國立政治大學資訊科學系

Department of Computer Science

National Chengchi University

碩士學位論文

Master's Thesis

具可控監管與交易保密功能之區塊鏈支付系統

Blockchain-based Confidential Payment System with Controllable Regulation

廖御辰

Yu Chen, Liao

指導教授：左瑞麟 博士

Advisor: Raylin Tso, Ph.D.

中華民國 111 年 8 月

Aug, 2022

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

具可控監管與交易保密功能之區塊鏈支付系統

Blockchain-based Confidential Payment System with Controllable Regulation

碩士生： 廖御辰　Student： Yu Chen, Liao

指導教授： 左瑞麟 博士　Advisor： Raylin Tso, Ph.D.

國立政治大學

資訊科學系

碩士學位論文

A Thesis

submitted to Department of Computer Science

National Chengchi University

in partial fulfillment of the Requirements

for the degree of

Master

in

Computer Science

中華民國 111 年 8 月

Aug, 2022

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

致謝

在就讀研究所的期間，首先要感謝我的指導教授 — 左瑞麟教授，老師不僅在學術上指導我們，也在研究所期間給予我們很多其他的幫助。在論文上老師總是能點出我沒有思考到的方向，並且給予改善的建議。感謝曾一凡教授，在 meeting 時總是能夠給予我們許多啟發，在論文的方面也都可以給予我許多提點。也謝謝子源學長，不僅僅是在論文寫作上給予我很大的幫助，犧牲自己的時間幫助我完成論文，在平時也都幫我們解決許多研究的問題，在生涯規劃的方面也給予我們很多有用的建議。感謝仁傑學長，在密碼學、數學上都可以解決我的疑惑，也總是在 meeting 時提出許多我沒有想到的方向。謝謝實驗室的所有學長姐、同學、學弟以及助理們，在研究所期間從大家的身上都學到了很多，也留下了許多愉快的回憶。感謝從大學到現在的同窗李亦修，陪我度過了研究所的不同階段，帶我接觸到許多虛擬貨幣的知識。也祝福現在在日本進修的實驗室夥伴一切順利。

謝謝我的家人，給予我這麼好的環境，讓我可以心無旁鶩的專注在研究上。謝謝游賀媜，總是一直陪伴、支持著我。

廖御辰 謹誌

政治大學資訊科學系

資訊安全實驗室

111 年 8 月

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

摘要

基於區塊鏈的支付系統（如比特幣等），由於其交易可公開驗證的特性而有廣泛的應用。此外，為了保護使用者在區塊鏈上的隱私，如一交易的收發人以及交易金額等，隱私保護貨幣如門羅幣及 Zerocash 等也因此被提出。然而，在缺乏任何監管措施的情形下，過度的隱私保護系統卻有可能遭到惡意的濫用。因此，如何同時兼顧隱私保護但又保有適度的監管能力是一個非常重要的問題。在此篇論文中，我們提出一個具可控監管與交易保密功能之區塊鏈支付系統。為了保護使用者的隱私且同時進行可控監管，我們使用了門檻同態加密系統來對使用者的交易金額進行加密。此加密使用監管者的門檻加密金鑰，在保護交易隱私的同時也限制了監管者對一交易金額進行解密的能力。此外，藉由同態加密的特性，我們能夠在不對交易金額解密的情形下，更新使用者在經過一交易後的餘額，進而達到保持使用者隱私的目的。我們所提出的系統也能滿足安全性的需求，並且我們也實作了此系統的原型用以效能分析。

關鍵字：可歸責性，區塊鏈，保密交易，門檻加密

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

Abstract

Blockchain-based payment system (e.g., Bitcoin) is wildly adopted in many scenarios due to the transaction details are publicly accessible. In addition, blockchain-based anonymous payment systems (e.g., Monero and Zerocash) have been proposed to further protect on-chain privacy, such as the balance of the sender and receiver, and the amount of the transaction. However, without any regulation, overly privacy-preserving systems will sometimes be abused for malicious behavior. How to strike a balance between the needs for regulation and privacy become a important issue on such systems. In this paper, we proposed a blockchain-based confidential payment system with controllable regulation. To protect user's privacy as well as perform controllable regulation, we realized the proposed system by utilizing threshold homomorphic encryption to encrypt user's transaction values and balance. The encryption is done with regulators' thresholdized keys and thus limits regulators' ability to decrypt a transaction. In addition, with the homomorphic property, we can update the user's balance without decrypting the transaction value or the user's balance and thus preserve on-chain privacy. The proposed system also satisfies the security requirements and and a prototype implementation is provided for the performance analysis.

Keywords:Accountability, Blockchain, Confidential Transaction, Threshold Encryption

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

Contents

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

List of Tables

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

List of Figures

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

List of Definitions

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

1 Intorduction

Blockchain has given rise to many applications because of its immutability, verifiability, and decentralization. Amoung all of them, Financial service like cryptocurrency is one of the most popular utilizations on blockchain. Although cryptocurrency has many advantages over traditional finance tools like better accessibility and instant remittance, privacy is a concern due to the nature of blockchain. In a blockchain system, in order to verify the correctness of a transaction, nodes need to have access to the detail of a transaction like the address of sender, receiver and transaction amount. It may not be suitable to reveal sensitive information to everyone in some scenarios. Most cryptocurrencies like Bitcoin [Nak08] and Ethereum [But14] employed pseudonym mechanisms which allow users to create and send transaction with different addresses to offer anonymity of some degree.

However, studies have shown that using a method like transaction graph analysis [CSL+18] and address clustering [SHL+21], attackers can de-anonymize a user to trace his or her transactions. In some cryptocurrencies like bitcoin, users are able to create different anonymous accounts and use them as multiple payers in one transaction. With graph analysis, attacker can turn every account and transaction flow into a graph and treat different payer accounts in one transaction as one entity. Following the whole transaction graph on the blockchain, the attacker can thus de-anonymize different anonymous accounts. With the above example, we can see that simply creating multiple anonymous accounts is not sufficient. To further improve privacy, decentralized anonymous payment (DAP) schemes like Monero [SAL+17] and Zerocash [BCG+14] are proposed. By using cryptographic techniques like zero-knowledge proof and ring signature, DAP schemes offer a strong level of privacy that hide the sender, receiver, and amount in a transaction. Specifically, Monero uses ring signature to hide the sender inside multiple decoy transactions and uses stealth address to hide the receiver. One can

DOI:10.6814/NCCU202201524

only observed that some unknown amount of coins is moved from one of the member inside the decoy group to a one-time stealth address without knowing which specific address it is. Zerocash on the other hand puts users' coins inside a "shielded pool" and utilizes zero-knowledge proof to prove one's right to spend certain amount of coins inside the pool without reveal the actual value.

Although DAP schemes can largely enhance users' privacy, they may also be abused without any form of regulation. Since there is no authority who controls the user's identity, and details of every transaction in DAP schemes are hidden, it is hard to perform any regulatory measure. Study [WOD18] has shown that cryptocurrency may be used to conduct illegal acts such as money laundering and terrorist funding.

A simple way to tackle this problem is by introducing an additional regulator to the DAP scheme that can decrypt and check every transaction before appending it to the blockchain. Although this measure can eliminate the regulatory concerns mentioned above, it also gives regulators the ability to access details of every transaction. Furthermore, since every transaction must pass through the regulator before appending to the blockchain, the regulator is able to secretly exclude certain transactions from the system to gain an unfair advantage for others and thus harm the system. To eliminate the problem of a central regulator, another way to solve the regulatory problem is by adding zero-knowledge proofs in every transaction. The zero-knowledge proofs make sure a transaction follows certain regulatory policies like limiting the transaction frequency or other anti-money laundering measures. By using zero-knowledge proofs, we can achieve regulatory requirements without introducing any central authorities like a regulator.

However, we claim that zero-knowledge proof is not sufficient for all regulatory needs in practice. In order to utilize zero-knowledge proof, we need to provide rules beforehand and encode them in the system. Any transaction that passes the rule check cannot be monitored or analyzed afterward. According to the United Nations [PW11], only as little as 0.1% to 0.3% of money laundering is detected with the current anti-money laundering policy. Being able to analyze transactions afterward to find the money launderer is important to tackle money laundering [MvF15], and solely relying on certain regulatory policies is not enough.

DOI:10.6814/NCCU202201524

To track a transaction after it has settled, the system requires regulators to analyze transactions. Since we don't want the regulators to fully control the system, some methods are required to limit the regulators' power. The term controllable regulation means that in order to perform regulation, the regulators must fulfill some preset conditions. The conditions is set to control the regulators' power and thus prevent them from overly interfering in the system.

1.1 Contribution

In this thesis, we try to find the balance between privacy and regulation. We do not want the regulator to simply check every transaction's content before appending it to the blockchain, and yet we still want the regulator to be able to perform regulatory tasks if necessary. To achieve this goal, first, we encrypt the transaction amount and user's balance with the regulator's key in order to maintain privacy. With it, we can ensure that anyone besides the regulator will not be able to observe the transaction amount. Second, we partition the role of a validator and a regulator. Validators only check the correctness of a transaction and are responsible for updating the blockchain, as with nodes in some blockchain systems. We use zero-knowledge proof to validate the correctness of a transaction without leaking the actual transaction detail. As for updating the blockchain, we make use of homomorphic encryption to correctly update the user's balance without decrypting it. By utilizing zero-knowledge proof and homomorphic encryption, we can make sure that the validator does not have to decrypt every transaction before validating it and updating the blockchain. Regulators on the other side are special entities that do not participate in any transactions and are only responsible for regulatory tasks. To prevent the regulator from decrypting any transaction at will, we set the number of regulators and a threshold number t at the system setup. Only if at least t regulators participate in the decryption process can the transaction amount be revealed. We make use of threshold encryption to split the private key between regulators, and they must go through a 'voting' process in order to decrypt a transaction. With threshold encryption,

DOI:10.6814/NCCU202201524

we can control the power of regulators in the system. At the end of this thesis, we also give a proof-of-concept implementation of the proposed system. Note that we only hide the transaction amount but leave the sender and the receiver in plaintext. This is due to the need for accountability. If we hide the transaction as well as the sender and the receiver, the regulators will have no choice but to decrypt every transaction in order to supervise the system. This not only leaves our threshold design in vain but also hurt the privacy of the users.

1.2 Related Works

Two seminal confidential blockchain payment schemes are Cryptonote [Sab13] and Zerocoin [IM13]. Cryptonote utilizes traceable ring signatures to hide the sender and recipient of a transaction and gives rise to the Monero protocol which uses similar techniques to hide the sender, recipient as well as the transaction amount. Zerocoin uses zero-knowledge proof and an accumulator scheme to break the linkage between different transactions. Ben-Sasson et al. propose Zerocash [BCG+14] based on Zerocoin. Zerocash formulates decentralized anonymous payment (DAP) schemes and provides strong anonymity by using zk-SNARKs [Pet19] and commitment schemes. On Ethereum, the AZTEC protocol proposed by Zachary J. Williamson [Wil18] uses zero-knowledge proof and commitment scheme to provide verifiable confidential transaction that can hide the transaction amount.

