

Article

RZcoin: Ethereum-Based Decentralized Payment with Optional Privacy Service

Hong Zhao ^{1,2}, Xue Bai ¹, Shihui Zheng ¹ and Licheng Wang ^{1,*}

¹ State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; marcus.zhao@tron.network (H.Z.); bupt_bx_xxaq2013@bupt.edu.cn (X.B.); shihuizh@bupt.edu.cn (S.Z.)

² Technical Department, Golden Siv Technology Limited, Level 7, K11 ATELIER Victoria Dockside, 18 Salisbury Rd, Tsim Sha Tsui, Hong Kong, China

* Correspondence: wanglc@bupt.edu.cn

Received: 6 May 2020; Accepted: 24 June 2020; Published: 27 June 2020



Abstract: As the blockchain 2.0 platform, Ethereum's turing complete programming language and smart contract components make it play an important role in the commercialization of blockchain. With the further development of blockchain applications, the privacy and security issues of Ethereum have gradually emerged. To solve this problem, we proposed a blockchain privacy protection model called RZcash in the previous work. It implements the dynamically updateable and verifiable hiding of the asset information in Ethereum, namely the account balance and transaction amount. However, RZcash does not pay attention to the key redundancy problem that may be caused by the creation of secret accounts. In addition, the large size of proofs gives it high communication costs. In response to these problems, we further improve RZcash. For the key redundancy problem, we construct a new signature scheme based on the ciphertext equivalent test commitment. Moreover, we use the Schnorr signature and bulletproof to improve the corresponding proof scheme in RZcash, thereby reducing the size of proof. Based on these improvements, we propose a decentralized payment system, called RZcoin, based on Ethereum. Finally, we implement the algorithm model of RZcoin and evaluate its security and performance. The results show that RZcoin has higher security and Lower communication cost than RZcash.

Keywords: blockchain; privacy protection; zero-knowledge proof; publicly verifiable

1. Introduction

1.1. Background

With the rise of Bitcoin [1], the cryptocurrency system gradually attracted widespread attention worldwide. Since then, systems such as Ethereum [2] have begun to emerge. In June 2019, Facebook took the lead in publishing a white paper on the global cryptocurrency project called Libra [3]. According to the white paper, Libra will be the simple, borderless currency and financial infrastructure that serves billions of people around the world. The emergence of Libra has led governments to recognize the need for blockchain research and applications and start developing their own legal digital currency. However, at the recent hearing on Libra, the privacy of cryptocurrency system was questioned. The privacy disclosure has become a key issue that must be resolved in the process of legalizing the cryptocurrency.

Throughout the history of cryptocurrency, privacy has always been the design goal of these systems. From the beginning, privacy protection has been the focus of researchers. In 1982, David Chaum, known as the father of the digital currency, proposed Ecash [4]. Chaum used

self-designed blind signature technology to implement an anonymous communication mechanism. Ecash became the first anonymous digital currency system to achieve untrackable, protecting users' privacy while ensuring currency circulation. However, because it relies on the assistance of central agencies (i.e., banks) to solve problems such as double-spending, it was not widely used in the end. B-money is the first truly decentralized cryptocurrency in the world, and was proposed by Dai in 1998 [5]. It is the first to introduce the concept of PoW (Proof of Work) proposed by Adam in [6] into the issuance of digital currency. The decentralized structure helps these systems solve privacy disclosure issues caused by centralization. However, in order to achieve distributed consensus, Dai proposed that the transaction must be publicly broadcast through the whole network. It weakens the privacy of cryptocurrency system to some extent. In 2004, with the advent of RPoW (Reusable Proofs of Work) [7], the development of the cryptocurrency turned a corner. Based on B-money, Nakamoto introduces technologies such as timestamp [8–11], Merkle tree [12] and the longest chain mechanism to solve the double-spending problem. The privacy setting of cryptocurrency system that subsequently appeared (e.g., Ethereum) is basically based on Ecash and B-money. The core technology of these systems is blockchain. This concept originates from "Bitcoin: A Peer-to-Peer Cash Payment System" published by Nakamoto in the cryptography mailing list of cryptography Info Page at the end of 2008. It is essentially a distributed ledger that shares and stores data in a decentralized manner. The birth of blockchain benefits from the privacy needs of digital currencies. Its technical structure gradually formed and improved in the development process of these currency systems. However, these blockchain-based systems only cut off the connection between the identity of users and the transaction script. Sensitive information such as the asset value is still exposed to the network. In the anonymous mechanism, the transaction address is hidden by a hash function. Users can generate multiple addresses to participate in the transaction. However, due to the uniqueness of the hash value and the accuracy of the behavior analysis of users, it is possible to reconstruct the connection between the identity and the account address by analyzing a large number of transactions. Moreover, since the transaction amount is public throughout the network, malicious adversaries may concentrate on attacking transactions with high value. This will make the transaction script easily destroyed, resulting in the disclosure of important data such as business information.

