**	**TEE**	**Blockchain**
Invocation	$data_u^1 \rightarrow c_u^1$; $c_u^1 \rightarrow \mathsf{Tx}_u^1$; $data_u^2 \rightarrow c_u^2$; $c_u^2 \rightarrow \mathsf{Tx}_u^2$; $data_u^3 \rightarrow c_u^3$; $c_u^3 \rightarrow \mathsf{Tx}_u^3$;			c_b;
Computation			$c_u^1 \rightarrow data_u^1$; $c_u^2 \rightarrow data_u^2$; $c_u^3 \rightarrow data_u^3$; $c_{blockchain} \rightarrow m_{blockchain}$; $\mathsf{Exec}(data_u^1, data_u^2, data_u^3, m_{blockchain}) \rightarrow m'_{blockchain}, m'_t$; $m'_b, m'_t \rightarrow c'_{blockchain}, c'_t$;	
Consensus				$\mathsf{Tx}_u^1 \rightarrow chaindata$; $\mathsf{Tx}_u^2 \rightarrow chaindata$; $\mathsf{Tx}_u^3 \rightarrow chaindata$; $c'_{blockchain} \rightarrow chaindata$; $c'_t \rightarrow chaindata$;
Response		$c'_t \rightarrow voteresult$		

tem [58, 59]. Once a contract-based manager is deployed successfully, the voting logic is loaded into a TEE, and corresponding secret keys are privately generated and stored inside TEEs. The encrypted state is then confirmed by the blockchain nodes. This offers the e-voting protocol *confidentiality*, *neutrality*, *auditability* and *accountability*. Firstly, voters' inputs c_u^1, c_u^2, and c_u^3 are encrypted, and intermediate parameters (e.g., $m_{blockchain}$) are privately processed through TEEs. External attackers cannot obtain sensitive information, and thus confidentiality is achieved. Secondly, the predefined voting logic only occurs in the decentralized network when certain conditions are satisfied, bringing neutrality to the access control management. Thirdly, if a voter wants to vote for a candidate, she needs, in advance, to encrypt his choice using TEE's public key and merge the encrypted choice to a transaction Tx. After that, she calls the contract by sending Tx. Due to the protection of encryption, the voter's choice is kept secret, while some metadata, such as the transaction sender and receiver, remain visible and immutable, ensuring that the user's voting is accountable.

We admit that *public verifiability* as one of the fundamental properties of the blockchain system is difficult to achieve in the context of encryption. Contracts that are executed inside TEE make the execution procedure lack public verifiability. Only the nodes who install TEE and that are given corresponding keys can verify the correctness of contract executions. However, unencrypted transaction metadata (e.g., transaction sender, transaction receiver, value) is publicly verifiable, which makes it possible to verify the absence of double spending.

Sub-procedures. A TCSC-based voting system mainly consists of two sub-procedures: *deployment stage* and *execution stage*. In the deployment stage, all the operational code and the initial state are coded into a TCSC. This stage includes two steps.

a. Compile. Contract binary codes are compiled into enclave codes. Since an enclave has only a small quantity of trusted zones for application code and data (the protected memory is 128MB, and only 96MB is usable for an enclave in the current version of Intel SGX [61]), a contract has to determine the boundary of these zones and identify corresponding zones used for privacy-critical functionalities. In particular, the e-voting contract needs to define: *the scope of the secret state*, *the scope of the public state*, *the approach to access the secret state* and *the approach to access the outside state*.

In SGX, Enclave Definition Language (EDL) defines trusted components, untrusted components, and corresponding interfaces between them, which takes charge of translation from contract code to enclave code. It provides two functionalities: Enclave Calls (ECALLs) and Outside Calls (OCALLs). ECALLs define the functions inside the enclave that are used to expose APIs for untrusted applications to call in. OCALLs specify untrusted functions outside the enclave where the enclave code is able to invoke. In our example, the total number of votes cast for a candidate cannot be revealed until the voting has ended. Thus, the total number of votes cast is defined at the access point ECALLs, and is thereby hidden from the public and can only be revealed once the voting procedure has been completed.

b. Load. Afterwards, EDL files will load into an enclave, which is stored in the Enclave Page Cache (EPC). From a micro perspective, the first step is to call the ECREATE instruction for creating an enclave. This will allocate memory inside the Enclave Page Cache (EPC). Then, enclave code and data are added to pages in EPC by calling the EADD instruction. Finally, when the instruction EINIT completes successfully, an enclave's $INIT$ attributes become true, and the above instructions cannot be used anymore.

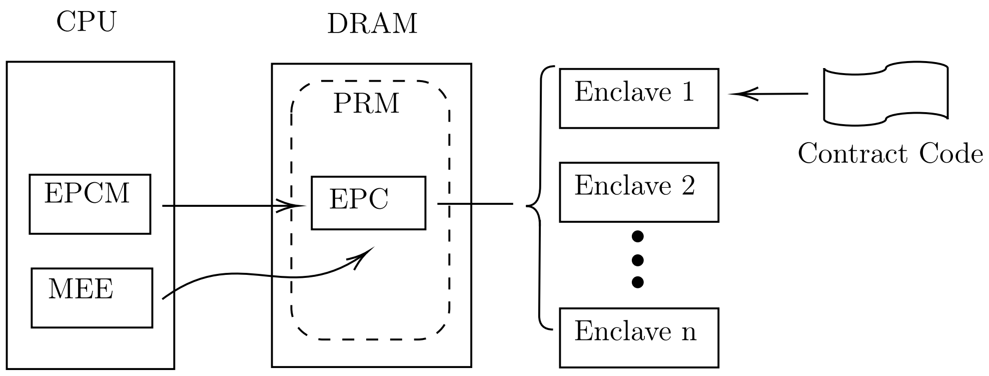


Figure 4.2: A diagram on loading the contract code to multiple enclaves

It is worth mentioning that SGX allows multiple enclaves to run on one system at the same time. This is achieved by dividing the EPC into 4 KB pages and assigning them to different enclaves. Each EPC has its own SGX Enclave Control Structure (SECS), one or more Thread Control Structures (TCS), corresponding Save State Areas (SSAs), Signature Structures (SIGSTRUCT), and Version Array Pages. The SECS and TCS hold the metadata for each enclave, which is mainly used for managing simultaneous threads. SIGSTRUCT is responsible for the signature and sealing identities of the enclave.

After a successful deployment, the initial state and operational code of this contract will be replicated among blockchain nodes. This means the e-voting logic cannot be changed. But, the state of functionalities can be transferred to parties who have been granted permission with a message-call [213].

In the execution stage, voters call the deployed TCSC to finish the voting. This process has three main features.

Execution Isolation. SGX-enabled processors protect the integrity and confidentiality of computation inside an enclave by isolating enclave code/data from the outside environment. To have a better understanding, we will introduce several new concepts: Enclave Page Cache Map (EPCM), Processor Reserved Memory (PRM), and Memory Encryption Engine (MEE). Enclave Page Cache Map (EPCM) is used to perform EPC security checks, which contains the status for each page listed in EPC. EPC performs actions inside the Processor Reserved Memory (PRM) located in Dynamic Random Access Memory (DRAM), while EPCM is a look-up table stored inside a CPU with enclave-related data. Since the data stored in PRM is encrypted, a new tool named Memory Encryption Engine (MEE) [98] is necessary to decrypt data inside the CPU in real time. Here, we emphasize that the MEE key is generated at the boot time and stored within a CPU. The CPU is thereby the only place of the system that can read the plaintext data stored in the enclave.

Before an enclave starts to execute the voting operation, it needs to fetch the current contract state c_b from the blockchain. Then, the CPU starts to execute the plaintext contract in the enclave mode. External attackers cannot obtain the knowledge of sensitive information since the MEE key never leaves TCB. To prevent data leakage, a CPU cannot directly deal with failures such as an interrupt, fault (for example, a page fault) or VM exit when executing enclave code. Considering the protection of private data, the CPU first needs to switch from enclave code to ring 3 [47] code by performing an Asynchronous Enclave Exit (AEX) and then execute the above operations.

Hardware-based Access Control. One important aspect of Intel SGX's functionality is that the code inside an enclave can access the particular enclave state by performing additional checks on memory semantics [163]. Back to our example, an enclave executes the voting operation only when the following four requirements are fulfilled: (1) The processor runs in enclave mode; (2) The requested page is part of the same enclave; (3) The page access is through the correct specific virtual address; (4) The code semantics successfully pass the check (the invocation is initiate by the authorized address). In a word, the CPU is acting as a security guard in the TCSC, providing a hardware-based access control mechanism.

Transaction-based Accountability. After obtaining results from TEEs, the consensus algorithm starts to reach an agreement. Specifically, when a miner receives a newly mined block, he will re-execute all transactions inside the block to obtain the newly transferred state. Here, we emphasize that only the TEE result is encrypted, the other metadata of the transaction remains unencrypted, and the "re-execute" operation is necessary to verify the absence of double spending. Once enough blockchain miners receive the block and re-execute transactions, the voting results and the transactions triggering the contract execution will eventually reach the final agreement. When all the voting procedures have ended, the teller can fetch the final encrypted state and obtain the final voting result. Meanwhile, the transactions can be used as evidence to trace the voter's behaviour.

4.1.2 Key Management

Various keys are involved in the contract execution, including TEE internal keys such as the attestation key and TEE service keys for state encryption/decryption (see Figure 4.3). We will continue to use the above confidential e-voting contract (see Section 4.1.1.2) to explain the mechanism of key management. In terms of key generation and usage, the workflow is shown as follows. Initially, SGX generates an MEE key at boot, which is placed in special registers and destroyed at system reset. This MEE key is utilized for memory encryption and decryption of enclaves. At the same time, SGX generates a report key, an attestation key and a sealing key for further actions. We summarize these keys and their usage in Table.4.3.

Then, TEE creates an enclave, and this enclave generates a key pair (pk_{tee}, sk_{tee}) for protecting the user's input and a symmetric key $\mathsf{k}_{blockchain}$ for protecting the enclave output. After that, $voter1$, $voter2$, $voter3$ and $teller$ generate their key pairs $(pk_{voter1}, sk_{voter1})$, $(pk_{voter2}, sk_{voter2})$, $(pk_{voter3}, sk_{voter3})$ and $(pk_{teller}, sk_{teller})$ respectively. Subsequently, $voter1$, $voter2$ and $voter2$ send an encrypted voting choice c_u^1, c_u^2 and c_u^3, respectively, by using the public key pk_{tee} that belongs to an enclave, in which the corresponding private key sk_{tee} is stored in enclaves. Then, the enclave receives ciphertexts (c_u^1, c_u^2 and c_u^3) attached to transactions, and then decrypts them using sk_{tee}. Afterwards, the enclave fetches the newest state from a blockchain and decrypts it using $\mathsf{k}_{blockchain}$, and starts to execute the contract. Then, TEE encrypts current voting state as $c'_{blockchain}$ using $\mathsf{k}_{blockchain}$, and final voting result as c'_t using pk_{teller}. After that, in the consensus stage, $c'_{blockchain}$ and c'_t will reach agreement on the blockchain nodes. Finally, $teller$ fetches c'_t and

Table 4.2: Varieties of TEE internal keys and their main functionalities

TEE Key	**Functionalities**	**Remarks**
MEE key	Stored inside a CPU; used to encrypt (decrypt) data before writing (reading) it to (from) RAM.	Different enclaves in the same TEE platform share one MEE key.
Report key	Derived by EGETKEY instruction; used to generate a MAC tag for the measurement.	Different enclaves in the same TEE platform share one report key;
Attestation key	Stored in tamper-resistant hardware; only used to produce attestation signatures.	Different enclaves in the same TEE platform share one attestation key;
Sealing key	Stored in tamper-resistant hardware; used to migrate secrets between enclaves.	Different enclaves in the same TEE platform may share the sealing key depending on key policies.

decrypts it to obtain the final result using sk_{teller}.

Table 4.3: Enclave service keys and their main functionalities

TEE Key	**Functionalities**	**Remarks**
Enclave asymmetric key (pk_{tee}, sk_{tee})	Generated by an enclave; used to encrypt (decrypt) data that comes from voters.	pk_{tee} is shared among all the votes.
Enclave symmetric key $\mathsf{k}_{blockchain}$	Generated by an enclave; used to encrypt (decrypt) the blockchain state.	$\mathsf{k}_{blockchain}$ is privately stored inside an enclave.

Key management dilemma. The private keys in TEE-assisted systems are crucial but hard to manage. On the one hand, putting the service keys (e.g., sk_{tee} and $\mathsf{k}_{blockchain}$) in a single TEE contributes to key security. However, it also raises the risk of a single point of failure. On the other hand, sharing the private key among multiple TEEs offers practical availability but (as a sacrifice) increases key exfiltration risk. State-of-the-art TCSC systems provide a range of technologies to avoid this dilemma. Ekiden [53] designs a distributed key generation (DKG) [89] protocol using a secret sharing scheme [185]. Even if one key is compromised, an adversary still cannot obtain the entire key. However, this solution does not completely solve the key leakage issues. The final keys are assembled and replicated among all the end-TEEs. If an adversary compromises an end-TEE, exposing the entire contract state becomes a trivial task. The key rotation technology, adopted by Fastkitten [64], Phala [101] partially solves the above issues by providing a short-term key in every epoch. An adversary cannot corrupt a future or previously committed state, which minimizes the possibility of key exposure to attackers and further helps the layer-two system to achieve forward secrecy. Also, some projects such as COMMITEE [72] mitigate these key issues by providing each TEE with a secret key. Even if a certain TEE's private key were stolen, this only would affect the smart contract running on that compromised TEE. Furthermore, Phala Network [101], equips each contract with an asymmetric key called the *contract key*, which also enhances key security to a certain degree. CONFIDE [215] mitigates this issue by proposing a decentralized key management protocol. Two types of keys are involved in this protocol: the *asymmetric private key* used to decrypt confidential transactions from clients and the *symmetric states root key* used for state encryption/decryption between the confidential engine and storage service. However, these technologies are too complicated for widespread adoption and cannot completely solve the key issues. Suppose an attacker steals the attestation key somehow. She might consequently generate the attestation materials to deceive the user with a fake fact: the contract has been executed. Even worse, if a root key stored in the tamper-resistant hardware (e.g., Memory Encryption Engine Key in Intel SGX) is compromised, all aforementioned key technologies for protecting service keys become useless.

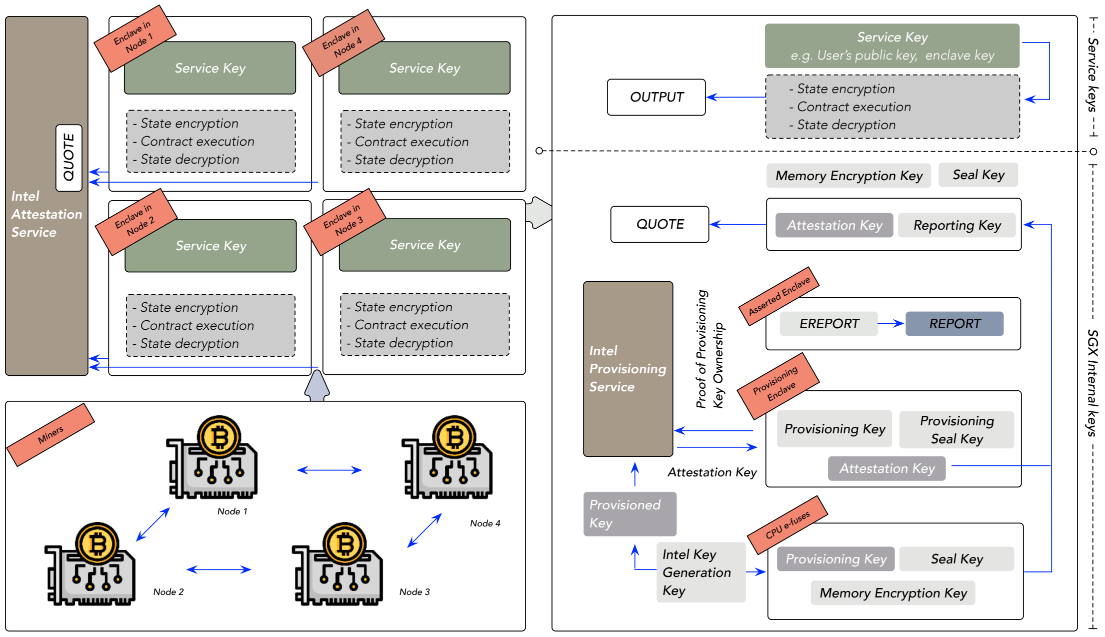


Figure 4.3: Key usage in a TCSC system

4.2 TCSC Classification and Evaluation

This section provides a systematization of existing TEE-assisted confidential smart contracts driven from academic work and *in production* projects. Based on their operating mechanisms and methods of combination, we divide them into two main categories: *layer-one* solution and *layer-two* solution. Then, we establish a unified evaluation framework that takes into consideration two factors: *privacy-preserving properties* and *blockchain intrinsic features*.

4.2.1 System Classification

TCSC systems can be categorized into two classes according to the place where contract execution and consensus procedure operate: *layer-one solution* and *layer-two solution*.

4.2.1.1 Layer-One Solution

A *layer-one solution* enables blockchain nodes to run contracts in their isolated areas and to conduct the consensus. This approach combines the consensus pro-

Table 4.4: Selected TEE-assisted confidential smart contract systems

Project	**TEEs**	**Main Functionalities**
Ekiden [53]	Intel SGX	Ekiden proposes a TEE-blockchain hybrid architecture where the contract computation is separated from consensus.
PDOs [31]	Intel SGX	PDOs leverages Intel SGX to run the contract code off-chain; a user is acting as a channel for communication between the SGX enclave and the blockchain.
ShadowEth [218]	Intel SGX	ShadowEth establishes a confidential platform protected by TEEs off the public blockchain for contract execution and storage
Phala [101]	Intel SGX	Phala proposes an interoperable smart contract network by combining TEEs and query responsibility segregation architecture [167]
Enigma [2, 71]	Intel SGX	Enigma implements a so-called Secret Network, a proof-of-stake blockchain that is built on top of the Cosmos SDK [60] by using Tendermint consensus [197]
Fabric [33]	Arm Trustzone	Fabric proposes a modular architecture that conceptually decouples the traditional chaincode executions into two types of functionalities containing *transaction ordering* and *state execution*.
CCF [179]	Intel SGX	CCF implements a leader-based consensus protocol (in particular, RAFT [165]) and provides secure memory databases and remote procedure call (RPC) services
CONFIDE [215]	Intel SGX	CONFIDE separates transactions into *public* and *confidential* transactions; confidential transactions are processed in a confidential engine, whereas public transactions are processed in normal procedures

cedure and contract execution, either logically or physically. The reason why we call this method *layer-one* is that all executions are completed in the same layer of the blockchain network, both in terms of physical space and time. The key to such an approach is to equip every blockchain node with a TEE. While this certainly requires greater integration efforts, it also comes with several advantages. In particular, a smart contract can implement stateful functionalities that receive arguments and update the state instantly; a smart contract can directly access the local ledger, greatly saving time often wasted in communications.

In the layer-one execution pattern, the operations of ledger update (consensus) and state transition (smart contract) are coupled. Like Ethereum [213], smart

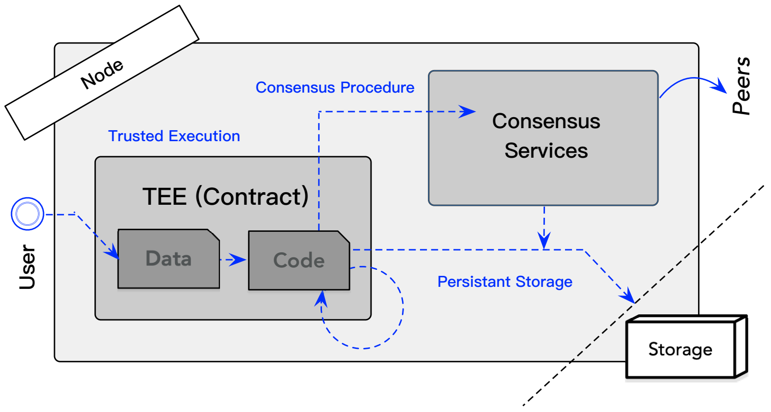


Figure 4.4: Layer-one execution model

contracts run inside blockchain nodes. Assume that a user plans to use the private contract, then he only needs to upload data to the blockchain service and wait for results. The remaining procedures are fully completed by TEE-assisted distributed nodes. The TEE in these nodes acts as a *black box* for data processing, and outputs targeted results without any leakage of sensitive data. This approach greatly improves the convenience for users due to its easy access and management. As illustrated in Figure 4.4, a generic data flow is as follows: A contract creator deploys the code into the blockchain and bootstraps the system. Then, a user sends his encrypted input data to an arbitrary blockchain node. His request is confidentially executed by inside TEEs that are connected with smart contracts. After a full cycle of execution, encrypted results are broadcast to peers. After encrypted results are confirmed by other blockchain nodes through the consensus algorithm, the user who sent the request can fetch results from the blockchain.

4.2.1.2 Layer-Two Solution

A *layer-two solution* is a straightforward approach to combining the TEE and blockchain to offer smart contracts with confidentiality while retaining scalability. In such systems, the operations of smart contracts are decoupled from their underlying blockchain systems and are executed in an independent layer outside the blockchain systems.

In a general *layer-two solution*, the blockchain is used as a dispute resolution layer.

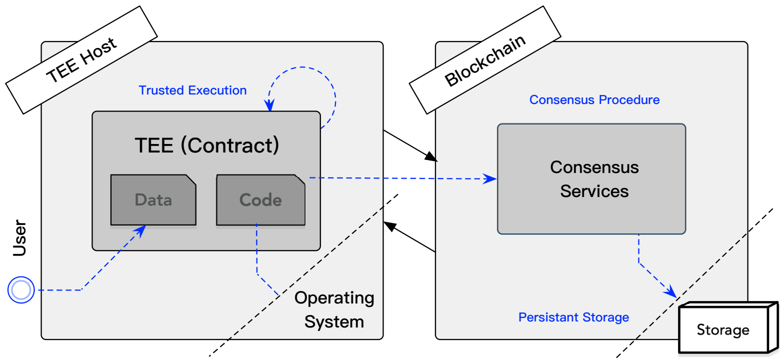


Figure 4.5: Layer-two execution model

The smart contract is executed outside the blockchain, making TEEs act as an agent between users and blockchain systems. Suppose that a user aims to use a private contract. He first needs to compile original contract codes, push (compiled) binary codes to a TEE, and then upload execution results to the public ledger. As illustrated in Figure 4.5, we extract a generic data flow as follows: a user sends the encrypted input data to a TEE-powered node. Then, the TEE correspondingly decrypts the input data and executes the contract. Subsequently, the encrypted execution results are sent to the blockchain platform for verification and storage. Finally, the user fetches and decrypts blockchain-confirmed results.

Summary. The *layer-one* solution and *layer-two* solution share some same principles: (i) the contracts are executed in an isolated secure area; (ii) the state-changing in a contract is based on an external message call, usually represented as sending a transaction by a user; (iii) the final encrypted state is confirmed by blockchain nodes and finally stored on the chain.

The main difference between layer-one and layer-two solutions lies in the position to execute the confidential contract. In the layer-one solution, the blockchain miner is required to provide a TEE service, making the miner execute a confidential state, and its consensus procedure is executed in the same machine. In contrast, in the layer-two solution, the state and corresponding procedure are performed independently.

For a clear understanding, we make a comparison of the original blockchain system

Table 4.5: A comparison of L_1 and L_2 solutions

	Ethereum	L_1 Solution	L_2 Solution
EVM and consensus in same machine	✓	✓	✗
EVM and consensus in same TEE	-	✗	✗
Contract execution publicly verifiable	✓	✗	✗
Contract execution peer verifiable	✓	✓	✗
Consensus procedure publicly verifiable	✓	✓	✓

(e.g., Ethereum), layer-one (L_1) solution, and layer-two (L_2) solution. As shown in Table 4.5, Ethereum runs smart contracts (in Ethereum Virtual Machine) and consensus procedures in the same machine of distributed nodes. All the contract and transaction operations are publicly verifiable due to their total transparency. The layer-one solution performs such operations (contract execution and consensus) in the same machine, but the contract operations are separate from consensus procedures. In contrast, the layer-two solution makes both of them operate independently. The contracts are executed outside the blockchain network, while state consensus happens inside the blockchain nodes.

4.2.2 Evaluation Framework

This section establishes a unified evaluation framework. Ideally, moving smart contract execution into TEEs brings additional security and privacy and maintains the original benefits of blockchain. Thus, we identify desirable properties in two main categories: *privacy-preserving properties* and *blockchain intrinsic features*.

4.2.2.1 Privacy-Preserving Properties

The property of confidentiality is the most distinguished feature in TEE-assisted confidential smart contracts.

A1. *Specification hidden.* The source code of a smart contract is hidden during the deployment and the subsequent synchronization and execution.

A2. *Input privacy.* The inputs fed into a confidential smart contract are hidden from the public. An adversary cannot see the plaintext of the inputs without a private key.

A3. *Output privacy.* The output returned from a confidential smart contract is kept private. Only the authorized user who holds the private key can access the plaintext.

A4. *Procedure privacy.* The execution procedure is hidden from unauthorized parties. An adversary cannot learn operation knowledge inside a TCSC.

A5. *Address unlinkability.* The pseudonymity of blockchain does not entail strong privacy guarantees [8, 153]. An adversary may break the address unlinkability by observing users' activities. This property ensures that transactions are unlinkable.

A6. *Address anonymity.* The contract caller's identity (a user who calls a smart contract) is hidden from an anonymity set [36].

4.2.2.2 Blockchain Intrinsic Features

The TCSC also inherits some features given by original blockchain systems. We summarize these features as follows.

A7. *Code immutability.* Once a contract is successfully deployed, its source code cannot be altered.

A8. *(Private) state consistency.* The execution happens at a certain blockchain height, where all execution outputs stay the same across different nodes, despite these outputs being encrypted.

A9. *Contract interoperability.* A smart contract can call another contract and be called by others.

A10. *High availability.* A smart contract is continuously accessible without a single point of failure.

A11. *Decentralized execution.* A smart contract runs over a decentralized network, making the same operations execute many times in different blockchain nodes, which guarantees fairness.

A12. *Automatic execution.* A smart contract can be automatically executed once conditions are satisfied.

A13. *Gas mechanism.* Operations running on smart contracts are charged with gas fees [213].

A14. *Explicit invocation.* Each invocation will be formatted as a transaction and stored on the blockchain.

A15. *Public verifiability.* The procedure of contract execution and the result are publicly verifiable.

A16. *Consensus verifiability.* The consensus procedure on the (confidential) state is publicly verifiable.

A17. *Transaction transparency.* This property ensures that transactions that trigger the execution of TCSC are immutable, and cannot be cancelled, tampered with, or reversed by any involved parties.

A18. *Transaction unforgeability.* Transaction unforgeability is defined as the inability to forge a valid transaction that deceives the blockchain to accept this forged transaction.

4.2.3 L1 and L2 Evaluation

4.2.3.1 Layer-One Solution

Privacy-preserving properties indicate that contract states and the procedure of contract executions are hidden from the public. To achieve privacy, layer-one systems execute these confidential contracts inside TEEs in every distributed node. CCF [179], Fabric [33] and CONFIDE [215] follow this straightforward design, where confidential contracts are loaded to a TEE of each consensus node, which encrypts both the inputs and outputs of contract states, together with their operating logic and predefined rules. Enigma [2] introduces a secret network layer, and this layer allows users to submit their transactions together with encrypted data to miners. We also notice that current layer-one solutions only focus on internal procedures rather than the linkability and anonymity of addresses and transactions. This indicates that TCSC only protects the contents that have been loaded into TEEs, while the data that relates to outside users is out of scope.

Layer-one systems inherit most of the features empowered by blockchain. More precisely, the properties of *code immutability*, *high availability*, *explicit invocation*, *decentralized execution*, *automatic execution* and *consensus verifiability* remain the same because basic contract executions still rely on their underlying blockchain systems. Also, the property of (confidential) *state consistency* in Enigma [2], CCF [179] and Fabric [33] remains unchanged. The states and executions from these systems follow the procedures of online consensus processes. Then, the returned results from inside TEEs must still be confirmed on-chain. This makes their actions effectively perform the same functions as a normal smart contract, except that the contents of states are transmitted from plaintext to ciphertext. However, the property of *contract interoperability* is lost since the contracts are executed in isolated TEEs.

4.2.3.2 Layer-Two Solution

The *confidential execution* is an essential property. In layer-two systems such as [31, 101, 218], the contract computations run inside Intel SGX enclaves, while TZ4Fabric [157] moves contract executions into ARM Trusted Zone. Since the contract state transition process happens inside the TEE, any intermediate states remain invisible to the outside. Meanwhile, to achieve full life-cycle security for a smart contract, the input sent to a TEE and the output returned from a TEE are also required to be encrypted. For example, in ShadowEth [218], PDOs [31], Phala [101] and Hybridchain [209], the contract invocation arguments are encrypted with the TEE public key. They can only be decrypted within the enclave. Also, before transferring execution results to the blockchain (or users), the intermediate (or final) states in an enclave are encrypted. Some variants also enhance the privacy-preserving properties from other aspects. In Phala [101], only authorized queries to the contract will be answered. The smart contract source codes in ShadowEth [218] are hidden during deployment and synchronization. This further reduces the possibility of data leakage in subsequent contract execution. Consid-

Table 4.6: Comparison of $L1$, $L2$ system and normal smart contract (SC)

Properties	**Sub-properties**	$L1$	$L2$	SC
	Specification hidden	✓̷	✓̷	✗
Confidentiality Property	Input privacy	✓	✓	✗
	Output privacy	✓	✓	✗
	Procedure privacy	✓	✓	✗
	Address unlinkability	✓̷	✓̷	✗
	Address anonymity	✓̷	✓̷	✗
	Code immutability	✓̷	✓̷	✓
Intrinsic Feature	State consistency	✓	✓	✓
	Contract interoperability	✓̷	✓̷	✓
	High availability	✓̷	✓̷	✓
	Decentralized execution	✓̷	✓̷	✓
	Automatic execution	✓	✓	✓
	Gas mechanism	✓̷	✓̷	✓
	Explicit invocation	✓̷	✓̷	✓
	Public verifiability	✗	✗	✓
	Consensus verifiability	✓	✓	✓
	Trans transparency	✓	✓	✓
	Trans unforgeability	✓	✓	✓

The character ✓ represents that the item completely holds this property; ✗ means that the item does not own this property; ✓̷ denotes that the item partly has this property. The underline signifies the common properties.

ering a fixed address may expose the user who calls the contract, PDOs [31] also allow the user to use pseudonym addresses for submitting a transaction with TEE outputs to the blockchain.

As for the intrinsic feature, ShadowEth [218] and Taxa [196] introduce a distributed service to manage the contracts, achieving the properties of *code immutability*, *high availability* and *decentralized execution*. Meanwhile, the layer-two systems satisfy *state consistency* because the encrypted states of contracts in different blockchain nodes are eventually consistent when reaching a successful agreement. Intuitively, the contracts deployed in layer-two systems should retain the features given by original blockchain systems. However, some fundamental properties are lost to a certain degree when using layer-two solutions. For example, most layer-two systems lose contract interoperability since each contract is executed on different machines. Among all the evaluated systems, only Phala [101] identifies this issue and proposes a command query responsibility segregation architecture to ensure interoperability. Also, public verifiability is a crucial property for the blockchain since it allows each contract invocation, and the correctness of contract execution to be publicly verifiable. Unfortunately, contracts are executed in TEEs and outputs are encrypted. To check whether TEE has executed contracts following loaded contract source specifications is a non-trivial task.

4.2.3.3 Properties Summary

Common Properties. In both $L1$ and $L2$ systems, the contract state transition process happens inside the TEE, and any intermediate state remains invisible to outside programs. Therefore, the procedure of executing a contract is hidden from unauthorized third parties. Meanwhile, to achieve full life-cycle security for a smart contract, the input sent to a TEE and the output returned from a TEE are also required to be encrypted. Even if an adversary can see encrypted states in the blockchain, they cannot access the plaintext or learn any information about how the plaintext hidden inside the encrypted state is transferred, without fetching corresponding keys. Briefly, these properties ensure that no additional information is leaked to an adversary during the process of *contract execution* and *data transmission* of honest parties.

Meanwhile, a TEE-assisted contract as a special smart contract inherits many distinguished features of original blockchain systems. Firstly, TCSC inherits the state triggering mechanism from smart contracts. State-changing is based on an external message call, which is usually represented as sending a transaction by a user. This transaction can be further used as a piece of evidence to audit operations inside the smart contract. In a word, both $L1$ and $L2$ system hold the properties of *Procedure privacy*, *State consistency*, *Trans transparency* and *Trans unforgeability*, as shown in Table 4.6.

Shortcomings. Still, a TCSC faces some issues. Firstly, it lacks public verifiability. On the one hand, contracts are executed inside TEE, and the outputs are usually encrypted, which lacks public verifiability as compared to traditional blockchain systems. The attestation service can only guarantee that the encrypted outputs indeed come from a TEE. However, neither users nor the blockchain nodes can learn whether the TEE is compromised or executes contracts following the predefined specification. Even if many TEEs can re-execute the same contract with the same setup (e.g., the same private key) to check outputs, this inevitably increases the risk of key leakage. On the other hand, the precise architectures of some chips are still unclear for some TEE products, such as Intel SGX [61]. TEE-assisted solutions force the user to put too much trust in the manufacturers of this hardware. Some users even argue that Intel may have reduced the security of SGX to improve performance to cater for market demand [67]. Additionally, the attestation service used to prove that a program runs inside TEEs is *centralized* and *nontransparent*. A compromised provider has the ability to insert fake IDs, and further, steal the confidential state in smart contracts.

4.3 TCSC Formal Treatment

This section begins with the syntax of TCSC. Then, the security properties are formalized.

4.3.1 TCSC Syntax

As we discussed in Section 4.1, there are several approaches to achieving confidentiality for a smart contract, including cryptographic approaches backed by mathematics, and TEE-based approaches. Without loss of the generality, we, therefore, provide a general definition called *privacy-preserving smart contract*, rather than *TEE-assisted confidential contract*. This definition is modified from Definition 6.

Definition 7 ($\widehat{PPSC}$)**.** *A privacy-preserving smart contract (PPSC) is a private state machine built on top of a blockchain system and can be modeled by a tuple* $(\mathcal{S}, \mathcal{S}', \mathcal{P}, \mathcal{T}, \mathbb{B})$ *under a transition function*

$$f : \mathcal{S}', \mathcal{P}' \overset{\mathbb{B}}{\leftarrow} \mathcal{S} \otimes \mathcal{P} \otimes \mathcal{T},$$

where $\mathcal{S}$ *represents a set of private states with the initial state* s*,* $\mathcal{P}$ *is a public state space with the initial state* p*,* $\mathcal{S}'$ *is the new state set after the specified operations,* $\mathcal{T}$ *means a set of publicly visible transactions that can trigger the execution of a contract, and* $\mathbb{B}$ *represents a blockchain system which provides the execution environment.*

- **Deploy.** $(\langle opcode\rangle, \langle reqcode\rangle, s, p) \leftarrow \langle bytecode\rangle \otimes \mathsf{Tx}$. The deployment is triggered by a transaction Tx, where $\mathsf{Tx} \in \mathcal{T}$. It takes as input the binary code $\langle bytecode\rangle$ and outputs a private state s, and a public state p. In particular, the contract is compiled into instruction codes $\langle opcode\rangle$ and $\langle reqcode\rangle$, where $\langle opcode\rangle$ specifies the operation set to be executed and $\langle reqcode\rangle$ defines the conditions depending on which the operation of $\langle opcode\rangle$ can be conducted.