To solve the regulatory issue, many studies proposed different methods to allow regulation in DAP schemes. Garman et al. [GGM16] included users tracing and coins tracing in Zerocash using zk-SNARKs. Lin et al. proposed a decentralized condition anonymous payment system DCAP [LHH+20] which hides the sender and recipient of a transaction. However, the manager in DCAP needs to obtain the real addresses of the sender and recipient to validate each transaction and append the transaction on the blockchain, causing the problem that the manager can reject any transactions. Cecchetti et al. presented Solidus [CZJ+17] that uses publicly-verifiable oblivious RAM. Solidus

DOI:10.6814/NCCU202201524

allows only the users' banks to obtain their identity, but others can only learn the identities of the banks. Therefore, this is suitable for the modern financial system. PGC proposed by Chen et al. [CMT$^+$20] uses zero-knowledge proof to ensure transactions which the transaction amounts are hidden follow certain policies. Xue et al. [XLN$^+$22] also proposed a payment system that uses zero-knowledge proof to make sure compliance with certain policies of a transaction. While using zero-knowledge proof can maintain decentralization while providing regulatory means, it may not be sufficient for all the regulatory needs as we described in the previous paragraph.

1.3 Organization

The thesis is organized as follows: Chapter 2 presents the cryptographic primitives that will be used in the construction of our system as well as in the security description. Chapter 3 first gives a high-level view of our system and then defines the system model and security model. Chapter 4 presents an in-depth view of our proposed system. Chapter 5 provides the security description of the system, including authenticity, confidentiality, balance and consistent amount. Chapter 6 presents an efficiency analysis of the proposed system with a proof-of-concept implementation. Chapter 7 concludes our proposed system and provides directions for future work.

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

2 Preliminaries

2.1 Commitment scheme

Commitment scheme allows one to commit to a value in a hidden form and later reveals the value to show that it is indeed the one committed. Informally, a commitment scheme consists of commit phase and reveal phase. In commit phase, one chooses a value and commit to it. In reveal phase, one reveals the committed value and other auxiliary information for others to verify the commitment.

Formally, commitment scheme consists of three algorithms $(Setup, Commit, Open)$:

- $Setup(\lambda) \to pp$. On inputting of a security parameter λ, it outputs public parameters pp. Here pp includes message space M, commitment space C and ramdom space R. pp will be an implicit input in the following algorithms.

- $Commit(m, r) \to c$. On inputting of a message m and a random number r, the commit algorithm outputs a commitment c to m.

- $Open(c, r, m) \to 1/0$. On inputting of a commitment c, a random number r and a message m, the reveal algorithm outputs 1 if $c = Commit(m, r)$ and 0 otherwise.

Definition 2.1 (Correctness of Commitment scheme). *For all $pp \leftarrow Setup(\lambda)$, $m \in M$,*

$$\Pr[Open(Commit(m, r), r, m) = 1] = 1.$$

Definition 2.2 (Hiding). *For a probabilistic polynomial time adversary $\mathcal{A}$, a Commitment scheme satisfies Hiding if $\mathcal{A}$'s advantage over the following experiment is negligible.*

DOI:10.6814/NCCU202201524

```
Experiment $\mathcal{E}_{Hiding}$
1 : $pp \leftarrow Setup(\lambda)$
2 : $(m_0, m_1) \leftarrow \mathcal{A}(pp)$
3 : $b \xleftarrow{\$} \{0,1\}; r \xleftarrow{\$} R$
4 : $c \leftarrow Commit(m_b, r)$
5 : $b' \leftarrow \mathcal{A}(c)$
6 : return $b' = b$
```

We define $\mathcal{A}$'s advantage over $\mathcal{E}_{Hiding}$ as follows,

$$Adv_{\mathcal{A}}^{Hiding}(\lambda) := \left|\Pr[\mathcal{E}_{Hiding} = 1] - \frac{1}{2}\right|.$$

Definition 2.3 (Binding)**.** *For a probabilistic polynomial time adversary $\mathcal{A}$, a Commitment scheme satisfies Binding if $\mathcal{A}$'s advantage over the following experiment is negligible.*

```
Experiment $\mathcal{E}_{Binding}$
1 : $pp \leftarrow Setup(\lambda)$
2 : $(c, m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(pp)$
3 : return $m_0 \neq m_1 \wedge c = Com(m_0, r_0) = Com(m_1, r_1)$
```

We define $\mathcal{A}$'s advantage over $\mathcal{E}_{Binding}$ as follows,

$$Adv_{\mathcal{A}}^{Binding}(\lambda) := \Pr[\mathcal{E}_{Binding} = 1].$$

Pedersen commitment. Below we review the Pedersen commitment scheme [Ped92]:

1. $Setup(\lambda) \rightarrow pp$. On inputting of a security parameter λ, it outputs public parameters pp. Here pp includes a group G of order q and it's two generators g, h. The message space M and random space R are $\mathbb{Z}_q$, and commitment space C is G.

2. $Commit(m, r) \rightarrow c$. On inputting of a message $m \in \mathbb{Z}_q$ and a random number $r \xleftarrow{\$} \mathbb{Z}_q$, it outputs a commitment $c \leftarrow g^m h^r$.

3. $Open(c, r, m) \rightarrow 1/0$. On inputting of a commitment c, a random number r and a message m, it outputs 1 if $c = g^m h^r$ and 0 otherwise.

DOI:10.6814/NCCU202201524

2.2 Public-Key Encryption

Public-Key Encryption consists of three algorithms $(Setup, KeyGen, Encrypt, Decrypt)$:

- $Setup(\lambda) \rightarrow pp$. On inputting of a security parameter λ, it outputs public parameters pp. Here pp also consists of message space M and ciphertext space C.
- $KeyGen(pp) \rightarrow (pk, sk)$. On inputting of public parameters pp, it outputs a pair of public/private keys (pk, sk).
- $Encrypt(pk, m) \rightarrow c_m$. On inputting of a public key pk, message m, the algorithm outputs a ciphertext c_m of m under pk.
- $Decrypt(sk, c_m) \rightarrow m/\perp$. On inputting of a secret key sk, a ciphtertext c_m, the algorithm outputs a plaintext message m or a special symbol $\perp$ denotes decryption fail.

Definition 2.4 (Correctness of Public-Key Encryption)**.** *For all $pp \leftarrow Setup(\lambda)$, $(pk, sk) \leftarrow KeyGen(pp)$, and $m \in M$,*

$$\Pr[Decrypt(sk, Encrypt(pk, m)) = m] = 1.$$

Definition 2.5 (IND-CPA)**.** *For a probabilistic polynomial time adversary $\mathcal{A}$, a Public-Key Encryption scheme satisfies indistinguishability under chosen plaintext attack (IND-CPA) if $\mathcal{A}$'s advantage over the following experiment is negligible.*

```
Experiment $\mathcal{E}_{IND-CPA}$
1 : $pp \leftarrow Setup(\lambda)$
2 : $(pk, sk) \leftarrow KeyGen(pp)$
3 : $(m_0, m_1) \leftarrow \mathcal{A}(pk)$
4 : $b \xleftarrow{\$} \{0, 1\}$
5 : $c \leftarrow Encrypt(pk, m_b)$
6 : $b' \leftarrow \mathcal{A}(pk, c)$
7 : return $b' = b$
```

DOI:10.6814/NCCU202201524

We define $\mathcal{A}$'s advantage over $\mathcal{E}_{IND-CPA}$ as follows,

$$Adv_{\mathcal{A}}^{IND-CPA}(\lambda) := \left|\Pr[\mathcal{E}_{IND-CPA} = 1] - \frac{1}{2}\right|.$$

2.3 Threshold Cryptosystem without a Trusted Party

Informally, in threshold cryptosystems [Ped91], a secret key is split among n members, and a ciphertext can only be decrypted if more than t members ($1 \leq t \leq n$) cooperate. In particular, if the process of selection and distribution of a secret key can be done without assistance from a trusted party, then this threshold cryptosystem is further called threshold cryptosystem without a trusted party.

Formally, a threshold cryptosystem without a trusted party consists of six algorithms $(Setup, GenPK, GenSharedSK, Encrypt, Decrypt, CombineShare)$:

- $Setup(\lambda) \rightarrow pp$. On inputting of a security parameter λ, it outputs public parameters pp. Here, the threshold value t is implicitly contained in pp.
- $GenPK(pp) \rightarrow pk, \{pk_1, pk_2, \cdots, pk_n\}$. On inputting of the public parameters pp, each member of the group generates his or her share of the public key pk_i where $i \in \{1, 2, \cdots, n\}$ and then broadcasts it to other users. After receiving all other's shares, each member can calculate and output the same public key pk.
- $GenSkShare(pp, pk_i, pk) \rightarrow sk_i$. On inputting of the public parameters pp, a share of public key pk_i, and the public key pk, each member of the group outputs his or her share of the secret key sk_i.
- $Encrypt(pk, m) \rightarrow c$. On inputting of the public key pk and the message m, it outputs the ciphertext c of the message.
- $DecryptShare(sk_i, c) \rightarrow sh_i$. On inputting of a share of the secret key sk_i and the ciphertext c, it outputs a share of the message sh_i of member i.
- $CombineShare(sh_1, sh_2, \cdots, sh_j) \rightarrow m/\perp$. On inputting of j shares of the message m for some $j \geq t$, it outputs the message m of the ciphertext c, otherwise it outputs $\perp$ indicating decryption fail.

DOI:10.6814/NCCU202201524

2.4 Additive Homomorphic Encryption

Additive homomorphic encryption is a cryptosystem that permits the user to perform additive computation in the form of a ciphertext. Formally, additive homomorphic encryption consists of five algorithms $(Setup, KeyGen, Encrypt,$
$Decrypt, Add)$:

- $Setup(\lambda) \rightarrow pp$. On inputting of the security parameters λ, it outputs public parameters pp.
- $KeyGen(pp) \rightarrow (pk, sk)$. On inputting of the public parameters pp, it outputs a pair of public/private keys (pk, sk).
- $Encrypt(pk, m) \rightarrow c$. On inputting of a public key pk and a message m, it outputs a ciphertext c of the message m.
- $Decrypt(sk, c) \rightarrow m/\perp$. On inputting of a secret key sk and a ciphertext c, output the origin message m of ciphertext c or a special symbol $\perp$ denotes decryption fail.
- $Add(c_1, c_2) \rightarrow c_{1+2}$. On inputting of two ciphertext c_1, c_2 related to the same public key pk, it outputs a ciphertext c_{1+2} such that $Decrypt(sk, c_{1+2}) = Decrypt(sk, c_1) + Decrypt(sk, c_2)$.