The above problems not only affect the legalization of digital currency, but also hinder the application of the blockchain in the industrial Internet and other fields. China's legal digital currency will be issued soon. The government has also begun to actively develop the blockchain technology. Therefore, it is urgent to build a blockchain system with complete privacy protection services. In response to this demand, we proposed a blockchain privacy protection solution called RZcash in 2019. It realizes the hiding of account balance and transaction amount in Ethereum by integrating the existing cryptographic technology [13]. Specifically, in RZcash, we use Pedersen commitment to achieve dynamic hiding of asset information, use the rangeproof scheme based on the two-party ring signature to provide evidence for the legality of asset commitments, and use the zero-knowledge proof based on ECDSA to ensure that secret accounts are created correctly. We proved that RZcash has good performance while ensuring security and privacy. However, due to the large size of rangeproof and zero-knowledge proof, its communication cost is higher. In addition, RZcash does not pay attention to the key redundancy problem that may be caused by the creation of secret accounts. These problems may hinder the practical application of RZcash, so it needs to be further improved.

1.2. Motivation and Contribution

In this regard, we proposed a series of improvements in this paper. Based on RZcash, we further improve its core algorithms and propose a decentralized payment scheme called RZcoin that provides the optional privacy protection service. When users need the service, they must create a new secret account through the creation transaction. The account can be traded with other secret accounts or with open accounts (i.e., the external account of Ethereum). The former is called the secret transaction, and the latter is called the semi-secret transaction. In general, our main contributions are as follows:

- We further improve the privacy protection mechanism called RZcash. For the key redundancy problem, we propose a signature scheme called CEs based on the ciphertext equivalent test scheme [14]. It achieves the creation of secret accounts without regenerating the private key. In addition, compared to ECDSA, its signature size is reduced by 80%. In addition, we use Schnorr signature to reconstruct the zero-knowledge proof of RZcash and reduce the size of proof by 1/3. We also use bulletproof to reduce the size of rangeproofs.
- We combine the improved privacy protection mechanism with Ethereum and propose a decentralized payment called RZcoin with optional privacy services. In RZcoin, the hidden asset can be publicly verifiable and dynamically updated. Users can neither overspend nor forge their assets. All proofs required for transactions involving the privacy service (i.e., secret and semi-secret transactions) are attached to the "extraData" module of transactions.
- We implement the above decentralized payment scheme based on the Ethereum test chain and evaluate its performance. At the same time, we also propose the adversarial model it faces and give the corresponding security analysis. The result shows that the scheme can provide the privacy service to users with good performance while ensuring security.

1.3. Related Work

By summarizing the existing research results, we observe that the privacy service structure of the blockchain system is mainly divided into two modes: off-chain and on-chain.

In the off-chain model, the private data is stored and calculated in the off-chain environment, thereby avoiding the privacy disclosure caused by the on-chain storage. For example, the Keep project initiated by Luongo et al. expands the basic application of smart contracts through the form of private data containers [15]. In addition, it establishes an access channel between Ethereum smart contracts and the private data through secure multiparty computing (SMPC). In 2017, Microsoft proposed an Ethereum-based blockchain framework called Confidential Consortium (CCF) [16]. It uses local trusted execution environments (TEEs, such as Intel SGX) to provide the secure off-chain operating and storage space for the code and data of the blockchain system. Although these two projects are still in the theoretical stage, they provide a unique idea for the privacy protection research of the blockchain.