- **Transfer.** $(s', p') \overset{\mathbb{B}}{\leftarrow} \langle opcode\rangle \otimes \langle reqcode\rangle \otimes s \otimes p \otimes \mathsf{Tx}'$. By sending a transaction Tx' with some inputs (optional), the current private state s is transited to a new private state s', and the current public state p is transited to a new public state p' under the guidance of the instruction codes $\langle opcode\rangle$ and $\langle reqcode\rangle$. The new private state s' is returned only when Tx' satisfies the conditions defined in $\langle reqcode\rangle$, *i.e.*, $\mathsf{Tx}' \in \langle reqcode\rangle$.

- **Read.** $s'' \overset{\mathbb{B}}{\leftarrow} \langle opcode\rangle \otimes \langle reqcode\rangle \otimes s' \otimes \mathsf{Tx}''$. By sending a query transaction Tx'' through the blockchain system $\mathbb{B}$, the private state s'' is returned only when Tx'' satisfies the conditions predefined in $\langle reqcode \rangle$, *i.e.*, $\mathsf{Tx}'' \in \langle reqcode \rangle$.

In this definition, the instruction code including $\langle opcode \rangle$ and $\langle reqcode \rangle$, the transactions Tx, Tx' and Tx'', the state p and p' are completely transparent. In contrast, state s, s' and s'' and their changes are hidden from the public.

4.3.2 Security Properties

Based on the above syntax, we provide four security properties: *state-privacy*, *state-consistency*, *transaction-transparency* and *transaction-unforgeability*. We emphasize that these properties are held by both $L1$ and $L2$ systems (see Table 4.6).

State-privacy. The state-privacy guarantees that the contract state is protected from the public. Only the user who satisfies predefined conditions can learn the confidential state. This property in TCSC is achieved by the hardware-based access control mechanism and hardware-based runtime isolation. We give an informal definition as follows. No PPT adversaries $\mathcal{A}$ can successfully obtain private states s, s' and s'' without sending a transaction or sending an unauthorized transaction.

State-consistency. The state-consistency ensures that a smart contract shares the same confidential data view after the operation code is executed. This property in TCSC is achieved by the consensus algorithm. We give an informal definition as follows. No PPT adversaries $\mathcal{A}$ can obtain two valid confidential state $s^{\star}$ and $s^{\star\star}$ for executing **Transfer** algorithm once.

Transaction-transparency. The transaction-transparency ensures that transactions triggering the execution of PPSC can be freely queried without being cancelled, tampered with, or reversed by any involved parties. We give an informal definition as follows. No PPT adversaries $\mathcal{A}$ can obtain two transactions $\mathsf{Tx}^{\star}$ and $\mathsf{Tx}^{\star\star}$ for executing **Transfer** algorithm once.

Transaction-unforgeability. The transaction-unforgeability guarantees transactions (as evidence) are reliable and authentic without being forged or cheated. We give an informal definition as follows. No PPT adversaries $\mathcal{A}$ can forge a valid transaction $\mathsf{Tx}^{\star}$ that can trigger the execution of **Transfer** algorithm.

4.4 TTP-II : TCSC-based TTP

In this section, we give a definition of TTP-II protocol. Then, we show a comparison between the TTP-I construction and TTP-II construction.

4.4.1 General Construction

Essentially, TTP-II is defined as a customized confidential smart contract. Thus, a TTP-II protocol consists of two main types of roles: *protocol users* and *confidential smart contract*. The protocol users are composed of both the message sender and the message receiver. The confidential smart contract is used to aid or replace a TTP to maintain secret auxiliary information. In a normal case, the auxiliary information stored in a TTP-II protocol is confidential, and only the authorized user who satisfies predefined conditions can learn this auxiliary information by sending a transaction. Meanwhile, this transaction will be used as evidence to prove the fetching of the confidential state. A generic construction is shown as follows[2].

Comparison with Construction of TTP-I. In order that the reader can quickly and simply be familiarized with TTP-II construction, the algorithms that are different from TTP-I's are denoted with (**); the algorithms that are identical to TTP-I's are denoted with (=).

At first, as a conventional security protocol, a TTP-II protocol needs a global setup and a key generation for the message sender and message receiver.

(=) **System Setup** $pms \leftarrow \mathsf{Setup}(1^\lambda)$. The algorithm takes as input a security parameter λ, and outputs the system parameter pms, which are implicit inputs for the rest of the algorithms.

(=) **Key Generation** $(sk, pk), (sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. The algorithm takes as input a security parameter λ, and outputs a key pair (sk, pk) for encryption and decryption, and a transaction key pair $(sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}})$.

[2]Note that our general construction mainly aims to capture the critical features of TTP-II protocol. Some sub-algorithms may be omitted or changed as the different manifestations in our instance. For example, the "Encryption" is replaced by "Credential Generation" in our credential anonymity revocation system.

Then, a smart contract is deployed, with outputting a contract identity $\widehat{c}$, an initial state $\overline{s}$, the operational code $\langle opcode \rangle$, and the execution condition $\langle reqcode \rangle$. The logic of a TPP is coded into $\langle opcode \rangle$, and the execution condition of the logic is coded into $\langle reqcode \rangle$. This step is finished by calling $\widehat{\mathcal{PPSC}}$.**Deploy** defined in Section 4.3.1. Next, a message sender conducts the pre-operation using the receiver's identity/key with auxiliary data *auxdata* stored in the deployed smart contract. Two examples of pre-operation are message encryption and signature generation. Here, we emphasize that, at this stage, only the message sender knows *auxdata*. Should the message receiver wish to learn *auxdata*, he must seek the assistance of the contract; this assistance is represented as calling a function defined in $\langle opcode \rangle$ by sending a transaction.

(=)**Transaction Generation** $\mathsf{Tx} \leftarrow \mathsf{Sign}(sk_{\mathsf{Tx}}, metadata, auxdata)$. The algorithm takes as input a transaction *metadata*, an auxiliary data *auxdata*, a private signing key sk_{Tx}, and outputs a transaction Tx.

(**)**TTP-II Operation** $s' \overset{\mathbb{B}}{\leftarrow} \mathsf{TtpOperate}(\widehat{c}, \overline{s}, \mathsf{Tx})$. The algorithm takes as input a contract identity $\widehat{c}$, current contract state $\overline{s}$, a transaction Tx, and outputs the transferred state s'. This algorithm is completed by calling the algorithm $\widehat{\mathcal{PPSC}}$.**Transfer** described in Section 4.3.1.

Encryption $ct \leftarrow \mathsf{Enc}(pk, auxdata, m)$. The algorithm takes as input pk, an auxiliary data *auxdata* and a message m, and outputs a ciphertext ct. This algorithm is completed in the local client of users. At this stage, only the sender knows this private auxiliary data.

(**)**State Read** $s' \overset{\mathbb{B}}{\leftarrow} \mathsf{Read}(\widehat{c}, \mathsf{Tx})$. The algorithm takes as input a contract identity $\widehat{c}$, a transaction Tx, and outputs a new state s'. This algorithm is conducted by a message receiver. The receiver has to interact with the blockchain to obtain auxiliary data. This algorithm is completed by calling the algorithm $\widehat{\mathcal{PPSC}}$.**Read** described in Section 4.3.1.

Decryption $m \leftarrow \mathsf{Dec}(sk, auxdata, ct)$. The algorithm takes as input sk, an auxiliary data *auxdata* extracted from s', ct, and outputs a message m. This algorithm is completed in the local clients of users.

Inspection $true/false \overset{\mathbb{B}}{\leftarrow} \mathsf{Inspect}(\mathsf{Tx})$. This algorithm takes as input Tx, and

Table 4.7: Comparison of TTP-I construction and TTP-II construction

Algorithms	**TTP-I Protocol**	**TTP-II Protocol**
TTP-Operation	• SC is used as a TTP. • The changes of a state are transparent. • All entities can see the state, and only authorized entities can change the state. • The transactions that trigger the change of a state are transparent.	• TCSC is used as a TTP. • The changes of a state are private. • Only authorized entities can see and change the state. • The transactions that trigger the change of a state are transparent.
State Read	The state can be accessed by reading the blockchain.	The state transfer operations are needed.
Inspection	The state transfer operations are traceable by checking the transaction.	The state transfer operations are traceable by checking the transaction.

returns the legality of the **Transfer** operation.

The transactions that trigger the execution of a contract in the **State Transfer** operation can be used as evidence to indicate users' misbehaviours, which significantly reduces the probability of TTP or a user acting maliciously.

4.4.2 TTP-I and TTP-II Comparison

Similar to TTP-I, TTP-II employs a customized smart contract as a TTP, but such a contract is privacy-preserving (details are shown in Table 4.7). The contract state and the changes of the state are hidden from the public. In particular, the state inside a contract can only be revealed to authorized parties who called a specific pre-defined method by using a publicly traceable transaction.

Chapter 5

Security Protocols Using TTP-II

This chapter proposes two security protocols using the concept of TTP-II: a credential anonymity revocation system (Section 5.1), and an accountable decryption system (Section 5.2). In both protocols, TTP is replaced by an ideal contract-based "decentralized black box" providing secrecy, correctness, and fairness.

5.1 Credential Anonymity Revocation System

5.1.1 Problem Statement

Anonymity revocation was first discussed by Von Solms and Naccache [202], as they pointed out that Chaum's blind signatures [48] could potentially lead to nonpunishable crime. Subsequently, anonymity revocation has been studied comprehensively, especially in e-cash systems designed to combat money laundering and blackmailing [24, 39, 41]. The idea of adding anonymity revocation to anonymous credential systems was first proposed by Camenisch and Lysyanskaya [38], where they offered an optional anonymity tracing approach to find the identity of pseudonymous tokens involved in suspicious transactions. In general, anonymity revocation in a credential system allows an issuer to find out the person who owns an anonymous credential.

The blindness issuance property of an anonymous credential system prevents an issuer from completing the task of anonymity revocation by themselves. The party who helps the issuer to reveal the identity is referred to as revelator. Intuitively, there are two parties that can act as the revelator: the user (credential holder) and the judge (trusted third party). Voluntary anonymity revocation by the user is usually straightforward. The issuer cannot link the identity, the message and the resulting signature together unless the user does. One typical example is Microsoft's U-Prove [169]. In such a system, the issuing protocol and the showing protocol are unlinkable. Even if the issuer colludes with the verifier, it cannot associate the message with the resulting signature. The only possibility is that the user chooses to lift anonymity. Meanwhile, lifting anonymity by a judge, which is inspired by fair blind signature scheme [80], is widely used in systems such as [40, 73, 170, 175]. Taking ABC4Trust [175] as an example, it introduces an inspector to uncover the user who created a presentation token to prevent abuse.

However, several flaws remain in the aforementioned approaches for revocation anonymity. Firstly, revealing anonymity through the credential holder relies too heavily on the user's will, which ultimately leads to the non-availability problem. This means if a user behaves maliciously and rejects to cooperate with the issuer, the issuer will never learn the relationship between the identity and the credential. Furthermore, even if the user is honest, they may be offline, resulting in the failure of blindness removal. Meanwhile, in most previous proposals revealing anonymity through the judge lacks transparency, which raises several security concerns: (1) even without the user's consent, the issuer and the judge may conspire to map the credential to the real identity of the user; (2) the judge confronts a single point of failure. More importantly, the user has no auxiliary information to detect whether the judge has been compromised or not. These challenges lead to the following question:

Is it possible to build an anonymity revocation mechanism that satisfies these requirements: (1) the process of lifting anonymity is transparent and auditable; (2) the revelator always accepts revealing anonymity if necessary?

This chapter gives a positive answer to this question. Instead of using a trusted third party, we use a neutral and privacy-preserving smart contract (PPSC) as the revelator to revoke the blindness. The self-execution property of the smart contract ensures the neutrality of the revelator. This means the neutral blockchain is always honest and willing to revoke anonymity whenever it is needed by the issuer. Meanwhile, our PPSC-based approach allows anonymity revocation in an auditable manner. More precisely, the anonymity tracing must interact with the PPSC that "lives" on the blockchain and automatically renders the progress auditable. Such revocation progress is recorded in a blockchain transaction which is publicly visible. This auditability provided by the smart contract calling records avoids the misuse of revocation and reduces potential collusion problems to a great extent. Furthermore, the transparent contract calling records provide the user with auxiliary information to detect whether the issuer has been compromised.

5.1.2 Related Work

In the last few decades, a series of works [4, 80, 120, 192], have been proposed in the field of anonymity revocation, especially in e-cash systems. Brickell *et al.* [35] introduced the first trustee-based tracing electronic cash system, in which the coin owner can be revealed by several publicly appointed trustees. Camenisch *et al.* [39] proposed an anonymous digital payment system with a passive anonymity-revoking trustee. In their system, the trustee only needs to be involved in the anonymity-revoking progress rather than regular transactions such as opening a new account. Jakobsson and Yung [106] presented an e-money system that makes the value of funds and user anonymity revocable with the consumer rights organisations, even given an extreme condition that an active attacker gets the bank's key or forces the bank to release the money.

In 1995, a fair blind signature scheme was first proposed by Stadler *et al.* [192]. It involved a judge and allowed this judge to deliver information to the signer to link the issuing session and resulting message-signature pair. Later, Jakobsson and Yung [107] pointed out that the reused session identifier may make the anonymity revocation invalid, and proposed a fair blind signature scheme that guarantees the one-to-one mapping revocability between the issuing session and the resulting signature. Thereafter, Hufschmitt *et al.* [105] presented a formal security model for fair blind signatures in the random oracle model. Then, based on Hufschmitt's model, Fuchsbauer *et al.* [80] proposed a fair blind signature scheme that is not based on the random oracle model. To the best of our knowledge, Camenisch and Lysyanskaya [38] were the first to use anonymity revocation in the credential system. They offered an optional approach to trace the identity of the pseudonymous token for transactions. After that, practical systems like IBM's Identity Mixer [40], ABC4Trust [175] started to consider the anonymity revocation. An interesting revocation approach is traceable anonymous certificate [130]. It allows one sub-issuer to verify the ownership of a user and another sub-issuer to validate the contents. Then, these two issuers collaborate to map the certificate to its real identity.

However, the aforementioned anonymity revocation approaches have drawbacks: the repudiation and the lack of auditability in the revocation progress. The assumption that the revelator always remains honest is unrealistic. The revelator may be offline when it is needed, may conspire with the issuer to seek profits, or even be entirely controlled by an attacker. Our scheme is the first to use a PPSC as the revelator to solve the above problems. The self-executing nature of the contract ensures the neutrality of the revelator. The transparent contract calling records guarantee that the revelator's revocation progress is auditable.

5.1.3 Construction Overview

An auditable blind credential system has six participants (see Figure 5.1): *issuer*, *user*, *verifier*, *tracer*, *inspector* and *PPSC platform*. The user is the holder of a credential. The issuer is in charge of blindly issuing a credential. The verifier is responsible for checking the validity of the credential. The tracer is used to reveal the relationship between the credential and the user's identity. It should be noted that we introduced the concept of tracer and allow both the issuer and the verifier to act as the tracer. The inspector is used to check suspicious revocation activities and report them. The PPSC platform is employed as a revelator to provide the revocation service.

The PPSC platform can run based on either $L1$ system or $L2$ system (we refer readers to Section 4.2.1 for more details), which includes two types of nodes: *TEE-powered nodes* and *consensus nodes*. The TEE-powered nodes are composed of the contract-TEE and the key manager TEE. The contract-TEE is used to execute the smart contract and then encrypt the resulting state with the key from the key

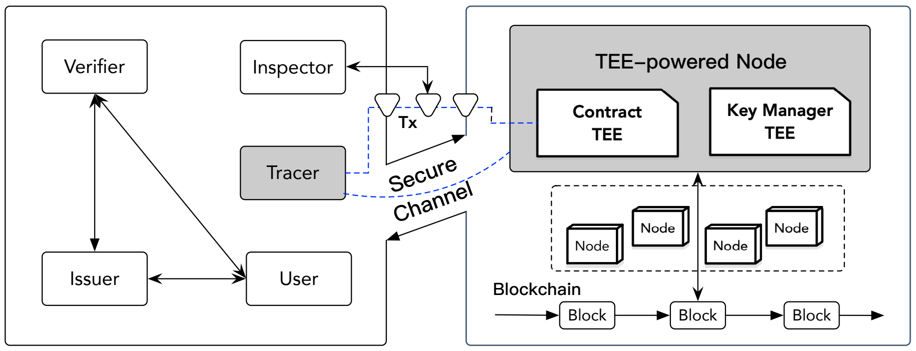


Figure 5.1: The overview of the auditable blind credential system

manager TEE. Consensus nodes are used to achieve an agreement on the encrypted state of the smart contract.

A basic auditable blind credential system works as follows. The system sets up parameters and key pairs for the issuer, the user, and the tracer. Then, the system calls TEE-powered blockchain nodes to obtain a PPSC $\widehat{c}$, in which the method name, arguments, and returned data are externally invisible. After that, the user interacts with the issuer to obtain an anonymous credential. Next, the user shows the credential to the verifier who wants to check validity. So far, due to the blind issuance, neither the issuer nor the verifier knows the relationship of the credential and its holder's identity. In the revocation stage, the tracer first builds an encrypted and authenticated channel with the contract-TEE (one crucial property of remote attestation [152] in TEE, see Section 2.3). Then, given the user's identifier or the anonymous credential, $\widehat{c}$ lifts the blindness and returns the result to the tracer bearing a transaction[1].

Due to the protection of the encrypted channel, transaction contents, including the input and the output, are kept secret. However, the invoking records of the transaction remain visible and become immutable because of the confirmation by consensus nodes. Alternatively, any entity can see the fact that the tracer is interacting with the contract, but nobody, except the tracer, knows the exact data in the transaction. Subsequently, the inspector scans the blockchain to collect the

[1]Here, this transaction must be initiated from a specific identity owned by the tracer. When contract-TEE executes the functionalities of blindness lifting, it first checks whether the caller has permission to call this function.

tracer's calling records and inspect the suspicious credential tracing activity.

5.1.4 Concrete Instantiation

This section presents an instantiation based on Abe's blind signature scheme [4] and the PPSC platform of Ekiden [52]. For security and efficiency purposes, we slightly modify Abe's scheme by using elliptic curve cryptography. As a result, all the following arithmetic operations are based on the addition of points, unless otherwise noted. Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that generates a group parameter. Define $(p, q, g, h) \leftarrow \mathcal{G}(1^{\lambda})$, where p is a large prime number, (g, h) are generators of subgroup of order q in $\mathbb{Z}_p$. Define $\mathcal{H}_1 : \{0,1\}^{\star} \rightarrow \langle g \rangle$ and $\mathcal{H}_2, \mathcal{H}_3 : \{0,1\}^{\star} \rightarrow \{0,1\}^{|q|}$. $\mathcal{H}_1$ refers to mapping an arbitrary string to an element of the subgroup $\langle g \rangle$. $\mathcal{H}_2$ and $\mathcal{H}_3$ all refer to mapping an arbitrary string to an element of $\mathbb{Z}_q$ with a fixed length.

Key Generation The issuer generates a public key y and a tag key z, where $x \in \mathbb{Z}_q, y = g^x \, mod \, q$, and $z = \mathcal{H}_1(p, q, g, h, y)$. A user generates a key pair (γ, ξ), where $\gamma \in \mathbb{Z}_q$ and $\xi = g^{\gamma} \, mod \, q$. To simplify the instantiation, we use the session identifiers to represent the user's identity and allow one user to generate multiple identities $(\gamma_1, \xi_1), (\gamma_2, \xi_2), \ldots (\gamma_n, \xi_n)$. Similarly, the tracer generates the session key pair (ι, τ), where $\iota \in \mathbb{Z}_q$ and $\tau = g^{\tau} \, mod \, q$. Note that the tracer's session key is only used to establish authenticated channels to the contract-TEE.

Contract Registration The system compiles pieces of code of a smart contract $\widehat{c}$ and sends its bytecode to a TEE-powered blockchain node. The code contains the logic of revealing anonymity (Equations 5.1 and 5.2). Then, the TEE-powered blockchain node first loads bytecode into the contract-TEE. Then, the contract-TEE creates a new contract identifier $\widehat{c}$, obtains a fresh internal contract key pair $(pk_{cid}^{in}, sk_{cid}^{in})$ and an internal state key k_{state} from key manager TEE (see Section 4.1.2 for more details on TEE key management). Thereafter, the contract-TEE outputs an encrypted initial contract state $state_{init} = \mathsf{SE.Enc}(\mathsf{k}_{state}, \overrightarrow{state_0})$ and an attestation Ω_{cid}^0, where Ω_{cid}^0 is used to prove the correctness of this initialization. After that, the TEE-powered blockchain node gets a proof π of Ω_{cid}^0 by the attestation service and push the final composition $(\widehat{c}, pk_{cid}^{in}, state_{init}, \Omega_{cid}^0, \pi)$ to blockchain consensus nodes. Then, blockchain consensus nodes will accept this

Role	Symbol	Description
user	γ	user's private key
	ξ	user's public key (user's identity)
issuer	x	issuer's private key
	y	issuer's public key
	z	issuer's tag key
tracer	ι	tracer's private key used to establish channels
	τ	tracer's private key used to establish channels
verifier	ζ_1	user's credential
inspector	$tran$	transaction to recognize suspicious activities.
PPSC	$\widehat{c}$	contract identifier
	$\Omega^0_{cid}, \Omega^1_{cid}$	TEE attestations
	sk^{in}_{cid}	contract private key
	pk^{in}_{cid}	contract public key
	k_{state}	contract state key
	x_t	ppsc-based tracer's private key
	y_t	ppsc-based tracer's public key

Table 5.1: Notions in auditable blind credential system

smart contract if all attestations and proofs are verified successfully. As for parameters' registration, given the common parameters (p, q, g, h) and a public key of an issuer y, $\widehat{c}$ takes a random number x_t under $\mathbb{Z}_q$ as the private tracing key and generates the corresponding public key $y_t = g^{x_t} \, mod \, q$. The private key x_t is secretly held, which can only be accessed by the PPSC $\widehat{c}$ internally.

5.1.4.1 Blind Issuance

Credential Generation Credential generation is an interactive protocol that involves only the user and issuer, which means that it runs independently from the PPSC. A basic version is shown as follows: A user first chooses a blinding factor γ and computes $z_u = z^{1/\gamma}$. Then, she sends (z_u, g^γ) to the issuer. Next, the issuer blindly issues a signature, bringing a pair (z_u, y_t) into (z, y_t^γ). By this approach, the user can obtain a valid signature since she can do the conversion by taking γ-th power. The signature is left blind $g^\gamma \not\Leftrightarrow y_t^\gamma$, since γ is not known by the issuer. The identity and the signature can be traced through exponentiation, namely, $(y_t^\gamma)^{1/x_t} = g^\gamma$ and $(g^\gamma)^{x_t} = y_t^\gamma$. However, these protocols heavily rely on assumptions that all users are honest in choosing a unique blinding factor γ. In our protocol, we follow the idea in [4], and provide a security-enhanced version by adding randomness υ to the blinding factor. Details can be found below.

1. A user chooses a blinding factor γ and computes $z_u = z^{1/\gamma}$ and proves to the issuer that $\log_g \xi$ is equal to $\log_{z_u} z$.

2. An issuer randomly generates υ, and computes $z_1 = y_t^\upsilon$ and $z_2 = z_u/z_1$. Then, he proves to the user that z_1 is made as it should be.

3. Based on y, z_1, z_2, the issuer and the user engage in an interactive proof protocol. For the issuer, the protocol is a witness indistinguishable proof of knowledge of

   $$log_g^y \vee (log_g^{z_1} \wedge log_h^{(z_u/z1)}).$$

   The issuer converts the proof into

   $$log_g^y \vee (log_g^{\zeta_1} \wedge log_h^{(z/\zeta_1)})$$

   by raising them with the blinding factor λ under the standard diversion technique [164]. The converted proof is eventually transformed to a signature with the Fiat-Shamir technique.

4. The issuer stores ξ^υ as the identity of this session. Clearly, ξ^υ is easy to map to ξ, since the issuer knows the υ.

5. The user outputs $cred_u$ with Σ, say, $\Sigma = (\zeta_1, \rho, \overline{\omega}, \sigma_1, \sigma_2, \delta)$ is the signature for the message m.

Credential Verification Credential verification, proceeding after the operation of credential generation, is another interactive protocol that runs independently of blockchain, involving only the user and the verifier. We consider a credential

Issuer (x, υ) | **User** (x_t, m)

$r \in \mathbb{Z}_q^\star$

$z_u = r^{\gamma^{-1}}$

$\xi = g^\gamma$

$$\xleftarrow{z_u, \zeta}$$

$\upsilon \in \mathbb{Z}_q^\star$

$z_1 = y_t^\upsilon, z_2 = z_u / z_1$

$u, s_1, s_2, d \in \mathbb{Z}_q^\star$

$a = g^u; b_1 = g^{s_1} z_1^d$

$b_2 = h^{s_2} z_2^d$

$$\xrightarrow{z_1, a, b_1, b_2}$$

$\zeta_1 = z_1^\gamma, \zeta_2 = z/\zeta_1$

$t_1, t_2, t_3, t_4, t_5 \in \mathbb{Z}_q^\star$

$\alpha = a g^{t_1} y^{t_2}$

$\beta_1 = b_1^\gamma g^{t_3} \zeta^{t_5}$

$\beta_2 = b_2^\gamma h^{t_4} \zeta^{t_5}$

$\varepsilon = \mathcal{H}_2(\zeta_1 | \alpha | \beta_1 | \beta_2 | m)$

$e = \varepsilon - t_2 - t_5 \ mod \ q$

$$\xleftarrow{e}$$

$c = e - d \ mod \ q$

$r = u - cx \ mod \ q$

$$\xrightarrow{r, c, s_1, s_2, d}$$

$\rho = r + t1 \ mod \ q$

$\varpi = c + t_2 \ mod \ q$

$\sigma_1 = \gamma s1 + t3 \ mod \ q$

$\sigma_2 = \gamma s2 + t4 \ mod \ q$

$\delta = d + t5 \ mod \ q$

$(\zeta_1, \rho, \varpi, \sigma_1, \sigma_2, \delta)$

ξ^υ

Figure 5.2: A simplified Abe's scheme [4]

(Σ, m) is valid if it fulfils:

$$\overline{\omega} + \delta = \mathcal{H}_2(\zeta_1 | g^{\rho} y^{\overline{\omega}} | g^{\sigma_1} \zeta_1^{\delta} | h^{\sigma_2} (z/\zeta_1)^{\delta} | m) \ mod \ q.$$

Here, the **Correctness** of credential generation and verification is easy to check as we have,

$$\begin{aligned} g^{\rho} y^{\overline{\omega}} = g^{r+t_1} y^{c+t_2} = g^{r+cx} g^{t_1} y^{t_2} = a g^{t_1} y^{t_2} = \alpha, \\ g^{\sigma_1} \zeta_1^{\delta} = g^{\gamma s_1 + t_3} z_1^{\gamma \zeta} = \beta_1, \\ h^{\sigma_2} (z/\zeta_1)^{\delta} = h^{\gamma s_3 + t_3} z_1^{\gamma \zeta} = (g^{s_1} z_1^{d})^{\gamma} g^{t_3} z_1^{\gamma t_5} = b_2^{\gamma} h^{t_4} \zeta_2^{t_5} = \beta_2, \\ \mathcal{H}_2(\zeta_1 | g^{\rho} y^{\overline{\omega}} | g^{\sigma_1} \zeta_1^{\delta} | h^{\sigma_2} (z/\zeta_1)^{\delta} | m) \ mod \ q \\ = \mathcal{H}_2(\zeta_1 | \alpha | \beta_1 | \beta_2 | m) \ mod \ q \\ = \varepsilon. \end{aligned}$$

Meanwhile, we have,

$$\begin{aligned} \overline{\omega} + \delta = (c + t_2 \ mod \ q) + (d + t_5 \ mod \ q) \\ = e + t_2 + t_5 (\ mod \ q) \\ = \varepsilon. \end{aligned}$$

5.1.4.2 Auditable Revocation

TTP-II Operation (Credential Tracing) Credential Tracing is an interactive protocol that involves the tracer, the TEE-powered blockchain node and blockchain consensus nodes. It aims to find a credential identifier, which covers the following sub-protocols:

(1) A tracer first fetches the pk_{cid}^{in} of the tracing contract $\widehat{c}$, and then encrypts the input of a seesion identifier ξ^{υ} (user's identity ξ and issuer's random number $\upsilon \in \mathbb{Z}_q^{\star}$) as $inpt_c = \mathsf{PKE.Enc}(pk_{cid}^{in}, \xi^{\upsilon})$, and sends $\widehat{c}$ within $inpt_c$ to a

TEE-powered blockchain node.

(2) To start the process of the execution, the TEE-powered blockchain node first loads the contract $\widehat{c}$, the input $inpt_c$ and the previous encrypted state $state_{init}$ into the contract-TEE.

(3) The contract-TEE decrypts inp_c and $state_{init}$ with the keys (e.g., sk_{cid}^{in}, k_{state}) from the key manager TEE, saying that, $\xi^{\upsilon} = \mathsf{PKE.Dec}(sk_{cid}^{in}, inpt_c)$, and $x_t = \mathsf{SE.Dec}(\mathsf{k}_{state}, state_{init})$. Then, it starts to execute the anonymity tracing function and outputs I_{cred} and state $state_t$. Observing that,

$$I_{cred} = (\xi^{\upsilon})^{x_t} = g^{\gamma\upsilon x_t} = y_t^{\gamma\upsilon} = z_1^{\gamma} = \zeta_1. \tag{5.1}$$

(4) The contract-TEE calculates $outp_{new}^{TEE} = \mathsf{PKE.Enc}(pk_{cid}^{out}, I_{cred})$ and a new encrypted state $state_{new}^{TEE} = \mathsf{SE.Enc}(\mathsf{k}_{state}, state_t)$. Then, it sends $state_{new}^{TEE}$, $outp_{new}^{TEE}$ and the proper attestation *quote* to the tracer through a secure channel established by the tracer's session keys (ι, τ).

(5) The tracer acknowledges the reception by calling back the TEE-powered blockchain node, which triggers the contract-TEE to send the transaction $tran = (\widehat{c}, outp_{new}^{TEE}, state_{new}^{TEE}, \Omega_{cid}^{1}, \pi)$ to the blockchain. The signature π is used to protect the integrity of the transaction, and the attestation Ω_{cid}^{1} is used to prove that $outp_{new}^{TEE}$ and $state_{new}^{TEE}$ come from a TEE.

(6) Once the consensus nodes confirm the transaction *tran*, the contract-TEE decrypts $outp_{new}^{TEE}$ and $state_{new}^{TEE}$ as $outp_{new}^{t}$ and $state_{new}^{t}$ and then sends them to the tracer through the mentioned secure channel.

(7) The tracer parses $outp_{new}^{t}$ and $state_{new}^{t}$ and ultimately learns I_{cred}.

Among all sub-protocols, we emphasize that sub-protocol (5) and (6) (Figure 5.3) are atomic operations, and we refer to [52] for more details. Also, we highlight two main features. Firstly, the transaction *tran* will be confirmed by the consensus nodes mentioned in sub-protocol (6). Thus, the contract invoked eventually becomes immutable and auditable. Second, the output $outp_{new}^{t}$ and state $state_{new}^{t}$ are kept secret through the full life cycle of the execution and transmission.

Here, the **Correctness** of credential tracing is easy to check.

In step (1) and step (3), we have

$$\mathsf{PKE.Dec}(sk_{cid}^{in}, \mathsf{PKE.Enc}(pk_{cid}^{in}, \xi^{v})) = \xi^{v},$$

and,

$$\mathsf{SE.Dec}(\mathsf{k}_{state}, \mathsf{SE.Enc}(\mathsf{k}_{state}, \overrightarrow{state_0})) = \overrightarrow{state_0}.$$

In step (4) and step (7), we have

$$\mathsf{PKE.Dec}(sk_{cid}^{out}, \mathsf{SE.Enc}(pk_{cid}^{out}, I_{cred})) = I_{cred},$$

and,

$$\mathsf{SE.Dec}(\mathsf{k}_{state}, \mathsf{SE.Enc}(\mathsf{k}_{state}, state_t)) = state_t.$$

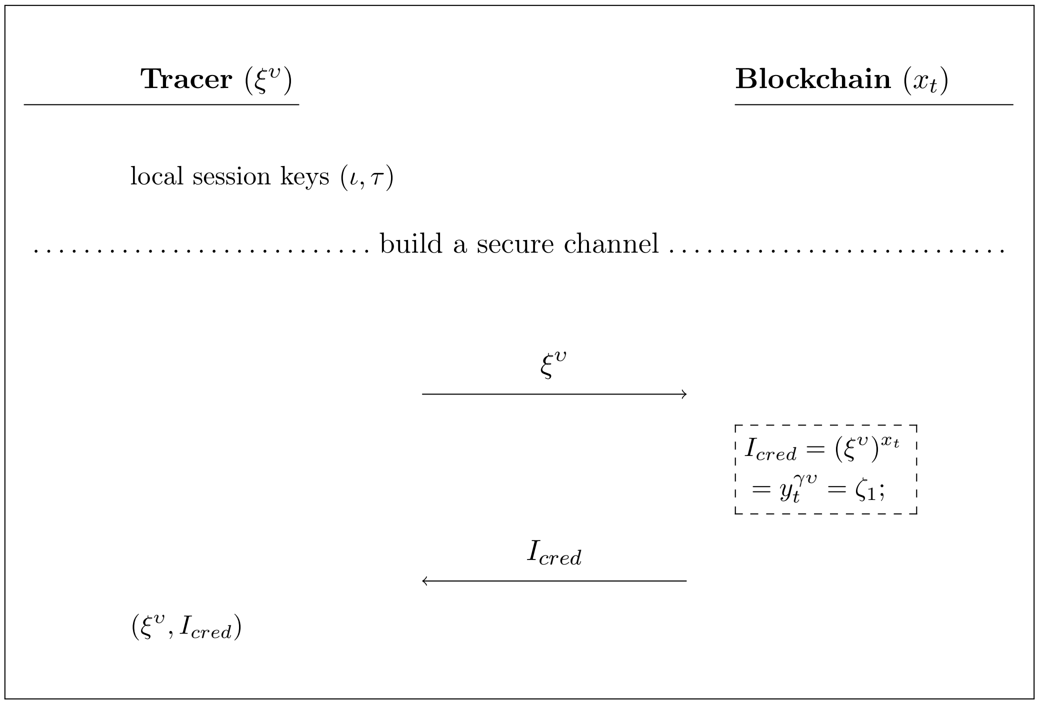


Figure 5.3: The diagram of credential tracing protocol

TTP-II Operation (Identity Tracing) Identity tracing aims to find a user's identifier, given a certain credential identifier (e.g., ζ_1). It is achieved by tracing the session identifier. Observes that,

$$I_{id} = \zeta_1^{1/x_t} = z_1^{\lambda/x_t} = y_t^{\upsilon\lambda/x_t} = g^{\upsilon\lambda} = \xi^{\upsilon}. \tag{5.2}$$

Since the session identifier ξ^{υ} is stored or published by the issuer in the credential generation stage, the tracer can instantly identify the user who issued the credential.

Inspection Given the inspector type (identity tracing or credential tracing) and the smart contract identifier $\widehat{c}$, the inspector scans the blockchain to collect all transactions (e.g., *tran*) related to this contract. Then, the inspector checks all these transactions to recognize suspicious activities.

5.1.5 Implementation and Evaluation

We have implemented a proof of concept of our instantiation. Next, we report on our implementations with their performance.