2.5 Digital Signature

Digital signature consists of three algorithms $(Setup, KeyGen, Sign, Verify)$:

- $Setup(\lambda) \rightarrow pp$. On inputting of a security parameter λ, it outputs public parameters pp. Here pp also includes message space M and signature space Σ.
- $KeyGen(pp) \rightarrow (pk, sk)$. On inputting of a security parameter λ, the key generation algorithm returns a public/secret key pair (pk, sk).

DOI:10.6814/NCCU202201524

- $Sign(sk, m) \to (\sigma_m)$. On inputting of a secret key sk and a message m, the signing algorithm returns a digital signature σ_m of the message m under sk.

- $Verify(pk, m, \sigma_m) \to 1/0$. On inputting of a public key pk, a message m and a signature σ_m, the verify algorithm returns 1 if σ_m is a valid signature of m under sk and returns 0 otherwise.

Definition 2.6 (Correctness of Digital Signature)**.** *For all $pp \leftarrow Setup(\lambda)$, $(pk, sk) \leftarrow KeyGen(pp)$, and $m \in M$,*

$$\Pr[Verify(pk, m, Sign(sk, m)) = 1] = 1.$$

Definition 2.7 (EUF-CMA)**.** *For a probabilistic polynomial time adversary $\mathcal{A}$, a Digital Signature scheme satisfies existential unforgeability under the chosen message attacks (EUF-CMA) if $\mathcal{A}$'s advantage over the following experiment is negligible.*

```
Experiment $\mathcal{E}_{EUF-CMA}$
1 : $pp \leftarrow Setup(\lambda)$
2 : $(pk, sk) \leftarrow KeyGen(pp)$
3 : $\mathcal{A}$ queries for the signatures of $Q = \{m_1, m_2, \cdots, m_n\}$
4 : $(m^*, \sigma^*) \leftarrow \mathcal{A}(pk)$
5 : return $Verify(pk, m^*, \sigma^*) = 1 \wedge m^* \notin Q$
```

We define $\mathcal{A}$'s advantage over $\mathcal{E}_{EUF-CMA}$ as follows,

$$Adv_{\mathcal{A}}^{IND-CPA}(\lambda) := \Pr[\mathcal{E}_{EUF-CMA} = 1].$$

2.6 Zero-knowledge Proofs

For a language $L := \{x \mid \exists w \text{ s.t. } R(x, w) = 1\}$, where w is a witness of x, zero-knowledge proofs allow a prover who possesses w to prove $x \in L$ without leaking any additional knowledge about w. In particular, a zero-knowledge proof scheme must satisfies the following properties:

DOI:10.6814/NCCU202201524

1. *Completeness*: If a statement $x \in L$, then the prover must succeed in convincing the verifier.

2. *Soundness*: If a statement $x \notin L$, then the prover must fail to convince the verifier.

3. *Zero-knowledge*: Besides the correctness of a statement, the process must not reveal any additional information.

Σ-protocols [CG15; Kra03] can be regarded as interactive zero-knowledge proofs with three phases: *Commitment*, *Challenge*, and *Response*, interacting between a prover P and a verifier V.

Let G be a multiplicative cyclic group with generator g, the following we introduce a flow of a Σ-protocol that proves the possession of $x \in \mathbb{Z}_q$ such that $y = g^x$:

1. *Commitment*: P randomly chooses $r \in \mathbb{Z}_q$ and calculate $T = g^r$. Afterward, P sends T to V.

2. *Challenge*: V chooses a random $c \in \mathbb{Z}_q$ and send c to P as a challenge.

3. *Response*: P calculates a response $s = r - cx \mod p$ and sends the response to V.

4. V verifies the response by validating whether $y^c g^s$ is equal to T, if so, outputs 1. Otherwise, V outputs 0.

Here we note that by using the Fiat-Shamir transform [FS86] (i.e., using the hash of commitment produced by P as a challenge, without the interaction with V), the above protocol can become a non-interactive protocol.

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

3 Confidential Blockchain Payment System with Regulation

3.1 System Description

To reach our goal of protecting user privacy while being regulated, we introduce a novel confidential blockchain payment system with regulation. As shown in Fig. 3.1, we focus on distributing the power of regulators and separating the role of regulators and validators. The description of each entity is as follows:

- *Users*: Users have their own ledger public keys and secret keys. They can send and receive transactions via their keys. The balance of each user is encrypted by the regulators' public key as well as their own public keys.

- *Validators*: Validators are entities for validating each transaction, but they only validate the correctness of a transaction instead of its legitimacy. Validators may also initiate a transaction. In this system, we intentionally separate the role of regulators from validators so that even without regulators, the system can still function like a normal blockchain payment system. Actually, validators can be regarded as miners of blockchains like Bitcoin and Ethereum.

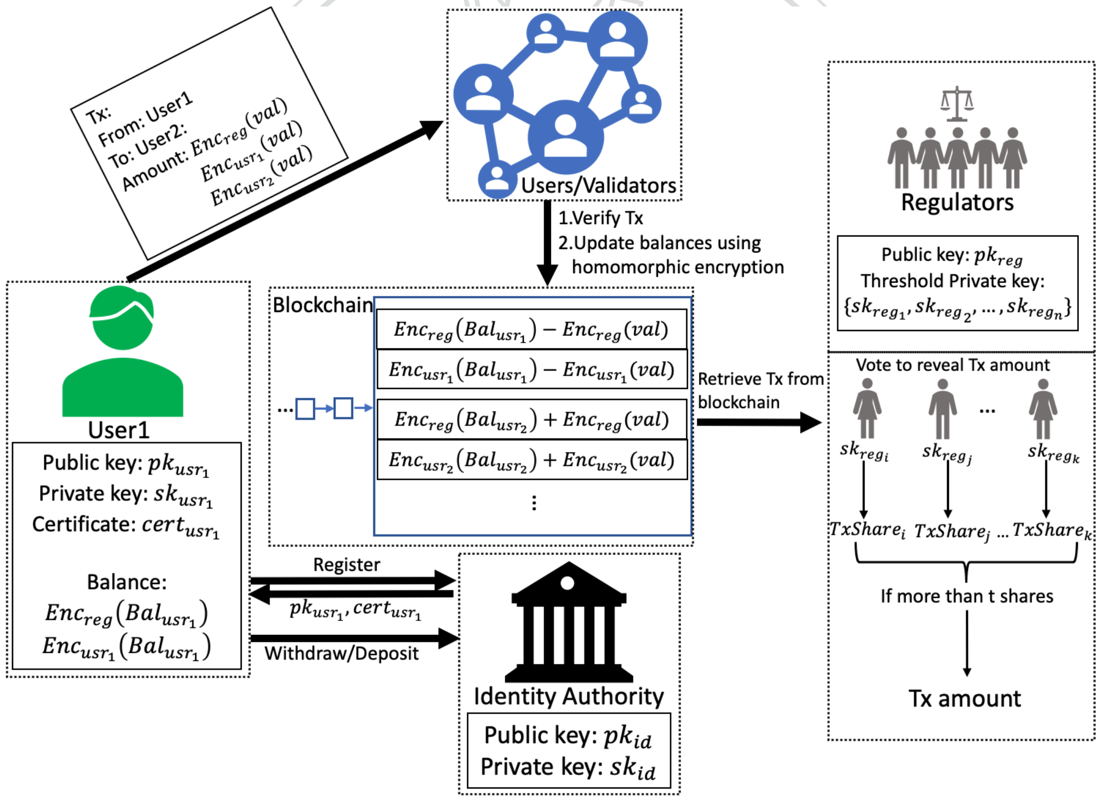
- *Regulators*: Regulators are special nodes on the system who are capable of checking the legitimacy of certain transactions. They cannot be the sender or receiver of a transaction. The regulators share one common public key, and the corresponding secret key is thresholdized and split between each regulator.

- *Identity Authority*: Identity authority is in charge of linking a user's public key to

DOI:10.6814/NCCU202201524

his real identity. It can also send a transaction to adjust user's balance if a user deposits assets onto the blockchain. The identity authority essentially works as a bridge between any data on-chain and off-chain. It can be a commercial bank that holds a user's identity and assets in real life. Note that the identity authority can only know the amount of a transaction initiated or received by it, that is, for a deposit or withdrawal and can not see any other transaction's amount on chain.

Note that in the concrete construction of the system, the public key is required to be additive homomorphic in order to correctly update the balance of both parties of a transaction.



Tx:
From: User1
To: User2:
Amount: $Enc_{reg}(val)$
$Enc_{usr_1}(val)$
$Enc_{usr_2}(val)$
Users/Validators
1.Verify Tx
2.Update balances using homomorphic encryption
Regulators
Public key: pk_{reg}
Threshold Private key: $\{sk_{reg_1}, sk_{reg_2}, \dots, sk_{reg_n}\}$
Blockchain
$Enc_{reg}(Bal_{usr_1}) - Enc_{reg}(val)$
$Enc_{usr_1}(Bal_{usr_1}) - Enc_{usr_1}(val)$
$Enc_{reg}(Bal_{usr_2}) + Enc_{reg}(val)$
$Enc_{usr_2}(Bal_{usr_2}) + Enc_{usr_2}(val)$
Retrieve Tx from blockchain
Vote to reveal Tx amount
sk_{reg_i} sk_{reg_j} sk_{reg_k}
$TxShare_i$ $TxShare_j$ ... $TxShare_k$
If more than t shares
Tx amount
User1
Public key: pk_{usr_1}
Private key: sk_{usr_1}
Certificate: $cert_{usr_1}$
Balance:
$Enc_{reg}(Bal_{usr_1})$
$Enc_{usr_1}(Bal_{usr_1})$
Register
$pk_{usr_1}, cert_{usr_1}$
Withdraw/Deposit
Identity Authority
Public key: pk_{id}
Private key: sk_{id}

Figure 3.1: The description of the proposed systems

The workflow of our system can be simplified as follows: Depending on the security parameter, the size of the regulators n is defined. The system also define a threshold number t where $t \leq n$. The threshold t decides the minimum number of regulators that need to cooperate together in order to decrypt one transaction. The higher the number

DOI:10.6814/NCCU202201524

is, the more difficult it is for the regulators to decrypt a transaction. Thus preventing the regulators from arbitrarily decrypting any transactions. Our system uses ElGamal encryption with threshold private keys [DF89] as it is easy to construct an additive homomorphic ElGamal encryption for our needs. As a special entity on the blockchain, the regulators first generate the shared regulator public key, then they calculate each one's own private key based on t and the public key. Part of the transaction and user's balance will be encrypted by the regulators' key for regulatory purpose.