Most of the existing research results achieved privacy protection in an on-chain mode, such as Dash [17], Monero [18], Zerocash [19] and so on. This type of system mainly implements real-time hiding of transaction information by using cryptographic technologies such as commitment. The key technology on which they are based is divided into two main categories. The one is ring signature [20]. Its typical use case is Monero. It hides the transaction amount through confidential transaction and uses a new type of ring signature called MLSAG to hide participants in the transaction [21–23]. The ring signature cannot only hide the source of the transaction, but also prove the correctness of hidden amount. The other is zero-knowledge proof [24]. In 2015, BenSasson et al. further developed the zk-SNARKs technology on the basis of zero-knowledge proof, which enables people to succinctly and non-interactively prove that they know certain information without revealing specific knowledge. Zcash is created based on zk-SNARKs. Subsequently, zero-knowledge proof has gradually become the mainstream technology for privacy protection. Mimblewimble [25], PLONK [26], Halo [27], Sonic [28] and other blockchain privacy protection projects were proposed. These projects are to improve the existing zk-SNARKs, in order to reduce the number of trusted settings needed for its operation, so as to improve the efficiency and universality. Although zero-knowledge proof is inefficient and difficult to implement, it is still considered to be the most effective solution to privacy issues of blockchain.

Compared to Bitcoin, the privacy requirement of Ethereum is more complex. In addition to hiding the transaction amount and participants, Ethereum also needs to hide the account balance and transaction details of the smart contract. In the past two years, Mobius [29], Zether [30], AZTEC [31], PGC [32] and other privacy protection schemes for Ethereum were proposed. However, because most of their privacy services are provided based on smart contracts, the expensive consumption of GAS prevents it from being put into practical use. In this regard, the privacy protection mechanism we

proposed refers to the idea of off-chain computing. It encapsulates the encryption and verification procedures involving secret transactions in TEEs, and performs the additional calculations required for secret transactions in the off-chain environment. The privacy protection mechanism we proposed has good portability and can be combined with existing zero-knowledge virtual machines to delegate some computing work to the off-chain semi-trusted server.

1.4. Roadmap

The rest of this paper proceeds as follows. In Section 2, we introduce some basic knowledge used in this paper. In Section 3, we present our main originality, i.e., the ciphertext-equivalent signature (CEs) for RZcoin. In Section 4, we present the system model and the adversarial model of RZcoin. In Section 5, we introduce the interactive details of the involved roles of RZcoin. In Section 6, we give the security analysis of RZcoin according to the system model given in Section 4. In Section 7, we implement the core model of RZcoin and provide the performance analysis, especially the improvements to the corresponding proof sizes. Finally, we summarize the full text and point out the future research direction in Section 8.

2. Preliminary

2.1. Ethereum System

Ethereum is known as the blockchain 2.0 platform. It is widely used because of its turing-complete scripting language, account-based transaction model and practical smart contract components. According to the description in its white paper, the architecture of Ethereum is mainly divided into seven layers: application, contract, consensus, protocol, network, data and storage, which are arranged in order from top to bottom as shown in Figure 1. Among them, the data layer mainly deals with various types of data in Ethereum. It is the gathering place of cryptographic technology and the focus of our research. Our privacy protection mechanism is mainly applied to this layer.