5.1.5.1 Implementation

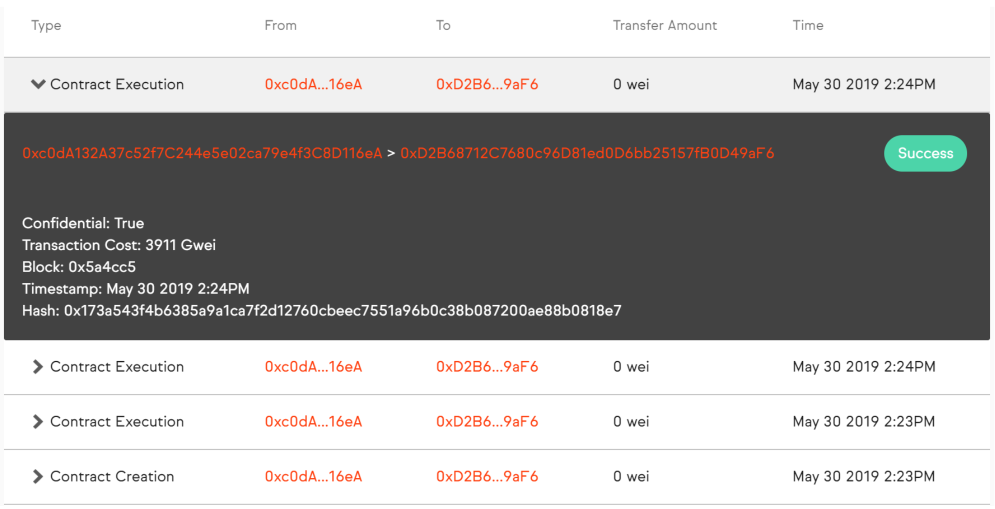
We focus on implementing the blind issuance protocols and the anonymity revocation smart contracts and leave the implementation of TEE-related protocols to the Oasis Devnet [52]. Specifically, our implementation is divided into two modules: *the issuing module* and *the tracing module*. The issuing module covers the protocol of credential generation and credential verification, and it is realised by Python with 168 lines of code. The issuing module is responsible for blindly issuing credentials and verifying the issued ones. Meanwhile, the tracing module which performs the protocol of credential tracing and identity tracing is achieved by Solidity[2] with 449 lines of code and deployed in Oasis Devnet (version 1.0). The tracing module allows the tracer to uncover the identity of a credential or the credential of a specific user.

```
    // example code in Solidity;
```

[2]Solidity is an object-oriented, high-level language for implementing smart contracts.

```
mapping (address => uint256) private CredentialTraceResults;
function CredentialRevocation (uint256 upsilon) {
    CredentialTraceResults[upsilon] = power(upsilon, xt, p);
}
```

Two key properties are highlighted in our implementation: *the full protection of private state* and *the auditable anonymity tracing records*. The full protection of the private state is represented as that input data and the output data in the contract are kept secret in the full life cycle. For example, as it is shown in the example code, the parameter of *CredentialTraceResults* is designed to privately store the relationship of the identity and credential. The other entities cannot read them unless through an end-to-end secure channel that has been established with the contract-TEE. The auditability of anonymity tracing records is evident in that all the smart contract invoking records are publicly visible and immutable (Figure 5.4 is an example of smart contract creation and invoking). In addition, we provide a web-based client to present an interactive process of credential and identity tracing. Full codes are shown in the repository[3].



Type	From	To	Transfer Amount	Time
Contract Execution	0xc0dA...16eA	0xD2B6...9aF6	0 wei	May 30 2019 2:24PM

0xc0dA132A37c52f7C244e5e02ca79e4f3C8D116eA > 0xD2B68712C7680c96D81ed0D6bb25157fB0D49aF6

Confidential: True
Transaction Cost: 3911 Gwei
Block: 0x5a4cc5
Timestamp: May 30 2019 2:24PM
Hash: 0x173a543f4b6385a9a1ca7f2d12760cbeec7551a96b0c38b087200ae88b0818e7

Type	From	To	Transfer Amount	Time
Contract Execution	0xc0dA...16eA	0xD2B6...9aF6	0 wei	May 30 2019 2:24PM
Contract Execution	0xc0dA...16eA	0xD2B6...9aF6	0 wei	May 30 2019 2:23PM
Contract Creation	0xc0dA...16eA	0xD2B6...9aF6	0 wei	May 30 2019 2:23PM

Figure 5.4: A screenshot of credential anonymity revocation records

[3]https://github.com/typex-1/auditable-credential-core

5.1.5.2 Evaluation

Our performance evaluation covers five operations: tracing parameter generation, credential issuing, credential verifying, credential tracing and identity tracing (see Table 5.2). All experiments are conducted on a Dell precision 3630 Tower with 16GB of RAM and one 3.7GHz six-core i7-8700K processor running Ubuntu 18.04. Experiments are measured in seconds through wall clock run time. To have an accurate and fair result, we repeat experiments for each execution 300 times and calculate its average. Also, to simplify the performance evaluation, we measure the running time of each step and accumulate them together if there are many steps involved. Note that all operations take much less than one second to complete, and credential issuing is the main performance bottleneck. This operation takes more time than others because issuing a new credential requires many interactions between users and issuers. Fortunately, this bottleneck can be ignored in real applications because a credential is issued only once but can be verified or traced multiple times.

We then examine the operating cost. Similar to the performance testing, the cost evaluation covers five mentioned credential operations. Table 5.2 shows the data size and the cost of these operations measured in gas under an elliptic curve with 128 bits security level. An analysis of the data size and cost points to some trends. The data size of the parameter generation is the largest since this operation needs to register the group parameters to the smart contract. Surprisingly, the cost of the parameter generation is not the largest, as this operation does not cover complex computations. On the contrary, the data size of the operation of the credential issuing and verifying is zero, and there is no gas cost since these operations are executed independently of the blockchain. Meanwhile, credential tracing and identity tracing have static gas cost since the length of input data of these operations is constant, and the data handling procedure is fixed. In our scheme, a one-time elliptic-curve exponentiation (see Equations (5.1) and (5.2)) is adequate to conduct the complete tracing activity. The gas cost of the one-time computation is quite lower and easier to adopt by users when compared with other blockchain-based applications such as [36, 191], where they have massive elliptic-curve exponentiation operations and significant cost.

Table 5.2: The performance, input data size, gas cost and latency evaluation

Operations	**Performance**(second)	**Size**(byte)	**Cost**(gas)	**Latency**(second)
Parameter generation*	0.00084	260	20672	14.781
Credential issuing	0.00740	0	0	1.601
Credential verifying	0.00232	0	0	1.175
Credential tracing*	0.00306	132	390261	17.538
Identity tracing*	0.00455	132	388944	18.905

* TEE-related operations

Finally, we conduct latency testing, as latency is an essential consideration for adopting a system. For our implementation, the latency includes blockchain confirmation time, network request time and network response time. It is observed that the latency of credential issuing and identity verifying is much smaller than that in other operations. The main reason behind this is that these two operations independently run from the blockchain and do not wait for the block to be confirmed. Meanwhile, the average latency of credential tracing and identity tracing is approximately eighteen seconds, which would be a primary drawback of our system. Given these latency constraints, our system, at least built on the current version of Oasis Devnet, is not suitable for applications that require fast credential tracing or identity tracing. However, for several privacy-priority applications, such as medical record tracing systems, our scheme provides a powerful framework to protect patient's privacy.

5.1.6 Security Discussion

This section provides a security discussion. Note that we only prove the security of the credential tracing, considering that identity tracing has the same mechanism.

Given a valid session identifier ξ^v, if an adversary obtains a corresponding credential identifier ζ_1 without being audited, there are four possibilities. (i) An adversary has successfully accessed the private key x_t, which is stored in the TEE and locally calculated the elliptic-curve exponentiation as shown in Equations (5.1) to conduct the complete tracing activity without interacting with the blockchain. (ii)

An adversary has compromised the DDH-based secure channel and obtained the tracing resulting $outp^t_{new}$ and $state^t_{new}$. (iii) An adversary has successfully forged a valid credential identifier $\zeta_1^\star$ independently from the blockchain, where $\zeta_1^\star$ meets the conditions: $\zeta_1^\star \neq \zeta_1$ and $true \leftarrow \mathsf{FBS.MatchSig}(\zeta_1, \zeta_1^\star)$. (iv) An adversary has called the smart contract $\widehat{c}$ and successfully hid the invoked transactions from the inspector.

Scenario (i) contradicts our assumption that the TEE provides an isolated secure environment. The proof of Scenario (ii) is done by contradiction. Suppose that there exists an adversary $\mathcal{A}$ that compromised the secure channel with success probability $\mathsf{Adv}^{\text{sc}}_{\mathcal{A}}(n)$, where $\mathsf{Adv}^{\text{sc}}_{\mathcal{A}}(n)$ is not negligible. Then, based on $\mathsf{Adv}^{\text{sc}}_{\mathcal{A}}(n)$ of the adversary $\mathcal{A}$, we can construct another adversary $\mathcal{B}$ to solve a DDH problem with non-negligible advantage [25]. Scenario (iii) indicates two properties: an adversary has successfully forged a signature, and the forged signature and the original signature can be linked to one identity. These properties violate the unforgeability and signature traceability of a fair blind signature scheme, which has been proved secure by Abe [4]. For Scenario (iv), if an adversary can successfully hide the invoked transaction, that indicates the transaction does not eventually appear in the ledger. However, it contradicts our assumption that the blockchain meets the *Liveness* property, which requires that the submitted valid transactions will eventually be included in the ledger (see Assumption 2 in Section 2.2).

5.1.7 Example Applications

Our scheme may be used for privacy medical record protection, specifically for unrestricted research purposes. A medical record is supposed to be very sensitive in cases such as HIV and sexually transmitted infections. A hospital might share the medical record with a research institution without patients' permission, thereby causing information leakage. Our mechanism allows the hospital to show the patient records without knowing patients' real identities so that the privacy of the patient is protected. In the case of a family genetic disorder, patients may disclose their identities to the research institution by invoking the PPSC. This invocation represented a transaction that can be publicly traced to prevent misuse.

5.1.8 Conclusion

Anonymity credentials and anonymity revocation were proposed several decades ago, but they have not yet gained significant adoption. Potential obstacles are the lack of auditability and neutrality for the revocation process. In this chapter, we proposed a blockchain-powered traceable anonymous credential framework. Our approach allows the issuer to blindly issue a credential, then leverages a PPSC that acts as a revelator to trace the credential. Importantly, all these tracing activities are auditable due to the immutable smart contract calling records provided by the public ledger. The auditability and neutrality guaranteed by the blockchain avoid misuse of tracing and potential collision problems to a great extent.

5.2 Accountable Decryption System

5.2.1 Problem Statement

Accountable cryptographic protocol is increasingly crucial in sensitive personal data protection. We focus on the following scenario. Law enforcement or intelligence agencies may demand access to personal encrypted data held by service providers, and sometimes even require access to the communication metadata that is closely related to sensitive information of individuals. In most cases, a granted warrant is needed from a legal authority. However, data owners have no way to know when and how law enforcement collects and accesses their sensitive data. In particular, abuses of granted warrants of decryption may easily happen since overseers cannot verify whether practical investigation activities match the scope permitted in the document. Therefore, the accountability mechanism is a critical *after-the-fact* remediation technique to deter investigators since it provides instant evidence to detect malicious or deviant behaviours, which increases the transparency of warrant execution.

However, achieving accountability is tricky, and it requires additional roles involved. Investigators cannot autonomously convince others of the accountability of their actions. They need to resort to one or more neutral trusted parties, usually named *judge(s)*, to audit their actions. More specifically, an accountability mechanism requires each investigator to generate evidence on their warrant executions. A judge then examines this evidence to detect dishonest behaviours or declare the examined participant compliant. This approach relies heavily on the faithful cooperation of the judge and the investigator, as a malicious judge or dishonest investigator may undermine the accountability mechanism. If the investigator rejects to cooperate with the judge to provide the required evidence, or if the judge themselves examine fake evidence or apply the wrong examination procedure, outsides cannot audit investigators' decryption actions. In this chapter, we generalize the above example as a standard case in which an investigator obtains an order from a court, and his access of users' data needs to be audited by the judge. The discussed challenges lead to the following research question:

Is it possible to design an accountability mechanism guaranteeing that (1) the judge honestly checks the evidence; (2) the investigator does not refuse to provide the evidence trail of their actions?

Based on the previous discussion, the answer would intuitively be “NO”. Firstly, it is difficult to guarantee that a judge will always be secure and reliable. Even if the judge claims to be neutral, he faces the threat of being attacked or provided with misleading evidence. Once the judge is compromised, the accountability mechanism fails as it cannot be applied. Undoubtedly, multiple judges may mitigate such concerns, but the judge collusion issue cannot be effortlessly overcome. Secondly, asking the investigator to neutrally create a piece of honest evidence also confronts difficulties. The isolated local execution environment makes it potentially easy and profitable for the investigator to generate fake evidence while incurring a low risk of being detected. Several proposals [5, 57] employed certain trusted hardware to aid the evidence generation. Intuitively, physical hardware is more secure and reliable since the evidence logic and its measurements are hardcoded in non-volatile storage. However, the risk of compromised hardware still exists [127].

In this chapter, we propose *Fialka*, a novel transaction-triggering accountability framework using a privacy-preserving smart contract (PPSC) to make investigators accountable for executing decryption calls. Our framework prevents the decryption queries evidence from being maliciously generated (e.g., hidden) while guaranteeing the authenticity of the evidence. More precisely, *Fialka* combines PPSC with an IND-CCA secure public key encryption (PKE) scheme [122] at the protocol level to construct an accountability mechanism. PPSC cryptographically hides a secret random number used as an additional decryption key, where external investigators have to interact with PPSC for the execution of the decryption warrant. The secret key will be extracted by invoking decryption-related smart contracts, which consequently generate transaction-based evidence as an on-chain record. After that, another smart contract plays the role of the judge, who transparently checks the transaction to decide whether the decryption is legal in a specific setting. Accountability is thereby achieved. Additionally, our framework inherits the benefit of high availability from the underlying blockchain protocol.

5.2.2 Related work

The smart contract-based accountability approach has been studied comprehensively recently. Xu *et al.* [214] proposed a remotely decentralized data auditing scheme for network storage service, where accountability is achieved by involving a smart contract as a third-party auditor to notarize the integrity of outsourced data. Azaria *et al.* proposed MedRec [10], in which an Ethereum [213] smart contract is used as a meta-data agent to manage the permission of data usage, making patients' choices accountable. Neisse *et al.* [161] proposed a blockchain-based framework for data accountability and provenance tracking. However, a pure smart contract cannot provide a complete accountable protocol since it cannot guarantee the authenticity of the input (i.e., the submitted evidence). In other words, even if the smart contract is neutral and trustful, a client may provide fake evidence to the smart contract without being detected.

Several solutions have been proposed to ensure the authenticity of the submitted evidence. Among them, equipping entities with secure hardware devices [5, 127, 180] is an attractive approach. Alder *et al.* [5] employed Intel SGX [61] to produce a verifiable measurement of the resource usage in each function invocation. Luo *et al.* [145] applied SGX with blockchain to a data sharing scheme, where the decryption process also relied on the confidentiality of secure hardware devices. The hardware-based approach is intuitively reliable and robust since trusted hardware devices cannot change evidence generation rules once loaded. However, security cannot be guaranteed when adversaries successfully attack the hardware. The approach using multiple hardware may mitigate such security concerns to a certain extent. Unfortunately, the efficiency issue and incentives issue cannot be easily overcome. Another promising approach is directly employing the protocol execution result as evidence, such as using the ciphertext and the private key as evidence. A typical example is accountable-authority identity-based encryption [96, 99, 132], where a judge can decide whether a PKG is malicious by showing cryptographic proofs that contain the decryption key. However, such an approach lacks practicality.

5.2.3 General Construction

5.2.3.1 System Overview

Our system consists of four entities (see Fig 5.5.a): *common users (sender/receiver)*, *investigator*, *key management smart contract (PPSC-KM)*, and *auditor smart contract (PPSC-AD)*. PPSC-KM is used to manage investigators' decryption keys. PPSC-AD is employed as a "judge" to decide whether the event of the investigator's decryption is conducted under the court-issued order.

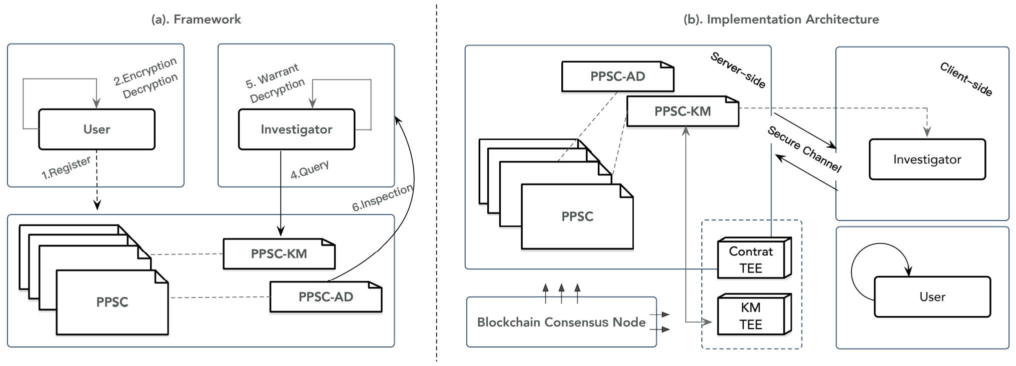


Figure 5.5: Fialka system framework (a) and architecture (b)

A detailed workflow is shown as follows. The sender encrypts messages with a random number, which is hidden in PPSC-KM, and then it sends the encrypted message to the receiver. The receiver decrypts the ciphertext as normal. Meanwhile, the investigator who obtained a court-issued order decrypts the ciphertext by fetching the random number from PPSC-KM. When a query is sent to PPSC-KM, the actions will be recorded through a transaction as evidence. Next, PPSC-AD will check evidence to report malicious decryption. In our protocol, PPSC-KM and PPSC-AD are, respectively, abbreviated as $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$ for simplicity. Formally, we provide the general construction as follows.

Setup $(pms, \widehat{\mathsf{c}}_{\mathsf{km}}, \widehat{\mathsf{c}}_{\mathsf{ad}}) \leftarrow \mathsf{Setup}(1^\lambda, codes)$. The algorithm takes as input a security parameter λ and binary codes $codes$, and returns public parameters pms and two contracts $\widehat{\mathsf{c}}_{\mathsf{km}}$, $\widehat{\mathsf{c}}_{\mathsf{ad}}$.

Key Generation $(pk, sk, tk) \leftarrow \mathsf{KeyGen}(pms)$. The algorithm takes as input pms, and returns the receiver's key pair (pk, sk), and a secret tag key tk. Then, it sends tk and accountability policies $\breve{P}$ to added to $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$, respectively.

Encryption $ct \leftarrow \mathsf{Encrypt}(tk, pk, m)$. This algorithm takes as input tk, pk, and a message m, and then returns a ciphertext ct.

State Read (Warrant Decryption) $(m, \mathsf{Tx}) \overset{\mathbb{B}}{\leftarrow} \mathsf{WDecrypt}(r, r_1, \mathsf{s}, ct)$. This algorithm takes as input a random number r, r_1, and s (including tk), calls the algorithm **Transfer** described in Section 4.3.1, and outputs $m \in \mathcal{M}$.

Decryption $m \leftarrow \mathsf{Decrypt}(sk, ct)$. The algorithm takes as input sk, ct, and returns the plain message $m \in \mathcal{M}$.

Inspection $true/false \overset{\mathbb{B}}{\leftarrow} \mathsf{Inspect}(\mathsf{Tx}, \breve{P})$. This algorithm takes as input accountability policies $\breve{P}$ and Tx, and returns the inspection result. The result *true* indicates that the authorized decryption is legitimately executed under the warrant and vice versa.

The procedure of **Decryption** represents normal decryption run by offline users, whereas **Warrant Decryption** is run by investigators, who are forced to leave evidence each time of decryption. Meanwhile, the access control conditions in $\widehat{\mathsf{c}}_{\mathsf{km}}$ and the accountability policies in $\widehat{\mathsf{c}}_{\mathsf{ad}}$ are set as the same. We notice that the logic of **Warrant Decryption** might be confusing: the $\widehat{\mathsf{c}}_{\mathsf{km}}$ has defined the access control conditions for investigators. Is accountability necessary for investigators' decryption? We clarify that access control and accountability in our system play distinct roles. The access control condition in $\widehat{\mathsf{c}}_{\mathsf{km}}$ is similar to an order issued by the court, which describes actions that an investigator should perform but has not done yet, whereas the accountability policies in $\widehat{\mathsf{c}}_{\mathsf{ad}}$ are responsible for checking the actions an investigator has completed (e.g., whether an investigator has executed the decryption under a warrant). We define *malicious decryption* as: the investigator's decryption does not match the actions permitted in issued orders.

5.2.3.2 Security Definitions

Our *Fialka* system is denoted by Π, and above algorithms are abbreviated as: Set, Gen, Reg, Enc, Dec, WDec, and Insp, respectively. We assume that an investigator has already obtained a warrant from a court, and his access to users' plaintext needs to be audited by the judge. Inspired by [129], the investigator should obtain fair treatment, neither being framed for the legitimate investigation nor being escaped from the punishment for wrongdoings. We capture two properties with respect to accountability: *fairness* and *completeness*.

Fairness. This property prevents the judge from framing honest investigators. An honest investigator should follow pre-defined policies and return *true*. We consider an adversary $\mathcal{A}$ who imitates an honest investigator, uses his identity, creates perjuries, and then maliciously executes the warrant to attempt to frame such an investigator.

Definition 8 (Fairness)**.** *Fialka satisfies fairness if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $adv_{\mathcal{A},\Pi}^{\partial_{fair}}(\lambda) < negl(\lambda)$ where $adv_{\mathcal{A},\Pi}^{\partial_{fair}}(\lambda)$ is the advantage of $\mathcal{A}$ wins the game ∂_{fair} defined as,*

- **Initialization***. The system configures the parameters as normal, and creates $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$ by running the algorithm Set. Then, $\mathcal{C}$ generates the secret key tk by running the algorithm Gen. Next, $\mathcal{C}$ registers tk and decryption policies $\breve{P}$ to $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$, respectively.

- **Actions***. At each round, the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ execute the following algorithms. (1) $\mathcal{A}$ generates the key pair (sk_a, pk_a) by running the algorithm Gen. (2) $\mathcal{C}$ inputs the public key pk_a, message m, a random numbers r and a secret key tk, and then obtains the ciphertext ct by running the algorithm Enc. (3) $\mathcal{C}$ runs the algorithm **Transfer**, and then returns r_2 and Tx to $\mathcal{A}$. (4) $\mathcal{A}$ inputs r_2, the ciphertext ct, and outputs the message m by running the algorithm WDec. (5) $\widehat{\mathsf{c}}_{\mathsf{ad}}$ executes the algorithm Insp with the input Tx, and return the inspection result.

- **Challenge**. Assume that $\mathcal{A}$ executes above actions at most for l times, and obtains a set $\mathcal{T} = \{\mathsf{Tx}_0, \mathsf{Tx}_1, ..., \mathsf{Tx}_l\}$. $\mathcal{A}$ wins if $\mathcal{A}$ generates a transaction $\mathsf{Tx}^\star$ satisfying the conditions: $false \leftarrow \mathsf{Insp}(\mathsf{Tx}^\star, \breve{P}) \wedge \mathsf{Tx}^\star \notin \mathcal{T}$.

Completeness. This property guarantees that the judge always punishes users who misbehave. To define *completeness*, we consider an adversary $\mathcal{A}$ aims to evade the responsibility of illegally executing the authorized decryption; he has done decryption that does not match the actions permitted in issued orders.

Definition 9 (Completeness)**.** *Fialka satisfies completeness, if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl(\lambda)$ such that $adv_{\mathcal{A},\Pi}^{\eth_{comp}}(\lambda) < negl(\lambda)$, where $adv_{\mathcal{A},\Pi}^{\eth_{comp}}(\lambda)$ is the advantage of $\mathcal{A}$ wins $\eth_{comp}$ defined as,*

- **Initialization** and **Actions**. The steps are kept the same, with the fairness game labelled with ($\star$).

- **Challenge**. Assume that $\mathcal{A}$ executes the above action at most for l times, and then obtains a set of ciphertext-transaction tuple $\{\mathcal{C}, \mathcal{T}\} = \{(ct_0, \mathsf{Tx}_0), (ct_1, \mathsf{Tx}_1), ..., (ct_l, \mathsf{Tx}_l)\}$. $\mathcal{A}$ wins if $\mathcal{A}$ successfully generates a new tuple $(ct^\star, \mathsf{Tx}^\star)$ that satisfying the conditions: $true \leftarrow \mathsf{Insp}(\mathsf{Tx}^\star, \breve{P})$ $\wedge\ \mathsf{WDec}(r, \mathsf{s}, ct^\star) = m^\star \wedge (ct^\star, \mathsf{Tx}^\star) \notin \{\mathcal{C}, \mathcal{T}\}$.

5.2.4 Concrete Instantiation

In this section, we present an instantiation of *Fialka* based on Kiltz's PKE protocol [122] and the Oasis Devnet [1, 54]. Kiltz's PKE is an efficient and IND-CCA secure scheme with a tight security reduction, while Oasis Devnet is an SGX-backed PPSC platform with rigorous security proof under the Universal Composability (UC) framework [42]. In this instance, PPSC-KM manages a secret random number as the investigator's decryption key and its access permission through the SGX enclave, and PPSC-AD audits transactions and then reports the investigator's malicious decryption. Specifically, a decryption key used for investigation is loaded in PPSC-KM and hidden in an enclave, which forces the outside investigator to fetch it, and further leaves transaction-based evidence that will be audited by PPSC-AD. Note that SGX-based PPSC is an example of hiding the secret key inside the hardware, and other approaches can also achieve the same goal, such as cryptographically hiding the secret key by ZKP. Our framework is compatible with various aforementioned PPSC technologies [36, 110, 126, 182, 226] (see Section 4.2.1). Also, our construction can easily be extended to other accountable PKE protocols without significant modifications.

Setup $(pms, \widehat{\mathsf{c}}_{\mathsf{km}}, \widehat{\mathsf{c}}_{\mathsf{ad}}) \leftarrow \mathsf{Setup}(1^\lambda, codes)$. The algorithm takes as input a security parameter λ, and returns public parameters including the multiplicative cyclic group $\mathbb{G}$ with prime order p. Then, it chooses two collision-resistant hash functions

$\mathcal{H}_1 : \{0,1\}^\star \to \mathbb{Z}_p$ and $\mathcal{H}_2 : \mathbb{G} \times \mathbb{G} \to \mathbb{Z}_p$. Next, it takes as input contract binary *codes*, and calls the algorithm **Deploy** (defined in Section 4.3.1), and finally returns two contracts $\widehat{\mathsf{c}}_{\mathsf{km}}$ and $\widehat{\mathsf{c}}_{\mathsf{ad}}$.

Key Generation $(pk, sk, tk) \leftarrow \mathsf{KeyGen}(pms)$. The algorithm is run by the sender and receiver. The receiver runs the algorithm to generate his key pair (pk, sk), and the sender runs the algorithm to obtain a secret tag key tk.

$$\begin{aligned} tk, x_1, x_2, y_1, y_2 \leftarrow \mathbb{Z}_p^*; \\ \text{Choose } (g_1, g_2, z) \in \mathbb{G},\ \text{satisfying } g_1^{x_1} = g_2^{x_2} = z; \\ u_1 \leftarrow g_1^{y_1}; u_2 \leftarrow g_2^{y_2}; pk \leftarrow (\mathbb{G}, p, g_1, g_2, z, u_1, u_2); sk \leftarrow (x_1, x_2, y_1, y_2). \end{aligned}$$

Next, the tag key tk is registered into $\widehat{\mathsf{c}}_{\mathsf{km}}$. The policies $\breve{P}$ are added to $\widehat{\mathsf{c}}_{\mathsf{ad}}$ by the means of external message calls (see Section 4.3.1). The privacy of tk and s are protected by the SGX enclave. More details can be found in our implementation.

Encryption $ct \leftarrow \mathsf{Encrypt}(tk, pk, m)$. This algorithm is run by the sender. It takes as input tk, pk, and a message m, returns a ciphertext ct.

$$\begin{aligned} pk = (\mathbb{G}, p, g_1, g_2, z, u_1, u_2);\ r_1, r \leftarrow \mathbb{Z}_p; \\ r_2 \leftarrow \mathsf{H}_1(tk|r); C_1 \leftarrow g_1^{r1}; C_2 \leftarrow g_2^{r2};\ \tau \leftarrow \mathsf{H}_2(C_1, C_2);\ V \leftarrow r_1; \\ D_1 \leftarrow z^{\tau r_1} u_1^{r_1}; D_2 \leftarrow z^{\tau r_2} u_2^{r_2}; K \leftarrow z^{r_1 + r_2}; E \leftarrow mK; \\ ct \leftarrow (C_1, C_2, D_1, D_2, E, V). \end{aligned}$$

State Read **(Warrant Decryption)** $(m, \mathsf{Tx}) \overset{\mathbb{B}}{\leftarrow} \mathsf{WDecrypt}(r, r_1, \mathsf{s}, ct)$. This algorithm is run by the investigator. It takes as input r, r_1 and the private state s (including tk), and then calls the **Transfer** algorithm to execute the function $r_2 \leftarrow \mathsf{H}_1(tk|r)$ in an isolated environment provided by the SGX. This calling progress is represented in the form of a transaction Tx.

$$\begin{aligned} \textit{Parse ct as}(C_1, C_2, D_1, D_2, E, V); \\ r_2, \mathsf{Tx} \leftarrow \mathbf{Transfer}(\mathsf{s}, r); \\ K'' = z^{r_1 + r_2};\ m \leftarrow E(K'')^{-1}. \end{aligned}$$

Decryption $m \leftarrow \mathsf{Decrypt}(sk, ct)$. This algorithm is run by the receiver. It takes as input the receiver's secret key sk, the ciphertext ct, and returns $m \in \mathcal{M}$.

$$\begin{aligned} Parse\ ct\ as(C_1, C_2, D_1, D_2, E, V); \\ s_1, s_2 \leftarrow \mathbb{Z}_p;\ \tau \leftarrow \mathsf{H}_2(C_1, C_2); \\ K' \leftarrow \frac{C_1^{x_1+s_1(\tau x_1+y_1)} C_2^{x_2+s_2(\tau x_2+y_2)}}{D_1^{s_1} D_2^{s_2}}; \\ m \leftarrow E(K')^{-1}. \end{aligned}$$

Inspection $true/false \overset{\mathbb{B}}{\leftarrow} \mathsf{Inspect}(\mathsf{Tx}, \breve{P})$. This algorithm is run by $\widehat{\mathsf{c}}_{\mathsf{ad}}$. It takes as input $\breve{P}$ and Tx, and returns inspection result $true/false$. The $true$ indicates the warrant decryption satisfying policies and vice versa.

Here, the **correctness** of our construction is easy to check as we have

$$\begin{aligned} K' = & \frac{C_1^{x_1+s_1(\tau x_1+y_1)} C_2^{x_2+s_2(\tau x_2+y_2)}}{D_1^{s_1} D_2^{s_2}} = C_1^{x_1} C_2^{x_2} \left(\frac{C_1^{\tau x_1+y_1}}{z^{tr_1} u_1^{r_1}}\right)^{s_1} \left(\frac{C_2^{\tau x_2+y_2}}{z^{\tau r_2} u_2^{r_2}}\right)^{s_2} \\ = & C_1^{x_1} C_2^{x_2} \left(\frac{g_1^{r_1(\tau x_1+y_1)}}{g_1^{r_1(x_1\tau+y_1)}}\right)^{s_1} \left(\frac{g_2^{r_2(\tau x_2+y_2)}}{g_2^{r_2(x_2\tau+y_2)}}\right)^{s_2} = g_1^{r_1 x_1} g_2^{r_2 x_2}. \end{aligned}$$

Note that the random numbers s_1 and s_2 are used for implicitly testing if the ciphertext is consistent with tag τ [122]. We see that $K = z^{r_1+r_2} = g_1^{x_1 r_1 + x_1 r_2} = g_1^{x_1 r_1} g_2^{x_2 r_2}$. Then, we observe that $K = K' = K''$. Thus, both the receiver and investigator can obtain the message m by

$$\mathsf{Dec}(sk, ct) = E(K)^{-1} = mK(K)^{-1} = m.$$

5.2.5 Security Proof

Theorem 1 (Fairness)**.** *Assume that the SGX-based PPSC is secure, our construction Fialka satisfies the property of fairness.*

Proof: Suppose that there exists an adversary $\mathcal{A}$ who wins the fairness game $\eth_{\mathsf{fair}}$ with non-negligible advantage. Then, we transform an adversary $\mathcal{A}$ against *fairness* into adversaries against PPSC security. Next, we describe a sequence of games to finish the proof.

Lemma 1 (SGX-based PPSC [54, 135])**.** *Our SGX-based platform is a secure instantiation of PPSC whose protocols match the ideal functionality in the UC framework. More details can be found in [54].*

Game $\eth_0$. This is an unmodified game. Trivially, the winning probability of this game equals the advantage of $\mathcal{A}$ against the fairness game, namely, $adv_{\mathcal{A},\Pi}^{\eth_{\text{fair}}}(\lambda)$.

Game $\eth_1$. In this game, when $\mathcal{A}$ calls $\mathcal{C}$, we disallow $\mathcal{C}$ to call contract $\widehat{\mathsf{c}}_{\mathsf{km}}$.

Game $\eth_2$. In this game, when $\mathcal{A}$ calls $\mathcal{C}$, the transaction-based evidence is not allowed to be given to $\widehat{\mathsf{c}}_{\mathsf{ad}}$. Instead, the evidence is randomly selected for auditing.

Obviously, the winning probability of the game $\eth_2$, denoted as $adv_{\mathcal{A},\Pi}^{\eth_2}(\lambda)$, is negligible, since the transaction-based evidence is randomly selected. Next, to find out the differences between these games, we define the following events.