To link an account to its real identity, the system sets up a public key pair for the identity authority. Each user needs to register with the identity authority using their personal information to acquire their public key and certificate on the blockchain system. The certificate will be included in every transaction and checked by the validator using the public key of identity authority. The identity authority can also send a special transaction to the blockchain to adjust a user's balance if the user deposit assets into the blockchain. If a user want to withdraw assets out of the blockchain, he can send a transaction to the identity authority with the amount he wants to withdraw.

Each user's balance is stored as two copies on the blockchain, one encrypted by the regulators' public key and another encrypted by his own public key. We do so in order to preserve privacy and allow regulation. To initialte a transaction, the user first encrypts the transaction amount by the regulators' public key. In order for the recipient to see the amount, user also needs to encrypts the transaction amount with recipient's public key. In addition, the user needs to encrypt the amount with his own public key so that his balance can be correctly updated.

Since there are several ciphertexts inside a transaction, the transaction also needs to include zero-knowledge proofs for the validators to check the correctness of those ciphertexts. After generating the transaction, the user then signs the transaction with his private key and broadcasts it on the network. On receiving a transaction, the validators validate the transaction by checking its zero-knowledge proofs, signature, and certificate before appending it to the blockchain. Since we encrypt both the transaction amount and the balance, we rely on the additive homomorphic property of our encryption scheme to correctly update the balance.

DOI:10.6814/NCCU202201524

When a regulator wants to decrypt a transaction, he sends the transaction to other regulators with the reasons it needs to be decrypted. If a regulator agrees, he then generates a decryption share of the transaction using his own secret key and sends the decryption share to other regulators. After gathering more than t decryption shares of the transaction, the regulators can then decrypt the transaction. The process of collecting decryption shares can be analogous to a voting process between the regulators. With the voting process, we can prevent the regulators from arbitrarily decrypting every transaction.

3.2 System Model

The proposed system consists of seven algorithms ($Setup$, $IdSetup$, $RegSetup$, $UsrRegister$, $Send, Validate, Reveal, Deposit, Withdraw$), which are described as follows.

- $Setup(\lambda) \rightarrow pp$. On inputting a security parameter λ, $Setup$ algorithm outputs public parameters pp.
- $IdSetup(pp) \rightarrow (pk_{id}, sk_{id})$. On inputting the public parameters pp, $IdSetup$ returns a key pair pk_{id}, sk_{id} of the identity authority.
- $RegSetup(pp, n, t) \rightarrow (pk_{reg}, \{sk_{reg_1}, sk_{reg_2}, \cdots, sk_{reg_n}\})$. On inputting the public parameters pp, the group size of regulators n, and the intended threshold number of the system t, $RegSetup$ returns the public key of regulators pk_{reg} and the thresholdized private keys $\{sk_{reg_1}, sk_{reg_2}, \cdots, sk_{reg_n}\}$.
- $UsrRegister(pp, pk_{id}, sk_{id}, usrID_i) \rightarrow (pk_{usr_i}, sk_{usr_i}, cert_{usr_i})$. On inputting the public parameters pp, the key pair of the identity authority (pk_{id}, sk_{id}), and the user identity $usrID$ required by the system, $UsrRegister$ returns a public/private key pair of the user, (pk_{usr_i}, sk_{usr_i}) and the corresponding certificate $cert_{usr_i}$ generated by the identity authority. Note that this is an interactive algorithm between user and identity authority, and the identity authority only sees the certificate of a user but not the key pair.

DOI:10.6814/NCCU202201524

- $Send(pp, pk_{usr_i}, sk_{usr_i}, pk_{usr_j}, val, cert_{usr_i}) \rightarrow (tx, \sigma_{tx})$. On inputting the public parameters pp, the public/private key pair (pk_{usr_i}, sk_{usr_i}) of the sender, the receiver's public key pk_{usr_j}, a transaction amount val, and the certificate $cert_{usr_i}$ of the sender, $Send$ outputs a transaction tx and a corresponding signature σ_{tx} generated by the sender.

- $Validate(tx) \rightarrow 1/0$. On inputting a transaction, $Validate$ output 1 if tx is valid; otherwise, it outputs 0.

- $Reveal(pp, tx, \{sk_{reg_i}, sk_{reg_j}, \cdots, sk_{reg_k}\}) \rightarrow val/\perp$. On inputting the public parameters pp, a transaction tx, and a set of regulators' keys, $Reveal$ returns the original amount val of the transaction or $\perp$ indicates failure.

- $Deposit(pp, pk_{id}, sk_{id}, pk_{usr_i}, val) \rightarrow (tx_{dep}, \sigma_{tx_{dep}})$. On inputting the public parameters pp, the public key and secret key of identity authority, pk_{id}, sk_{id}, a user's public key pk_{usr_i} and a transaction amount val, $Deposit$ returns a special deposit transaction tx_{dep} and a corresponding signature $\sigma_{tx_{dep}}$ generated by the identity authority.

- $Withdraw(pp, pk_{usr_i}, sk_{usr_i}, pk_{id}, val, cert_{usr_i}) \rightarrow (tx_{wit}, \sigma_{tx_{wit}})$. $Withdraw$ algorithm is the same as $Send$ algorithm except the receiver is set to the identity authority.

3.3 Security Model

For our regulated confidential payment system, it should provide $Authenticity$, $Confidentiality$, and $Balance$ as other payment systems. $Authenticity$ requires that the sender of a transaction must own the account (knows the private key of the sender's account), nobody else is capable of making a transaction from this account. $Confidentiality$ requires that besides the sender, receiver, and regulators, no one can obtain any information about the hidden amount of a confidential transaction. $Balance$ requires that the balance of sender and receiver should be correctly updated with the

DOI:10.6814/NCCU202201524

transaction amount. In addition, we further consider the $ConsistentAmount$ property in the proposed system. More concretely, since the transaction amount will be encrypted by regulators' public key, sender's public key and receiver's public key, we need this property to ensure the transaction amount should be consistent when viewed by a sender, receiver, or regulators.

We formally define the properties by the following games interacted between a challenger C, an adversary $\mathcal{A}$. The adversary $\mathcal{A}$ can send C with following queries include $UsrRegister$, $Send$, and $Reveal$. After getting the queries, C checks the formality and forwards the queries to an oracle O^{CBP} of the payment system. Here, O^{CBP} is initialized by C, and O^{CBP} maintains a ledger followed by the system and updates it correspondingly with the queries from $\mathcal{A}$. Note that $\mathcal{A}$ cannot acquire the private inputs of a transaction. $\mathcal{A}$ can only query $Reveal$ to reveal the amount of a transaction sent by or sent to a user created by $\mathcal{A}$ with $UsrRegister$ and reveal the balance of a user created by $\mathcal{A}$. The $Reveal$ query reveals the amount or balance encrypted with the regulators' key, which is stored on the ledger.

Game - Authenticity:

- *Initialization*: C initializes an oracle O^{CBP} which maintains a ledger L and follows the system with different queries. O^{CBP} provides $\mathcal{A}$ with the view of L and also maintains a user list U that is created by $\mathcal{A}$ with $UsrRegister$ queries.

- *Queries*: $\mathcal{A}$ adaptively sends queries to C who then forwards the queries to O^{CBP} if they pass the formality test. O^{CBP} then updates the ledger corresponding to different queries and returns the result to C. Finally, C forwards the result to the $\mathcal{A}$.

- *Output*: $\mathcal{A}$ outputs a transaction tx' where the sender of tx' is denoted by S'. We say $\mathcal{A}$ wins the game if $Validate(tx') = 1$ and $S' \notin U$.

The advantage of $\mathcal{A}$ in winning the above game is defined as

$$Adv^{Aut}_{CBPS,\mathcal{A}}(\lambda) := \Pr[Validate(tx') = 1 \wedge S' \notin U].$$

DOI:10.6814/NCCU202201524

Definition 3.1 (Authenticity). *A confidential blockchain payment system satisfies authenticity if, for any PPT $\mathcal{A}$, $Adv^{Aut}_{CBPS,\mathcal{A}}(\lambda)$ is negligible.*

Game - Confidentiality:

- *Initialization*: C randomly selects $b \in \{0, 1\}$. Then, he/she also initializes an oracle O^{CBP} as the previous game.
- *Queries*: $\mathcal{A}$ can submit queries to C freely, but each time $\mathcal{A}$ submit $Send$ query to C, $\mathcal{A}$ also provides the value v inside the transaction to C. Upon receiving a $Send$ query from $\mathcal{A}$, C creates a transaction tx' where the sender and receiver are not in U. The value in tx' is set to v if $b = 0$ or a random number r if $b = 1$. Afterward, C returns tx' to $\mathcal{A}$.
- *Output*: $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. We say $\mathcal{A}$ wins the game if $b' = b$.

The advantage of $\mathcal{A}$ in winning the above game is defined as

$$Adv^{Con}_{CBPS,\mathcal{A}}(\lambda) := \Pr[b' = b] - \frac{1}{2}.$$

Definition 3.2 (Confidentiality). *A confidential blockchain payment system satisfies confidentiality if, for any PPT $\mathcal{A}$, $Adv^{Con}_{CBPS,\mathcal{A}}(\lambda)$ is negligible.*

Game - Balance:

- *Initialization, Queries*: These two phases are the same as phases in the Authenticity game.
- *Output*: $\mathcal{A}$ outputs a transaction tx'. Let val be the transaction amount in tx', S be the sender, and R be the receiver. In addition, let S's and R's balance difference before and after the transaction be S_{dif} and R_{dif}, respectively. We say $\mathcal{A}$ wins the game if $Validate(tx') = 1$ and the equation $-S_{dif} = R_{dif} = val$ does not hold.

The advantage of $\mathcal{A}$ in winning the above game is defined as

$$Adv^{Bal}_{CBPS,\mathcal{A}}(\lambda) := \Pr[Validate(tx') = 1 \wedge -S_{dif} = R_{dif} = val \text{ does not hold}].$$

DOI:10.6814/NCCU202201524

Definition 3.3 (Balance)**.** *A confidential blockchain payment system satisfies balance if, for any PPT $\mathcal{A}$, $Adv^{Bal}_{CBPS,\mathcal{A}}(\lambda)$ is negligible.*

Game - ConsistentAmount:

- *Initialization, Queries*: These two phases are the same as phases in the Authenticity game.
- *Output*: $\mathcal{A}$ outputs a transaction tx'. Let $v_{am,S}$, $v_{amo,R}$, and $v_{amo,reg}$ be the view of amount for sender, receiver, and regulators. We say $\mathcal{A}$ wins the game if $Validate(tx') = 1$ and $v_{am,S} = v_{amo,R} = v_{amo,reg}$ does not hold.