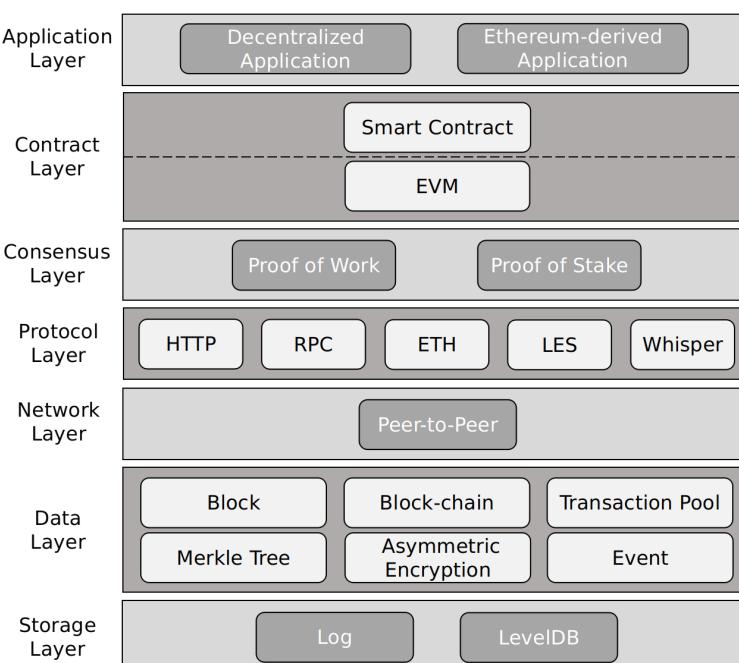


Figure 1. Ethereum Architecture.

We can see that the main body of the data layer is the transaction. It is stored in the transaction pool and packaged into blocks by the full node after passing the legality verification. Then it is recorded in the blockchain after multi-party consensus. As shown in Table 1, the transaction format

of blockchain system is basically the same, which mainly includes information such as the address of transaction participants, the transaction amount, the hash value and the signature of transaction. In addition, the transaction in Ethereum also includes the payload module. This module is mainly used to store other information required for transactions. In RZcoin, we use it to carry the extra data needed for secret transactions.

Table 1. Transaction Format of the Blockchain System.

| Transaction | |
|--------------|--|
| Hash | Hash value of the transaction |
| AccountNonce | Total number of transactions initiated by sender |
| GasInfo | Information about GAS of the transaction |
| Sender | Address of the transaction sender |
| Recipient | Address of the transaction recipient |
| Amount | Transaction amount |
| Timestamp | Creation time of the transaction |
| Signature | Signature data for the transaction |
| Payload | Other data of the transaction |

2.2. Number Theory Knowledge

As a basic tool in the public key cryptography, number theory played a very important role in the research and development of modern cryptography. The discrete logarithm problem (DLP) is the basic conclusion in it and is widely used in many public key cryptographic schemes. In 1985, Koblitz and Miller first proposed the creative idea of applying the elliptic curve to cryptography and constructed the discrete logarithm problem on elliptic curves (ECDLP) based on the Abelian addition group consisting of elliptic curve points [33]. The cryptographic schemes used in this paper are mainly based on ECDLP. The relevant definition is as follows.

Definition 1 (DLP [33]). *Let g be a generator of the cyclic group \mathbb{G} with prime order, i.e., $\mathbb{G} = \langle g \rangle$. Randomly select h from \mathbb{G} . If there exists a negligible function $\text{negl}(\lambda)$ for all probabilistic polynomial time (PPT) adversaries \mathcal{A} , where λ is a security parameter, such that*

$$P[h = g^a \mid a \leftarrow \mathcal{A}(g, h)] \leq \text{negl}(\lambda) \quad (1)$$

then we say that it is difficult to solve the discrete logarithm problem.

Let E be the elliptic curve group on the finite field F_q , where $q = p^m$. In this paper, we take m as 1. Let G be a generator of $E(F_q)$. Similar to the above, the definition of ECDLP is given below.

Definition 2 (ECDLP [33]). *Let G be the base point of group $E(F_q)$. Randomly select point H from $E(F_q)$. If for all PPT adversaries \mathcal{A} , there exists a negligible function $\text{negl}(\lambda)$ such that*

$$P[H = k \cdot G \mid k \leftarrow \mathcal{A}(G, H)] \leq \text{negl}(\lambda) \quad (2)$$

then we say that it is difficult to solve the elliptic curve discrete logarithm problem.

All privacy protection schemes used in this paper involve the point addition operation on elliptic curves. Therefore, we can provide security for our privacy protection solution based on ECDLP. For example, for the Pedersen commitment $CM = s \cdot G + r \cdot H$ we used, when the adversary only knows CM , G and H , he cannot obtain the secret value s in polynomial time because of ECDLP.

To solve the problem of the construction of secret accounts in RZcoin, we propose the ciphertext equivalent signature scheme called CEs based on the existing ciphertext equivalent scheme.

This scheme mainly relies on bilinear mapping in number theory [34]. The definition of bilinear mapping is as follows.