- ◇ $\mathbb{E}[a1]$: forging evidence. The event $\mathbb{E}[a1]$ implies that the adversary $\mathcal{B}_1$ forges a valid transaction Tx^* without update $\widehat{\mathsf{c}}_{\mathsf{km}}$, denoted as $\neg$**Transfer**.

$$\left.\begin{array}{r} r_2, \mathsf{Tx}^* \overset{\mathbb{B}}{\leftarrow} \neg\mathbf{Transfer}(\mathsf{s}, r) \wedge \\ \mathsf{WDec}\left(r, r_1, \mathsf{s}, \mathsf{Enc}(tk, pk, m)\right) = m \wedge \\ false \overset{\mathbb{B}}{\leftarrow} \mathsf{Insp}(\mathsf{Tx}^*, \breve{P}) \end{array}\right] \Rightarrow \mathbb{E}[a1].$$

- ◇ $\mathbb{E}[a2]$: forging an inspection result. The event $\mathbb{E}[a2]$ implies that the adversary $\mathcal{B}_2$ forges an inspection result, where the originally "true" in the algorithm Insp is modified to be "false".

$$\left.\begin{array}{r} r_2, \mathsf{Tx} \overset{\mathbb{B}}{\leftarrow} \mathbf{Transfer}(\mathsf{s}, r) \wedge \\ \mathsf{WDec}\left(r, r_1, \mathsf{s}, \mathsf{Enc}(tk, pk, m)\right) = m \wedge \\ false \overset{\mathbb{B}}{\leftarrow} \mathsf{Insp}(\mathsf{Tx}, \breve{P}) \end{array}\right] \Rightarrow \mathbb{E}[a2].$$

Game $\eth_0 \approx$ **Game** $\eth_1$. The winning condition for $\eth_0$ is equal to the winning condition for $\eth_1$ if and only if the event $\mathbb{E}[a1]$ does not happen. The probability of $\mathbb{E}[a1]$ happening is identical to the advantage of breaking the promise of transaction-unforgeability (see the security property defined in Section 4.3.2). Thus, we have

$$|\Pr[\eth_0] - \Pr[\eth_1]| = \Pr[\mathbb{E}[a1]] = adv^{\eth_{\text{unforg}}}_{\mathcal{B}_1,\Pi}(\lambda).$$

Game $\eth_1 \approx$ **Game** $\eth_2$. The winning condition for $\eth_1$ is equal to the winning condition for $\eth_2$ if and only if the event $\mathbb{E}[a2]$ does not happen. We consider the possibility of $\mathbb{E}[a2]$, and it is identical to the advantage of breaking the promise of state-consistency (see the security property defined in Section 4.3.2). Thus, we obtain

$$|\Pr[\eth_1] - \Pr[\eth_2]| = \Pr[\mathbb{E}[a2]] = adv^{\eth_{\text{cons}}}_{\mathcal{B}_2,\Pi}(\lambda).$$

Putting everything together, we conclude that

$$\begin{aligned} adv^{\eth_{\text{fair}}}_{\mathcal{A},\Pi}(\lambda) \leq & \Pr[\mathbb{E}[a1]] + \Pr[\mathbb{E}[a2]] + adv^{\eth_2}_{\mathcal{B},\Pi}(\lambda) \\ \leq & adv^{\eth_{\text{unforg}}}_{\mathcal{B}_1,\Pi}(\lambda) + adv^{\eth_{\text{cons}}}_{\mathcal{B}_2,\Pi}(\lambda) + adv^{\eth_2}_{\mathcal{A},\Pi}(\lambda) \leq negl(\lambda). \end{aligned}$$

□

Theorem 2 (Completeness)**.** *Assume that SGX-based PPSC is secure and Kiltz's full PKE scheme [122] is secure against chosen-ciphertext attacks, Fialka satisfies completeness.*

Proof: Suppose that there exists an adversary $\mathcal{A}$ who wins the completeness game $\eth_{\text{comp}}$ with non-negligible probability. Then, we transform an adversary $\mathcal{A}$ against *completeness* into adversaries against the PPSC security and IND-CCA security of Kiltz's PKE scheme. We describe a sequence of games to conduct the proof.

Game $\eth_0$. This is the unmodified completeness game. The winning probability equals the advantage of $\mathcal{A}$ against *completeness* game, namely, $adv^{\eth_{\text{comp}}}_{\mathcal{A},\Pi}(\lambda)$.

Game $\eth_1$. In this game, when the adversary calls $\mathcal{C}$, we disallow contract $\widehat{\mathsf{c}}_{\mathsf{ad}}$ to execute the algorithm Insp, and then $\widehat{\mathsf{c}}_{\mathsf{ad}}$ outputs *true* to the adversary.

Game $\Game_2$. In this game, we disallow $\mathcal{A}$ calls $\mathcal{C}$, and thus **Transfer** in $\widehat{\mathsf{c}}_{\mathsf{km}}$ cannot be executed, indicating $\mathcal{A}$ cannot obtain secret key from blockchain.

Clearly, without querying the smart contract, the adversary's advantage of winning $\Game_2$ equals the advantage of breaking the CCA security of PKE. The adversary against security of Kiltz's PKE scheme $adv^{\Game_{\mathrm{CCA}}}_{\mathcal{B},\Pi}(\lambda)$ is negligible, and the proof is given in [122]. To find out the difference between these games, we define the events: (1) $\mathbb{E}[b1]$: blocking the transaction-based evidence. The adversary $\mathcal{B}_1$ fetches the key from the blockchain and successfully hides the transaction $\mathsf{Tx}^{\star}$ that is used for validation in the algorithm Insp. (2) $\mathbb{E}[b2]$: forging an inspection result. The adversary $\mathcal{B}_2$ forges an inspection result by executing $\neg\mathsf{Insp}$, where $\neg\mathsf{Insp}$ means the malicious behaviours of inspection, and it modifies the $false$ result as $true$. (3)$\mathbb{E}[b3]$: breaking the security of PPSC. The adversary $\mathcal{B}_3$ obtains a valid private key without invoking the blockchain.

Game $\Game_0 \approx$ **Game** $\Game_1$. The winning conditions for $\Game_0$ equals the winning conditions for $\Game_1$ if neither event $\mathbb{E}[b1]$ nor event $\mathbb{E}[b2]$ happen. Thus, we have $|\Pr[\Game_0]-\Pr[\Game_1]| = \Pr[\mathbb{E}[b1]]+\Pr[\mathbb{E}[b2]]$. We then consider the happening probabilities of $\mathbb{E}[b1]$ and $\mathbb{E}[b2]$. The happening of $\mathbb{E}[b1]$ implies that the adversary $\mathcal{B}_1$ hides the transaction evidence, which contradicts the assumption of transparency properties. Thus, the winning advantage of $\mathbb{E}[b1]$ is identical to breaking the promise of transaction-transparency (see the security property defined in Section 4.3.2). If the event $\mathbb{E}[b2]$ happens, indicating that the adversary $\mathcal{B}_2$ breaks the state-consistency of PPSC, the possibility is identical to the advantage of breaking the promise of state-consistency (see the security property defined in Section 4.3.2). Thus, we have $\Pr[\mathbb{E}[b1]] = adv^{\Game_{\mathrm{tran}}}_{\mathcal{B}_1,\Pi}(\lambda)$ and $\Pr[\mathbb{E}[b2]] = adv^{\Game_{\mathrm{cons}}}_{\mathcal{B}_2,\Pi}(\lambda)$.

Game $\Game_1 \approx$ **Game** $\Game_2$. The winning condition for $\Game_1$ is equal to the winning condition for $\Game_2$ if and only if the event $\mathbb{E}[b3]$ does not happen. The possibility of $\mathbb{E}[b3]$ is identical to the advantages of breaking the promise of state-privacy (see the security property defined in Section 4.3.2). Thus, $|\Pr[\Game_1] - \Pr[\Game_2]| = \Pr[\mathbb{E}[b3]] = adv^{\Game_{\mathrm{privacy}}}_{\mathcal{B}_3,\Pi}(\lambda)$.

Combining everything together, we obtain that

$$\begin{aligned} adv_{\mathcal{A},\Pi}^{\eth\text{comp}}(\lambda) \leq & \Pr[\mathbb{E}[b1]] + \Pr[\mathbb{E}[b2]] + \Pr[\mathbb{E}[b3]] + adv_{\mathcal{B},\Pi}^{\eth\text{no-query}}(\lambda) \\ \leq & adv_{\mathcal{B}_1,\Pi}^{\eth\text{tran}}(\lambda) + adv_{\mathcal{B}_2,\Pi}^{\eth\text{cons}}(\lambda) + adv_{\mathcal{B}_3,\Pi}^{\eth\text{privacy}}(\lambda) + adv_{\mathcal{B},\Pi}^{\eth\text{CCA}}(\lambda) \leq negl(\lambda). \end{aligned}$$

□

5.2.6 Implementation

In this section, we discuss the implementation[1] of our instantiation based on the SGX-based PPSC platform Oasis Devnet [1, 54] (version 2.0). Our implementation (see Figure 5.5.b) has two components: *client-side* and *server-side*. The client side is run by the sender, receiver, and investigator, while the server side is run by the PPSC platform. The client-side covers four algorithms: Set, Gen, Enc, Dec. They are implemented by 1000+ lines of JavaScript codes in total, containing the packages of `client` and `client-connector`. The `client` implements basic operations executed by end-users at local, while `client-connector` builds a bridge between the client-side and the server-side. The server-side consists of two pieces of PPSCs: PPSC-KM and PPSC-AD. PPSC-KM covers the algorithms CGen, Reg and Trans[2], while PPSC-AD includes the algorithm Insp. Both are implemented in Rust. PPSC-KM protects private decryption keys by using the enclave technology from Intel SGX [61], while PPSC-AD determines whether the decryption is legal or not by checking the security policies.

To be specific, after a successful deployment of the contract PPSC-KM and PPSC-AD, the evidence inspection algorithm Insp and the investigator's key generation algorithm (by revoking Trans) as well as their access conditions, will be compiled as the binary codes and replicated to enclaves [61] in SGX-powered blockchain nodes. Then, an encrypted contract state containing the investigator's key $\mathsf{H}_1(tk|r)$ reaches an agreement across distributed blockchain nodes. After that, to obtain the key from PPSC-KM, two requirements must be fulfilled: (1) A transaction with the input satisfying access conditions should be provided; (2) An encrypted and authenticated channel connected to enclaves should be established (after a

[1]A demo site and reference source code are accessible at `http://www.fialka.top`.

[2]Trans (**Transfer** algorithm) calculates the investigator's key, and it belongs to WDec.

successful attestation [1, 54, 61]). Then, an invocation in the form of a transaction will remain visible and immutable. Each entity can see/witness the progress of obtaining the investigator's key, but no entity, except the contract caller, knows the exact output (key) of the smart contract. Subsequently, PPSC-AD audits transactions through an internal query to detect suspicious activities. Essentially, the privileges of the Trans algorithm are protected and managed at a CPU-level by Intel SGX. Only designated investigators should be allowed to access this secret key. We also notice that our implementation only provides one-off auditing since it can only trace records when the first time an investigator extracts the secret key. Our implementation provides a prototype to demonstrate feasibility.

5.2.7 Evaluation

We first provide the performance evaluation on average CPU-time, representing the consumed time since the operation starts. The evaluation contains all the algorithms, and the testing environment is set as follows. The client-side runs on a Dell precision 3630 Tower with 16 GB of RAM and one 3.7GHz six-core i7-8700K processor running Ubuntu 18.04. The server-side runs on a blockchain node, which is provided by the Oasis SDK [1, 54].

Table 5.3: The average cpu-time, gas cost and latency evaluation

Operations	**CPU-time**/ms	**Cost**/gas	**Latency**/ms
Set	1.16	-	-
Gen	50.04	-	-
CGen†	0.0880	5129943	5683
Reg	0.0104	494553	3960
Enc	102.35	-	-
Dec	64.86	-	-
Trans	0.0325	342514	3643
Insp	0.0027	251971	2450

†: CGen means contract generation

CPU-time. The evaluation results illustrate several critical points. The offline operation Enc is the most time-consuming operation since the encryption covers seven exponentiations. The offline operation Dec takes approximately half the time of that in encryption because it processes four exponentiations. On the contrary, blockchain-related operations CGen, Reg, Trans and Insp take much less than offline operations since they do not have group mathematics computation. In particular, the operation Insp is the fastest operation, which indicates the efficiency of our accountability protocol. However, CPU-time is close to the testing environment, inefficient to convince that our framework is practical. Therefore, we provide further evaluations on *gas cost* and *latency* for real-world scenarios.

Gas cost. The gas cost measures the amount of computational effort that a blockchain takes to execute an algorithm. The gas cost evaluation includes the operations of CGen, Reg, Trans, and Insp. The operation CGen costs the most gas among all since the initial configuration of a smart contract has to be loaded. Fortunately, this bottleneck can be ignored because each contract is created only once and can be reused multiple times. The cost of Reg is relatively high since the public parameters are needed to store on smart contracts. The cost of Trans and Insp are relatively low due to simple online calculations, which indicates that our accountability protocol is financially feasible[3]. In real-world settings, different investigators can call the functions in the same PPSC for decryption and auditing simultaneously. To demonstrate the practicability of our system, we simulate a distributed environment by increasing the number of invocations from different investigators. In particular, we test the gas cost of Trans and of Insp with a maximum of 1000 simultaneous invocations. As shown in Figure 5.6, the outputs remain relatively stable under variations, and the average cost of Trans is approximately 340k while that of Insp is about 250k. It matches our intuitive expectation since the gas cost is theoretically independent of the number of investigators. Based on these results, our accountability framework is practically affordable and can be widely adopted.

Latency. Our latency test covers all blockchain-related operations including CGen, Reg, Trans and Insp. Among them, CGen is the most time-consuming, as the contract codes need to be compiled into the blockchain. The operation Reg also takes a long time because all parameters have to be configured into contracts. In

[3]Estimates on the real value of gas cost are omitted since the Oasis token has not been officially released at the time of writing.

contrast, the operations Trans and Insp are in low latency because they do not have sophisticated on-chain computations. We also provide a simulation by increasing the invocations in a distributed environment. Our simulation includes the two most frequently used functions in PPSC, namely Trans and Insp. As shown in Figure 5.6, the results turn out that the latency stably increases along with the growing number of invocations. Theoretically, numerous invocations will impose a heavy burden on the distributed network, which may even cause the network failure or transaction stuck. We set an upper bound of invoking transactions with 1000 users at the peak. The testing results confirm our expectations.

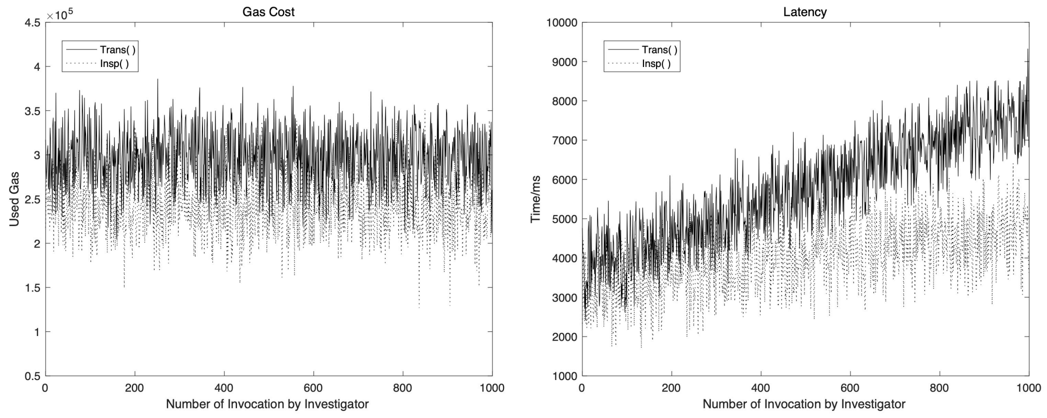


Figure 5.6: The gas and latency evaluation

5.2.8 Conclusion

In this chapter, we proposed *Fialka*, a novel transaction-triggering accountable decryption system based on PPSC. Our system utilized PPSC to trace and detect the decryption evidence, which makes warrant execution accountable. To the best of our knowledge, we presented the first PPSC-based accountability mechanism with formal definitions and proofs. The security analysis showed that our system holds accountability properties encompassing fairness and completeness. The implementation based on Oasis Devnet with the detailed evaluation indicated that our system is feasible and applicable.

Chapter 6

Blockchain System Enhancement

This chapter provides two blockchain algorithms that enhance the performance, scalability and usability of current blockchain systems.

6.1 A Weak Consensus Algorithm

6.1.1 Introduction

The consensus mechanism is a critical component in distributed systems, providing a powerful means of establishing agreement as to the network's current state. With the promotion of blockchain, consensus mechanisms obtain tremendous attention due to their influential roles in secure token transferring. Generally, two mainstream types of consensus algorithms are identified [14][203], namely, the classic Byzantine Fault Tolerant (BFT) protocols [45][46] and the newly proposed Nakamoto consensus (NC) [84] such as PoW [30, 158], PoS [119], PoA [70, 207], etc. However, blockchain systems adopting these algorithms suffer from low-performance issues due to massive communication or intensive computation. For example, Bitcoin requires competitive computations to decide the valid chain, whose rate is limited to 7 transactions per second (TPS) [158]. Such limitations greatly hurdle the widespread adoption in real scenarios. This leads us back to their core mechanisms.

BFT protocols have been proposed to achieve consensus even when some replicas (less than 1/3) are Byzantine faulty. BFT protocols require the negotiation process for final decisions. A typical system, PBFT [46], is illustrated in Figure 6.1.a. The leader sends a proposal to replicas, and replicas distribute their replies. Then, after receiving valid replies over the predefined threshold, a replica broadcasts his status (whether ready for the new state) to others. Once the received commit messages exceed the threshold, a decision is made. Time consumption in such a process is unpredictably unstable due to factors like network delay. Interactive communications consequently limit the performance of the BFT consensus and increase the communication overhead.

Nakamoto consensus, patterned after Bitcoin [158], has received remarkable attention due to its simplicity. NC does not need a closed committee. Instead, it allows all participants to get involved in the consensus process. NC protocols remove the interactive model and adopt a competition rule – *the longest chain wins.* As shown in Figure 6.1.b, blocks generated by miners are randomly attached to their ancestors. Only the chain with the most descendants survives, whereas other competitive sub-chains are abandoned. The finality is progressively achieved by letting blocks bury deep enough. Thus, conflict solving in NC seriously slows down the confirmation of blocks.

We observe that protocols based on these two types of consensus mechanisms follow the same principle that *only one block is deemed as confirmed at one round (equal block height in NC).* This greatly constrains the overall performance since the procedure of conflict solving, and total ordering serving for *strong consistency* costs much more time than expected. Such mechanisms hurdle their widespread adoptions [125][91], particularly, for some high performance required scenarios [211]. To mitigate such a limitation, we ask the following question,

> Is it possible to propose a consensus algorithm to improve performance by weakening the guarantee of consistency?

Intuitively, the answer should be "No". State consistency is the core property of the consensus mechanism. Strict consistency ensures that the distributed network reaches an agreement on the total order of transactions in the presence of fault maintainers and adversarial network delay. All distributed nodes have the same global view at each specific height. This guarantees that states are transited in an organized and managed way, supporting upper-layer establishments like smart contracts. Disordered transactions, on the contrary, indicate ambiguous states where users may feel confused when invoking the blockchain service. For example, Alice sends a transaction to Bob. If this transaction is stored in more than one block, Bob cannot know which position provides a valid transaction. However, in some scenarios, strict consistency is not firmly required, such as the blockchain-based certificate system. The main target of the certificate prover is to confirm that a certificate is indeed stored in the chain. The specific position of this certificate does not matter; even a duplicated storage of certificates is allowed. It should be noted that the partial consistency in several DAG-structure projects [11][216] is still sensitive to the position of transactions since their upper-layer applications, such as token transferring, are still based on a fixed sequence of transactions.

In this chapter, we propose a new consensus mechanism, called *weak consensus*, to fit the aforementioned scenarios. Our design weakens the guarantee of strict consistency and relaxes the property of persistence [84]. Weak consensus guarantees that the relative sequences of blocks in one individual chain remain consistent with that in the other chains. As illustrated in Figure 6.1.c, node B creates a serial of blocks $1, 2, 3, 4$. Our goal is to ensure that the sequence of $(B1 \rightarrow B2 \rightarrow B3 \rightarrow B4)$ can be correctly maintained across chains, no matter how many blocks (generated by other nodes) are inserted between them. Blocks in our model are required to receive replies from other nodes, saying that they have successfully stored the blocks. Whenever a block collects *commit* messages more than the threshold, it is deemed as confirmed. To demonstrate the robustness of our consensus, we formally define the properties *relative persistence* and *liveness*, inspired by [84]. Relative persistence focuses on the relationship between a predecessor and its successor, ensuring the correctness of the relative position. Liveness guarantees that all nodes would eventually agree on the relationship of blocks. Furthermore, we apply this algorithm to a blockchain system called *Sphinx*, with a full implementation.

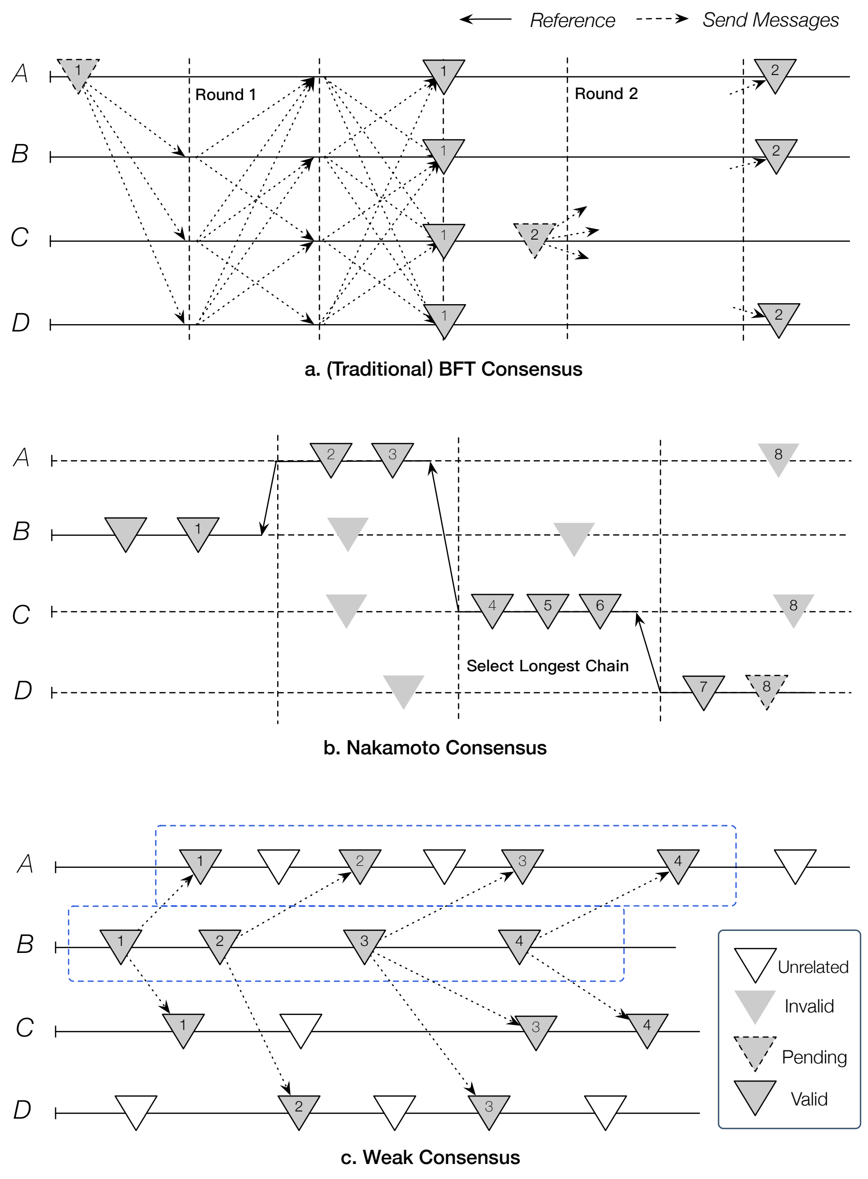


Figure 6.1: Consensus mechanisms diagram

6.1.2 Weak Consensus Algorithm

This section provides the security assumptions and the general construction of our consensus algorithm with corresponding security properties.

6.1.2.1 Notations

We denote the nodes in our protocol as N and identify each of them as $\{N_0, N_1, \dots, N_n\}$, where n is the index of committee members satisfying $n = 3f + 1$ (f represents the Byzantine nodes). Let i be a growing integer satisfying, $0 \leq i \leq r$ where r is the index of states and j be an integer satisfying $0 < j \leq n$. Assume that $\mathbb{M}$ is the message space, $\mathbb{S}$ is the state space and $\mathbb{R}$ is the reference space. $M \in \mathbb{M}$, is the message proposed by some node. PF is the proof of successful insertion of M. S represents a confirmed state satisfying $S \in \mathbb{S}$. $S^i_{N_j}$ is the i-th confirmed state in the node N_j, where $S^i_{N_j} \in \{\mathbb{S} | S^0_{N_1}, \dots, S^r_{N_1}; \dots; S^1_{N_n}, \dots, S^r_{N_n}\}$. These two parameters are used to locate a specific state in the network. $S^r_{\{N_0,\dots,N_n\}}$ refers to the states received from other nodes in current round r. $\Downarrow$ is the reference which indicates the relative positions between two states. Specifically, $\Downarrow^B_A$ is the reference pointing from B to A. It defines a *happens-before* relationship that A happens before B. More specifically, $\Downarrow^{S^y_{N_j}}_{S^x_{N_j}}$ means that $S^x_{N_j}$ is an ancestor of $S^y_{N_j}$ where $0 < x < y \leq r$. Further, $\Downarrow^{S^i_{N_j}}_{S^{i-1}_\star}$ represents a set of references including the edges from the state $S^i_{N_j}$ to states $S^{i-1}_{N_0}, \dots, S^{i-1}_{N_n}$ (*a.k.a.*, the out-degree edges of $S^i_{N_j}$). Correspondingly, $\Downarrow^{S^i_\star}_{S^{i-1}_{N_j}}$ contains all the edges from states $S^i_{N_0}, \dots, S^i_{N_n}$ to the state $S^{i-1}_{N_j}$ (*a.k.a.*, the in-degree edges of $S^{i-1}_{N_j}$).

6.1.2.2 Security Assumption

We assume that honest nodes will always conduct honest behaviours, where the messages sent to peers are correct. As for the underlying network, we follow the implicit assumption of a partial synchronous network. In particular, the network of honest nodes in our system is well-connected, and the communication channels between honest nodes are unobstructed. Messages from honest broadcasters may be delayed, but they will eventually arrive at others within the known maximum delay δ [171]. Our algorithm follows the basic design of classic BFT-style protocols, with the aim to tolerate one-third of Byzantine nodes. Specifically, we assume that there are $3f + 1$ nodes in total, and the number of participating nodes is fixed. It indicates that the dynamics of peer participation, or churn, are out of our consideration. Also, we assume that at least $2f$ of them perform honestly, where f is the number of Byzantine nodes.

6.1.2.3 Protocol Overview

The protocol is modelled as a state machine which is replicated across distributed nodes. Each node in the network maintains a message log containing the *accepted* message and the current state. Meanwhile, in our algorithm, a node must maintain the states received from other nodes. We present our protocol by following the description of PBFT [45][46][193]: the protocol proceeds in rounds, and each round has three phases, namely *Pre-Prepare*, *Prepare* and *Commit.* We provide the overview of our protocol as follows.

- **Pre-Prepare.** The primary node receives client requests and inserts such messages into the local chain. Then, the node creates a *Pre-Prepare* message to claim the relative position between two client messages. Subsequently, it broadcasts the signed message to peers.

- **Prepare.** The node receives the *Pre-Prepare* message and checks integrity, correctness, and validity. When the received *Pre-Prepare* message passes the verification procedure, the node updates his local-stored state and broadcasts the replied *Prepare* message to claim the correct relative position. Otherwise, the node aborts it.

- **Commit.** If any node receives a quorum $2f + 1$ of valid *Prepare* messages from other nodes (possibly including his own) within a specified time interval, this node confirms the proposed decision by broadcasting a replied *Commit* message. When a node collects more than $2f + 1$ *Commit* messages, this node transits the state and replies to clients with updated states.

Complementary mechanism. Our protocol aims to achieve an eventual confirmation of the relative relationship between two states. It is possible that the relative relationship cannot be committed due to the lack of $2f + 1$ matched *Com-*

Table 6.1: Comparison between PBFT algorithm and our algorithm

	PBFT algorithm [45]	**Our algorithm**
Request Stage	- A leader is required. A client sends a request to the primary node. If the primary node has changed/rotated, it will broadcast the request message to all replicas.	- No leader exists in our algorithm. Every node acts in similar behaviours. A client sends a request to a random node, where a request message is represented as the relative position.
Normal Case	- *Pre-Prepare:* The primary node puts the pending requests in total order and initiates agreement by sending *Pre-Prepare* message to all replicas. - *Prepare:* Replica acknowledges the receipt of a *Pre-Prepare* message by sending *Prepare* message to other replicas. - *Commit:* Replica acknowledges the reception of $2f$ *Prepare* message matching a valid pre-prepare by broadcasting the *Commit* message to peers.	**Every node** executes the following actions in parallel. - *Pre-Prepare:* The same algorithm with PBFT. - *Prepare:* The same algorithm with PBFT. - *Commit:* The same algorithm with PBFT.
View Change	- It ensures that the system can always proceed by allowing replicas to change the leader, so as to not wait indefinitely for a faulty primary.	- No need for the view change progress. Alternatively, the complementary mechanism was adapted to ensure the correct relative persistence to be held.
Garbage collection	- The checkpoint mechanism is used to ensure the safety condition to be held.	- The timeout mechanism is used to ensure that the liveness condition is held.

mit messages. Figure 6.2 provides an example to explain the flaws. We assume that there are three nodes, and the relative position of states $S^1_{N_2}$ and $S^2_{N_2}$, saying $\Downarrow^{S^2_{N_2}}_{S^1_{N_2}}$, in the node N_2 have already been confirmed by the node N_3. By the design of previous protocols, Sphinx achieves an agreement by receiving $2f+1$ replies. However, the node N_1 may store the conflicting states, namely, $\Downarrow^{S^1_{N_1}}_{S^2_{N_1}}$. Note that, here, we assume $\Downarrow^{S^1_{N_2}}_{S^2_{N_2}} = \Downarrow^{S^1_{N_1}}_{S^2_{N_1}}$, indicating that the states sent from N_2 are finally confirmed in the node N_1, and accordingly subscripts are changed. Without re-pulling the state from other nodes when the bound set in the counter is exceeded, the conflicting states will never be able to be reversed. Thus, the complementary mechanism re-attaches the state $S^2_{N_1}$, and the updated $S^2_{N_1}$ will replace the out-dated version. The relative positions in N_1 are thus returned to correct positions, saying $\Downarrow^{S^2_{N_1}}_{S^2_{N_1}} = \Downarrow^{S^2_{N_2}}_{S^1_{N_2}} = \Downarrow^{S^2_{N_3}}_{S^1_{N_3}}$. The above procedure is compulsory for N_1. If N_1 cannot accept relative positions from honest nodes N_2 and N_3, the new messages that follow N_1 will not be accepted by N_2 and N_3 since his *Pre-Prepare* message is based on the latest state from others. More details are provided in our security analysis.

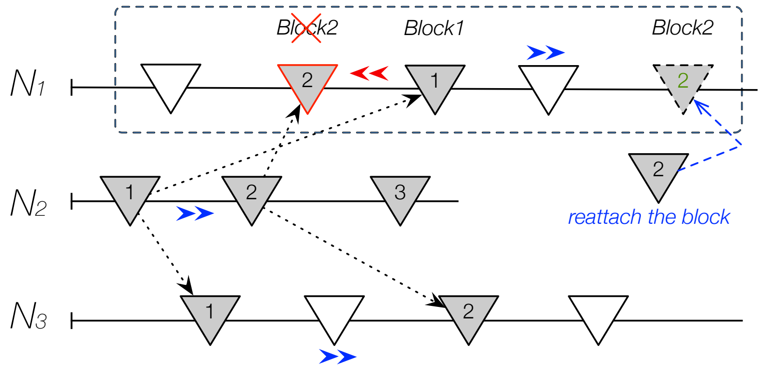


Figure 6.2: Complementary mechanism diagram

Highlighted Differences. Our protocol differs from PBFT in four aspects (see Table.6.1): (a) Our protocol is an asynchronously leaderless Byzantine agreement protocol. Instead of relying on a single leader, Sphinx removes the leader-associated phases, enabling every participant involved in the consensus procedures. Thus, every participant does not need to wait for the latest state synchronized from others. (b) Every consensus node in the network conducts the similar behaviours (*pre-prepare*, *prepare*, *commit*). These nodes independently proceed but mutually interact with each other by cross-references. (c) We remove auxiliary mechanisms (such as *checkpoint*, *view-change*) of PBFT protocols. Instead, we provide a brief complementary mechanism to solve conflicts like reserved positions between two states. (d) Our protocol weakens the assumption of *strong consistency*, with the gain of higher performance and lower confirmation time. We follow the *liveness* property from PBFT. Further, we introduce a new security definition *relative persistence*, which is inspired by the properties of *agreement* [45] and *persistence* [84].

6.1.2.4 Security Properties

Our algorithm weakens the strong guarantee of consistency by reaching a partial consistency instead of a linear consistency. The procedure of total ordering is no longer needed in our consensus. Each node individually creates new states and simultaneously stores the remote (other's) states. We allow one state to be stored in multiple positions across parallel chains. The key idea is to keep the consistency of the relative position between the two states. Based on that, we formalize our algorithms by two properties: *relative persistence* and *liveness*. *Relative persistence* means the relative positions of two states are irreversible once enough honest nodes report it as confirmed. *Liveness* means once the relative position between two states is confirmed by one honest node, it should eventually be confirmed by all the other honest nodes in the network.

Definition 1 (Relative persistence)**.** *Weak consensus algorithm achieves the property of relative persistence, if for all relationship in* $\mathbb{R}$*, there exists a negligible function* $negl(\lambda)$ *such that* $adv_{\mathbb{N}}^{\mathbb{R}}(\lambda) < negl(\lambda)$*, where* $adv_{\mathbb{N}}^{\mathbb{R}}(\lambda)$ *is the advantage in which the decisions on the same relationship* $\Downarrow_{S_{N_i}^{x}}^{S_{N_i}^{y}}$ *made by any two honest nodes are conflicting. Here,* $0 < x < y \leq r$*.*

The *relative persistence* property ensures that as soon as the relative position between two states has been confirmed by an honest node, this relationship will ultimately be confirmed in every node in the network with high probability. The property guarantees that the relative positions remain consistent between two states across paralleled chains.

Definition 2 (Liveness)**.** *Weak consensus algorithm achieves the liveness property, if for all PPT honest nodes, there exists a negligible function* $negl(\lambda)$ *such that*

$adv_{\mathbb{N}}^{\mathbb{R}}(\lambda) < negl(\lambda)$, where $adv_{\mathbb{N}}^{\mathbb{R}}(\lambda)$ is the advantage that the honest node does not accept the correct relationship $\Downarrow_{S_{N_i}^{x}}^{S_{N_i}^{y}}$. Here, $0 < x < y \leq r$.

The *liveness* property guarantees that all nodes eventually agree on a unique relationship regarding each chain. The unique relationship represents the relative position between a state and its ancestor. The term *eventually* indicates that it may take a sufficient amount of time (within the upper bound of δ) to reach the agreement. The property ensures that a state will either be abandoned or accepted instead of permanently pending status.

6.1.3 Sphinx System

In this section, we first introduce the cryptographic building blocks used to build the scheme. Then, we present a high-performance blockchain system (*Sphinx*) that adopts the weak consensus algorithm.

6.1.3.1 Entities.

Sphinx mainly consists of two types of nodes, namely *blockchain node* and *client node*. The client node is the creator of the message and allows sending a request to the ledger recorder and waiting for replies after the consensus is completed. The blockchain node is responsible for two functionalities: *record* and *validate*. The former functionality is used to record the local chain. Another one is employed to check the correctness of the recording progress.

6.1.3.2 System Design

The state in Sphinx is instantiated as the block, which is validated and confirmed by other nodes in the network. The message inherits the classic blockchain structure, which includes the fields of address, timestamp, metadata *etc.* Every chain has two types of references in our system: the one pointing to its own parent block and the other pointing to all other nodes. Explicitly, our system embraces cross-referencing to increase the blockchain's security, where multiple nodes simultaneously generate their own chains in parallel and validate the blocks of other nodes. The cross-reference ensures that each chain can mutually validate others' behaviours, such as whether they maintain a consistent sequence of blocks by checking their previous Merkle roots. This mechanism guarantees the consistency of relative positions between two blocks. The concrete protocol is presented as follows.

Pre-Prepare. Each node maintains an individual message pool and an independent ledger. When a node N_p receives a request (message) M from the client, it first checks the message's syntax to ensure the correct execution. If passed, it sorts the received client messages in the local pool. These messages are ordered by their timestamps (*e.g.,* Lamport timestamps [45]), in which the latest messages have higher priorities than the earlier ones. The first received message is processed in the algorithm, whereas the conflicting/duplicated messages are discarded. Based on the ordered sequence, the node assigns a valid sequence number SN to M. We emphasize that the sequence number SN is a local variable used to represent the index r of a single chain. It is different from the global variable SN defined in BFT algorithm [45][46]. The index r in our system helps to locate a block in the local chain while locating a block in the global view requires an additional parameter of chain id N_j to form a coordinate (r, N_j).