The advantage of $\mathcal{A}$ in winning the above game is defined as

$$Adv^{ConAmo}_{CBPS,\mathcal{A}}(\lambda) := \Pr[Validate(tx') = 1 \wedge v_{am,S} = v_{amo,R} = v_{amo,reg} \text{ does not hold}].$$

Definition 3.4 (ConsistentAmount)**.** *A confidential blockchain payment system satisfies consistentAmount if, for any PPT $\mathcal{A}$, $Adv^{ConAmo}_{CBPS,\mathcal{A}}(\lambda)$ is negligible.*

DOI:10.6814/NCCU202201524

4 The Proposed System

In this chapter, we describe the details of the proposed system.

Setup. With the input of a security parameter λ, this algorithm chooses a generator g of a group G with prime order q. Then, it sets the public parameter as $pp = (q, g, G)$.

IdSetup. The identity authority randomly chooses private key $sk_{id} \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets public key as $pk_{id} = g^{sk_{id}}$.

RegSetup. The regulators invoke this algorithm to get their shared public key and their own thresholdized private keys. Let n be the number of regulators, and $t \le n$ be the threshold size. The regulators are described as $R = \{R_1, R_2, \cdots, R_n\}$. Let $C(m, r)$ denote a commitment on $m \in \{0, 1\}^*$ with a random r. The regulators generate a shared public key pk_{reg} as follows:

1. For each $R_i \in R$, R_i randomly chooses $x_i \overset{\$}{\leftarrow} \mathbb{Z}_q$ and computes $h_i = g^{x_i}$. Afterward, R_i chooses another random $r_i \in \mathbb{Z}_q$ and generates a commitment $C_i = C(h_i, r_i)$. Finally R_i broadcasts (C_i, r_i, h_i) to all members belong to R.

2. After all members in R have broadcast their commitments, each R_i opens all commitments $\{C_1, C_2, \cdots, C_n\}$.

3. The shared public key pk_{reg} is set as $pk_{reg} = \prod_{i=1}^{n} h_i = g^{\sum_{i=1}^{n} x_i}$.

Although all regulators have obtained the public key pk_{reg}, they cannot compute the corresponding private key $sk_{reg} = \sum_{i=1}^{n} x_i$ unless they collude with each other. Therefore, each R_i in R has to run the following procedure to obtain his/her thresholdized private key:

DOI:10.6814/NCCU202201524

1. R_i randomly generates a $t-1$ degree polynomial $f_i(z) = f_{i,0} + f_{i,1}z + \cdots + f_{i,t-1}z^{t-1}$ with $f_{i,0} = x_i$ (i.e., $f_i(0) = x_i$).

2. R_i calculates $F_{i,j} = g^{f_{i,j}}$ for $j = 0, \cdots, t-1$ and broadcast $(F_{i,1}, \cdots, F_{i,t-1})$ to other regulators. Here we note that $F_{i,0} = g^{f_{i,0}} = g^{x_i} = h_i$ is already known by other members.

3. R_i sends $s_{i,j} = f_i(j)$ to R_j for $j = 1, \cdots, n$ via a secure channel.

4. R_i verifies $s_{j,i}$ sent from R_j by checking $g^{s_{j,i}} \overset{?}{=} \prod_{\ell=0}^{t-1} F_{j,\ell}^{i^\ell}$. If failed, R_i publish $s_{j,i}$ and terminates.

5. Finally, R_i computes his or her thresholdized private key $sk_{reg_i} = \sum_{j=1}^{n} s_{j,i}$. Then, if more than t members cooperate, they can decrypt a ciphertext encrypted with pk_{reg}.

UsrRegister. In this system, each user has to register to the identity authority before the transaction for accountability purposes. Note that the identity requirement of a user can be different depending on the system implementation.

The initial balance of every user is set to zero and encrypted by the regulators' public key and the user's own public key. We use additive homomorphic ElGamal as the encryption scheme, so the balance can be correctly updated while encrypted. We use the certified key generation protocol [AFM⁺14] in our system; that is, each user's public key has to be certified by the identity authority. The protocol runs between user usr and the identity authority id as follows:

1. usr randomly chooses k in $[0, q-1]$, computes $z = g^k$ and sends to id. usr also sends his or her identity information required by the system to id.

2. After receiving the identity information and z from usr. id first verifies the identity information. If failed, abort the protocol, else id picks random k' in $[0, q-1]$ and computes $cert_{usr} = z \cdot g^{k'}$, $\bar{x} = k' + cert_{usr} \cdot sk_{id}$. Afterward, id sends $(cert_{usr}, \bar{x})$ to usr. Note that when using in elliptic curve setting, we need an additional function to map $cert$ from a point on elliptic curve to its underlying

DOI:10.6814/NCCU202201524

finite field when calculating $\bar{x}$. It can be simply implemented as an encoding function that map an element of G to a positive number.

3. After receiving $(cert_{usr}, \bar{x})$ from id, usr sets his or her private key as $sk_{usr} = \bar{x} + k$ and public key as $pk_{usr} = g^{sk_{usr}}$. Here, anyone can easily verify the correctness of the public key by using $cert_{usr}$ as follows:

$$\begin{aligned} pk_{usr} = g^{sk_{usr}} &= g^{k+k'+cert_{usr}\cdot sk_{id}} \\ &= g^{k+k'} \cdot g^{cert_{usr}\cdot sk_{id}} \\ &= cert_{usr} \cdot {pk_{id}}^{cert_{usr}}. \end{aligned}$$

We also note that usr's public key pk_{usr} and certificate $cert_{usr}$ should be included in every transaction such that anyone can verify them.

4. The balance of usr is set to zero and encrypted with the regulators' public key pk_{reg} and usr's own public key pk_{usr} with random numbers r_{init_r} and r_{init_u}. The random number used here will be updated when the user receives or sends a new transaction. Finally, the initial balances after encryption are as follows:

$$renc\text{-}bal_{usr} = (g^{r_{initr}}, g^0 \cdot {pk_{reg}}^{r_{initr}}).$$
$$uenc\text{-}bal_{usr} = (g^{r_{initu}}, g^0 \cdot {pk_{usr}}^{r_{initu}}).$$

Send. This algorithm enables a user to send a transaction without revealing the amount. The sender and receiver are still visible for the purpose of regulation. To achieve this requirement, we use ElGamal encryption to encrypt the transaction amount and the balance of every user. Since ElGamal encryption is additive homomorphic, we are able to correctly maintain the balance of each account even if the transaction is hidden. In addition, to ensure a sender follows the protocol during generating a transaction, the sender must generate zero-knowledge proofs to prove the correctness of a transaction.

Let sender usr_i's encrypted balances be $renc\text{-}bal_{usr_i} = (g^{r_{i_r}}, g^{bal_{usr_i}} \cdot {pk_{reg}}^{r_{i_r}})$ and $uenc\text{-}bal_{usr_i} = (g^{r_{i_u}}, g^{bal_{usr_i}} \cdot {pk_{usr_i}}^{r_{i_u}})$. Let receiver usr_j's encrypted balances be

DOI:10.6814/NCCU202201524

$renc\text{-}bal_{usr_j} = (g^{r_{jr}}, g^{bal_{usr_j}} \cdot {pk_{reg}}^{r_{jr}})$ and $uenc\text{-}bal_{usr_j} = (g^{r_{ju}}, g^{bal_{usr_j}} \cdot {pk_{usr_j}}^{r_{ju}})$. Here, bal_{usr_i} and bal_{usr_j} are the balances of usr_i and usr_j respectively. The process of sending a transaction is as follows:

1. The sender usr_i first encrypts the amount val with the regulator's public key pk_{reg}, usr_i's own public key pk_{usr_i} and usr_j's public key pk_{usr_j}. usr_i randomly chooses $r_1, r_2, r_3 \overset{\$}{\leftarrow} [0, q-1]$ and calculates

$$e_{reg} = (g^{r_1}, g^{val} \cdot {pk_{reg}}^{r_1});\ e_{usr_i} = (g^{r_2}, g^{val} \cdot {pk_{usr_i}}^{r_2});\ e_{usr_j} = (g^{r_3}, g^{val} \cdot {pk_{usr_j}}^{r_3}).$$

   Here we include e_{reg} to a transaction so that the regulator can retrieve the transaction amount and the balance of a user if necessary. We also include e_{usr_i} and e_{usr_j} in a transaction so both usr_i and usr_j and view the transaction amount and their latest balance on chain and generate the correct zero-knowledge proofs based on it.

2. usr_i needs to prove that vals encrypted in e_{reg}, e_{usr_i} and e_{usr_j} are the same and are using the correct public keys. Formally, usr_i generate a proof π_{eq} for the statement $(e_{reg}, e_{usr_i}, e_{usr_j}, g, pk_{reg}, pk_{usr_i}, pk_{usr_j}) \in L_{eq}$, where the language L_{eq} is defined as:

$$\begin{aligned} L_{eq} := \{&(e_{reg}, e_{usr_i}, e_{usr_j}, g, pk_{reg}, pk_{usr_i}, pk_{usr_j}) \mid \\ &\exists (r_1, r_2, r_3, val) \ \text{ s.t } \ e_{reg} = (g^{r_1}, g^{val} \cdot {pk_{reg}}^{r_1}) \wedge \\ &e_{usr_i} = (g^{r_2}, g^{val} \cdot {pk_{usr_i}}^{r_2}) \wedge e_{usr_j} = (g^{r_3}, g^{val} \cdot {pk_{usr_j}}^{r_3})\}. \end{aligned}$$

   Sigma protocol for L_{eq}. We can utilize sigma protocol to generate such proof:

   (a) Generate random $k_1, k_2, k_3, k_4 \overset{\$}{\leftarrow} [0, q-1]$ and calculate

$$\begin{aligned} e_1 &= g^{k_1},\ e_2 = g^{k_4} \cdot pk_{reg}^{k_1}, \\ e_3 &= g^{k_2},\ e_4 = g^{k_4} \cdot pk_{usr_i}^{k_2}, \\ e_5 &= g^{k_3},\ e_6 = g^{k_4} \cdot pk_{usr_j}^{k_3}. \end{aligned}$$

DOI:10.6814/NCCU202201524

(b) Calculate challenge c using hash function h as

$$c = h(g, pk_{reg}, pk_{usr_i}, pk_{usr_j}, e_1, e_2, e_3, e_4, e_5, e_6).$$

(c) Generate response as

$$t_1 = k_1 + cr_1, t_2 = k_2 + cr_2, t_3 = k_3 + cr_3, t_4 = k_4 + c \cdot val.$$

(d) Set $\pi_{eq} = (e_1, e_2, e_3, e_4, e_5, e_6, t_1, t_2, t_3, t_4)$.