Definition 3 (Bilinear Mapping). *Given an addition cyclic group \mathbb{G}_1 of prime order q and a multiplication cyclic group \mathbb{G}_2 , we define a mapping relationship $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ such that it satisfies the following properties:*

- *Bilinear : $\forall P, Q \in \mathbb{G}_1, \forall a, b \in Z_q^*, e(a \cdot P, b \cdot Q) = e(P, Q)^{ab}$ is established, where $Z_q^* = \{1, \dots, q - 1\}$;*
- *Non-degenerate : $\exists P, Q \in \mathbb{G}_1, e(P, Q) \neq 1 \in \mathbb{G}_2$ is established;*
- *Computability : $\forall P, Q \in \mathbb{G}_1$, there is an efficient polynomial time algorithm for computing $e(P, Q)$.*

If all the above properties are satisfied, the mapping relationship e is called a bilinear mapping.

2.3. Pedersen Commitment

In this paper, we use the Pedersen commitment scheme based on elliptic curves to hide the asset information [35]. Its inherent homomorphism can prove the linear relationship between the hidden asset.

Definition 4 (Pedersen Commitment [35]). *For the elliptic curve group $E(F_q)$, randomly pick $G, H \xleftarrow{R} E(F_q)$. \log_G^H is unknowable. The Pedersen commitment scheme contains the following three algorithms:*

- *$pp \leftarrow \text{Setup}(1^\lambda)$. For security parameter 1^λ , it outputs public parameters pp which will exist as implicit input in the following algorithms.*
- *$CM \leftarrow \text{Com}(s, r)$. For the secret value $s \in Z_q^*$, pick the random number $r \xleftarrow{R} Z_q^*$, then output:*

$$CM = s \cdot G + r \cdot H \quad (3)$$

- *$\{0, 1\} \leftarrow \text{Open}(CM, s, r)$. If Equation (3) holds, the output of this algorithm is 1. Otherwise the output is 0.*

On the basis of the above scheme, Bunz et al. proposed the Pedersen vector commitment in the bulletproof scheme published in 2018 to further strengthen the scalability of confidential transactions [36]. Bulletproof uses this scheme to achieve the hiding of the secret set s . In short, it constructs the corresponding Pedersen commitment for each element in the set of secret values, thereby achieving complete hiding and computable binding of all secret values. In the following definition, the concept of vector refers to a set of n elements.

Definition 5 (Pedersen Vector Commitment [36]). *For the elliptic curve group $E(F_q)$, randomly pick $\{\mathbf{G} = (G_1, G_2, \dots, G_n), H\} \xleftarrow{R} E(F_q)$, where n is the size of vector and \log_G^H is unknowable. A Pedersen vector commitment scheme contains the following three algorithms:*

- *$pp \leftarrow \text{Setup}(1^\lambda)$. For security parameter 1^λ , it outputs public parameters pp which will exist as implicit input in the following algorithms.*
- *$CM \leftarrow \text{Com}(\mathbf{s}, r)$. For the secret vector $\mathbf{s} \in Z_q^n$, pick the random number $r \xleftarrow{R} Z_q^*$, then output:*

$$CM = \mathbf{s} \cdot \mathbf{G} + r \cdot H = r \cdot H + \sum_{i=1}^n s_i \cdot G_i \quad (4)$$

where Z_q^n denotes the set of vectors containing n elements in Z_q .

- *$\{0, 1\} \leftarrow \text{Open}(CM, \mathbf{s}, r)$. If Equation (4) holds, the output is 1. Otherwise the output is 0.*

It was proved that because of the hardness of ECDLP, both of these commitment schemes are unconditionally hiding and computationally binding. The definition of these properties is as follows.