Next, it inserts the message M into its local chain. The insertion mainly merges the hash of M and the states of remote blocks $(S^r_{N_0}, S^r_{N_1}, \ldots, S^r_{N_n})$ into a new Merkle Tree [181]. Next, it signs the Merkle root and appends this block to a public log by generating a proof PF, which contains three types of proofs: (a) P_M is used to prove that the tree contains M; (b) P_S is used to prove $(S^r_{N_0}, S^r_{N_1}, \ldots, S^r_{N_n})$ exists in trees; (c) P_B is used to prove the tree is an extension of old blocks.

The message M and states $(S^r_{N_0}, S^r_{N_1}, \ldots, S^r_{N_n})$ are stored in the leaves of the Merkle tree from left to right in chronological order. Thus, the proof can be easily calculated. A Merkle tree contains the items of M and states $(S^r_{N_0}, S^r_{N_1}, \ldots, S^r_{N_n})$. In our design, these items are solely stored at leaves. After that, the current node broadcasts the *Pre-Prepare* message $\langle \mathsf{Pre\text{-}Prepare}, M, SN, S^r_{N_p}, PF, \Downarrow^{S^r_\star}_{S^{r-1}_{N_p}} \rangle$ to other nodes, where $\Downarrow^{S^r_\star}_{S^{r-1}_{N_p}}$ represents the relative position between the r-th state and $(r-1)$-th state.

Prepare. Assume that a random node N_q receives the *Pre-Prepare* message $\langle \mathsf{Pre\text{-}Prepare}, S^r_{N_p}, M, PF, \Downarrow^{S^r_\star}_{S^{r-1}_{N_p}} \rangle$ from the node N_p. The node N_q validates the correctness of *Pre-Prepare* message. The algorithm checks whether: (a) the signature of $S^\star$ is correct; (b) the message M has been inserted to N_p; (c) the previous state $S_q^\star$ has been inserted to N_p; (d) the state of the claimed message has no conflict. When the received *Pre-Prepare* message is checked successfully, the node N_q updates his local-stored state S_p, and inserts the received message to his local log. Then, it generates and broadcasts a reply *Prepare* message $\langle \mathsf{Prepare}, SN, d(SN)\rangle$, in which $d(SN)$ means the hash digest of the state. Otherwise, the node aborts the *Pre-Prepare* message. Note that the same *Pre-Prepare* message can only be accepted once, and the duplicated *Pre-Prepare* message will be discarded.

Commit. If a node receives a quorum $2f + 1$ of valid *Prepare* messages from different nodes (possibly including his own), it confirms the proposed message and broadcasts a *Commit* message $\langle \mathsf{Commit}, SN, d(SN)\rangle$. Then, this node collects the *Commit* messages from all nodes. Once exceeding the threshold $(2f+1)$, the node accepts the updated state. Then, this node replies to the client with new states.

Complementary Mechanism. The aforementioned phases are insufficient since some malicious $(2f - 1)$ nodes may store the wrong (opposite) relative positions, making our system fail in satisfying the security property of relative persistence. It is possible that the relative relationship cannot be committed due to the lack of $2f+1$ matched *Commit* messages. The states, as a result, will never be terminated. To solve this problem, we introduce our complementary methods. On one side, each message is embedded with a counter. If the message fails due to the lack of enough confirmation, the procedure of rebroadcasting will be launched, and the counter increases each time of a retry. If the accumulated value is greater than the bound set in the counter, the node will pull the newest state from other nodes and accept the reversed relationship and rebroadcast it. If a node collects more than $2f + 1$ *Commit* messages on the reversed position, the node replies to clients with updated states. Otherwise, the message will be aborted, and the node will accordingly send a *Failure* message to the client. On the other side, when the waiting time exceeds the predefined time bound in the counter, the message is aborted with a *timeout* message sent to the client. The complementary mechanism is essential to achieve the properties of *relative persistence* and *liveness*.

6.1.4 Implementation

We have implemented the system in Go language[1] with 32,000+ lines of code. We have developed full functionalities of a classic blockchain system, including account configuration, consensus mechanism, peer-to-peer network, user interface, *etc.* We employ Go's built-in hash function *SHA-256* and elliptic curve digital signature algorithm *secp256k1*. Here, we focus on key functions to present a skeleton of our implementation. Example code segments together with the workflow are illustrated in Figure 6.3. To be specific, ValidateState validates the changed state after the state transition, such as the receipt roots and state roots. The function will return a database handle if the validation turns out a success. Otherwise, an error is returned. ValidateBody validates the uncle blocks and verifies their header's receipts. The headers are assumed to be already validated at this point. NewBlockChain returns a fully initialized blockchain by loading information in the database. It initializes default validators. FastSyncCommitHead inserts the committed head block to others by the form of hashes. GetBlocksFromHash returns the block corresponding to hash and up to $n-1$ ancestors. InsertHeaderChain attempts to insert the headers of parallel chains into the local chain. Insert inserts or rejects a new header of the block into the current local chain. Confirm aims to ensure whether threshold conditions are satisfied by a block.

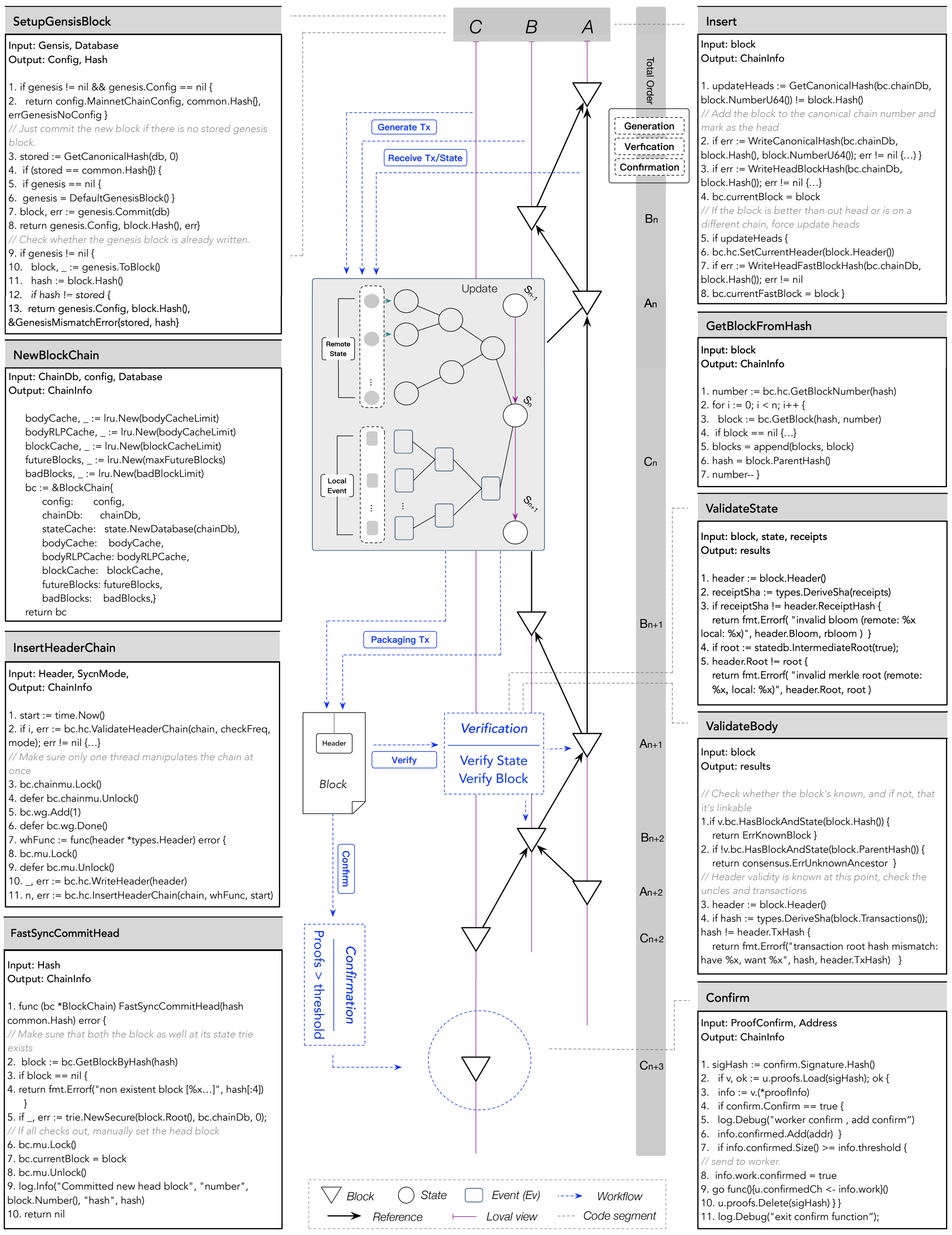
6.1.5 Security Analysis

This section proves that our protocol satisfies *relative persistence* and *liveness*. We assume that honest nodes are consistent for their commits, which means if an honest node accepts a relative state, all his commits in this iteration and the following iterations are consistent. Formally, we define the above intuition in Lemma 1.

Lemma 1. *Suppose a node N_i is the first honest node to commit a relationship $\Downarrow_{S^x_{N_i}}^{S^y_{N_i}}$ for the relative positions between the y and x. In all subsequent iterations, all commits from this node are valid decisions on the relationship $\Downarrow_{S^x_\star}^{S^y_\star}$.*

Theorem 1. ***(Relative persistence)*** *If the relative position of two states y and x is accepted by the node N_i in iteration r and by the node N_j in $r+1$, respectively,*

[1]Go is an open-source programming language supported by Google



SetupGensisBlock

```
Input: Gensis, Database
Output: Config, Hash

1. if genesis != nil && genesis.Config == nil {
2.   return config.MainnetChainConfig, common.Hash{},
errGenesisNoConfig }
// Just commit the new block if there is no stored genesis
block.
3. stored := GetCanonicalHash(db, 0)
4.  if (stored == common.Hash{}) {
5.  if genesis == nil {
6.  genesis = DefaultGenesisBlock() }
7. block, err := genesis.Commit(db)
8. return genesis.Config, block.Hash(), err}
// Check whether the genesis block is already written.
9. if genesis != nil {
10.   block, _ := genesis.ToBlock()
11.   hash := block.Hash()
12.   if hash != stored {
13.  return genesis.Config, block.Hash(),
&GenesisMismatchError{stored, hash}
```

NewBlockChain

```
Input: ChainDb, config, Database
Output: ChainInfo

    bodyCache, _ := lru.New(bodyCacheLimit)
    bodyRLPCache, _ := lru.New(bodyCacheLimit)
    blockCache, _ := lru.New(blockCacheLimit)
    futureBlocks, _ := lru.New(maxFutureBlocks)
    badBlocks, _ := lru.New(badBlockLimit)
    bc := &BlockChain{
        config:      config,
        chainDb:     chainDb,
        stateCache:   state.NewDatabase(chainDb),
        bodyCache:    bodyCache,
        bodyRLPCache: bodyRLPCache,
        blockCache:   blockCache,
        futureBlocks: futureBlocks,
        badBlocks:    badBlocks,}
    return bc
```

InsertHeaderChain

```
Input: Header, SycnMode,
Output: ChainInfo

1. start := time.Now()
2. if i, err := bc.hc.ValidateHeaderChain(chain, checkFreq,
mode); err != nil {...}
// Make sure only one thread manipulates the chain at
once
3. bc.chainmu.Lock()
4. defer bc.chainmu.Unlock()
5. bc.wg.Add(1)
6. defer bc.wg.Done()
7. whFunc := func(header *types.Header) error {
8. bc.mu.Lock()
9. defer bc.mu.Unlock()
10. _, err := bc.hc.WriteHeader(header)
11. n, err := bc.hc.InsertHeaderChain(chain, whFunc, start)
```

FastSyncCommitHead

```
Input: Hash
Output: ChainInfo

1. func (bc *BlockChain) FastSyncCommitHead(hash
common.Hash) error {
// Make sure that both the block as well at its state trie
exists
2.  block := bc.GetBlockByHash(hash)
3. if block == nil {
4. return fmt.Errorf("non existent block [%x…]", hash[:4])
    }
5. if _, err := trie.NewSecure(block.Root(), bc.chainDb, 0);
// If all checks out, manually set the head block
6. bc.mu.Lock()
7. bc.currentBlock = block
8. bc.mu.Unlock()
9. log.Info("Committed new head block", "number",
block.Number(), "hash", hash)
10. return nil
```

Insert

```
Input: block
Output: ChainInfo

1. updateHeads := GetCanonicalHash(bc.chainDb,
block.NumberU64()) != block.Hash()
// Add the block to the canonical chain number and
mark as the head
2. if err := WriteCanonicalHash(bc.chainDb,
block.Hash(), block.NumberU64()); err != nil {...} }
3. if err := WriteHeadBlockHash(bc.chainDb,
block.Hash()); err != nil {...}
4. bc.currentBlock = block
// If the block is better than out head or is on a
different chain, force update heads
5. if updateHeads {
6. bc.hc.SetCurrentHeader(block.Header())
7. if err := WriteHeadFastBlockHash(bc.chainDb,
block.Hash()); err != nil
8. bc.currentFastBlock = block }
```

GetBlockFromHash

```
Input: block
Output: ChainInfo

1. number := bc.hc.GetBlockNumber(hash)
2. for i := 0; i < n; i++ {
3.   block := bc.GetBlock(hash, number)
4.  if block == nil {...}
5. blocks = append(blocks, block)
6. hash = block.ParentHash()
7. number-- }
```

ValidateState

```
Input: block, state, receipts
Output: results

1. header := block.Header()
2. receiptSha := types.DeriveSha(receipts)
3. if receiptSha != header.ReceiptHash {
    return fmt.Errorf( "invalid bloom (remote: %x
local: %x)", header.Bloom, rbloom )  }
4. if root := statedb.IntermediateRoot(true);
5. header.Root != root {
    return fmt.Errorf( "invalid merkle root (remote:
    %x, local: %x)", header.Root, root )
```

ValidateBody

```
Input: block
Output: results

// Check whether the block's known, and if not, that
it's linkable
1.if v.bc.HasBlockAndState(block.Hash()) {
    return ErrKnownBlock }
2. if !v.bc.HasBlockAndState(block.ParentHash()) {
    return consensus.ErrUnknownAncestor  }
// Header validity is known at this point, check the
uncles and transactions
3. header := block.Header()
4. if hash := types.DeriveSha(block.Transactions());
hash != header.TxHash {
    return fmt.Errorf("transaction root hash mismatch:
have %x, want %x", hash, header.TxHash)   }
```

Confirm

```
Input: ProofConfirm, Address
Output: ChainInfo

1. sigHash := confirm.Signature.Hash()
2.   if v, ok := u.proofs.Load(sigHash); ok {
3.   info := v.(*proofInfo)
4.   if confirm.Confirm == true {
5.   log.Debug("worker confirm , add confirm")
6.   info.confirmed.Add(addr)  }
7.   if info.confirmed.Size() >= info.threshold {
// send to worker.
8.  info.work.confirmed = true
9. go func(){u.confirmedCh <- info.work}()
10. u.proofs.Delete(sigHash) } }
11. log.Debug("exit confirm function");
```

Figure 6.3: Sphinx implementation diagram

their decisions on the relationship are the same, represented as $\Downarrow^{S_i^y}_{S_i^x} = \Downarrow^{S_j^y}_{S_j^x}$.

Proof. We prove the theorem by contradiction induction. We assume the relationship $\Downarrow^{S_i^y}_{S_i^x}$ is accepted by the node N_i. Similarly, we assume the $\Downarrow^{S_j^y}_{S_j^x}$ is accepted by the node N_j. We show that, in current iteration and all subsequent iterations, the reported relationships from N_i and N_j are consistent, namely, $\Downarrow^{S_i^y}_{S_i^x} = \Downarrow^{S_j^y}_{S_j^x}$, and no valid relationships other than the reported one can be agreed upon.

Suppose the property of relative persistence is violated, which indicates the relationship holds $\Downarrow^{S_i^y}_{S_i^x} \neq \Downarrow^{S_j^y}_{S_j^x}$. From the above assumption, we know the relationship $\Downarrow^{S_i^y}_{S_i^x}$ from the node N_i has been accepted, which means N_i received $2f + 1$ valid *Commit* replies in the current iteration. Among these replies, at least one of the commits comes from an honest node (assume N_k). Thus, N_k must have received a *right* relative position between two states and forwarded this relationship to all other nodes. If $\Downarrow^{S_k^y}_{S_k^x} \neq \Downarrow^{S_j^y}_{S_j^y}$, other honest nodes can immediately detect the mismatch of relationships from different proposals. A wrong-located state (with the same sequence number) is in a reversed position. The nodes then reattach the state until they obtain a consistent relationship. Dishonest nodes will never collect more than $2f + 1$ valid *Commit* messages unless the majority of honest nodes become traitors in the current and following iterations. However, this situation contradicts *Lemma 1*. Thus, we have $\Downarrow^{S_i^y}_{S_i^x} = \Downarrow^{S_j^y}_{S_j^x}$. □

Now, we move to *liveness* property and explain how our algorithm guarantees all honest nodes agree on the same relative relationship and reach the termination.

Theorem 2. ***(Liveness)*** *If a correct relationship* $\Downarrow^{S_i^y}_{S_i^x}$ *is committed, every honest node will eventually accept it.*

Proof. Suppose that the property of *liveness* is violated, where only a small fraction of nodes (less than the threshold) or even none of the nodes accept the final decision on the relationship $\Downarrow^{S_\star^y}_{S_\star^x}$ between states y and x. Equally, a majority of nodes reject or have no responses for the commit decision after a sufficient period of time. We show that, in all subsequent rounds, the malformed relative relationship from dishonest nodes will never be accepted.

If a randomly selected group of dishonest nodes (more than $\lfloor \frac{3f+1}{2} \rfloor$) reject the decision of the relationship $\Downarrow_{S_\star^x}^{S_\star^y}$, an honest node N_i will never be terminated. This is because the state of N_i has to be confirmed by enough nodes, which requires at least $2f+1$ valid *Commit* messages from other nodes. The confirmed message is based on the other honest nodes' states. If the majority of nodes reject a correct proposal, any honest node cannot get confirmed, either. Thus, when times out, it must reverse the relative relationship and restart the generation phase to achieve the final confirmation. □

6.1.6 Evaluation

Experimental Configurations. Our experiments are conducted on 8 Dell R730 rack servers in a local cluster with dual 2.1 GHz Opteron CPUs. The bandwidth is connected with 1 Gbps switched Ethernet. The operating system is running based on Ubuntu 16.04.1 LTS version. Meanwhile, all nodes in our experiment are deployed on the same machine and connected to different ports to simulate a globally distributed system.

Performance Evaluation. The throughput represents the rate of transactions being confirmed in a certain time interval. We adopt the log-based approach [223] and the concept of transactions per second (TPS) to measure the rate. In our performance experiments, we set the production of transactions at a constant rate and calculate the confirmation time of transactions in a fixed time. The time is measured in seconds via wall clock running time. To achieve a fair test result, we repeat 300 times. The results show that the average throughput of Sphinx reaches 43k TPS with 8 full nodes and drops to around 5000 TPS given 64 full nodes. Even if the TPS drops sharply, our system is still greatly faster than Ethereum. In particular, given the same testing environment, Ethereum (version 1.9.25, released on December 11, 2020) only reaches 355, 311, 268, 91 TPS under the setting of 8, 16, 32, 64 nodes, respectively. To explore the reason behind the high throughout, we further evaluate the performance of each individual algorithm.

Firstly, in the *Pre-Prepare* stage, the transaction is added to a block. The evaluation results (see Figure 6.4) show that it takes approximately 15 milliseconds to finish the block generation algorithm, making our system reach an extremely high throughput. In contrast, Ethereum needs over 50 milliseconds to generate a block. The high-speed generation rate of our system is due to the parallel processing mechanism. Each node has the ability to instantly generate the block, even if it does not consist of the latest state header. In contrast, for mining a block, the node in Ethereum has to wait for the latest block header broadcast by other nodes, which causes a severe delay.

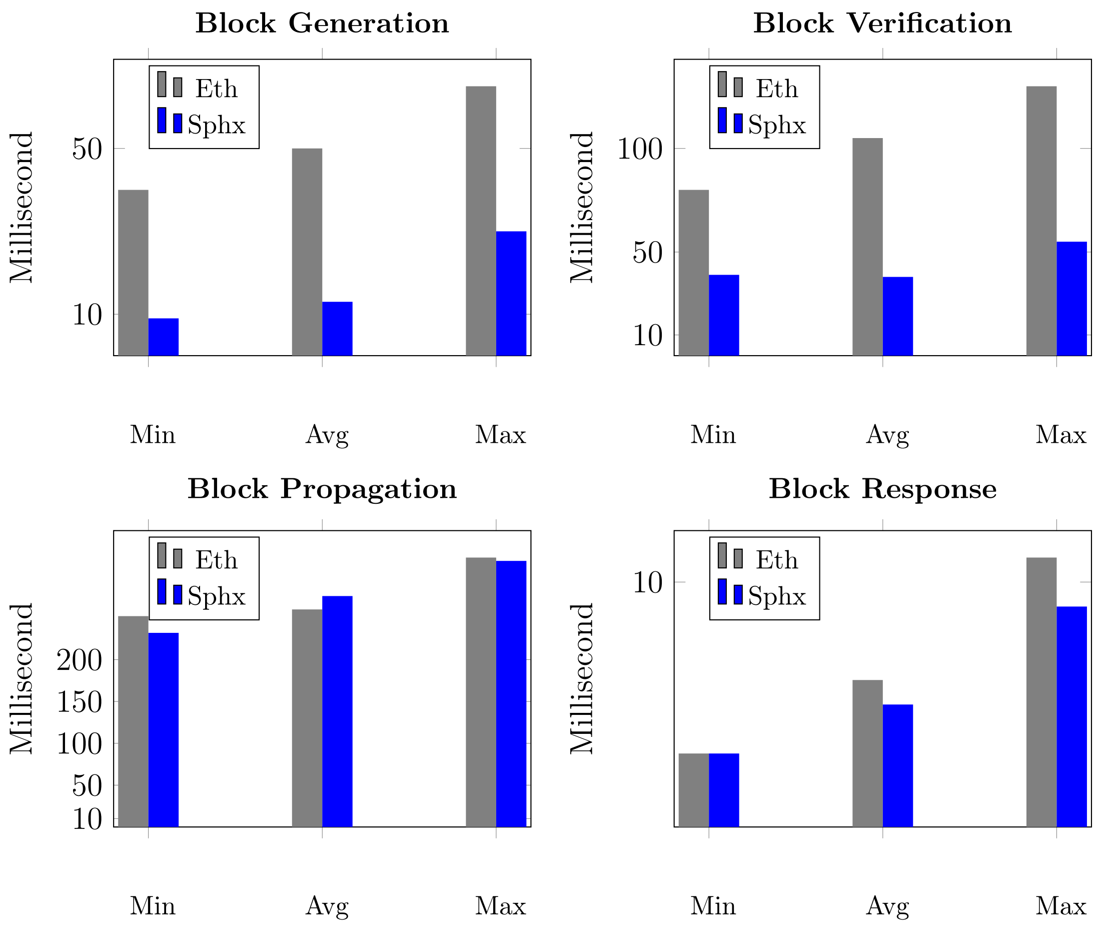
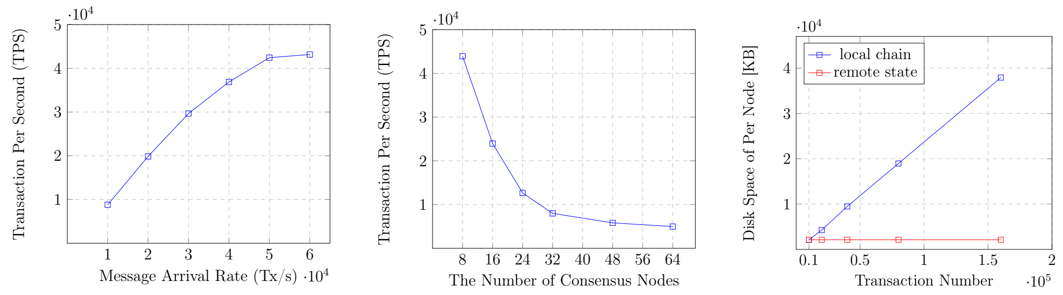


Figure 6.4: Execution time of different operations in Ethereum and Sphinx

Then, we consider the verification time and message broadcasting time in the *Prepare* stage. The verification time represents the length of time in verifying states. In particular, it covers the time of checking the validity of signatures, the correctness of proofs, and the non-conflicts of messages. The results show that our verification algorithm takes approximately 200 milliseconds, which is remarkably efficient compared to Ethereum. The broadcasting time indicates the length of time in propagating a message from one node to another. We assume that all nodes share the same timestamp, where such configurations can be achieved by NTP service [155]. When the message is generated by a node, it is broadcast to other nodes in the network. We adopt the concept of *coverage rate* to represent the percentage of reached nodes. Our experiment results show that it takes around 500 milliseconds to achieve 100% coverage rate for a message, which is roughly the same as Ethereum's testing results. The algorithm is the main bottleneck of our system.

Furthermore, we check the response time of a message. The response time represents the length of time in replying to a message from any chain node to the client. This operation averagely takes around 15 milliseconds, which is fast and efficient in our implementation.

Figure 6.5: Scalability and disk space evaluation



(a) TPS under different message arrival rates (b) TPS under different size of nodes (c) Space usage on each Sphinx node

Scalability. The scalability in Sphinx is used to describe the capability to handle an increasing number of transactions. To study its scalability, we set the block size at 4 Megabytes (MB) and the block generation rate at 3s. Then, we run experiments with the following configurations: (i) increase the transaction arrival rate with a fixed number of nodes (as shown in Figure 6.6(a), the transaction arrival rate increases from 10^4 to $6 * 10^4$); (ii) increase the number of nodes with a fixed transaction arrival rate (as shown in Figure 6.6(b), the number of nodes increases from 8 to 64). To obtain an accurate testing result, again, we repeat experiments 300 times and calculate their average performance.

The first experiment attempts to evaluate the average TPS over different transaction arrival rates. We set 8 nodes participating in the system and start the testing at a rate of 1250 Tx/s for each node (total arrival rate = 10^4 Tx/s). Then, we increase the rate in a fixed interval, as it is shown in Figure 6.6(a). The evaluation shows that when the arrival rate is less than 50k, the throughput increases linearly. When the arrival rate gets close to or above the saturation point (50k Tx/s), the throughput flats out at around 43k TPS. The reason is that the propagation latency becomes the primary bottleneck when the arrival rate is over 50k Tx/s, which makes the influence of the arrival rate negligible. Meanwhile, the number of transactions pending in the verification phase also affects the final results.

The second experiment attempts to evaluate the throughput with respect to an increasing number of nodes. Our algorithm, implemented based on PBFT, is only suitable for permissioned blockchain. We limit the size of the committee to an upper bound of 64, and the number of these nodes will remain stable. We send transactions at a fixed rate (total arrival rate = $5*10^4$ Tx/s) and adjust participated nodes from 8 to 64. Figure 6.6(b) shows the throughput of the system drops down as the number of participants increases. Merely increasing the number of participants improves the concurrency but scarifies the performance of individual nodes. Each of them has to wait for enough replies from other nodes (reaching the threshold). Thus, the scalability of Sphinx cannot be improved without limitation.

Disk Space. To check the feasibility, we also provide evaluations of disk space. In Sphinx, the storage of the node grows each time when the messages are appended to the local chain. Meanwhile, for checking the correctness of the behaviours from other nodes, each auditor must store the state received from other nodes. Thus, we consider space evaluation in two aspects: (i) the size of the local chain and (ii) the size of the remote state. We assume that there are eight nodes with the transaction creation rate of 100 messages/second. Then, we monitor the space usage of each node and analyze their growth rates. As shown in Figure 6.6(c), the results indicate that the size of the local chain grows linearly with the increased transactions. On average, the size of the local chain in each node grows at the rate of 0.212 KB per message. In contrast, the size of the remote state is static, which is independent of the scale of transactions. This is easy to understand because the disk usage of remote states relies on the scale of the participated nodes, where the number is fixed in the initial configuration.

6.1.7 Use Cases of Our Consensus

This section explores two potential applications to demonstrate the feasibility and applicability of the proposed consensus.

Certificate System. Issuing and verifying certificates are slow and complicated. Errors and fraud also threaten the usability of certificates. A blockchain-based certificate system uploads certificate metadata to the blockchain to achieve reliable management. However, current systems such as Bitcoin suffer from extremely low performance, making the certificate confirmed slowly. This greatly limits the wide adoption of classic blockchain systems. We observed that the key idea behind a certificate system is to store certificates transparently rather than to sort their orders. Thus, our system can perfectly meet the requirements: (i) the transaction data on the chain proves the existence of uploaded certificates; (ii) the high-performance system without linear ordered sequence makes it applicable to large-scale certificate scenarios.

Log System. Blockchain technology provides a new approach to the log system since it provides a publicly accessible bulletin board. Recording the logs that are generated by the software onto a distributed storage system greatly improves security. The irreversibility of the blockchain system guarantees that the uploaded logs cannot be easily falsified. However, the low performance of current blockchain systems significantly retards the procedures (upload/store/download/change) of logs. This impedes their applications to business scenarios. Our weak consensus benefits current approaches with the ability to prove the existence as well as high performance of processing, enabling blockchain-based log systems more practical.

6.1.8 Related Work

Our scheme adopts the model of *parallel chains* [78], where each node maintains their own chains. Generally, two types of consensus mechanisms are adopted in our model: the variants of BFT protocols called *leaderless BFT protocols*, and the modified NC protocols named *extended Nakamoto consensus*.

Leaderless BFT protocols. *Hashgraph* [13] was proposed with the leaderless BFT mechanism. Each node maintains a separate chain, but they are required to interact via the gossip protocol mutually. The node that receives the synchronization information creates a message locally to record the history and then broadcasts it to peers. Other nodes iterate the same procedure. Hashgraph achieves the consensus through an asynchronous Byzantine consensus. However, disseminated information containing all previous histories is heavy, which significantly increases communication overhead. Parallel chains in *DEXON* [50] confirm each other through the reference field called `ack`, and achieve consensus through these references. The consensus consists of three steps. Firstly, each block is deterministically arranged into one single chain by comparing the residue of its hash value. Then, DEXON employs the technique of a variant BFT protocol borrowed from Algorand [92] as its single-chain consensus. Finally, it proposes a sorting mechanism to determine the total order of all blocks across parallel chains. In the case of a network delay, these steps will easily be congested, which will negatively affect the entire system. *Aleph* [81] is a leaderless BFT distributed system. Each node concurrently issues units (carrying messages). These units are organized in different sets. Units in these sets undergo a voting algorithm. The unit is considered to be valid if it receives more votes than the threshold. However, this procedure is time-consuming.

Extended Nakamoto Consensus. *OHIE* [216] shares similarities in composing multiple parallel chains. OHIE adopts classic Nakamoto consensus for each individual chain. Miners need to calculate a puzzle to generate blocks and additionally have to sort received blocks. Blocks arrive at a global order across all parallel chains, thus achieving consistency. However, the total ordered sequence limits the upper bound of performance due to the strong assumption of consistency. *Prism* [11] structures the network with three types of blocks: proposer blocks, transaction blocks, and voter blocks. These blocks replace the functionalities of a common block in Nakamoto consensus. The consistency is achieved by sorting all transaction blocks. The total ordering is ensured by its proposer blocks, which are selected by voter blocks. However, even decoupled functionalities reach their upper bounds, the procedure of a total ordering algorithm still becomes the bottleneck of throughput. *Chainweb* [147] aims to scale Nakamoto consensus by maintaining multiple parallel chains. It is based on a PoW consensus that incorporates each other's Merkle roots to increase the hash rate. Each chain in the network mines the same cryptocurrency, which can be transferred cross-chain via a simple payment verification. The method reduces several coins from one chain and creates equal amounts on another chain. However, Kiffer *et al.* [121][78] argue that Chainweb utilizing Nakamoto consensus is bounded by the same throughput under the same consistency guarantee.

6.1.9 Conclusion

In this chapter, we proposed a weak consensus algorithm by relaxing the strong consistency promise. We applied our algorithm to a high-performance blockchain system. The system runs in parallel chains, where all transactions and blocks are concurrently processed. We further defined the security of *relative persistence* and *liveness*, and then proved that our system achieves these properties. Also, we provided a full implementation with all layered components, including P2P network and ledger structure. The evaluation indicated that our system achieves high performance, with approximately 43k TPS in total. Finally, we explored potential applications to demonstrate feasibility and applicability.

6.2 An Offline Delegatable Payment System

6.2.1 Introduction

The interest in decentralized cryptocurrencies has grown rapidly recently. Bitcoin [158], as the first and most famous system, has attracted massive attention. Subsequently, a handful of cryptocurrencies, such as Ethereum [213], Namecoin [6] and Litecoin [222], were proposed. Blockchain-based cryptocurrencies significantly facilitate the convenience of payment by providing a decentralized online solution for customers. However, merely online processing of transactions confronts the problem of low performance and high congestion. Offline delegation provides an alternative way to mitigate the issue by enabling users to exchange the coin without having to connect to an online blockchain platform [97]. Unfortunately, decentralized offline delegation still confronts risks caused by unreliable participants. Misbehaviours may easily happen due to the absence of effective supervision. To be specific, let us start from a real scenario: imagine that Bob, the son of Alex, a wild teenager, wants some digital currency (*e.g.*, BTC) to buy a film ticket. According to current decentralized cryptocurrency payment technologies [158][213], Alex has two delegation approaches:

- *Coin-transfer.* Alex asks for Bob's BTC address and then transfers a specific number of coins to Bob's address. In such a scenario, Bob can only spend the received coins from Alex.
- *Ownership-transfer.* Alex directly gives his own private key to Bob. Then, Bob can freely spend coins using such a private key. In this situation, Bob obtains all coins that are saved in Alex's address.

We observe that both approaches suffer drawbacks. For the first approach, coin-transfer requires a global consensus of the blockchain, which makes the payment time-consuming [118]. For example, a confirmed transaction in Bitcoin [158] takes around one hour (6 blocks), making coin-transfer lose the essential property of real-time. For the other approach, ownership-transfer highly relies on the honesty of the delegatee. The promise between the delegator and delegatee depends on their trust or relationship. But it is weak and unreliable. The delegatee may spend all coins in the address for other purposes. Back to the example, Alex's original intention is to give Bob 200 μBTC to buy a film ticket, but Bob may spend all coins to purchase his favourite toys. That means Alex loses control of the rest of the coins. These two types of approaches represent most of the mainstream schemes ever aiming to achieve a secure delegation, but neither of them provides a satisfactory solution. This leads to the following research problem:

> Is it possible to build a secure offline peer-to-peer delegatable system for decentralized cryptocurrencies?

The answer would intuitively be "NO". Without interacting with the online network, used coins confront the risk of being spent twice after another successful delegation. This is because a delegation is only witnessed by the owner and delegatee, where no authoritative third parties perform the final confirmation. The pending status leaves a window for attacks in which a malicious coin owner could spend this delegated transaction before the delegatee uses it. Even if a third party can be introduced as a judge between the delegator (owner) and delegatee to secure transactions, He faces the threat of being compromised or provided with misleading assure. Furthermore, the approach that uses a third party contradicts the real intention of decentralized cryptocurrency systems.