Note that when encrypting using different public keys, it is safe to reuse the random number in Elgamal encryption [BBS02]. That is, we can compress e_{reg}, e_{usr_i}, e_{usr_j} as $(g^{r_1}, g^{val} \cdot {pk_{reg}}^{r_1}, g^{val} \cdot {pk_{usr_i}}^{r_1}, g^{val} \cdot {pk_{usr_j}}^{r_1})$. π_{eq} generated with sigma protocol can also be compressed as $(e_1, e_2, e_4, e_6, t_1, t_4)$ where $e_1 = g^{k_1}$, $e_2 = g^{k_4} \cdot pk_{reg}^{k_1}, e_4 = g^{k_4} \cdot pk_{usr_i}^{k_1}, e_2 = g^{k_4} \cdot pk_{usr_j}^{k_1}$ and $t_1 = k_1 + cr_1, t_4 = k_1 + c \cdot val$

3. usr_i needs to prove that the amount encrypted in e_{reg}, e_{usr_i}, e_{usr_j} are within the range $[0, m]$ where m is much smaller than the group size used in the modular arithmetic. Since we have proved vals across the three ciphertexts are the same, it is sufficient to proof val inside e_{reg} is within the range. usr_i generate a proof π_{range} for the statement $(g, pk_{reg}, m, e_{reg}) \in L_{range}$ which is defined as follows:

$$\begin{aligned} L_{range} = \{&(g, pk_{reg}, m, e_{reg}) \mid \\ &\exists r_1, val \text{ s.t. } e_{reg} = (g^{r_1}, g^{val} \cdot {pk_{reg}}^{r_1}) \wedge 0 \le val \le m\}. \end{aligned}$$

Range proof for L_{range}. Here we utilized Bulletproof [BBB+18] range proof protocol to generate a proof π_{range} for L_{range}. Bulletproof protocol can generate a zero-knowledge range proof for a committed value using Pedersen commitment. Since the second part of an Elgamal encryption ciphertext is in the same form of a Pedersen commitment, that is, $g^{val} \cdot {pk_{reg}}^{r_1}$ in e_{reg}, we can generate a range proof of val with it. Note that in Bulletproof, we require the discrete logarithm relation between g, pk_{reg} to be unknown to the prover for the soundness of the proof to hold. Since the secret key of the regulator, sk_{reg} where $g^{sk_{reg}} = pk_{reg}$, is thresholdized, only if all the regulators collude can they retrieve sk_{reg}. Hence

DOI:10.6814/NCCU202201524

it is safe to generate a range proof using e_{reg}. We refer the details of the proving procedure to Section 4.2 of [BBB+18].

4. usr_i needs to generate a zero-knowledge proof π_{bal} to prove his encrypted balances after the transaction is greater than or equal to 0. The encrypted balances can be calculated by the validators on the blockchain using additive Homomorphism of our Elgamal encryption scheme, where

$$\begin{aligned} renc\text{-}bal'_{usr_i} &= Add(renc\text{-}bal_{usr_i}, (e_{reg})^{-1}) \\ &= (g^{r_{i_r}}, g^{bal_{usr_i}} \cdot pk_{reg}{}^{r_{i_r}}) \times (g^{-r_1}, g^{-val} \cdot pk_{reg}{}^{-r_1}) \\ &= (g^{r_{i_r}-r_1}, g^{bal_{usr_i}-val} \cdot pk_{reg}{}^{r_{i_r}-r_1}). \end{aligned}$$

$$\begin{aligned} uenc\text{-}bal'_{usr_i} &= Add(uenc\text{-}bal_{usr_i}, (e_{usr_i})^{-1}) \\ &= (g^{r_{i_u}}, g^{bal_{usr_i}} \cdot pk_{usr_i}{}^{r_{i_u}}) \times (g^{-r_2}, g^{-val} \cdot pk_{usr_i}{}^{-r_2}) \\ &= (g^{r_{i_u}-r_2}, g^{bal_{usr_i}-val} \cdot pk_{usr_i}{}^{r_{i_u}-r_2}). \end{aligned}$$

Since we have proved that vals in e_{reg} and e_{usr_i} are the same, and all of the previous transactions also hold the same vals when encrypting with different public keys. It is sufficient to proof one of $renc\text{-}bal'_{usr_i}$ and $uenc\text{-}bal'_{usr_i}$ is within the range.

Intuitively, we can use the second part of $renc\text{-}bal'_{usr_i}$ to generate a range proof using Bulletproof like in π_{range}. However, we don't actually know the value of random number r_{i_r} in $renc\text{-}bal_{usr_i}$ since it is derived from previous transactions, and we need $r_{i_r} - r_1$ as a witness of the commitment when using Bulletproof.

To solve the problem, we first refresh the random number $r_{i_u} - r_2$ in $uenc\text{-}bal'_{usr_i}$ to a newly selected $r' \overset{\$}{\leftarrow} [0, q-1]$ and calculate $uenc\text{-}bal''_{usr_i}$ as $(g^{r'}, g^{bal_{usr_i}-val} \cdot pk_{usr_i}{}^{r'})$. Here we need to prove that bal_{usr_i} remains the same when refreshing the random number. We can achieve this with the knowledge of sk_{usr_i}. Formally, we generate a proof π_{bal_r} for the

DOI:10.6814/NCCU202201524

statement $(\mathit{uenc\text{-}bal}'_{usr_i}, \mathit{uenc\text{-}bal}''_{usr_i}, g, pk_{usr_i}) \in L_{bal_r}$, where the language L_{bal_r} describes that the two ciphertexts $\mathit{uenc\text{-}bal}'_{usr_i}, \mathit{uenc\text{-}bal}''_{usr_i}$ can be decrypted to the same value with the knowledge of sk_{usr_i}.

Sigma protocol for L_{bal_r}. Let X, Y denote $g^{r_{iu}-r_2}$ and $g^{bal_{usr_i}-val} \cdot {pk_{usr_i}}^{r_{iu}-r_2}$ in $\mathit{uenc\text{-}bal}'_{usr_i}$. X', Y' denote $g^{r'}$ and $g^{bal_{usr_i}-val} \cdot {pk_{usr_i}}^{r'}$ in $\mathit{uenc\text{-}bal}''_{usr_i}$. $X/X' = g^{r_{iu}-r_2-r'}$, $Y/Y' = {pk_{usr_i}}^{r_{iu}-r_2-r'}$. We can then use the knowledge of sk_{usr_i} to proof that $(X/X')^{sk_{usr_i}} = Y/Y'$. Since that only if the values encrypted inside $\mathit{uenc\text{-}bal}'_{usr_i}$ and $\mathit{uenc\text{-}bal}_{usr_i}$ are the same will the equation holds, the proof is sufficient for L_{bal_r}. The procedure of the proof are as follows:

(a) Generate random $k_5 \overset{\$}{\leftarrow} [0, q-1]$ and calculate $e_7 = (X/X')^{k_5}$.

(b) Calculate challenge c using hash function h as

$$c = h(X/X', Y/Y', e_7).$$

(c) Generate response as

$$t_5 = k_5 + c \cdot sk_{usr_i}.$$

(d) Set $\pi_{bal_r} = (e_7, t_5)$.

5. Although now we have $\mathit{uenc\text{-}bal}''_{usr_i}$ which the random number in it is known, we still cannot directly generate a range proof with it since we know the discrete logarithm relation between g and pk_{usr_i}. To solve the problem, we can generate a new Pedersen commitment with the same value and random number as in $\mathit{uenc\text{-}bal}''_{usr_i}$ using g and pk_{reg}. We set $pc_{bal} = g^{bal_{usr_i}-val} \cdot {pk_{reg}}^{r'}$. Since we do not know the logarithm relation between g and pk_{reg}, we are able to generate a range proof from it. We again use sigma protocol to prove that the value we commit and the random number we use are the same as $\mathit{uenc\text{-}bal}''_{usr_i}$. Let $\mathit{uenc\text{-}bal}''_{usr_i 2}$ denote the second part of $\mathit{uenc\text{-}bal}''_{usr_i}$. Formally, we generate a proof π_{bal_k} for the statement $(\mathit{uenc\text{-}bal}''_{usr_i 2}, pc_{bal}, g, pk_{usr_i}, pk_{reg}) \in L_{bal_k}$, where the language L_{bal_k} is defined as:

DOI:10.6814/NCCU202201524

$$\begin{aligned} L_{bal_k} := \{ & (uenc\text{-}bal''_{usr_i 2}, pc_{bal}, g, pk_{usr_i}, pk_{reg}) \mid \\ & \exists (bal_{usr_i} - val, r') \quad \text{s.t.} \\ & uenc\text{-}bal''_{usr_i 2} = (g^{bal_{usr_i} - val} \cdot {pk_{usr_i}}^{r'}) \wedge pc_{bal} = g^{bal_{usr_i} - val} \cdot {pk_{reg}}^{r'} \}. \end{aligned}$$

Sigma protocol for L_{bal_k}. The processes for sigma protocol are as follows:

(a) Generate random $k_6, k_7 \overset{\$}{\leftarrow} [0, q-1]$ and calculate

$$e_8 = g^{k_6} \cdot {pk_{usr_i}}^{k_7}, e_9 = g^{k_6} \cdot {pk_{reg}}^{k_7}.$$

(b) Calculate challenge c using hash function h as

$$c = h(e_8, e_9, g, pk_{usr_i}, pk_{reg}).$$

(c) Generate response as

$$t_6 = k_6 + (bal_{usr_i} - val) \cdot c, t_7 = k_7 + r' \cdot c.$$

(d) Set $\pi_{bal_k} = (e_8, e_9, t_6, t_7)$.

6. Finally, with proofs π_{bal_r} and π_{bal_k}, we can generate a range proof π_{bal} of $bal_{usr_i} - val$ with commitment $pc_{bal} = g^{bal_{usr_i} - val} \cdot {pk_{reg}}^{r'}$ using Bulletproof protocol.

7. usr_i generates a signature σ for the transaction tx where

$$\begin{aligned} tx = & (pk_{usr_i}, pk_{usr_j}, e_{reg}, e_{usr_i}, e_{usr_j}, \\ & \pi_{eq}, \pi_{range}, uenc\text{-}bal''_{usr_i}, pc_{bal}, \pi_{bal_r}, \pi_{bal_k}, \pi_{bal}, cert_{usr_i}). \end{aligned}$$

and broadcast the signature σ and tx to the blockchain.

Validate. After the validators receive a transaction, they use the algorithm to validate the transaction. Although the transaction amount is encrypted, we can ensure the correctness of the transaction by verifying the zero-knowledge proofs. We can also update the balances of both sender and receiver by using the additive homomorphic ElGamal encryption. The processes for a validator V to validate a transaction tx are as follows:

DOI:10.6814/NCCU202201524

1. V first validates the signature σ of the transaction with sender's public key pk_{usr_i}. If failed, abort the algorithm.