Theorem 1 (Unconditionally Hiding [35]). *If a commitment $\text{Com}(s, r)$ will not reveal any useful information about s , it is considered to be unconditionally hiding. That is, for any PPT adversary \mathcal{A} , his advantage satisfies:*

$$\text{Adv}_{\mathcal{A}}(\lambda) = P \left[b = b' \middle| \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda); \\ (s_0, s_1) \leftarrow \mathcal{A}(pp); \\ b \stackrel{R}{\leftarrow} \{0, 1\}, r \stackrel{R}{\leftarrow} Z_q^*; \\ CM \leftarrow \text{Com}(s_b, r); \\ b' \leftarrow \mathcal{A}(pp, CM). \end{array} \right] = \frac{1}{2} \quad (5)$$

Remark 1. *The proof of this theorem can be referred to [35]. The so-called unconditional hiding means that any two commitments are computationally indistinguishable. That is to say, even if the commitment CM is constructed by a value randomly selected from the set (s_0, s_1) known to the adversary, he can only guess the value contained in the CM correctly with half the probability.*

Theorem 2 (Computationally Binding [35]). *If a commitment $\text{Com}(s, r)$ can be opened to two different messages, ECDLP can be easily solved. That is, for any PPT adversary \mathcal{A} , there is a negligible function $\text{negl}(\lambda)$ that makes his advantage satisfy:*

$$\text{Adv}_{\mathcal{A}}(\lambda) = P \left[CM_1 = CM_2 \middle| \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda); \\ (s_1, s_2, r_1, r_2) \leftarrow \mathcal{A}(pp); \\ CM_1 \leftarrow \text{Com}(s_1, r_1), \\ CM_2 \leftarrow \text{Com}(s_2, r_2). \end{array} \right] \leq \text{negl}(\lambda) \quad (6)$$

Remark 2. *The proof of this theorem can be referred to [35]. The theorem means that the content of commitment cannot be changed after it is generated. That is to say, for the commitment CM_1 that was generated based on s_1 , the adversary cannot use the different value s_2 to construct the commitment CM_2 to satisfy $CM_1 = CM_2$.*

2.4. Zero-Knowledge Proof

As early as 1985, Goldwasser, Micali, and Rackoff came up with a groundbreaking idea called “zero-knowledge proof”. The original proof system is an interactive two-party protocol in which the prover can convince the verifier that certain statement is correct without providing the verifier with any useful information. In 1988, a non-interactive zero-knowledge proof scheme (NIZK) was proposed [37]. It extends the application of zero knowledge proof and lays a foundation for its application in blockchain.

In the existing blockchain privacy protection schemes, zero-knowledge proof is used to provide evidence for the legality of confidential transactions. Moreover, it works with the public key signature scheme to achieve accountability. Currently, zero-knowledge proof is considered to be the most effective solution to blockchain privacy issues. Therefore, for the hidden asset in RZcoin, we choose a zero-knowledge proof to provide evidence for its legitimacy.

2.4.1. Rangeproof

In RZcoin, we use bulletproof to implement rangeproofs to further reduce the size of proof [36]. Bulletproof optimizes the inner-product argument proposed by Bootle [38], reduces its communication complexity from $6\log_2 n$ to $2\log_2 n$, and constructs a non-interactive zero-knowledge proof scheme with low communication complexity based on it. By constructing the inner-product operation of the Pedersen vector commitments of two polynomials $L(x)$ and $R(x)$, bulletproof transforms the rangeproof of s equivalently into the special zero coefficient problem of inner product. In bulletproof, \mathbf{L} denotes the Pedersen vector commitment of $L(x)$; \mathbf{R} denotes the Pedersen vector commitment of $R(x)$. That is, if and only if s is within a certain range, the zero coefficient of $\langle \mathbf{L}, \mathbf{R} \rangle = \sum_{i=1}^n L_i R_i$ has a

specific form. This construction is implemented using the homomorphic property of Pedersen vector commitment. The definition of bulletproof is as follows.

Definition 6 (Bulletproof [36]). *For the input value a and global parameters $G, H \in E(F_q)$, this scheme can prove that $a \in [0, 2^n - 1]$ is true for the commitment $CM = a \cdot G + r \cdot H$. The value a here corresponds to the secret value s to be hidden in our scheme. In this definition, in order to ensure comparability with the details of scheme in [36], we choose to use a to denote the secret value. It mainly contains the following three algorithms:*