In this chapter, we propose *DelegaCoin*, an offline delegatable electronic cash system. The trusted execution environments (see Section 2.3 for more details on TEEs) are utilized to play the role of a *virtual agent*. TEEs prevent malicious delegation of coins (*e.g.,* double-delegation of the same coins). As shown in Figure.6.6, the proposed scheme allows the owner to delegate his coins without interacting with the blockchain or any trusted third parties. The owner can directly delegate specific amounts of coins to others by sending them through a secure channel. This delegation can only be executed once under the supervision of delegation policy inside TEEs.

6.2.2 Related Work

Decentralized Cryptocurrency System. Blockchain-based cryptocurrencies facilitate the convenience of payment by providing a decentralized online solution for customers. Bitcoin [158] was the first and most popular decentralized cryptocurrency. Litecoin [222] modified the PoW mechanism by using the Script algorithm and shortened the block confirmation time. Namecoin [6] was the first hard fork of Bitcoin to record and transfer arbitrary names (keys) securely. Ethereum [213] extended Bitcoin by enabling state-transited transactions. Zcash [114] provided a privacy-preserving payment solution by utilizing zero-knowledge proofs. CryptoNote-style schemes [217], instead, enhanced privacy by adopting ring-signatures. However, slow confirmation of transactions retards their wide adoption from developers and users. Current cryptocurrencies, with ten to hundreds of transactions [158, 223], cannot rival mature payment systems such as Visa or PayPal that can process thousands of transactions. Thus, various methods have been proposed for better throughput. The scaling techniques can be categorized in two ways: (i) On-chain solutions that aim to create highly efficient blockchain protocols, either by reconstructing structures, connecting chains [219] or via sharding the blockchain [204]. However, on-chain solutions are typically not applicable to existing blockchain systems (require a hard fork). (b) Off-chain (layer 2) solutions that regard the blockchain merely as an underlying mechanism and process transactions offline [97]. Off-chain solutions operate independently on top of the consensus layer of blockchain systems, not changing their original designs. In this chapter, we explore the second avenue.

Payment Delegation. Payment delegation plays a crucial role in e-commercial activities, and it has been comprehensively studied for decades. Several widely adopted approaches are such that, using credit cards (Visa, Mastercard, etc.), reimbursement, third-party platforms (like PayPal [212], AliPay [144]). These schemes allow users to delegate their cash spending capability to their devices or other users. However, these delegation mechanisms heavily rely on a centralized party that needs a significant amount of trust. Decentralized cryptocurrencies, like Bitcoin [158] and Ethereum [213], remove the role of trusted third parties, making the payment reliable and guaranteed by distributed blockchain nodes. However, such payment is time-consuming since online transactions need to get confirmed by the majority of participated nodes. The delegation provides the decentralized cryptocurrency with an efficient payment approach to delegate the coin owner's spending capability. The cryptocurrency delegation using SGX was first explored in [148], where they only focused on credential delegation in the fair exchange. Teechan [140] provided a full-duplex payment channel framework that employed TEEs, in which the parties can pay each other without interacting with the blockchain in a bounded time. However, Teechan requires a complex setup: the parties must commit a *multisig* transaction before the channel started. In contrast, our scheme is simple and more practical.

6.2.2.1 Secure Hardware

In our scheme, parties will have access to TEEs, which serve as isolated environments to guarantee the integrity and confidentiality of inside code and data. To capture the secure functionality of TEEs, inspired by [77] we define TEEs as a black-box program that provides interfaces exposed to users. The abstraction is given as follows. Note that, due to the scope of usage, we only capture remote attestation of TEEs and refer to [77] for a full definition.

Definition 10. *A secure hardware functionality* HW *for a class of probabilistic polynomial time (PPT) programs* $\mathcal{P}$ *includes algorithms:* Setup, Load, Run, $\mathsf{RunQuote}$, $\mathsf{QuoteVerify}$.

- $\mathsf{HW.Setup}(1^\lambda)$: The algorithm takes as input a security parameter λ, and outputs the secret key sk_{quote} and public parameters pms.

- $\mathsf{HW.Load}(pms, P)$: The algorithm loads a stateful program P into an enclave. It takes as input a program $P \in \mathcal{P}$ and public parameters pms, and outputs a new enclave handle hdl.

- $\mathsf{HW.Run}(hdl, in)$: The algorithm runs enclave. It inputs a handle hdl that relates to an enclave (running program P) and an input in, and outputs execution results out.

- $\mathsf{HW.RunQuote}(hdl, in)$: The algorithm executes programs in an enclave and generates an attestation quote. It takes as input hdl and in, and executes P on in. Then, it outputs $quote = (hdl, tag_P, in, out, \sigma)$, where tag_P is a measurement to identify the program running inside an enclave and σ is a corresponding signature.

- $\mathsf{HW.QuoteVerify}(pms, quote)$: The algorithm verifies the quote. It first exe-

Table 6.2: Featured notations

Symbol	**Item**	**Functionalities**
$\mathcal{O}$	Delegator	also known as coin owner, the person who sends the coins
$\mathcal{D}$	Delegatee	the person who receives the coins
$\mathcal{B}$	Blockchain	an ideal blockchain environment
Tx	Transaction	the transaction in blockchain network
$E_{\mathcal{O}}/E_{\mathcal{D}}$	Enclave	the Delegatee's/Delegator's Intel enclave instance
TEE	TEEs	a real TEEs environment, sometimes used in superscript for indication
hdl	Handle	an intermediate parameter when initiating TEEs
$quote$	Quote	a flag to request operations when running TEEs
pms	Parameters	intermediate parameters when running TEEs
pk/sk	Key pair	the public key and private key to encrypt/decrypt states
vk/sk_{sign}	Key pair	the key pair to identify a specific entity (delegatee)
key_{seal}	Private key	a sealing key used to export the state to the trusted storage
r	Private key	a symmetric encryption key r
ct_r	Ciphertext	the ciphertext under a symmetric encryption key with r inside TEEs
b	Account balance	the subscript $\mathsf{init}/\mathsf{deposit}/\mathsf{update}$ means the status in different stages
c	Encrypted balance	the balance that has been encrypted and transferred
σ	Signature	a valid signature, the subscript indicates its corresponding signer
HW	Hardware	a ideal and secure hardware functionality used in proofs
$\mathcal{P}$	Program space	a program that contains a set of algorithms, instantiated as P
O	Oracle	an environment that can provide ideal functionalities
$\mathcal{U}(\cdot)$	Oracle	a universal oracle can provide simulated answers
$\mathcal{S}$	Simulator	an ideal environment that can simulate some behaviours
$\mathcal{A}$	Adversary	an adversary who has some ability to launch attacks
λ	Security parameter	a type of parameter to adjust the security level of algorithms
$negl(\lambda)$	Negligible function	a function to show the negligible differences in security proofs
Exp	Experiment	an experiment that show the game and operations in proofs
PKE	Algorithm	an IND-CCA2 secure public key encryption scheme
S	Algorithm	an existentially unforgeable (EUF-CMA) signature scheme
SE	Algorithm	a IND-CPA secure symmetric encryption scheme

cutes P on in to get out. Then, it takes as input pms, $quote = (hdl, tag_P, in, out, \sigma)$, and outputs 1 if the signature σ is correct. Otherwise, it outputs 0.

Correctness. The HW scheme is correct if the following properties hold: For all program $\mathcal{P}$, all input in

- Correctness of $\mathsf{HW.Run}$: for any specific program $P \in \mathcal{P}$, the output of $\mathsf{HW.Run}(hdl, in)$ is deterministic.
- Correctness of $\mathsf{RunQuote}$ and $\mathsf{QuoteVerify}$:

$$\Pr[\mathsf{QuoteVerify}(pms, \mathsf{RunQuote}(hdl, in)) \neq 1] \leq negl(\lambda).$$

Remote attestation in TEEs provides functionality for verifying the execution and corresponding output of a certain code run inside the enclave by using a signature-based quote. Thus, remote attestation unforgeability security [77] is defined similarly to the unforgeability of a signature scheme.

Definition 11 (Remote Attestation Unforgeability (RemAttUnf))**.** *A* HW *scheme is RemAttUnf secure if all PPT adversaries, there exists a negligible function* $negl(\lambda)$ *such that*

$$\Pr\left[\mathsf{G}^{\mathsf{RemAttUnf}}_{\mathcal{A},\mathsf{S}}(\lambda) = 1\right] \leq negl(\lambda),$$

where $\mathsf{G}^{\mathsf{RemAttUnf}}_{\mathcal{A},\mathsf{HW}}(\lambda)$ *is defined as follows:*

```
$\mathsf{G}^{\mathsf{RemAttUnf}}_{\mathcal{A},\mathsf{HW}}(\lambda)$
1 : $pms \leftarrow \mathsf{HW.Setup}(1^\lambda)$;
2 : $L_{quote} \leftarrow \{\}$; // $\mathcal{C}$ initializes the list $L_{quote}$
3 : $hdl \leftarrow \mathsf{HW.Load}(pms, Q)$; // $\mathcal{A}$ loads program $P$ into enclave and gets back the handle $hdl$
4 : $quote \leftarrow \mathsf{HW.Run\&Quote}(hdl, in_{\{0,\dots,n\}})$; // $\mathcal{A}$ chooses input to run the algorithm and gets the quote
5 : $L_{quote} \leftarrow quote$; // $\mathcal{C}$ repeats step 4, and adds the quote into list $L_{quote}$
6 : $(in^\star, quote^\star) \leftarrow \mathcal{A}^{\mathcal{O}(hdl,\cdot)}(pms)$; // $\mathcal{O}(hdl,\cdot)$ works as the same with step 4
7 : return $(\mathsf{HW.QuoteVerify}(pms, quote^\star) = 1) \wedge quote^\star \notin L_{quote}$; // return true
        if the $quote^\star$ is not in $L_{quote}$ but is valid
```

6.2.3 DelegaCoin

In DelegaCoin, three types of entities are involved: coin owner (or delegator) $\mathcal{O}$, coin delegatee $\mathcal{D}$, and blockchain $\mathcal{B}$ (see Figure 6.6). The main idea behind DelegaCoin is to exploit the TEEs as trusted agents between the coin owner and coin delegatee. TEEs are used to maintain delegation policies and ensure faithful executions of the delegation protocol. In particular, TEEs guarantee that the coin owner (either honest or malicious) cannot arbitrarily spend the delegated coins. The workflow is described as follows. Firstly, both $\mathcal{O}$ and $\mathcal{D}$ initialize and run enclaves, and the owner $\mathcal{O}'s$ enclave generates an address `addr` for further transactions with a private key maintained internally. Next, $\mathcal{O}$ deploys delegation policies into the owner $\mathcal{O}'s$ enclave and deposits coins to the address `addr`. Then, $\mathcal{O}$ delegates coins to $\mathcal{D}$ by triggering the execution of delegation inside the enclave. Finally, $\mathcal{D}$ spends delegated transactions to the blockchain network $\mathcal{B}$. Note that the enclaves in our scheme are decentralized, meaning that each $\mathcal{O}$ and $\mathcal{D}$ has its own enclave without depending on a centralized agent, which satisfies the requirements of current cryptocurrency systems.

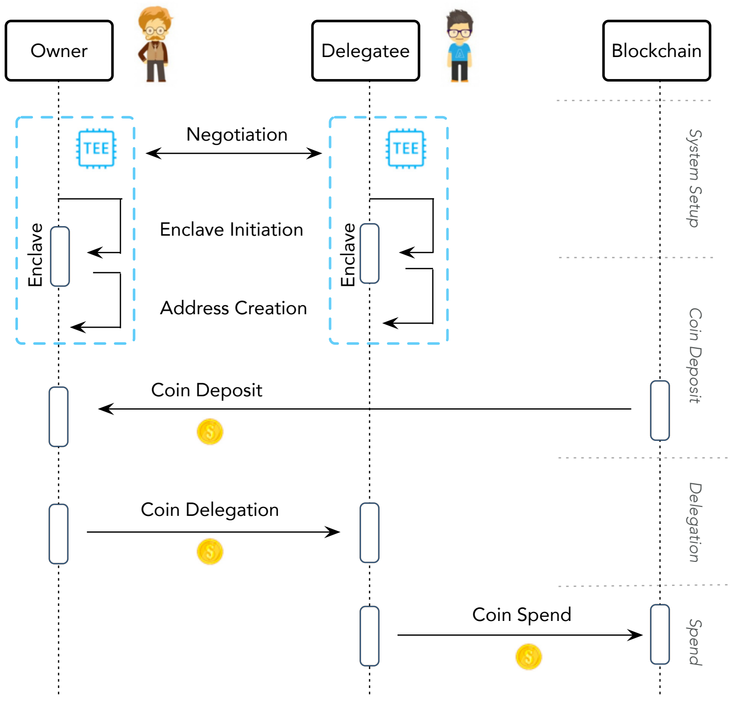


Figure 6.6: DelegaCoin workflow diagram

6.2.3.1 System Framework

System Setup. In this phase, the coin owner $\mathcal{O}$ and the delegatee $\mathcal{D}$ initialize their TEEs to provide environments for operations with respect to further delegation.

- *Negotiation.* $pms \leftarrow \mathsf{ParamGen}(1^\lambda)$: $\mathcal{O}$ agrees with $\mathcal{D}$ for pre-shared information. Here, λ is a security parameter.
- *Enclave Initiation.* $hdl_\mathcal{O}, hdl_\mathcal{D} \leftarrow \mathsf{EncvInit}(1^\lambda, pms)$: $\mathcal{O}$ and $\mathcal{D}$ initialize the enclave $E_\mathcal{O}$ and $E_\mathcal{D}$ with outputting the enclave handles $hdl_\mathcal{O}$ and $hdl_\mathcal{D}$.
- *Key Generation.* $(pk_{\mathsf{Tx}}, sk_{\mathsf{Tx}}), (pk_\mathcal{O}, sk_\mathcal{O}), key_{seal} \leftarrow \mathsf{KeyGen}^{\mathsf{TEE}}(hdl_\mathcal{O}, 1^\lambda)$ and $(pk_\mathcal{D}, sk_\mathcal{D}), (vk_{sign}, sk_{sign}), r \leftarrow \mathsf{KeyGen}^{\mathsf{TEE}}(hdl_\mathcal{D}, 1^\lambda)$: $\mathcal{O}$ and $\mathcal{D}$ run the enclaves $E_\mathcal{O}$ and $E_\mathcal{D}$ to create their internal keys. Key pair $(pk_{\mathsf{Tx}}, sk_{\mathsf{Tx}})$ is used for the transaction generation. Key pair $(pk_\mathcal{O}, sk_\mathcal{O})$ and $(pk_\mathcal{D}, sk_\mathcal{D})$ are used for remote assertion, while key_{seal} is a sealing key used to export the state to the trusted storage. The key pair (vk_{sign}, sk_{sign}) is used to identify a specific delegatee, while r is a private key for transaction encryption.
- *Quote Generation.* $quote \leftarrow \mathsf{QuoGen}^{\mathsf{TEE}}(sk_\mathcal{O}, vk_{sign}, pms)$: $\mathcal{O}$ generates a *quote* for requesting an encrypted symmetric encryption key from $\mathcal{D}$.
- *Key Provision.* $ct_r \leftarrow \mathsf{Provision}^{\mathsf{TEE}}(quote, sk_{sign}, pk_\mathcal{O}, pms)$: $\mathcal{O}$ proves to $\mathcal{D}$ that $E_\mathcal{O}$ has been instantiated with a *quote* to request an encrypted symmetric encryption key ct_r. The symmetric encryption is used to encrypt messages inside TEEs.
- *Key Extraction.* $r \leftarrow \mathsf{Extract}^{\mathsf{TEE}}(sk_\mathcal{O}, ct_r)$: $\mathcal{O}$ extracts a symmetric encryption key r from ct_r using $sk_\mathcal{O}$.
- *State Retrieval.* $b_{init} = \mathsf{Dec}^{\mathsf{TEE}}(key_{seal}, c_{init})$: Encrypted states are read back by the enclave $E_\mathcal{O}$ under key_{seal}, where b_{init} is the initial balance and c_{init} is the initial encrypted balance. This step prevents unexpected occasions that may destroy the state in TEEs memory.

Coin Deposit. The enclave $E_\mathcal{O}$ generates an address and its corresponding private key pk_{Tx} for the deposit. Afterwards, $\mathcal{O}$ sends coins to this address in the form of fund deposits.

- *Address Creation.* $addr \leftarrow \mathsf{AddrGen}^{\mathsf{TEE}}(1^{\lambda}, pk_{\mathsf{Tx}})$: $\mathcal{O}$ calls $E_{\mathcal{O}}$ to generate a transaction address $addr$. The private key sk_{Tx} of $addr$ is secretly stored inside TEEs and generated by an internal pseudo-random number.

- *Coin Deposit.* $b_{deposit} = \mathsf{Update}^{\mathsf{B}}(addr, b_{init})$: $\mathcal{O}$ generates an arbitrary transaction and transfers some coins to $addr$ as the deposits.

Coin Delegation. In this phase, neither $\mathcal{O}$ nor $\mathcal{D}$ interacts with the blockchain. $\mathcal{O}$ can instantly complete the coin delegation through offline transactions.

- *Balance Update.* $b_{update} \leftarrow \mathsf{Update}^{\mathsf{TEE}}(b_{deposit}, b_{\mathsf{Tx}})$: $E_{\mathcal{O}}$ checks current balance to ensure that it is enough for deduction. Then, $E_{\mathcal{O}}$ updates the balance.

- *Signature Generation.* $\sigma_{\mathsf{Tx}} \leftarrow \mathsf{TranSign}^{\mathsf{TEE}}(sk_{\mathsf{Tx}}, addr, b_{\mathsf{Tx}})$: $E_{\mathcal{O}}$ generates a valid signature σ_{Tx}.

- *Transaction Generation.* $\mathsf{Tx} \leftarrow \mathsf{TranGen}^{\mathsf{TEE}}(addr, b_{\mathsf{Tx}}, \sigma_{\mathsf{Tx}})$: $E_{\mathcal{O}}$ generates a transaction Tx using σ_{Tx}.

- *Coin Delegation.* $ct_{\mathsf{Tx}} \leftarrow \mathsf{TranEnc}^{\mathsf{TEE}}(\mathsf{r}, \mathsf{Tx})$: $\mathcal{O}$ sends encrypted transaction ct_{Tx} to $\mathcal{D}$.

- *State Seal.* $c_{update} \leftarrow \mathsf{Enc}^{\mathsf{TEE}}(key_{seal}, b_{update})$: Once completing the delegation, records c_{update} are permanently stored outside the enclave. If any abort or halt happens, a re-initiated enclave starts to reload missing information.

All algorithms in the step of **Coin Delegation** must be run as an atomic operation, meaning that either all algorithms finish or none of them finish. A hardware Root of Trust can guarantee this, and we refer to [61] for more details.

Coin Spend. $\mathsf{Tx} \leftarrow \mathsf{TranDec}^{\mathsf{TEE}}(r, ct_{\mathsf{Tx}})$: $\mathcal{D}$ decrypts $\mathsf{ct}_{\mathsf{Tx}}$ with r, and then spends Tx by forwarding it to the blockchain network.

Correctness. The DelegaCoin scheme is correct if the following properties hold: For all Tx, $b_{deposit}$, b_{update} and b_{Tx}.

- Correctness of Update:

$$\Pr\left[b_{\mathsf{Tx}} \neq (b_{deposit} - b_{update})\right] \leq negl(\lambda).$$

- Correctness of Seal:

$$\Pr[\mathsf{Dec}^{\mathsf{TEE}}(key_{seal}, \mathsf{Enc}^{\mathsf{TEE}}(key_{seal}, b_{init})) \neq b_{init}] \leq negl(\lambda).$$

- Correctness of $\mathsf{Delegation}$:

$$\Pr[\mathsf{TranDec}^{\mathsf{TEE}}(r, \mathsf{TranEnc}^{\mathsf{TEE}}(r, \mathsf{Tx})) \neq \mathsf{Tx}] \leq negl(\lambda).$$

6.2.3.2 Oracles for Security Definitions

We now define oracles to simulate an honest owner and a delegatee for further security definitions and proofs. Each oracle maintains a series of (initially empty) sets $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{C}$ which will be used later. Here, we use ($\mathsf{instruction}$;$parameter$) to denote both the instructions and inputs of oracles.

Honest Owner Oracle $\mathsf{O}^{\mathsf{owner}}$: This oracle gives the adversary access to honest owners. An adversary $\mathcal{A}$ can obtain newly delegated transactions or sealed storage with his customized inputs. The oracle provides the following interfaces.

- On input ($\mathsf{signature\ creation}$; $addr$), the oracle checks whether a tuple $(addr, \sigma_{\mathsf{Tx}}) \in \mathcal{R}_1$ exists, where $addr$ is an input of transactions. If successful, the oracle returns σ_{Tx} to $\mathcal{A}$; otherwise, it computes $\sigma_{\mathsf{Tx}} \leftarrow \mathsf{TranSign}^{\mathsf{TEE}}(sk_{\mathsf{Tx}}, addr, b_{\mathsf{Tx}})$ and adds $(addr, \sigma_{\mathsf{Tx}})$ to $\mathcal{R}_1$, and then returns σ_{Tx} to $\mathcal{A}$.
- On input ($\mathsf{quote\ generation}$; vk_{sign}), the oracle checks if a tuple $(vk_{sign}, quote) \in \mathcal{R}_2$ exists. If successful, the oracle returns $quote$ to $\mathcal{A}$; otherwise, it computes $quote \leftarrow \mathsf{QuoGen}^{\mathsf{TEE}}(sk_{\mathcal{O}}, vk_{sign}, pms)$ and adds $(vk_{sign}, quote)$ to $\mathcal{R}_2$, and then returns $quote$ to $\mathcal{A}$.

Honest Delegatee Oracle $\mathsf{O}^{\mathsf{delegatee}}$: This oracle gives the adversary access to honest delegatees. The oracle provides the following interfaces.

- On input ($\mathsf{key\ provision}$; $quote$), the oracle checks whether a tuple $(quote, ct_r) \in \mathcal{C}$ exists. If successful, the oracle returns ct_r to $\mathcal{A}$; otherwise, it computes $ct_r \leftarrow \mathsf{Provision}^{\mathsf{TEE}}(quote, sk_{sign}, pk_{\mathcal{O}}, pms)$, adds $(quote, ct_r)$ to $\mathcal{C}$, and then returns $(quote, ct_r)$ to $\mathcal{A}$.

HW Oracle: This oracle gives the adversary access to honest hardware. The oracle provides interfaces as defined as in Definition 10. Note that, to ensure that anything $\mathcal{A}$ sees in the real world can be simulated in the ideal experiment, we require that an adversary get access to HW **Oracle** through $\mathsf{O}^{\mathsf{delegatee}}$ and $\mathsf{O}^{\mathsf{owner}}$ rather than directly interacting with HW Oracle.

6.2.3.3 Threat Model and Assumptions

As for involved entities, we assume that $\mathcal{O}$ attempts to delegate coins to the delegatee. Each party may potentially be malicious. $\mathcal{O}$ may maliciously delegate an exceptional transaction, represented as sending the same transaction to multiple delegatees or spending delegated transactions before $\mathcal{D}$ spends them. $\mathcal{D}$ may also attempt to assemble an invalid transaction or double spend delegated coins. We also assume the blockchain $\mathcal{B}$ is robust and publicly accessible.

Regarding devices, we assume that TEEs are secure, which means that an adversary cannot access the enclave runtime memory and their hardware-related keys (*e.g.,* sealing key or attestation key). In contrast, we do not assume the components outside TEEs are trusted. For example, the adversary may control the operating system or high-level privileged software.

6.2.3.4 Security Goals.

DelegaCoin aims to employ TEEs to provide a secure delegatable cryptocurrency system. In brief, TEEs prevent malicious delegation in three aspects: (1) The private key of a delegated transaction and the delegated transaction itself are protected against the public. If an adversary learns any knowledge about the private key or the delegated transaction, he may spend the coin before the delegatee uses it; (2) The delegation executions are correctly performed. In particular, the spendable number of delegated coins must be less than (or at least equal to) that of original coins; (3) The delegation records are securely stored to guarantee consistency considering accidental TEEs failures or malicious TEEs compromises. DelegaCoin is secure if adversaries cannot learn any knowledge about the private key, the delegated transaction, and the sealed storage.

To capture such security properties, we formalize our system through a game inspired by [21]. In our game, a PPT adversary attempts to distinguish between a real world and a simulated (ideal) world. In the real world, the DelegaCoin algorithm works as defined in the construction. The adversary is allowed to access the transaction-related secret messages created by honest users through oracles, as in Definition 6.2.3.2. Obviously, the ideal world does not leak any useful information to the adversary. Since we model additional information explicitly to respond to the adversary, we construct a polynomial-time simulator $\mathcal{S}$ that can *fake* the additional information corresponding to the real result, but with respect to the fake TEEs. Thus, a universal oracle $\mathcal{U}(\cdot)$ in the ideal world is introduced to simulate the corresponding answers of $\mathcal{A}$ called in oracles in the real world. We give a formal model as follows, in which these two experiments begin with the same setup assumption.

Definition 12 (Security). *DelegaCoin is simulation-secure if for all PPT adversaries $\mathcal{A}$, there exists a stateful PPT simulator $\mathcal{S}$ and a negligible function $negl(\lambda)$ such that the probability of that $\mathcal{A}$ distinguishes between $\mathsf{Exp}^{\mathsf{real}}_{\mathcal{A},\mathsf{DelegaCoin}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{idea}}_{\mathcal{A},\mathsf{DelegaCoin}}(\lambda)$ is negligible, i.e.,*

$$\left|\mathsf{Pr}[\mathsf{Exp}^{\mathsf{real}}_{\mathcal{A},\mathsf{DelegaCoin}}(\lambda)=1]-\mathsf{Pr}[\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{A},\mathsf{DelegaCoin}}(\lambda)=1]\right|\leq negl(\lambda).$$

6.2.4 Formal Protocols

In this section, we present a formal model for our electronic cash system by utilizing the syntax of the HW model. In particular, we model the interactions of Intel SGX enclaves as calling to the HW functionality defined in Definition 10. The formal protocols are provided as follows.

The owner enclave program $\mathsf{P}_{\mathcal{O}}$ is defined as follows. The value tag_P is a measurement of the program $\mathsf{P}_{\mathcal{O}}$, and it is hardcoded in the static data of $\mathsf{P}_{\mathcal{O}}$. Let $\mathsf{state}_{\mathcal{O}}$ denote an internal state variable.

$\mathsf{P}_{\mathcal{O}}$:

```
$\mathsf{Exp}^{\mathsf{real}}_{\mathcal{A},\mathsf{DelegaCoin}}(\lambda)$
1 :  $pms \leftarrow \mathsf{ParamGen}(1^\lambda)$
2 :  $hdl_\mathcal{O}, hdl_\mathcal{D} \leftarrow \mathsf{EncvInit}(1^\lambda, pms)$
3 :  $(pk_\mathsf{Tx}, sk_\mathsf{Tx}), (pk_\mathcal{O}, sk_\mathcal{O}), key_{seal} \leftarrow \mathsf{KeyGen}^\mathsf{TEE}(hdl_\mathcal{O}, 1^\lambda)$
4 :  $(pk_\mathcal{D}, sk_\mathcal{D}), (vk_{sign}, sk_{sign}), \mathsf{r} \leftarrow \mathsf{KeyGen}^\mathsf{TEE}(hdl_\mathcal{D}, 1^\lambda)$
5 :  $quote \leftarrow \mathcal{A}(hdl_\mathcal{O}, vk_{sign}, pms)$
6 :  $ct_r \leftarrow \mathcal{A}^{\mathsf{Provision}^\mathsf{TEE}(sk_{sign})}(hdl_\mathcal{D}, quote, pk_\mathcal{O}, pms)$
7 :  $r \leftarrow \mathcal{A}^{\mathsf{Extract}^\mathsf{TEE}(sk_\mathcal{O})}(hdl_\mathcal{O}, ct_r)$
......................Setup Completed......................
8 :  $b_{init} = \mathsf{Dec}^\mathsf{TEE}(hdl_\mathcal{O}, key_{seal}, c_{init})$
9 :  $addr \leftarrow \mathsf{AddrGen}^\mathsf{TEE}(1^\lambda, pk_\mathsf{Tx})$
10 : $b_{deposit = \mathsf{Update}^\mathsf{B}(addr, b_{init})}$
11 : $b_{update \leftarrow \mathsf{Update}^\mathsf{TEE}(hdl_\mathcal{O}, b_{deposit}, b_\mathsf{Tx})}$
12 : $\sigma_\mathsf{Tx} \leftarrow \mathcal{A}^{\mathsf{TranSign}^\mathsf{TEE}(sk_\mathsf{Tx})}(hdl_\mathcal{O}, addr, b_\mathsf{Tx})$
13 : $\mathsf{Tx} \leftarrow \mathsf{TranGen}^\mathsf{TEE}(hdl_\mathcal{O}, addr, b_\mathsf{Tx}, \sigma_\mathsf{Tx})$
14 : $ct_\mathsf{Tx} \leftarrow \mathcal{A}^{\mathsf{TranEnc}^\mathsf{TEE}(\mathsf{r})}(hdl_\mathcal{O}, \mathsf{Tx})$
15 : $c_{update} = \mathcal{A}^{\mathsf{Enc}^\mathsf{TEE}(key_{seal})}(hdl_\mathcal{O}, b_{update})$
..................Delegation Completed..................
16 : $\mathsf{Tx} \leftarrow \mathsf{TranDec}^\mathsf{TEE}(hdl_\mathcal{D}, r, ct_\mathsf{Tx})$
17 : **return** $(\mathsf{Tx}, c_{update})$
```

```
$\mathsf{Exp}^{\mathsf{ideal}}_{\mathcal{A},\mathsf{DelegaCoin}}(\lambda)$
1 :  $pms \leftarrow \mathsf{ParamGen}(1^\lambda)$
2 :  $hdl^\star_\mathcal{O}, hdl^\star_\mathcal{D} \leftarrow \mathcal{S}(1^\lambda, pms)$
3 :  $(pk_\mathsf{Tx}, sk_\mathsf{Tx}), (pk_\mathcal{O}, sk_\mathcal{O}), key_{seal} \leftarrow \mathsf{KeyGen}^\mathsf{TEE}(hdl^\star_\mathcal{O}, 1^\lambda)$
4 :  $(pk_\mathcal{D}, sk_\mathcal{D}), (vk_{sign}, sk_{sign}), r \leftarrow \mathsf{KeyGen}^\mathsf{TEE}(hdl^\star_\mathcal{D}, 1^\lambda)$
5 :  $quote \leftarrow \mathcal{A}(hdl^\star_\mathcal{O}, vk_{sign}, pms)$
6 :  $ct_r \leftarrow \mathcal{A}^{\mathcal{S}^{\mathcal{U}(\cdot)}}(hdl^\star_\mathcal{D}, quote, \mathsf{pk}_\mathcal{O}, pms)$
7 :  $r \leftarrow \mathcal{A}^{\mathcal{S}^{\mathcal{U}(\cdot)}}(hdl^\star_\mathcal{O}, ct_r)$
......................Setup Completed......................
8 :  $b_{init} \leftarrow \mathcal{S}(hdl_\mathcal{O}, key_{seal}, c_{init})$
9 :  $addr \leftarrow \mathcal{S}(1^\lambda, pk_\mathsf{Tx})$
10 : $b_{deposit} = \mathcal{S}(addr, b_{init})$
11 : $b_{update} \leftarrow \mathcal{S}(hdl^\star_\mathcal{O}, b_{deposit}, 1^{|b_\mathsf{Tx}|})$
12 : $\sigma_\mathsf{Tx} \leftarrow \mathcal{A}^{\mathcal{S}^{\mathcal{U}(\cdot)}}(hdl^\star_\mathcal{O}, addr, b_\mathsf{Tx})$
13 : $\mathsf{Tx} \leftarrow \mathcal{S}(hdl^\star_\mathcal{O}, addr, 1^{|b_\mathsf{Tx}}|, \sigma_\mathsf{Tx})$
14 : $ct_\mathsf{Tx} \leftarrow \mathcal{A}^{\mathcal{S}^{\mathcal{U}(\cdot)}}(hdl^\star_\mathcal{O}, 1^{|\mathsf{Tx}|})$
15 : $c_{update} = \mathcal{A}^{\mathcal{S}^{\mathcal{U}(\cdot)}}(hdl^\star_\mathcal{O}, 1^{|b_{update}|})$
..................Delegation Completed..................
16 : $\mathsf{Tx} \leftarrow \mathcal{S}(hdl_{\mathcal{D}^\star}, r, ct_\mathsf{Tx})$
17 : **return** $(\mathsf{Tx}, c_{update})$
```

- On input (“init setup”, sid, vk_{sign}[2]):
  - Run $(pk_\mathcal{O}, sk_\mathcal{O}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and key_{seal}[3] $\leftarrow \mathsf{SE.KeyGen}(1^\lambda)$.
  - Update $state$ to $(sk_\mathcal{O}, vk_{sign})$ and output $(pk_\mathcal{O}, sid, vk_{sign})$.
- On input (“complete setup”, sid, ct_r, σ_r):
  - Look up the $state_\mathcal{O}$ to obtain the entry $(sk_\mathcal{O}, sid, vk_{sign})$. If no entry exists for sid, output $\perp$.
  - Receive the (sid, vk_{sign}) from $\mathcal{O}$ and check if vk_{sign} matches with the one in $state_\mathcal{O}$. If not, output $\perp$.
  - Verify signature $b \leftarrow \mathsf{S.Verify}(vk_{sign}, \sigma_r, (sid, ct_r))$. If $b = 0$, output $\perp$.
  - Run $r \leftarrow \mathsf{PKE.Dec}(sk_\mathcal{O}, ct_r)$.
  - Add the tuple (r, sid, vk_{sign}) to $state_\mathcal{O}$.

[2]We assume that the combination $(sid,\ vk_{sign})$, represented as the identity of a delegatee, has already been distributed before the system setup.

[3]Multiple enclaves from the same signing authority can derive the same key, since seal key is based on the enclave’s certificate-based identity.

- On input (“state retrieval”, sid):
  - Retrieve identity-balance pair (sid, c_{init}) from the sealed storage.
  - Run $b_{init} = \mathsf{SE.Dec}(key_{seal}, c_{init})$ and update $state_{\mathcal{O}}$ to (sid, b_{init})
- On input (“address generation”, 1^{λ}):
  - Run $(sk_{\mathsf{Tx}}, pk_{\mathsf{Tx}}) \leftarrow \mathsf{S.KeyGen}(1^{\lambda})$ and $addr \leftarrow \mathsf{AddrGen}^{\mathsf{TEE}}(1^{\lambda}, pk_{\mathsf{Tx}})$.
  - Update $(sk_{\mathsf{Tx}}, addr)$ to $state_{\mathcal{O}}$ and output $(pk_{\mathsf{Tx}}, addr)$.
- On input (“transaction generation”, $addr$):
  - Retrieve the private key sk_{Tx}.
  - Run $\sigma_{\mathsf{Tx}} \leftarrow \mathsf{S.Sign}(sk_{\mathsf{Tx}}, addr, b_{\mathsf{Tx}})$ and output a signature σ_{Tx}.
  - Run $\mathsf{Tx} \leftarrow \mathsf{TranGen}(addr, b_{\mathsf{Tx}}, \sigma_{\mathsf{Tx}})$ and update (sid, Tx) to $state_{\mathcal{O}}$.
- On input (“state update”, $addr$):
  - Check $b_{deposit}$ and b_{Tx}. If $b_{deposit} < b_{\mathsf{Tx}}$, output $\perp$.
  - Run $b_{update} \leftarrow \mathsf{Update}(b_{deposit}, b_{\mathsf{Tx}})$.
- On input (“start delegation”, $addr$):
  - Retrieve the provision private key r and Tx from $state_{\mathcal{O}}$.
  - Run $ct_{\mathsf{Tx}} \leftarrow \mathsf{SE.Enc}(r, \mathsf{Tx})$.
- On input (“state seal”, $addr$):
  - Run $c_{update} = \mathsf{SE.Enc}(key_{seal}, b_{update})$ and update $state_{\mathcal{O}}$ to $(addr, b_{update})$.
  - Store $addr$ and c_{update} to the sealed storage.