2. V validates the certificate of the sender using the identity authority's public key. If failed, abort the algorithm.

3. V calculates the balance of sender and receiver, $renc\text{-}bal'_{usr_i}$, $uenc\text{-}bal'_{usr_i}$ and $renc\text{-}bal'_{usr_j}$, $uenc\text{-}bal'_{usr_j}$ as follows.

$$\begin{aligned} renc\text{-}bal'_{usr_i} &= Add(renc\text{-}bal_{usr_i}, (e_{reg})^{-1}) \\ &= (g^{r_{ir}}, g^{bal_{usr_i}} \cdot {pk_{reg}}^{r_{ir}}) \times (g^{-r_1}, g^{-val} \cdot {pk_{reg}}^{-r_1}) \\ &= (g^{r_{ir}-r_1}, g^{bal_{usr_i}-val} \cdot {pk_{reg}}^{r_{ir}-r_1}). \end{aligned}$$

$$\begin{aligned} uenc\text{-}bal'_{usr_i} &= Add(uenc\text{-}bal_{usr_i}, (e_{usr_i})^{-1}) \\ &= (g^{r_{iu}}, g^{bal_{usr_i}} \cdot {pk_{usr_i}}^{r_{iu}}) \times (g^{-r_2}, g^{-val} \cdot {pk_{usr_i}}^{-r_2}) \\ &= (g^{r_{iu}-r_2}, g^{bal_{usr_i}-val} \cdot {pk_{usr_i}}^{r_{iu}-r_2}). \end{aligned}$$

$$\begin{aligned} renc\text{-}bal'_{usr_j} &= Add(renc\text{-}bal_{usr_j}, e_{reg}) \\ &= (g^{r_{jr}}, g^{bal_{usr_j}} \cdot {pk_{reg}}^{r_{jr}}) \times (g^{r_1}, g^{val} \cdot {pk_{reg}}^{r_1}) \\ &= (g^{r_{jr}+r_1}, g^{bal_{usr_j}+val} \cdot {pk_{reg}}^{r_{jr}+r_1}). \end{aligned}$$

$$\begin{aligned} uenc\text{-}bal'_{usr_j} &= Add(uenc\text{-}bal_{usr_j}, e_{usr_j}) \\ &= (g^{r_{ju}}, g^{bal_{usr_j}} \cdot {pk_{usr_j}}^{r_{ju}}) \times (g^{r_1}, g^{val} \cdot {pk_{reg}}^{r_1}) \\ &= (g^{r_{ju}+r_1}, g^{bal_{usr_j}+val} \cdot {pk_{usr_j}}^{r_{ju}+r_1}). \end{aligned}$$

4. V validates the zero-knowledge proofs in the transaction. This process ensures the correctness of the transaction. If failed, abort the algorithm.

   Validate π_{eq}. The processes of the validation for $\pi_{eq} = (e_1, e_2, e_3, e_4, e_5, e_6, t_1, t_2, t_3, t_4)$ are as follows.

(a) Calculate challenge c using hash function h as

$$c = h(g, pk_{reg}, pk_{usr_i}, pk_{usr_j}, e_1, e_2, e_3, e_4, e_5, e_6).$$

(b) Check if

$$\begin{aligned} g^{t_1} &\overset{?}{=} e_1 \cdot (g^{r_1})^c, \\ g^{t_2} &\overset{?}{=} e_3 \cdot (g^{r_2})^c, \\ g^{t_3} &\overset{?}{=} e_5 \cdot (g^{r_3})^c, \\ g^{t_4} \cdot (pk_{usr_i})^{t_1} &\overset{?}{=} e_2 \cdot \left(g^{val} \cdot (pk_{usr_i})^{r_1}\right)^c, \\ g^{t_4} \cdot (pk_{usr_j})^{t_2} &\overset{?}{=} e_4 \cdot \left(g^{val} \cdot (pk_{usr_j})^{r_2}\right)^c, \\ g^{t_4} \cdot (pk_{reg})^{t_3} &\overset{?}{=} e_6 \cdot \left(g^{val} \cdot (pk_{reg})^{r_3}\right)^c. \end{aligned}$$

Note that validators can get (g^{r_1}),(g^{r_2}),(g^{r_3}),$(g^{val} \cdot (pk_{usr_i})^{r_1})$,$(g^{val} \cdot pk_{usr_j})^{r_2})$,$(g^{val} \cdot (pk_{reg})^{r_3})$ in e_{reg},e_{usr_i} and e_{usr_j}.

Validate π_{range}. Validators check the range proof following the Bulletproof protocol. We again refer the details to Section 4.2 of [BBB+18].

Validate π_{bal_r}. The processes of the validation for $\pi_{bal_r} = (e_7, t_5)$ are as follows.

(a) Use $uenc\text{-}bal'_{usr_i} = (g^{r_{iu}-r_2}, g^{bal_{usr_i}-val} \cdot pk_{usr_i}{}^{r_{iu}-r_2})$ and $uenc\text{-}bal''_{usr_i} = (g^{r'}, g^{bal_{usr_i}-val} \cdot pk_{usr_i}{}^{r'})$ to calculate $X = g^{r_{iu}-r_2}/g^{r'} = g^{r_{iu}-r_2-r'}$ and $Y = (g^{bal_{usr_i}-val} \cdot pk_{usr_i}{}^{r_{iu}-r_2})/(g^{bal_{usr_i}-val} \cdot pk_{usr_i}{}^{r'}) = pk_{usr_i}{}^{r_{iu}-r_2-r'}$.

(b) Calculate challenge c using hash function h as $c = h(X, Y, e_7)$.

(c) Check if $X^{t_5} \overset{?}{=} e_7 \cdot Y^c$.

Validate π_{bal_k}. The processes of the validation for $\pi_{bal_k} = (e_8, e_9, t_6, t_7)$ using $uenc\text{-}bal''_{usr_i} = (g^{r'}, g^{bal_{usr_i}-val} \cdot pk_{usr_i}{}^{r'})$ and $pc_{bal} = g^{bal_{usr_i}-val} \cdot pk_{reg}{}^{r'}$ are as follows:

(a) Calculate challenge c using hash function h as $c = h(e_8, e_9, g, pk_{usr_i}, pk_{reg})$.

DOI:10.6814/NCCU202201524

(b) Check if

$$g^{t_6} \cdot (pk_{usr_i})^{t_7} \overset{?}{=} e_8 \cdot (g^{bal_{usr_i} - val} \cdot pk_{usr_i}{}^{r'})^c,$$
$$g^{t_6} \cdot (pk_{reg})^{t_7} \overset{?}{=} e_9 \cdot (pc_{bal})^c.$$

Validate π_{bal}. Validators check the range proof π_{bal} following the Bulletproof protocol.

The process of updating the blockchain is based on the underlying blockchain and can be changed depending on different setups. Note that the validators do not have the power to decrypt the transaction amount and the balances of the user. Thus they only check the correctness of a transaction but not the legitimacy.

Reveal. The regulators use this algorithm to decrypt the amount of a transaction or the balance of a certain user. Any of the regulators can initiate the process of decrypting a transaction. The regulator who initiates it acts as the master of the process and sends a decrypting request to other regulators. Once other regulators receive the request, they may decide whether to decrypt the transaction/balance or not. For a regulator reg_{mas} who initiates the algorithm, the process is as follows:

1. reg_{mas} sends a request $request_{dec}$ for decryption and a transaction id tx_{id} to all regulators.

2. For a regulator reg_i who received the request, he or she retrieves the encrypted amount of the transaction $e_{reg_{tx}} = (g^{r_{tx}}, g^{val_{tx}} \cdot pk_{reg}{}^{r_{tx}})$ from the blockchain with tx_{id}.

3. If reg_i decides to participate in decrypting the transaction, he or she calculates a share of the transaction $txShare_i = g^{r_{tx} \cdot sk_{reg_i}}$, where sk_{reg_i} is the private key share of reg_i. Then, he or she broadcasts $(txShare_i, i)$ to other regulators via a secure channel. Otherwise, reg_i broadcasts $\perp$ via a secure channel.

4. After a regulator receives more than t shares from other regulators, he or she randomly selects t shares from these shares to form a group S. Without loss

DOI:10.6814/NCCU202201524

of generality, let k be one of the share indexes randomly chosen from S. The regulator calculates

$$a_n = txShare_k^{\prod_{\substack{j \neq k \\ j \in S}} \frac{j}{j-k}} = g^{r_{tx} \cdot sk_{reg_k} \cdot \prod_{\substack{j \neq k \\ j \in S}} \frac{j}{j-k}}.$$

5. The regulator calculates $\prod_{n=1}^{t} a_n = g^{{pk_{reg}}^{r_{tx}}}$ and can thus retrieve $g^{val_{tx}}$ by using the second part of $e_{reg_{tx}}$. That is, $g^{val_{tx}} = (g^{val_{tx}} \cdot {pk_{reg}}^{r_{tx}}) \cdot (g^{{pk_{reg}}^{r_{tx}}})^{-1}$. Finally the regulator calculates the discrete logarithm of $g^{val_{tx}}$ to get the amount val_{tx}.

The regulators can decrypt the balance of a user by a similar process. Instead of a transaction id, reg_{mas} sends the public key of a user to other regulators.

Deposit. The deposit algorithm are called by the identity authority after a user make a deposit to it. After a user deposit some assets, the identity authority generate a special transaction to adjust the user's balance. The process of deposit are as follows:

1. The identity authority encrypts the amount val corresponding to the user's deposit with the regulator's public key pk_{reg}, and the user's public key pk_{usr_i}. Identity authority randomly chooses $r_1, r_2 \overset{\$}{\leftarrow} [0, q-1]$ and calculates

$$e_{reg} = (g^{r_1}, g^{val} \cdot {pk_{reg}}^{r_1});\ e_{usr_i} = (g^{r_2}, g^{val} \cdot {pk_{usr_i}}^{r_2}).$$

2. As in *Send* algorithm, the identity authority generate π_{eq} to prove that the values encrypted in e_{reg} and e_{usr_i} are the same and are using the correct public keys.

3. Finally, the identity authority generate a special transaction tx_{dep} and signed the transaction with it's secret key where

$$tx_{dep} = (pk_{id}, pk_{usr_i}, e_{reg}, e_{usr_i}, \pi_{eq}).$$

and broadcast the signature σ and tx_{dep} to the blockchain.

DOI:10.6814/NCCU202201524

Note that since the identity authority can create coins on chain based on a user's deposit, we don't need to check the balance of the identity authority.

Withdraw. A user calls the withdraw algorithm if he wants to make a withdrawal from the system. The withdraw transaction are the same as a normal transaction except the receiver are set to the identity authority. After the withdrawal transaction are validated and appended to the blockchain, a user can withdraw his assets from the identity authority.