The delegatee enclave program $\mathsf{P}_{\mathcal{D}}$ is defined as follows. The value $tag_{\mathcal{D}}$ is the measurement of the program $\mathsf{P}_{\mathcal{D}}$, and it is hardcoded in the static data of $\mathsf{P}_{\mathcal{D}}$. Let $state_{\mathcal{D}}$ denote an internal state variable. Also, the security parameter λ is hardcoded into the program.

$\mathsf{P}_{\mathcal{D}}$:

- On input (“init setup”, 1^λ):
  - Generate a session ID, $sid \leftarrow \{0,1\}^\lambda$.
  - Run $(pk_{\mathcal{D}}, sk_{\mathcal{D}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, and $(vk_{sign}, sk_{sign}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$.
  - Update $state_{\mathcal{D}}$ to $(sk_{\mathcal{D}}, sk_{sign})$ and output $(sid, pk_{\mathcal{D}}, vk_{sign})$.
- On input (“provision”, $quote, pk_{\mathcal{O}}, pms$):
  - Parse $quote = (hdl_{\mathcal{O}}, tag_P, in, out, \sigma)$, check that $tag_P == tag_{\mathcal{O}}$. If not, output $\perp$.
  - Parse $out = (sid, pk_{\mathcal{O}})$, and run $b \leftarrow \mathsf{HW.QuoteVerify}(pms, quote)$ on $quote$. If $b = 0$, output $\perp$.
  - Select a random number r and compute the algorithm $ct_r = \mathsf{PKE.Enc}(pk_{\mathcal{O}}, r)$ and $\sigma_r = \mathsf{S.Sign}(sk_{sign}, (sid, ct_r))$ and output (sid, ct_r, σ_r).
- On input (“complete delegation”, ct_{Tx}):
  - Retrieve r from $state_{\mathcal{D}}$.
  - Run $\mathsf{Tx} \leftarrow \mathsf{SE.Dec}(r, ct_{\mathsf{Tx}})$.

Setup. The following steps are based on the completed initialization of programs of the delegator $\mathsf{P}_{\mathcal{O}}$ and delegatee $\mathsf{P}_{\mathcal{D}}$. The delegatee $\mathcal{D}$ runs $hdl_{\mathcal{D}} \leftarrow \mathsf{HW.Load}(pms, P_{\mathcal{D}})$ and $(vk_{sign}, pk_{\mathcal{D}}) \leftarrow \mathsf{HW.Run}(hdl_{\mathcal{D}}, (\text{“init setup”}, 1^\lambda))$. Afterwards, $\mathcal{D}$ sends vk_{sign} to the delegator $\mathcal{O}$. Next, $\mathcal{O}$ runs $hdl_{\mathcal{O}} \leftarrow \mathsf{HW.Load}(pms, \mathsf{P}_{\mathcal{O}})$ to load the handle. Meanwhile, $\mathcal{O}$ calls $quote \leftarrow \mathsf{HW.Run\&Quote}(hdl_{\mathcal{O}}, (\text{“init setup”}, sid, vk_{sign}))$, and sends a $quote$ to $\mathcal{D}$. Then, $\mathcal{D}$ calls $(sid, ct_r, \sigma_r) \leftarrow \mathsf{HW.Run}(hdl_{\mathcal{D}}, (\text{“provision”}, quote, pk_{\mathcal{O}}, pms))$, and sends (sid, ct_r, σ_r) to $\mathcal{O}$, and $\mathcal{O}$ calls $\mathsf{HW.Run}(hdl_{\mathcal{O}}, (\text{“complete setup”}, vk_{sign}))$. At the end of completing setup, $\mathcal{O}$'s enclave $E_{\mathcal{O}}$ obtains the private key r used for the transaction delegation.

Deposit. $\mathcal{O}$ calls $c_{init} \leftarrow \mathsf{HW.Run}(hdl_{\mathcal{O}}, (\text{“state retrieval”}, sid))$. If c_{init} does not exist or equals to 0, $\mathcal{O}$ calls $addr \leftarrow \mathsf{HW.Run}(hdl_{\mathcal{O}}, (\text{“address generation”}, 1^\lambda))$

to create a new address $addr$. Then, $\mathcal{O}$ transfers coins to $addr$ through a normal blockchain transaction.

Delegation. $\mathcal{O}$ parses $hdl_{\mathcal{O}}$ and calls $E_{\mathcal{O}}$. Then, $E_{\mathcal{O}}$ retrieves $addr$. Afterwards, it calls $b_{update} \leftarrow \mathsf{HW.Run}(hdl_{\mathcal{O}}, (\text{“state update”}, addr))$. If the update algorithm returns false or failure, $E_{\mathcal{O}}$ aborts the following operations. Otherwise, it looks up the state to obtain sk_{Tx}, and runs $\mathsf{Tx} \leftarrow \mathsf{HW.Run}(hdl_{\mathcal{O}}, (\text{“transaction generation”}, addr))$ and outputs a transaction Tx. After that, the delegator's enclave $E_{\mathcal{O}}$ retrieves r and runs $ct_{\mathsf{Tx}} \leftarrow \mathsf{HW.Run}(hdl_{\mathcal{O}}, (\text{“start delegation”}, addr))$. Finally, $\mathcal{O}$ sends ct_{Tx} to $\mathcal{D}$.

Spend. $\mathcal{D}$ runs $\mathsf{Tx} \leftarrow \mathsf{HW.Run}(hdl_{\mathcal{D}}, (\text{“complete delegation”}, ct_{\mathsf{Tx}}))$. After that, $\mathcal{D}$ spends the received transaction Tx by forwarding it to the blockchain network. Then, a blockchain node parses $\mathsf{Tx} = (addr, pk_{\mathsf{Tx}}, metadata, \sigma_{\mathsf{Tx}})$ and runs $b \leftarrow \mathsf{S.Verify}^{B}(pk_{\mathsf{Tx}}, \sigma_{\mathsf{Tx}})$. If $b = 0$, output $\perp$. Otherwise, the node broadcasts Tx to other blockchain nodes.

6.2.5 Security Analysis

Theorem 3 (Security). *Assume that* SE *is IND-CPA secure,* PKE *is IND-CCA2 secure,* S *holds the EUF-CMA security, and TEEs are secure as in Definition 10, our DelegaCoin scheme is simulation-secure.*

Inspired by [77, 141], we use a simulation-based paradigm to conduct security analysis and explain the crux of our security proof as follows. We first construct a simulator $\mathcal{S}$ that can simulate the challenge responses in the real world. It provides the adversary $\mathcal{A}$ with a simulated delegated transaction, a simulated quote and sealed storage. The information that $\mathcal{A}$ can obtain is merely the instruction code and oracle responses queried by $\mathcal{A}$ in the real experiment. At a high level, the proof idea is simple: $\mathcal{S}$ encrypts zeros as the challenge message. In the ideal experiment, $\mathcal{S}$ intercepts $\mathcal{A}$'s queries to user oracle and provides simulated responses. It uses its $\mathcal{U}(\cdot)$ oracle to simulate oracles in the real world and sends the response back to $\mathcal{A}$ as the simulated oracle output. $\mathcal{U}(\cdot)$ and $\mathcal{S}$'s algorithms are described as follows.

Pre-processing phase. $\mathcal{S}$ simulates the pre-processing phase similar to in the real world. It first runs $\mathsf{ParamGen}(1^{\lambda})$ and records system parameters pms that are generated during the process. Then, it calls $\mathsf{EncvInit}(1^\lambda, \mathsf{pms})$ to create simulated enclave instances. $\mathcal{S}$ also creates empty lists $\mathcal{R}_1^\star$, $\mathcal{R}_2^\star$, $\mathcal{C}^\star$, $\mathcal{K}^\star$ and $\mathcal{L}^\star$.

KeyGen$^\star(1^\lambda)$ When $\mathcal{A}$ makes a query to $\mathbf{KeyGen(1^\lambda)}$ oracle, $\mathcal{S}$ responds the same way as in the real world except that now $\mathcal{S}$ stores all public keys queried in a list $\mathcal{K}^\star$. That is, $\mathcal{S}$ does the following algorithms.

- Compute and output $(pk_{\mathcal{O}}, sk_{\mathcal{O}}), (pk_{\mathsf{Tx}}, sk_{\mathsf{Tx}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.

- Store the keys $(pk_{\mathcal{O}}, sk_{\mathcal{O}}), (pk_{\mathsf{Tx}}, sk_{\mathsf{Tx}})$ in the list $\mathcal{K}^\star$.

Enc$^\star(key^\star, 1^{|msg^\star|})$[4] When $\mathcal{A}$ provides the challenge message $msg^\star$ for symmetric encryption, the following algorithm is used by $\mathcal{S}$ to simulate the challenge ciphertext.

- Compute and output $ct^\star \leftarrow \mathsf{SE.Enc}(key^\star, 1^{|msg^\star|})$.

- Store $ct^\star$ in the list $\mathcal{L}^\star$.

$\mathsf{O}^{owner\star}(\mathsf{signature\ creation};addr)$. When $\mathcal{A}$ takes a query to $\mathsf{O}^{\mathbf{owner}}$ oracle, $\mathcal{S}$ responds the same way as in the real world, except that $\mathcal{S}$ now stores all the $addr$ corresponding to the user's queries in a list $\mathcal{R}_1^\star$. That is, $\mathcal{S}$ does the following algorithms.

- Call $\mathsf{O}^{\mathbf{owner}}$ oracle with an input $(\mathsf{signature\ creation}; addr)$ and output σ_{Tx}.

- Store $(addr, \sigma_{\mathsf{Tx}})$ in the list $\mathcal{R}_1^\star$.

$\mathsf{O}^{owner\star}(\mathsf{quote\ generation}; vk_{sign})$. When $\mathcal{A}$ takes a query to the $\mathsf{O}^{\mathbf{owner}}$ oracle, $\mathcal{S}$ responds the same way as in the real world, excepting that $\mathcal{S}$ now stores all the $quote$ corresponding to the user's queries in a list $\mathcal{R}_2^\star$. That is, $\mathcal{S}$ does the following algorithms.

- Call the $\mathsf{O}^{\mathbf{owner}}$ oracle with an input $(\mathsf{quote\ generation};vk_{sign})$ and output $quote$.

- Store $(vk_{sign}, quote)$ in the list $\mathcal{R}_2^\star$.

$\mathsf{O}^{\mathbf{delegatee}\star}(\mathsf{key\ provision}; quote)$. When $\mathcal{A}$ takes a query to the $\mathsf{O}^{\mathbf{delegatee}}$ oracle, $\mathcal{S}$ responds the same way as in the real world, except that $\mathcal{S}$ now stores all the

[4]Here, $msg^\star$ is a wildcard character, representing any messages.

quote corresponding to the user's queries in a list $\mathcal{C}^\star$. That is, $\mathcal{S}$ does the following algorithm.

- Call $\mathsf{O}^{\mathbf{delegatee}}$ oracle with an input ($\mathsf{key\ provision}; quote$) and output ct_r.
- Store $(quote, ct_r)$ in the list $\mathcal{C}^\star$.

For a PPT simulator $\mathcal{S}$, we prove the security by showing that the view of an adversary $\mathcal{A}$ in the real world is computationally indistinguishable from its view in the ideal world. Specifically, we establish a series of **Hybrids** that $\mathcal{A}$ cannot be distinguished with non-negligible advantage.

Hybrid 0. $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{DelegaCoin}}(1^\lambda)$ runs.

Hybrid 1. As in *Hybrid 0*, except that $\mathbf{KeyGen}^\star(\mathbf{1}^\lambda)$ run by $\mathcal{S}$ is used to generate secret keys instead of $\mathbf{KeyGen}(\mathbf{1}^\lambda)$.

Proof: The proof is straightforward, storing corresponding answers in lists does not affect the view of $\mathcal{A}$. Thus, *Hybrid 1* is indistinguishable from *Hybrid 0*. $\square$

Hybrid 2. As in *Hybrid 1*, except that $\mathcal{S}$ maintains a list $\mathcal{C}^\star$ of all $quote = (hdl, tag_P, in, out, \sigma)$ output by $\mathsf{HW.Run\&Quote}(hdl_{\mathcal{O}}, in)$. And, when $\mathsf{HW.QuoteVerify}(hdl_{\mathcal{D}}, pms, quote)$ is called, $\mathcal{S}$ outputs $\perp$ if $quote \notin \mathcal{R}_2$. ($\mathcal{R}_2$ is a quote returned by the real world oracles that $\mathcal{A}$ has queried as defined in Section 6.2.3.2).

Proof: If a fake *quote* is produced, then the step $\mathsf{HW.QuoteVerify}(hdl_{\mathcal{O}}, pms, quote)$ in the real world would make it output $\perp$. Thus, *Hybrid 2* differs from *Hybrid 1* only when $\mathcal{A}$ can produce a valid *quote* without knowing $sk_{\mathcal{O}}$. Assume that there is an adversary $\mathcal{A}$ can distinguish between *Hybrid 2* and *Hybrid 1*. Obviously, this can be transformed to the ability against Remote Attestation, as in Definition 11. However, our assumption relies on the fact that the security of Remote Attestation holds. Therefore, *Hybrid 2* is indistinguishable from *Hybrid 1*. $\square$

Hybrid 3. As in *Hybrid 2*, except that when the $\mathsf{O}^{\mathbf{delegatee}}$ oracle calls $\mathsf{HW.Run}(hdl_{\mathcal{D}}, (\text{"provision"}, quote, pk_{\mathcal{O}}, pms))$, $\mathcal{S}$ replaces ct_r as an encryption of zeros, saying $\mathsf{PKE.Enc}(pk_{\mathcal{O}}, 1^{|r|})$.

Proof: The IND-CCA2 challenger provides the challenge public key $pk_{\mathcal{O}}$, and an adversary $\mathcal{A}$ provides two messages r and $1^{|r|}$, and further, the challenge returns an encryption of r or an encryption of $1^{|r|}$, which is represented $ct_{\star}$. $\mathcal{S}$ sets $ct_{\star}$ as the real output ct_r. For $ct_r \in \mathcal{C}$, $\mathcal{S}$ can use $\mathsf{O}^{\mathsf{delegatee}}$ as it used in the real world. However, for $ct_r \notin \mathcal{C}$, $\mathcal{S}$ neither has the oracles nor has $sk_{\mathcal{O}}$. But, the decryption oracle offered by the IND-CCA2 challenger can be used for any $ct_r \notin \mathcal{C}$. Under this condition, if $\mathcal{A}$ can still distinguish *Hybrid 3* and *Hybrid 2*, we can forward the answer corresponding to $\mathcal{A}$'s answer to the IND-CCA2 challenger. If $\mathcal{A}$ can distinguish between these two hybrids with a non-negligible probability, the IND-CCA2 security of PKE (see Definition 3) can be broken with a non-negligible probability. □

Hybrid 4. As in *Hybrid 3*, except that $\mathcal{S}$ maintains a list $\mathcal{R}_1^{\star}$ of all transaction signature σ_{Tx} output by $\mathsf{O}^{owner}(\mathsf{signature\ creation}; \mathsf{addr})$ for $addr \in \mathcal{R}_1$. When $b \leftarrow \mathsf{S}.verify^B(pk_{\mathsf{Tx}}, \sigma_{\mathsf{Tx}})$ is called, $\mathcal{S}$ outputs $\perp$, if $(addr, \sigma_{\mathsf{Tx}})$, as components of a Tx, do not belong to $\mathcal{R}_1$. Namely, $(\mathsf{addr}, \sigma_{\mathsf{Tx}}) \notin \mathcal{R}_1$.

Proof: If a transaction is given with an invalid signature, then the step $\mathsf{S.Verify}^B(pk_{\mathsf{Tx}}, \sigma_{\mathsf{Tx}})$ in the real world, would make it output $\perp$. Thus, *Hybrid 4* differs from *Hybrid 3* only when $\mathcal{A}$ can produce a valid signature on $addr$ which has never appeared before in the communication between $\mathcal{A}$ and oracles. Let $\mathcal{A}$ be an adversary who can distinguish *Hybrid 4* and *Hybrid 3*. We use it to break the EUF-CMA [94] security of the signature scheme $\mathcal{S}$. We get a verification key pk_{Tx} and access to $\mathsf{S.Sign}(sk_{\mathsf{Tx}}, \cdot)$ oracle from the EUF-CMA challenger. Whenever $\mathcal{S}$ signs a message using sk_{Tx}, it uses the $\mathsf{S.Sign}(sk_{\mathsf{Tx}}, \cdot)$ oracle. Also, our construction does not need direct access to sk_{Tx} sign; it is used only to sign messages for the oracle provided by the challenger. Now, if $\mathcal{A}$ can distinguish two hybrids, the only reason is that $\mathcal{A}$ generates a valid signature σ_{Tx}. Then, we can send such signature as forgery to the EUF-CMA [94] challenger. □

Hybrid 5. As shown in *Hybrid 4*, except that when the $\mathsf{O}^{\mathbf{owner}}$ oracle calls the function $\mathsf{HW.Run}(hdl_{\mathcal{O}}, (\text{“start delegation”}, addr))$, $\mathcal{S}$ replaces Enc with $\mathsf{Enc}^{\star}$.

Lemma 2. *If symmetric encryption scheme* SE *is IND-CPA secure, Hybrid 5 is indistinguishable from Hybrid 4.*

Proof: Whenever $\mathcal{A}$ provides a transaction Tx of its choice, $\mathcal{S}$ replies with zeros, e.g., $\mathsf{SE.Enc}(1^{|r|})$, which is shown as follows.

replace the encryption with zeros

	$\mathsf{Hybrid}_4(1^n)$	$\mathsf{Hybrid}_5(1^n)$
1 :	$\ldots$	$\ldots$
2 :	$ct_{\mathsf{Tx}} \leftarrow \mathsf{SE.Enc}(r, \mathsf{Tx})$	$ct_{\mathsf{Tx}} \leftarrow \mathsf{SE.Enc}(1^{|r|}, \mathsf{Tx})$
3 :	$\ldots$	$\ldots$

Assume that there is an adversary $\mathcal{A}$ who can distinguish the environments of *Hybrid 5* and *Hybrid 4*. Then, we build an adversary $\mathcal{A}^\star$ against IND-CPA secure of SE. Given a transaction Tx, if $\mathcal{A}$ distinguishes the encryption of r from the encryption of $1^{|r|}$, we forward the corresponding answer to the IND-CPA challenger. □

Hybrid 6. As in *Hybrid 5*, except that when the $\mathcal{A}$ calls $\mathsf{HW.Run}(\mathsf{hdl}_{\mathcal{O}}, (\text{"state seal"}, addr))$, $\mathcal{S}$ replaces Enc with $\mathsf{Enc}^\star$.

Proof: The Indistinguishability between *Hybrid 6* and *Hybrid 5* can be directly reduced to the IND-CPA property of SE, which is similar to Lemma 2 □

6.2.6 Implementation

We implement a prototype with three types of entities: *the owner node*, *the delegatee node*, and *the blockchain system*. The owner node and the delegatee node are separately running on two computers. The codes of these nodes are both developed in C++ using Intel® SGX SDK 1.6 under the operating system of Ubuntu 20.04.1 LTS. For the blockchain network, we adopt Bitcoin testnet [3] as our prototype platform. Specifically, we employ SHA-256 as the hash algorithm and ECDSA [109] with *secp256k1* [177] as the initial setting to sign transactions, which is the same as the configuration of the Bitcoin testnet.

Algorithm 1 Remote Attestation

```
Input: request$(quote, pms)$
Output: $b = 0/1$
parse the received $quote$ into $hdl, tag_P, in, out, \sigma$
verify the validity of $vk_{sign}$
run the algorithm HW.quoteVerify with an input $(pms, quote)$
verify the validity of $quote$
return the results $b = 1$ if it passes
```

Functionalities. We emphasize two main functionalities in our protocol, including *isolated transaction generation* and *remote attestation.* The delegation inside TEEs has full responsibility to govern the behaviours of participants. In particular, TEEs first call the function *sgx_create_enclave* and *enclave_init_ra* to create and initialize an enclave $E_{\mathcal{O}}$. Then, it derives the transaction key sk_{Tx} under the user's invocation.

Next, the system generates a bitcoin address and a transaction by called the function *create_address_from_string* and *generate_transaction*, respectively. $E_{\mathcal{O}}$ keeps sk_{Tx} in its global variable storage and signs the transaction with it while calling *generate_transaction.* The transaction can only be generated inside the enclave without exposing to the public. Afterwards, $E_{\mathcal{O}}$ creates a quote by calling the function *ra_network_send_receive*, and proves to the delegatee that its enclave has been successfully initialized and is ready for further delegation.

6.2.7 Evaluation

In this section, we evaluate the system regarding *performance* and *disk space.* To have an accurate and fair test result, we repeat the experiment for each operation 500 times and calculate the average value.

6.2.7.1 Performance

The operations of *public key generation* and *address create* cost approximately the same time since they are both based on the same type of basic cryptographic primitives. The operations of *transaction generation*, *state seal*, and *transaction decryption* spend more time than the aforementioned operations because they combine more complex cryptographic operations. We also observe that the *enclave initiation* spends much more time than (transactions) *key pair generations*. Fortunately, the time used on enclave initiation can be omitted since the enclave launches only once (one-time operation) each time. The *state update* spends the lowest time since most of the recorded messages are overlapped without changes and only a small portion of data requires an update. The operations of *coin deposit* and *transaction confirmation* depend on the configuration of Bitcoin testnet, varying from 10+ seconds to several minutes. Furthermore, we attach the time costs of the *state seal* operation under increased transactions in Figure.6.7 (right column). The time consumption grows slowly because a large portion of transactions are processed in batch. Remarkably, it costs less than 25 milliseconds to finish all operations of coin delegation, which is significantly lower than the online transaction of Bitcoin testnet. This indicates that our solution is efficient in transaction processing and practical coin delegation.

Table 6.3: The average performance of various operations

Phase	**Operation**	**Average Time / ms**
System setup	Enclave initiation	13.18940
	Public key generation (Tx)	0.34223
	Private key generation (Tx)	0.01119
Coin deposit	Address creation	0.00690
	Coin deposit	–
Coin delegation	Transaction generation	0.78565
	Remote attestation	19.50990
	State update	0.00366
	State seal	5.43957
Coin spend	Transaction decryption	–
	Transaction confirmation	–

6.2.7.2 Disk Space

In this section, we provide an evaluation of the disk space of the sealed state. We simulate the situation in DelegaCoin when more delegation transactions join the network. The transaction creation rate is set to be 560 transactions/second. We monitor space usage and the corresponding growth rate. Each transaction occupies approximately 700 KB of storage space. We test eight sets of experiments with an increased number of transactions in the sequence $1, 10, 100, 200, 400, 600, 800, 1000$. The results, as shown in Figure.6.7 (left column), indicate that the size of the disk usage grows linearly with increased delegation transactions. The reason is straightforward: the disk usage closely relates to the processed transactions that are stored in the list. In our configurations, the transaction generation rate stays fixed. Therefore, the used space is proportional to increased transactions.

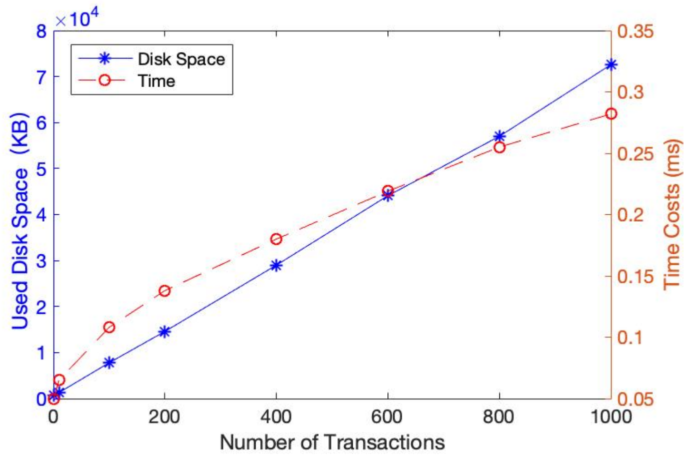


Figure 6.7: Used disk space and time-consuming of state seal

6.2.8 Conclusion

Decentralized cryptocurrencies such as Bitcoin [158] provide an alternative approach for peer-to-peer payments. However, such payments are time-consuming. In this chapter, we provided a secure and practical TEEs-based offline delegatable cryptocurrency system. TEEs are used as the primitives to establish a secure delegation channel and offer better storage protection of metadata (keys, policy). An owner can delegate coins through an offline transaction asynchronously with the blockchain network. A formal analysis, prototype implementation and evaluation demonstrated that our scheme is provably secure and practically feasible.

Chapter 7

Conclusion and Future Work

In this chapter, we provide a global perspective on the contribution of this thesis. We begin by summarizing the research conducted in each chapter. Then, we outline directions for future research.

7.1 Summary of Results

Security protocols [154] are defined as a series of security-related functions with message exchanges between two or multiple parties to achieve specific security goals. In practical implementations, security protocols generally rely on one or several powerful authorities, called the *trusted third party* (TTP), to achieve secure communications between different participants. However, the TTP in security protocols may not behave "as it should"; it may act maliciously due to hidden interests or having been compromised by malicious attackers. A malicious TTP can effortlessly destroy an entire security protocol, and this vulnerability has been widely exploited in real-world cryptosystems.

This thesis extends the research direction seeking to solve TTP issues and tries to address them by employing newly proposed blockchain and TEE technologies. Specifically, a smart contract-based TTP in the context of blockchain has been proposed to enhance existing TTP's transparency and further make cryptographic protocols more secure (Chapter 3); another smart contract&TEE-based TTP has been presented to protect existing TTPs' privacy, bringing other advanced security properties such as accountability. For each solution, two instances have been given to demonstrate feasibility and practicality (Chapter 4 and Chapter 5). Existing blockchain systems still suffer from scalability and usability issues. Thus, to make the blockchain-based TTP more practical and more likely to be adopted, two new blockchain algorithms with high performance, good scalability and usability have been proposed (Chapter 6). In the following section, we present a summary of the research results for each chapter.

Chapter 3 introduced a new concept that employs smart contracts as the TPP, called TTP-I. Unlike most of the current work employing complex protocols or algorithms to achieve a distributed TTP, the proposed approach uses the smart contract as a flexible TTP to secure existing security protocols. Subsequently, TTP-I was applied to CBE [83, 90, 142] and RBE [86, 87] protocols to demonstrate its feasibility and acceptability. It has been observed that TTP-I enhances security protocols in two ways. Firstly, TTP-I can be simply used as an autonomous subscriber that automatically informs participants and stores data when triggered by events. By adding transparent functions of validity checking and routing revocation requests of certificates, it effectively prevents the existing CA's misbehaviour in CBE protocols. Secondly, TTP-I can directly act as the trusted agent in decentralized applications, ensuring the authorities perform trusted transactions and agreements. As in the RBE example, by using a smart contract as a KC, all registration and query operations are publicly traceable and further make users' activities and KCs' behaviours accountable.

Chapter 4 introduced a new type of smart contract, called a TEE-assisted confidential smart contract (TCSC), which exploits TEEs to address privacy problems existing in TTP-I. In a nutshell, TCSC is defined as an ideal "decentralized black box" providing data secrecy, execution correctness, and fairness. To achieve a fair and privacy-preserving TTP, we first investigated the state-of-the-art technologies involved in the implementations of existing TCSC systems. Then, we showed how TCSC works and how the TEE keys are managed. Afterwards, we presented a unified framework to evaluate them. Based on their common features, the syntax of TCSC and corresponding assumptions were defined for building a general construction of TCSC-based protocol (called TTP-II protocol), followed by discussions of corresponding assumptions and security properties.

Chapter 5 presented two advanced security systems using TTP-II: a credential anonymity revocation system and an accountable decryption system.

- **Credential anonymity revocation system.** Anonymity revocation is essential for credential systems since unconditional anonymity is incompatible with pursuing and sanctioning credential misuse. However, current anonymity revocation approaches have shortcomings with respect to the auditability of the revocation process. By using TTP-II, we proposed a novel anonymity revocation approach, bringing neutrality and auditability.

- **Accountable decryption system.** To achieve accountability during the execution of a warrant to access users' sensitive data, a trusted authority is required, denoted as *judge*, to faithfully cooperate with participants (e.g., investigators). However, malicious judges or uncooperative participants may void the accountability mechanism in practice, for example, by fabricating fake evidence or by refusing to provide evidence. By using TTP-II, we proposed a novel accountable decryption system. The inherent neutralities of blockchain make TTP-II act as an accountable key manager and a transparent judge, which increases the transparency of warrant execution.

Both TTP-II systems were implemented and evaluated in extensive experiments. The evaluation results have demonstrated that our solutions are feasible and practical. Moreover, formal definitions and security analyses have theoretically proved the security of our proposed solutions.

Chapter 6 proposed two algorithms to enhance the blockchain performance/scalability as well as usability. As discussed, both TTP-I and TTP-II protocols suffer from the issues of poor scalability and low performance, which causes bad usability, and further limits their wide adoption especially in scenarios where high performance is required. The weak consensus algorithm provided a high-performance blockchain system by improving existing consensus algorithms. We have implemented the system with 32k+ lines of code, including all relevant components such as consensus, P2P, etc. The evaluations indicated that our system could reach a peak throughput of 43k TPS (with eight full nodes), which is significantly faster than current blockchain systems such as Ethereum, given the same experimental environment. Meanwhile, an offline delegatable cryptocurrency protocol was proposed to create a fast payment system for cryptocurrencies. The formal model and analysis, prototype implementation and corresponding evaluations have demonstrated that our proposed solution is provably secure and practically feasible.

As a result, this thesis represented a significant advance in the state-of-the-art of blockchain-based security protocols, consensus algorithms, and payment protocols, which provides a tremendous amount of assistance to cryptographers and blockchain researchers in both academia and industry. When designing a smart contract-based TTP, theoretical support must be considered. Our formalized model of smart contract protocol with well-defined syntax and assumptions (Chapter 3) could be easily used to prove the security of other upper-layer contract applications. Also, building advanced cryptographic protocols using TEE and contract naturally brought many security benefits. However, the corresponding security threat model remains to be uncovered. Our comprehensive investigation and modelling (Chapter 4) provided future researchers with a coherent view of existing TCSC systems. Additionally, two protocols using TCSC were presented (Chapter 5), and their implementation further proved the feasibility of the approach. In a nutshell, our research effectively promoted the development of contract-based security protocols, TEE-assisted smart contracts, TEE&contract-based security protocols, and blockchain systems.

Finally, it is worth pointing out that the weak consensus algorithm proposed in this thesis has been adopted by a Non-Fungible Token (NFT) project [74]. Using our algorithm, their team proposed an innovative Non-Fungible Token framework that differs from existing contract-based NFT projects [206], where NFT operations of transaction ordering and data storage are separated, equipping the system with properties of fast certification and low transaction fee.

7.2 Future Work

Improving the Transparency of TTP-II. In Chapter 5, we presented two advanced protocols based on TTP-II. However, TTP-II as a customized TEE-aided smart contract, lacks transparency. On the one hand, contracts are executed inside TEE, and the outputs are encrypted, which lacks public verifiability. The attestation service can only guarantee that encrypted outputs indeed come from a TEE. However, neither users nor the blockchain can learn whether the TEE is compromised or is faithfully executing contracts following the loaded contract specification. Even if many TEEs can re-execute the contract with the same setup (e.g., same private key) to check outputs, this inevitably increases the risk of key leakage. On the other hand, the precise architecture of chips is still unclear for several TEE products such as Intel SGX [61]. TEE-assisted solutions force users to put too much trust in manufacturers of this hardware. For example, some users argue that Intel may have reduced the security of SGX to improve performance to cater to market needs [67]. Additionally, the attestation service used to prove that a program runs inside TEEs is *centralized* and *non-transparent*. The service holder has the ability to insert fake IDs, and further steal the confidential state in smart contracts. Thus, in the future, we would like to build a publicly verifiable TTP-II.

Narrowing the Gap between Theory and Practice. In this thesis, all proposed schemes are proved to be theoretically secure. However, contract-based protocols, TEE&contract-based protocols and their underlying blockchain systems are inherently hybrid with a sophisticated mechanism. The assumptions used in our security proof are inevitably strong. Even if several prototype implementations are given, there is an insurmountable gap between the theoretical case and real application. Unexpected risks may still exist in practical scenarios. In the TTP-II-based anonymity revocation system outlined in Section 5.1, even though our scheme provides an auditable approach to trace credential anonymity, there are risks that the tracer's private key is stolen or misused, resulting in a failure of the whole mechanism. Thus, countermeasures to reduce these risks must be explored.

Appendix A

Resource Availability

- Auditable Credential System
  - **Code:** `https://github.com/typex-1/core`
  - **Data:** `https://github.com/typex-1/core/tree/master/test/result`
  - **Contributors:** Rujia Li (80%), Feng Liu (20%)
- Accountable Decryption System
  - **Code:** `https://github.com/fialka-1/demo`
  - **Website:** `http://www.fialka.top`
  - **Data:** `https://github.com/fialka-1/demo/tree/master/testarea`
  - **Contributors:** Rujia Li (50%), Feng Liu (50%)
- High-Performance Blockchain with Weak Consensus Algorithm
  - **Code:** `https://github.com/rjgeek/sphinx`
  - **Data:** `https://github.com/rjgeek/sphinx/tree/master/testdata`
  - **Contributors:** Rujia Li (40%), XueqianLu (40%), Qin Wang (20%)
- Offline Delegatable Cryptocurrency System
  - **Code:** `https://github.com/tees-projects/delegaCoin`
  - **Data:** `https://github.com/tees-projects/delegaCoin/tree/main/test_data`
  - **Contributors:** Rujia Li (50%), Xinrui Zhang (50%)

Bibliography

[1] Build your app on the world's leading privacy-first blockchain. `https://www.oasislabs.com/`. accessed: Dec., 2021.

[2] Enigma is securing the future of the web. `https://www.enigma.co/`. accessed: Apr., 2021.

[3] Testnet is an alternative bitcoin blockchain that developers use for testing. `https://www.blockchain.com/explorer/assets/btc-testnet`, 2020. accessed: Dec., 2021.

[4] M. Abe and M. Ohkub. Provably secure fair blind signatures with tight revocation. In C. Boyd, editor, Proceedings of the 2001 Annual International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT), pages 583–601. Springer Berlin Heidelberg, 2001.

[5] F. Alder, N. Asokan, A. Kurnikov, A. Paverd, and M. Steiner. S-faas: Trustworthy and accountable function-as-a-service using intel sgx. Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop (CCSW), 2019.

[6] M. Ali, J. Nelson, R. Shea, and M. J. Freedman. Blockstack: A global naming and storage system secured by blockchains. In Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC), pages 181–194, 2016.

[7] P. Ammann and S. Jajodia. Distributed timestamp generation in planar lattice networks. ACM Transactions on Computer Systems (TOCS), 11(3):205–225, 1993.

[8] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun. Evaluating user privacy in bitcoin. In Proceedings of the 2013 International Conference on Financial Cryptography and Data Security (FC), pages 34–51. Springer, 2013.

[9] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer. {SCONE}: Secure linux containers with intel {SGX}. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pages 689–703, 2016.

[10] A. Azaria, A. Ekblaw, and T. Vieira. Medrec: Using blockchain for medical data access and permission management. In Proceedings of the 2nd international conference on open and big data (OBD), pages 25–30. IEEE, 2016.

[11] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 585–602, 2019.

[12] K. Baghery. On the efficiency of privacy-preserving smart contract systems. In Proceedings of the 2019 International Conference on Cryptology in Africa (AFRICACRYPT), pages 118–136. Springer, 2019.

[13] L. Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Technical Report Technical Report SWIRLDS-TR-2016-01, Swirlds, 2016. accessed: Dec., 2021.

[14] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis. Sok: Consensus in the age of blockchains. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT), pages 183–198, 2019.

[15] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. In Proceedings

of the 2001 Annual International Cryptology Conference (CRYPTO), pages 1–18. Springer, 2001.

[16] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. ACM Transactions on Computer Systems (TOCS), 33(3):1–26, 2015.

[17] A. Beimel. Secret-sharing schemes: A survey. In Proceedings of the 2011 International Conference on Coding and Cryptology (IWCC), pages 11–46. Springer, 2011.

[18] M. Bellare and P. Rogaway. Collision-resistant hashing: Towards making uowhfs practical. In Proceedings of the Proceedings of the 1997 Annual International Cryptology Conference (CRYPTO), pages 470–484. Springer, 1997.

[19] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In Proceedings of the 1993 Annual ACM Symposium on Theory of Computing (STOC), pages 52–61, 1993.

[20] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In Proceedings of the 1988 Annual ACM Symposium on Theory of Computing (STOC), pages 351–371, 1988.

[21] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP), pages 499–516. IEEE, 2015.

[22] A. Bessani, J. Sousa, and E. E. Alchieri. State machine replication for the masses with bft-smart. In Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 355–362. IEEE, 2014.

[23] E.-O. Blass and F. Kerschbaum. Borealis: Building block for sealed bid auctions on blockchains. In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIACCS), pages 558–571, 2020.

[24] O. Blazy, S. Canard, G. Fuchsbauer, A. Gouget, H. Sibert, and J. Traoré. Achieving optimal anonymity in transferable e-cash with a judge. In Proceedings of the 2011 International Conference on Cryptology in Africa (AFRICACRYPT), pages 206–223. Springer, 2011.

[25] D. Boneh. The decision diffie-hellman problem. In Proceedings of the 1998 International Algorithmic Number Theory Symposium (ANTS), pages 48–63. Springer, 1998.

[26] D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Proceedings of the 2004 International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 223–238. Springer, 2004.

[27] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In Proceedings of the 2004 Annual International Cryptology Conference (CRYPTO), pages 41–55. Springer, 2004.

[28] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In Proceedings of the 2001 Annual International Cryptology Conference (CRYPTO), pages 213–229. Springer, 2001.

[29] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In J. Kilian, editor, Proceedings of the 2001 Annual International Cryptology Conference (CRYPTO), pages 213–229. Springer, 2001.

[30] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP), pages 104–121. IEEE, 2015.

[31] M. Bowman, A. Miele, M. Steiner, and B. Vavala. Private data objects: an overview, 2018. arXiv preprint.

[32] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. Journal of the ACM (JACM), 32(4):824–840, 1985.

[33] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti. Blockchain and

trusted computing: Problems, pitfalls, and a solution for hyperledger fabric, 2018. arXiv preprint.

[34] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf. Sanctuary: Arming trustzone with user-space enclaves. In Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS), 2019.

[35] E. F. Brickell, P. Gemmell, and D. W. Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In Proceedings of the 1995 ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995.

[36] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh. Zether: Towards privacy in a smart contract world. In Proceedings of the 2020 International Conference on Financial Cryptography and Data Security (FC), pages 423–443. Springer, 2020.

[37] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), pages 315–334. IEEE, 2018.

[38] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Proceedings of the 2001 International Conference on the Theory and Applications of Cryptographic Techniques (EROCRYPT), pages 93–118. Springer, 2001.

[39] J. Camenisch, U. Maurer, and M. Stadler. Digital payment systems with passive anonymity-revoking trustees. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, Proceedings of the 1996 European Symposium on Research in Computer Security (ESORICS), Lecture Notes in Computer Science, pages 33–43. Springer Berlin Heidelberg, 1996.

[40] J. Camenisch, S. Mödersheim, and D. Sommer. A formal model of identity mixer. In Proceedings of the 2010 International Workshop on on Formal Methods for Industrial Critical Systems (FMICS), pages 198–214. Springer, 2010.

[41] S. Canard and J. Traoré. On fair e-cash systems based on group signature schemes. In Proceedings of the 2003 Australasian Conference on Information Security and Privacy (ACISP), pages 237–248. Springer, 2003.

[42] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In Proceedings 2001 IEEE Symposium on Foundations of Computer Science (CSF), pages 136–145. IEEE, 2001.

[43] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In Proceedings of the 2003 International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 255–271. Springer, 2003.

[44] G. Carle. Wayback machine. `https://web.archive.org/web/20170829004310/http://www.ccs-labs.org/~dressler/teaching/netzsicherheit-ws0304/07_CryptoProtocols_2on1.pdf`, 4 2003. accessed: Apr.,2021.

[45] M. Castro and B. Liskov. Practical byzantine fault tolerance. In Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), pages 173–186, 1999.

[46] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. In Proceedings of the 2002 ACM Transactions on Computer Systems (TOCS), volume 20, pages 398–461. ACM, 2002.

[47] S. Cetola. A Method for Comparative Analysis of Trusted Execution Environments. PhD thesis, Portland State University, 2021.

[48] D. Chaum. Blind signatures for untraceable payments. In Proceedings of the 2nd Annual International Cryptology Conference (CRYPTO), pages 199–203. Springer, 1983.

[49] G. Chen, Y. Zhang, and T.-H. Lai. Opera: Open remote attestation for intel's secure enclaves. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 2317–2331, 2019.

[50] T.-Y. Chen, W.-N. Huang, P.-C. Kuo, H. Chung, and T.-W. Chao. DEXON:

A highly scalable, decentralized DAG-based consensus algorithm, 2018. arXiv preprint.

[51] Y. Chen, X. Ma, C. Tang, and M. H. Au. Pgc: Decentralized confidential payment system with auditability. In Proceedings of the 2020 European Symposium on Research in Computer Security (ESORICS), pages 591–610. Springer, 2020.

[52] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution, 2018. arXiv preprint.

[53] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EUROSP), pages 185–200. IEEE, 2019.

[54] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EUROSP), pages 185–200. IEEE, 2019.

[55] S. S. Chow. Removing escrow from identity-based encryption. In Proceedings of the 2009 International Workshop on Public Key Cryptography (PKC), pages 256–276. Springer, 2009.

[56] M. Conti, N. Dragoni, and V. Lesyk. A survey of man in the middle attacks. IEEE Communications Surveys & Tutorials, 18(3):2027–2051, 2016.

[57] D. Contractor and D. R. Patel. Accountability in cloud computing by means of chain of trust. International Journal of Network Security, 19(2):251–259, 2017.

[58] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene. Election verifiability for helios under weaker trust assumptions. In Proceedings of the 2014

European Symposium on Research in Computer Security (ESORICS), pages 327–344. Springer, 2014.

[59] V. Cortier, D. Galindo, R. Küsters, J. Mueller, and T. Truderung. Sok: Verifiability notions for e-voting protocols. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), pages 779–798. IEEE, 2016.

[60] Cosmos. The internet of blockchains. `https://cosmos.network/`. accessed: May, 2021.

[61] V. Costan and S. Devadas. Intel sgx explained. IACR Cryptology ePrint Archive, 2016(86):1–118, 2016.

[62] V. Costan, I. Lebedev, and S. Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In Proceedings of the 25th USENIX Security Symposium (USENIX Security), pages 857–874, 2016.

[63] R. Cramer, G. Hanaoka, D. Hofheinz, H. Imai, E. Kiltz, R. Pass, A. Shelat, and V. Vaikuntanathan. Bounded cca2-secure encryption. In Proceedings of the 2007 International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT), pages 502–518. Springer, 2007.

[64] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi. Fastkitten: Practical smart contracts on bitcoin. In Proceedings of the 2019 USENIX Security Symposium (USENIX Security), pages 801–818, 2019.

[65] A. W. Dent, B. Libert, and K. G. Paterson. Certificateless encryption schemes strongly secure in the standard model. In Proceedings of the 2008 International Workshop on Public Key Cryptography (PKC), pages 344–359. Springer, 2008.

[66] W. Diffie and M. E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory (TIT), 22:644–654, 1976.

[67] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont. Everything you should know about intel sgx performance on

virtualized systems. Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS), 3(1):1–21, 2019.

[68] L. Dong and K. Chen. Cryptographic Protocol: Security Analysis Based on Trusted Freshness. Springer, Berlin, Heidelberg, 2012.

[69] J.-E. Ekberg, K. Kostiainen, and N. Asokan. Trusted execution environments on mobile devices. In Proceedings of the 2013 ACM Conference on Computer and Communications Security (CCS), pages 1497–1498, 2013.

[70] P. Ekparinya, V. Gramoli, and G. Jourjon. The attack of the clones against proof-of-authority. In Proceedings of the 2020 Network and Distributed System Security Symposium (NDSS), pages 1–14, 2020.

[71] Enigma. The developer quickstart guide to enigma. `https://blog.enigma.co/the-developer-quickstart-guide-to-enigma-880c3fc4308`. accessed: Apr., 2021.

[72] A. Erwig, S. Faust, S. Riahi, and T. Stöckert. Commitee: An efficient and secure commit-chain protocol using tees. IACR Cryptology ePrint Archive, 2020:1486, 2020.

[73] A. Escala, J. Herranz, and P. Morillo. Revocable attribute-based signatures with adaptive security in the standard model. In Proceedings of the 2011 International Conference on Cryptology in Africa (AFRICACRYPT), pages 224–241. Springer, 2011.

[74] FASTBOX. Open a new nft world. `https://www.fastbox.org/`, 2021. accessed: Dec.,2021.

[75] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno. Komodo: Using verification to disentangle secure-enclave hardware from software. In Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), pages 287–305, 2017.

[76] Financials. Changan chain, the first independent and controllable blockchain technology system in china, was released today. `https://equalocean.com/briefing/20210127230021545`. EqualOcean, accessed: Dec., 2021.

[77] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov. Iron: functional encryption using intel sgx. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 765–782, 2017.

[78] M. Fitzi, P. Gazi, A. Kiayias, and A. Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. IACR Cryptology ePrint Archive, 2018:1119, 2018.

[79] G. Fuchsbauer, A. Plouviez, and Y. Seurin. Blind schnorr signatures and signed elgamal encryption in the algebraic group model. In Proceedings of the 2020 Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), volume 12106, page 63. Nature Publishing Group, 2020.

[80] G. Fuchsbauer and D. Vergnaud. Fair blind signatures without random oracles. In D. J. Bernstein and T. Lange, editors, Proceedings of the 2010 International Conference on Cryptology in Africa (AFRICACRYPT), pages 16–33. Springer Berlin Heidelberg, 2010.

[81] A. Gagol, D. Lesniak, D. Straszak, and M. Swietek. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT), pages 214–228, 2019.

[82] H. S. Galal and A. M. Youssef. Trustee: full privacy preserving vickrey auction on top of ethereum. In Proceedings of the 2019 International Conference on Financial Cryptography and Data Security (FC), pages 190–207. Springer, 2019.

[83] D. Galindo*, et al.* Improved certificate-based encryption in the standard model. In Proceedings of the 2008 International Conference on Security and Cryptography for Networks (SCN), volume 81, pages 1218–1226. Elsevier, 2008.

[84] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In Proceedings of the 2015 Annual International

Conference on the Theory and Applications of Cryptographic Techniques, pages 281–310. Springer, 2015.

[85] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Proceedings of the 2017 Annual International Cryptology Conference (CRYPTO), pages 291–323. Springer, 2017.

[86] S. Garg, M. Hajiabadi, M. Mahmoody, and A. Rahimi. Registration-based encryption: Removing private-key generator from ibe. In Proceedings of the 2018 Annual International Cryptology Conference (CRYPTO), pages 689–718. Springer, 2018.

[87] S. Garg, M. Hajiabadi, M. Mahmoody, A. Rahimi, and S. Sekar. Registration-based encryption from standard assumptions. In Proceedings of the 2019 International Workshop on Public Key Cryptography (PKC), pages 63–93. Springer, 2019.

[88] N. Gelernter, S. Kalma, B. Magnezi, and H. Porcilan. The password reset mitm attack. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), pages 251–267. IEEE, 2017.

[89] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Proceedings of the 1999 Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 295–310. Springer, 1999.

[90] C. Gentry. Certificate-based encryption and the certificate revocation problem. In Proceedings of the 2003 Annual International Cryptology Conference (CRYPTO), pages 272–293. Springer, 2003.

[91] A. Gervais, G. O. Karame, K. Wust, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 3–16. ACM, 2016.

[92] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th

Symposium on Operating Systems Principles (SOSP), pages 51–68. ACM, 2017.

[93] S. Goldfeder. Private smart contracts. In Proceedings of the 2018 Privacy Enhancing Technologies Symposium (PETS), 2018.

[94] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing, 17(2):281–308, 1988.

[95] R. Goyal and S. Vusirikala. Verifiable registration-based encryption. In Proceedings of the 2020 Annual International Cryptology Conference (CRYPTO), pages 621–651. Springer, 2020.

[96] V. Goyal, S. Lu, A. Sahai, and B. Waters. Black-box accountable authority identity-based encryption. In Proceedings of the 2008 ACM Conference on Computer and Communications Security (CCS), pages 427–436. ACM, 2008.

[97] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. SoK: Layer-two blockchain protocols. In Proceedings of the 2020 International Conference on Financial Cryptography and Data Security (FC), pages 201–226. Springer, 2020.

[98] S. Gueron. Memory encryption for general-purpose processors. IEEE Security & Privacy, 14(6):54–62, 2016.

[99] H. Guo, Z. Zhang, and et al. Generic traceable proxy re-encryption and accountable extension in consensus network. In Proceedings of the 2019 European Symposium on Research in Computer Security (ESORICS), pages 234–256. Springer, 2019.

[100] A. Gupta, N. K. Walia, and S. K. Guru. Cryptography algorithms: A review. International Journal of Engineering Development and Research, 2014.

[101] Y. Hang, Z. Shunfan, and J. Jun. Phala network: A confidential smart contract network based on polkadot. `https://crebaco.com/planner/admin/uploads/whitepapers/3580918phala-paper.pdf`, 2019. accessed: Dec., 2021.

[102] H. Huang, W. Kong, S. Zhou, Z. Zheng, and S. Guo. A survey of state-of-the-art on blockchains. ACM Computing Surveys (CSUR), 54:1 – 42, 2021.

[103] Q. Huang, D. S. Wong, and W. Susilo. P 2 ofe: Privacy-preserving optimistic fair exchange of digital signatures. In Proceedings fo the 2014 Cryptographers' Track at the RSA Conference (CT-RSA), pages 367–384. Springer, 2014.

[104] Q. Huang, G. Yang, D. S. Wong, and W. Susilo. A new efficient optimistic fair exchange protocol without random oracles. International Journal of Information Security, 11(1):53–63, 2012.

[105] E. Hufschmitt and J. Traoré. Fair blind signatures revisited. In T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto, editors, Proceedings of the 20007 International Conference on Pairing-Based Cryptography (ICPBC), Lecture Notes in Computer Science, pages 268–292. Springer, 2007.

[106] M. Jakobsson and M. Yung. Revokable and versatile electronic money. In Proceedings of the 3rd ACM Conference on Computer and Communications Security (CCS), 1996.

[107] M. Jakobsson and M. Yung. Distributed "magic ink" signatures. In W. Fumy, editor, Proceedings of the 1997 International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Lecture Notes in Computer Science, pages 450–464. Springer, 1997.

[108] M. Jansen, F. Hdhili, R. Gouiaa, and Z. Qasem. Do smart contract languages need to be turing complete? In Proceedings of the 2019 International Congress on Blockchain and Applications (BLOCKCHAIN), pages 19–26. Springer, 2019.

[109] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security, 1(1):36–63, 2001.

[110] A. Juels, A. Kosba, and E. Shi. The ring of gyges: Investigating the future of criminal smart contracts. In Proceedings of the 2016 ACM Conference on Computer and Communications Security (CCS), pages 283–295, 2016.

[111] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten. Arbitrum: scalable, private smart contracts. In Proceedings of the 27th USENIX Conference on Security Symposium, pages 1353–1370. USENIX Association, 2018.

[112] D. Kaplan, J. Powell, and T. Woller. Amd memory encryption. White paper, April, 2016.

[113] D. Kaplan, J. Powell, and T. Woller. Amd sev-snp: Strengthening vm isolation with integrity protection and more. White paper, Jan, 2020.

[114] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. An empirical analysis of anonymity in zcash. In Proceedings of 27th USENIX Security Symposium (USENIX Security), pages 463–477, 2018.

[115] P. A. Karger and A. J. Herbert. An augmented capability architecture to support lattice security and traceability of access. In Proceedings of the 1984 IEEE Symposium on Security and Privacy (SP), pages 2–2. IEEE, 1984.

[116] A. Kate and I. Goldberg. Distributed private-key generators for identity-based cryptography. In Proceedings of the 2010 International Conference on Security and Cryptography for Networks (SCN), pages 436–453. Springer, 2010.

[117] L. Kelly. China's public chang'an chain gets upgrade to accelerate processing. `https://forkast.news/headlines/china-blockchain-equip-blockchain-chip/`, Mar 2021. Forkast News, accessed: Dec., 2021.

[118] A. Kiayias and G. Panagiotakos. Speed-security tradeoffs in blockchain protocols. IACR Cryptology ePrint Archive, 2015:1019, 2015.

[119] A. Kiayias, A. Russell, B. David, and R. Oliynykov. kiayias2017ouroboros: A provably secure proof-of-stake blockchain protocol. In Proceedings of the 2017 Annual International Cryptology Conference (CRYPTO), pages 357–388. Springer, 2017.

[120] A. Kiayias and H.-S. Zhou. Concurrent blind signatures without random oracles. In R. De Prisco and M. Yung, editors, Proceedings of the 2006

International Conference on Security and Cryptography for Networks (SCN), pages 49–62. Springer Berlin Heidelberg, 2006.

[121] L. Kiffer, R. Rajaraman, and A. Shelat. A better method to analyze blockchain consistency. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 729–744, 2018.

[122] E. Kiltz. Chosen-ciphertext security from tag-based encryption. In Proceedings of the 2006 Theory of Cryptography Conference (TCC), pages 581–600. Springer, 2006.

[123] S. Kim, J. Han, J. Ha, T. Kim, and D. Han. Enhancing security and privacy of tor's ecosystem by using trusted execution environments. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI), pages 145–161, 2017.

[124] T. H.-J. Kim, L.-S. Huang, A. Perrig, C. Jackson, and V. Gligor. Accountable key infrastructure (aki) a proposal for a public-key validation infrastructure. In Proceedings of the 22nd International World Wide Web Conference (WWW), pages 679–690, 2013.

[125] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In Proceedings of the 25th USENIX Security Symposium (Usenix Security), pages 279–296, 2016.

[126] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In Proceedings of the 2016 IEEE symposium on security and privacy (SP), pages 839–858. IEEE, 2016.

[127] J. A. Kroll, J. Zimmerman, D. J. Wu, V. Nikolaenko, and E. W. Felten. Accountable cryptographic access control. In Proceedings of the 2018 Annual International Cryptology Conference (CRYPTO), volume 2018, 2018.

[128] A. Küpçü. Distributing trusted third parties. ACM SIGACT News, 44(2):92–112, 2013.

[129] R. Küsters, T. Truderung, and A. Vogt. Accountability: definition and

relationship to verifiability. In Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS), pages 526–535. ACM, 2010.

[130] T. Kwon. Privacy preservation with x. 509 standard certificates. Information Sciences, 181(13):2906–2921, July 2011.

[131] J. Lai, R. H. Deng, Y. Zhao, and J. Weng. Accountable authority identity-based encryption with public traceability. In Proceedings of the 2013 Cryptographers' Track at the RSA Conference (CT-RSA), pages 326–342. Springer, 2013.

[132] J. Lai and Q. Tang. Making any attribute-based encryption accountable, efficiently. In Proceedings of the 2018 European Symposium on Research in Computer Security (ESORICS), pages 527–547. Springer, 2018.

[133] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song. Keystone: An open framework for architecting trusted execution environments. In Proceedings of the 2020 European Conference on Computer Systems (EUROSYS), pages 1–16, 2020.

[134] C. Li, P. Li, D. Zhou, W. Xu, F. Long, and A. Yao. Scaling nakamoto consensus to thousands of transactions per second, 2018. arXiv preprint.

[135] R. Li, D. Galindo, and Q. Wang. Auditable credential anonymity revocation based on privacy-preserving smart contracts. In Proceedings of the 2019 International Workshop on Cryptocurrencies and Blockchain Technology (CBT), pages 355–371. Springer, 2019.

[136] R. Li, Q. Wang, F. Liu, Q. Wang, and D. Galindo. An accountable decryption system based on privacy-preserving smart contracts. In Proceedings of the 2020 International Conference on Information Security (ISC), pages 372–390. Springer, 2020.

[137] R. Li, Q. Wang, Q. Wang, D. G. Chacon, X. Zhang, and Y. Xiang. An offline delegatable cryptocurrency system. In Proceedings of the 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2021.

[138] R. Li, Q. Wang, Q. Wang, D. Galindo, and M. Ryan. Sok: Tee-assisted

confidential smart contract. In Proceedings of the 2022 Privacy Enhancing Technologies Symposium (PETS), 2022.

[139] R. Li, Q. Wang, X. Zhang, Q. Wang, D. Galindo, and Y. Xiang. Poster: An offline delegatable cryptocurrency system. In Proceedings of the 28th Annual Network and Distributed System Security Symposium (NDSS), 2021.

[140] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. In Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP), pages 63–79, 2019.

[141] Y. Lindell. How to simulate it–a tutorial on the simulation proof technique. Tutorials on the Foundations of Cryptography, pages 277–346, 2017.

[142] J. K. Liu and J. Zhou. Efficient certificate-based encryption in the standard model. In Proceedings of the 2008 International Conference on Security and Cryptography (SECRYPT), pages 144–155. Springer, 2008.

[143] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang. Combining graph neural networks with expert knowledge for smart contract vulnerability detection. IEEE Transactions on Knowledge and Data Engineering, 2021.

[144] L. Lu. How a little ant challenges giant banks? the rise of ant financial (alipay)'s fintech empire and relevant regulatory concerns. International Company and Commercial Law Review (2018), Sweet & Maxwell, ISSN, pages 0958–5214, 2018.

[145] Y. Luo, J. Fan, C. Deng, Y. Li, Y. Zheng, and J. Ding. Accountable data sharing scheme based on blockchain and sgx. In Proceedings of the 2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), pages 9–16. IEEE, 2019.

[146] W. Mao. Modern cryptography: theory and practice. Pearson Education India, 2003.

[147] W. Martino, M. Quaintance, and S. Popejoy. Chainweb: A proof-of-work parallel-chain architecture for massive throughput, 2018. Chainweb Whitepaper, accessed: Dec., 2021.

[148] S. Matetic, M. Schneider, A. Miller, A. Juels, and S. Capkun. Delegatee: Brokered delegation using trusted execution environments. In Proceedings of the 27th USENIX Security Symposium (USENIX Security), pages 1387–1403, 2018.

[149] S. Matsumoto and R. M. Reischuk. Ikp: Turning a pki around with decentralized automated incentives. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), pages 410–426. IEEE, 2017.

[150] U. Maurer. Modelling a public-key infrastructure. In Proceedings of the 1996 European Symposium on Research in Computer Security (ESORICS), pages 325–350. Springer, 1996.

[151] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP), page 1, 2013.

[152] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP), pages 1–1, Tel-Aviv, Israel, 2013. ACM.

[153] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In Proceedings of the 2013 AMC Internet Measurement Conference (IMC), pages 127–140, 2013.

[154] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. Handbook of applied cryptography. CRC press, 2018.

[155] D. L. Mills. Internet time synchronization: the network time protocol. IEEE Transactions on Communications (TCC), 39(10):1482–1493, 1991.

[156] M. Morbitzer, S. Proskurin, M. Radev, M. Dorfhuber, and E. Q. Salas. Severity: Code injection attacks against encrypted virtual machines. In

Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW), pages 444–455. IEEE, 2021.

[157] C. Müller, M. Brandenburger, C. Cachin, P. Felber, C. Göttel, and V. Schiavoni. Tz4fabric: Executing smart contracts with arm trustzone, 2020. arXiv preprint.

[158] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review, page 21260, 2008.

[159] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In Proceedings of the 2001 Annual International Cryptology Conference (CRYPTO), pages 41–62. Springer, 2001.

[160] A. Nehal and P. Ahlawat. Securing iot applications with op-tee from hardware level os. In Proceedings of the 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), pages 1441–1444. IEEE, 2019.

[161] R. Neisse, G. Steri, and I. Nai-Fovino. A blockchain-based approach for data accountability and provenance tracking. In Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES), page 14. ACM, 2017.

[162] G. S. Nicholas, Y. Gui, and F. Saqib. A survey and analysis on soc platform security in arm, intel and risc-v architecture. In Proceedings of the 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), pages 718–721. IEEE, 2020.

[163] G. Noubir and A. Sanatinia. Trusted code execution on untrusted platforms using intel sgx. In Proceedings of the 2016 annual Virus Bulletin International Conference (VB), 2016.

[164] T. Okamoto and K. Ohta. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In J.-J. Quisquater and J. Vandewalle, editors, Proceedings of the 1989 Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT), pages 134–149. Springer Berlin Heidelberg, 1990.

[165] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference (ATC), pages 305–319, 2014.

[166] M. Oxford, D. Parker, and M. Ryan. Quantitative verification of certificate transparency gossip protocols. In Proceedings of the 2020 IEEE Conference on Communications and Network Security (CNS), pages 1–9. IEEE, 2020.

[167] V. F. Pacheco. Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices. Packt Publishing Ltd, 2018.

[168] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report Technical Report TUD-BS-1999-02, Darmstadt University of Technology, 1999. accessed: Dec.,2021.

[169] C. Paquin and G. Zaverucha. U-prove cryptographic specification v1. 1. Technical report, Microsoft Corporation, 2011.

[170] S. Park, H. Park, Y. Won, J. Lee, and S. Kent. Traceable anonymous certificate. Technical Report Technical Report RFC5636, RFC Editor, Aug. 2009.

[171] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In Proceedings of the 2017 Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 643–673. Springer, 2017.

[172] S. Pinto and N. Santos. Demystifying arm trustzone: A comprehensive survey. ACM Computing Surveys (CSUR), 51(6):1–36, 2019.

[173] D. Polemi. Trusted third party services for health care in europe. Future Generation Computer Systems, 14(1-2):51–59, 1998.

[174] B. Qin, J. Huang, Q. Wang, X. Luo, B. Liang, and W. Shi. Cecoin: A decentralized pki mitigating mitm attacks. Future Generation Computer Systems, 107:805–815, 2020.

[175] K. Rannenberg, J. Camenisch, and A. Sabouri. Attribute-based credentials for trust. Identity in the Information Society, Springer, 2015.

[176] S. Raval. Decentralized Applications: Harnessing Bitcoin's Blockchain Technology. O'Reilly Media, Inc., 2016.

[177] C. Research. 2: Recommended elliptic curve domain parameters, 2000. Standards for Efficient Cryptography Group, accessed: Dec., 2021.

[178] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21:120–126, 1978.

[179] M. Russinovich, E. Ashton, C. Avanessians, M. Castro, A. Chamayou, S. Clebsch, M. Costa, C. Fournet, M. Kerner, S. Krishna, J. Maffre, T. Moscibroda, K. Nayak, O. Ohrimenko, F. Schuster, R. Schwartz, A. Shamis, O. Vrousgou, and C. M. Wintersteiger. CCF: A framework for building confidential verifiable replicated services. Technical Report Technical Report MSR-TR-2019-16, Microsoft, April 2019.

[180] M. Ryan. Making decryption accountable. In Proceedings of the 2017 Cambridge International Workshop on Security Protocols (SPW), pages 93–98. Springer, 2017.

[181] M. D. Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS), pages 1–14, 2014.

[182] S. S., B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, and M. Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In Proceedings of the 2019 ACM Conference on Computer and Communications Security (CCS), pages 1759–1776, 2019.

[183] A. Sabouri, I. Krontiris, and K. Rannenberg. Attribute-based credentials for trust (abc4trust). In Proceedings of the 2012 International Conference on Trust, Privacy and Security in Digital Business (TrustBus), pages 218–219. Springer, 2012.

[184] A. Sahai and B. Waters. Fuzzy identity-based encryption. In R. Cramer,

editor, Proceedings of the 2005 International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 457–473. Springer, 2005.

[185] A. Shamir. How to share a secret. Communications of the ACM, 22(11):612–613, 1979.

[186] A. Shamir. Identity-based cryptosystems and signature schemes. In Proceedings of the 1984 Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT), pages 47–53. Springer, 1984.

[187] Z. Shao. Fair exchange protocol of schnorr signatures with semi-trusted adjudicator. Computers & Electrical Engineering, 36(6):1035–1045, 2010.

[188] E. Shi, A. Perrig, and L. Van Doorn. Bind: A fine-grained attestation service for secure distributed systems. In Proceedings of the 2005 IEEE Symposium on Security and Privacy (SP), pages 154–168. IEEE, 2005.

[189] R. Sinha, S. Gaddam, and R. Kumaresan. Luciditee: A tee-blockchain system for policy-compliant multiparty computation with fairness. Cryptology ePrint Archive.

[190] R. Solomon and G. Almashaqbeh. smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption. IACR Cryptology ePrint Archive, 2021:133, 2021.

[191] A. Sonnino, M. Al-Bassam, S. Bano, and G. Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers, 2018. arXiv preprint.

[192] M. Stadler, J.-M. Piveteau, and J. Camenisch. Fair blind signatures. In Proceedings of the 2015 International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 209–219. Springer, 1995.

[193] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos. Performance modeling of pbft consensus process for permissioned blockchain network (hyperledger fabric). In Proceedings of the 36th IEEE Symposium on Reliable Distributed Systems (SRDS), pages 253–255. IEEE, 2017.

[194] N. Szabo. Smart contracts: building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought,(16), 18(2), 1996.

[195] N. Szabo. Trusted third parties are security holes. https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/ttps.html, 2001. accessed: July, 2021.

[196] Taxa. Taxa network: a universal logic layer for blockchain. https://taxa.network/, 2021. accessed: Dec., 2021.

[197] Tendermint. Building the most powerful tools for distributed networks. https://tendermint.com/. accessed: May, 2021.

[198] C.-C. Tsai, D. E. Porter, and M. Vij. Graphene-sgx: A practical library {OS} for unmodified applications on {SGX}. In Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC), pages 645–658, 2017.

[199] A. Unterweger, F. Knirsch, C. Leixnering, and D. Engel. Lessons learned from implementing a privacy-preserving smart contract in ethereum. In Proceedings of the 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pages 1–5, 2018.

[200] J. Vijayan. Godaddy breach exposes ssl keys of managed wordpress hosting customers. https://www.darkreading.com/attacks-breaches/godaddy-breach-exposes-ssl-keys-of-managed-wordpress-hosting-customers, Nov 2021. accessed: July, 2022.

[201] P. Voigt and A. Von dem Bussche. The eu general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing, 10:3152676, 2017.

[202] S. von Solms and D. Naccache. On blind signatures and perfect crimes. Computers & Security, 11(6):581–583, Oct. 1992.

[203] M. Vukolić. Rethinking permissioned blockchains. In Proceedings of the 2017 ACM Workshop on Blockchain, Cryptocurrencies and Contracts (ASIACCS'BCC), pages 3–7. ACM, 2017.

[204] G. Wang, Z. J. Shi, M. Nixon, and S. Han. SoK: Sharding on blockchain. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT), pages 41–61, 2019.

[205] Q. Wang and R. Li. A weak consensus algorithm and its application to high-performance blockchain. IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pages 1–10, 2021.

[206] Q. Wang, R. Li, Q. Wang, and S. Chen. Non-fungible token (NFT): Overview, evaluation, opportunities and challenges, 2021. arXiv preprint.

[207] Q. Wang, R. Li, Q. Wang, S. Chen, and Y. Xiang. Exploring unfairness on proof of authority: Order manipulation attacks and remedies. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (ASIACCS), pages 123–137, 2022.

[208] Q. Wang, R. Li, Q. Wang, and D. Galindo. Poster: Transparent certificate revocation for cbe based on blockchain. In Proceedings of the 41st IEEE Symposium on Security and Privacy (SP), pages 1–2, 2020.

[209] Y. Wang, J. Li, S. Zhao, and F. Yu. Hybridchain: A novel architecture for confidentiality-preserving and performant permissioned blockchain using trusted execution environment. IEEE Access, 8:190652–190662, 2020.

[210] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi. Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v. In Proceedings of the 2019 Annual Network and Distributed System Security Symposium (NDSS), 2019.

[211] D. Wenhao, T. Yufang, and X. Yan. A blockchain-based online game design architecture for performance issues. In Proceedings of the 2020 International Conference on Pattern Recognition and Artificial Intelligence (PRAI), pages 319–324. Springer, 2020.

[212] D. Williams. Introduction to paypal. Pro PayPal E-Commerce, pages 1–12, 2007.

[213] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151(2014):1–32, 2014.

[214] Y. Xu, J. Ren, Y. Zhang, C. Zhang, B. Shen, and Y. Zhang. Blockchain empowered arbitrable data auditing scheme for network storage as a service. Proceedings of the 2019 IEEE Transactions on Services Computing (TSC), 13(2):289–300, 2019.

[215] Y. Yan, C. Wei, X. Guo, X. Lu, X. Zheng, Q. Liu, C. Zhou, X. Song, B. Zhao, H. Zhang, and G. Jiang. Confidentiality support over financial grade consortium blockchain. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD), pages 2227–2240, 2020.

[216] H. Yu, I. Nikolić, R. Hou, and P. Saxena. OHIE: Blockchain scaling made simple. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), pages 90–105. IEEE, 2020.

[217] J. Yu, M. H. A. Au, and P. Esteves-Verissimo. Re-thinking untraceability in the cryptonote-style blockchain. In 2019 IEEE 32nd computer security foundations symposium (CSF), pages 94–9413. IEEE, 2019.

[218] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, and J. Xie. Shadoweth: Private smart contract on public blockchain. Journal of Computer Science and Technology (JCST), 33(3):542–556, 2018.

[219] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt. SoK: communication across distributed ledgers. Cryptology ePrint Archive, 2019.

[220] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi. Town crier: An authenticated data feed for smart contracts. In Proceedings of the 2016 ACM Conference on Computer and Communications Security (CCS), pages 270–282, 2016.

[221] R. Zhang, R. Xue, and L. Liu. Security and privacy on blockchain. ACM Computing Surveys (CSUR), 52(3):1–34, 2019.

[222] Z. Zhang, J. Yin, Y. Liu, and J. Liu. Deanonymization of litecoin through transaction-linkage attacks. In 2020 11th International Conference on Information and Communication Systems (ICICS), pages 059–065. IEEE, 2020.

[223] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu. A detailed and real-time performance monitoring framework for blockchain systems. In Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), pages 134–143. IEEE, 2018.

[224] S. Zheng. Why china's massive data leak is so chilling. https://www.bloomberg.com/news/newsletters/2022-07-11/why-china-s-massive-data-leak-is-so-chilling, July 2022. accessed: July, 2022.

[225] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu. Smart contract development: Challenges and opportunities. IEEE Transactions on Software Engineering, 2019.

[226] G. Zyskind, O. Nathan, and A. Pentland. Enigma: Decentralized computation platform with guaranteed privacy, 2015. arXiv preprint.