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

5 Security Analysis

In this chapter, we provide four security theorems to show that the proposed system satisfies $Authenticity$, $Confidentiality$, $Balance$, and $ConsistentAmount$.

Theorem 5.1. *The proposed system satisfies $Authenticity$ if the underlying signature scheme is existentially unforgeable under the chosen message attacks.*

Since we include a signature in every transaction, $\mathcal{A}$ breaks the unforgeability of the underlying signature scheme if $\mathcal{A}$ successfully forge a signature that can pass the validation.

Theorem 5.2. *The proposed system satisfies $Confidentiality$ if the underlying threshold cryptosystem and the public-key encryption scheme are indistinguishable under the chosen plaintext.*

Let q be the total number of queries $\mathcal{A}$ sends to C, and $\mathcal{A}$'s advantage in IND-CPA experiments of the underlying public-key encryption scheme is ϵ. The advantage of $\mathcal{A}$ in Confidentiality game is at most $4 \cdot q \cdot \epsilon$.

Theorem 5.3. *The proposed system satisfies $Balance$ if the underlying encryption scheme is additive-homomorphic and the zero-knowledge proof scheme is sound.*

Our scheme relies on homomorphic encryption to correctly update the balance. Providing that the zero-knowledge proof scheme is sound, we can be sure if π_{range} is valid, the transaction amount is within the range that will not break the modular arithmetic used in homomorphic encryption.

Theorem 5.4. *The proposed system satisfies $ConsistentAmount$ if the underlying zero-knowledge proof scheme is sound.*

If the zero-knowledge proof scheme is sound, once we verify π_{eq}, we can confirm that the amount viewed by sender, receiver and, verifiers are uniform, thus achieving $ConsistentAmount$.

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

6 Performance Evaluation

For the efficiency analysis, we implement a proof-of-concept system by using Rust language. The experiment was conducted on a MacBook Pro 2017 laptop (macOS Monterey version 12.3) with Intel(R) i5-7360U CPU clocked at 2.5GHz and 8GB of system memory.

In our implementation, we leverage BulletProof [BBB⁺18] as the zero-knowledge range proof protocol and sigma protocol with Fiat-Shamir [CG15; Kra03] transform as the zero-knowledge proof protocol. In addition, the elliptic curve we adopt is secp256k1 which has 256-bit security and is broadly used in cryptocurrencies like Bitcoin and Ethereum. Each point on the curve can be stored as 88 bytes, and the scalar of the finite field can be stored as 56 bytes.

Table 6.1: The computation cost of confidential transaction

	Size (bytes)	Generation time (ms)	Verify time (ms)
Confidential *tx*	2288	94	64

Here we set the max value of the transaction as $2^{\ell} - 1$ where $\ell = 64$. The computation cost of generating and verifying a transaction, as shown in Table 6.1, is efficient to compute. The transaction size is also feasible for the use of blockchain.

Table 6.2: The computation cost of decrypting threshold ElGamal

	Calculate share time (ms)	Combine share time (s)
Threshold ElGamal	0.2	224.8

Here we use threshold ElGamal encryption with 5 members and the threshold is set to 3. We encrypt a value of $10,000,000$ and use a simple brute force method in

DOI:10.6814/NCCU202201524

decryption. As shown in Table 6.2, the regulators can easily decrypt the transaction even if they need to compute the discrete logarithm problem when combining decryption shares. The decryption time is feasible for our use case since the regulators do not have to run the decryption algorithm with every transaction. In addition, the decryption time can be further reduced if run in parallel.

DOI:10.6814/NCCU202201524

7 Conclusion and Future Work

In this thesis, we propose a confidential payment system with controllable regulation based on blockchain. The regulation procedure includes a voting process in order to limit the power of the regulator. We hope the proposed system can strike a balance between privacy and regulation.

Below we describe possible improvements for future work. In modern financial systems, a transaction is often required to pass certain anti-money laundering checks before it can be carried out. Such feature can be done by adding other zero-knowledge proofs to check against the rules in a transaction. How to do so in our system without increase too many overheads is an open problem.

Another aspect that needs to consider is how to improve efficiency. Since the proposed system requires users to verify zero-knowledge proofs over the blockchain, it may cause additional computational costs. A possible solution is to adopt a curve that is more suitable for this usage, and how to find this curve is also an interesting problem.

DOI:10.6814/NCCU202201524

DOI:10.6814/NCCU202201524

Bibliography

[AFM+14] Giuseppe Ateniese, Antonio Faonio, Bernardo Magri, and Breno de Medeiros. “Certified Bitcoins.” In: *ACNS 2014*. Vol. 8479. LNCS. Springer. 2014, pp. 80–96 (cit. p. 24).

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, *et al.* “Bulletproofs: Short Proofs for Confidential Transactions and More.” In: *IEEE S&P 2018*. IEEE. 2018, pp. 315–334 (cit. pp. 27, 28, 32, 39).

[BBS02] Mihir Bellare, Alexandra Boldyreva, and Jessica Staddon. “Randomness Re-use in Multi-recipient Encryption Schemeas.” In: *Public Key Cryptography — PKC 2003*. Springer. 2002, pp. 85–99 (cit. p. 27).

[BCG+14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, *et al.* “Zerocash: Decentralized anonymous payments from Bitcoin.” In: *IEEE S&P 2014*. IEEE. 2014, pp. 459–474 (cit. pp. 1, 4).

[But14] Vitalik Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. `https://nft2x.com/wp-content/uploads/2021/03/EthereumWP.pdf`. 2014 (cit. p. 1).

[CG15] Pyrros Chaidos and Jens Groth. “Making Sigma-protocols non-interactive without random oracles.” In: *PKC 2015*. Vol. 9020. LNCS. Springer. 2015, pp. 650–670 (cit. pp. 13, 39).

[CMT+20] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Au. “PGC: Decentralized confidential payment system with auditability.” In: *ESORICS 2020*. Vol. 12308. LNCS. Springer, 2020, pp. 591–610 (cit. p. 5).

[CSL+18] Mauro Conti, E. Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. “A survey on security and privacy issues of Bitcoin.” In: *IEEE Commun. Surv. Tutorials*. Vol. 20. IEEE. 2018, pp. 3416–3452 (cit. p. 1).

[CZJ+17] Ethan Cecchetti, Fan Zhang, Yan Ji, *et al.* “Solidus: Confidential distributed ledger transactions via PVORM.” In: *CCS 2017*. ACM. 2017, pp. 701–717 (cit. p. 4).

[DF89] Yvo Desmedt and Yair Frankel. “Threshold cryptosystems.” In: *CRYPTO 1989*. Vol. 435. LNCS. Springer. 1989, pp. 307–315 (cit. p. 17).

DOI:10.6814/NCCU202201524

[FS86] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems." In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194 (cit. p. 13).

[GGM16] Christina Garman, Matthew Green, and Ian Miers. "Accountable privacy for decentralized anonymous payments." In: *FC 2016*. Vol. 9603. LNCS. Springer. 2016, pp. 81–98 (cit. p. 4).

[IM13] C. Garman I. Miers and A. D. Rubin M. Green. "Zerocoin: Anonymous distributed E-cash from Bitcoin." In: *IEEE S&P 2013*. IEEE. 2013, pp. 397–411 (cit. p. 4).

[Kra03] Hugo Krawczyk. "SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE protocols." In: *CRYPTO 2003*. Vol. 2729. LNCS. Springer. 2003, pp. 400–425 (cit. pp. 13, 39).

[LHH$^+$20] Chao Lin, Debiao He, Xinyi Huang, Muhammad Khurram Khan, and Kim-Kwang Raymond Choo. "DCAP: A secure and efficient decentralized conditional anonymous payment system based on blockchain." In: *IEEE Trans. Inf. Forensics Secur.* Vol. 15. 2020, pp. 2440–2452 (cit. p. 4).

[MvF15] Killian J. McCarthy, Peter van Santen, and Ingo Fiedler. "Modeling the money launderer: Microtheoretical arguments on anti-money laundering policy." In: *International Review of Law and Economics*. Vol. 43. Elsevier. 2015, pp. 148–155 (cit. p. 2).

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. `https://bitcoin.org/bitcoin.pdf`. 2008 (cit. p. 1).

[Ped91] Torben Pryds Pedersen. "A threshold cryptosystem without a trusted party." In: *EUROCRYPT*. Vol. 547. LNCS. Springer. 1991, pp. 522–526 (cit. p. 10).

[Ped92] Torben Pryds Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing." In: *Advances in Cryptology — CRYPTO '91*. Springer. 1992, pp. 129–140 (cit. p. 8).

[Pet19] Maksym Petkus. *Why and how zk-SNARK works*. `http://arxiv.org/abs/1906.07221`. 2019 (cit. p. 4).

[PW11] Thomas Pietschmann and John Walker. *Estimating Illicit Financial Flows Resulting From Drug Trafficking and other Transnational Organized Crimes*. `https://www.unodc.org/documents/data-and-analysis/Studies/Illicit_financial_flows_2011_web.pdf`. United Nations Office on Drugs and Crime, 2011 (cit. p. 2).

[Sab13] N. Van Saberhagen. *Cryptonote v 2.0*. `https://bytecoin.org/old/whitepaper.pdf`. 2013 (cit. p. 4).

DOI:10.6814/NCCU202201524

[SAL+17] Shi-Feng Sun, Man Ho Au, Joseph K. Liu, and Tsz Hon Yuen. “RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency Monero.” In: *ESORICS 2017*. Vol. 10493. LNCS. Springer. 2017, pp. 456–474 (cit. p. 1).

[SHL+21] Xuemin Sherman Shen, Cheng Huang, Dongxiao Liu, *et al.* “Data management for future wireless networks: Architecture, privacy preservation, and regulation.” In: *IEEE Netw.* Vol. 35. IEEE. 2021, pp. 8–15 (cit. p. 1).

[Wil18] Zachary J. Williamson. *The AZTEC protocol*. `https://raw.githubusercontent.com/AztecProtocol/AZTEC/master/AZTEC.pdf`. 2018 (cit. p. 4).

[WOD18] Rolf van Wegberg, Jan-Jaap Oerlemans, and Oskar van Deventer. “Bitcoin money laundering: Mixed results? An explorative study on money laundering of cybercrime proceeds using Bitcoin.” In: *J. Financial Crime*. Emerald Publishing Limited. 2018, pp. 419–435 (cit. p. 2).

[XLN+22] Liang Xue, Dongxiao Liu, Jianbing Ni, Xiaodong Lin, and Xuemin Sherman Shen. “Enabling regulatory compliance and enforcement in decentralized anonymous payment.” In: *IEEE Trans. Dependable Secur. Comput.* IEEE. 2022 (cit. p. 5).

DOI:10.6814/NCCU202201524