- $\{\mathbf{G} = (G_1, G_2, \dots, G_n), \mathbf{H} = (H_1, H_2, \dots, H_n), G, H\} \leftarrow \text{Setup}(1^\lambda)$. For security parameter 1^λ , it outputs public parameters $\{\mathbf{G}, \mathbf{H}, G, H\}$ which will exist as implicit input in the following algorithms. The definition of these parameters was introduced earlier.
- $\text{proof} \leftarrow \text{ProofGen}(a, r)$. The prover generates the rangeproof of a according to the following steps:
 - (1) Convert a to a binary string $\mathbf{a}_L \in \{0, 1\}^n$ and calculate $\mathbf{a}_R = \{\mathbf{a}_{L_{1-1}}, \dots, \mathbf{a}_{L_{n-1}}\}$;
 - (2) Choose blinding vectors $\mathbf{s}_L, \mathbf{s}_R \in Z_q^n$ and random parameters $r_1, r_2 \in Z_q^*$ to generate auxiliary parameters A, S .
 - (3) Use Fiat-Shamir heuristic to achieve non-interaction, i.e., use the hash function instead of the challenge interaction of y, z .
 - (4) Construct the two linear polynomials $L(x), R(x)$ and calculate:

$$T(x) = \langle L(x), R(x) \rangle \quad (7)$$

- (5) Use the polynomial identity test protocol to prove that Equation (7) holds and send proof = $\{\tau_x, \mu, T(x), \mathbf{L}, \mathbf{R}\}$.
- $\{0, 1\} \leftarrow \text{ProofVer}(\text{proof})$. The verifier needs to check that if $\mathbf{L}, \mathbf{R} \in \text{proof}$ are actually $L(x)$ and $R(x)$ and if Equation (7) holds. If the verification is passed, it means that a is indeed in the range $[0, 2^n - 1]$. Otherwise, it means that a is out of range and the commitment CM is illegal.

Remark 3. The specific details of the above definition can be found in [36]. For example, the symbols and calculation rules involved in bulletproof are described in detail in Section 2.3. For the first four steps included in the algorithm ProofGen , the corresponding calculation details can be found in the formula (36)–(45). In addition, the details of the fifth step can be found on page 17. In addition, the related content of algorithm ProofVer can be found on page 18.

2.4.2. Proof of Account Validity

In RZcoin, if users want to use the privacy protection service, they should create a new secret account for secret transactions. Although the account information will become visible throughout the network as the secret initial transaction recorded in the blockchain, the validity of new account cannot be verified directly because its balance is hidden in the commitment. Therefore, a proof must be attached to the secret initial transaction for validity verification.

Similar to the Ethereum, the account balance is zero during the initial creation period. Only through transactions with other accounts can the balance be increased. In other words, we only need to prove that the initial balance is zero. We implement this proof operation using the zero-knowledge proof based on ECDSA in RZcash. In this paper, we use the "key-prefixed" variant of Schnorr signature to reconstruct the scheme, thereby reducing the size of proof.

Definition 7 (The "key-prefixed" Schnorr signature [39]). *The "key-prefixed" variant of Schnorr signature is defined by the following three algorithms:*

- $(x, X) \leftarrow \text{KeyGen}(x)$. Let G be the base point of $E(F_q)$, the public key X is generated by:

$$X = x \cdot G \quad (8)$$

- $\sigma \leftarrow \text{Sign}(x, m)$. Pick the random number r , the signer calculates the random challenge c and the signature σ by the following steps. Here, Hash denotes a collision-resistant hash function. In this paper, we use SHA256 to achieve it.

$$R = r \cdot G, c = \text{Hash}(X, R, m), s = r + cx, \quad (9)$$

$$\sigma = (R, s) \quad (10)$$

- $\{0, 1\} \leftarrow \text{Ver}(\sigma, X, m)$. If Equation (11) holds, the signature σ is valid and the output is 1. Otherwise, the output is 0.

$$s \cdot G = R + c \cdot X \quad (11)$$

Based on the above scheme, we re-explain our original zero-knowledge proof as follows. To facilitate the subsequent algorithm description, we will call this scheme Szkp. Szkp mainly contains the following three algorithms:

- $pp \leftarrow \text{Setup}(1^\lambda)$. For security parameter 1^λ , it outputs public parameters pp which will exist as implicit input in the following algorithms.
- $z \leftarrow \text{zkpGen}(x, m)$. For the value x to be proved and the auxiliary parameter m , this algorithm calls the schnorr signature scheme to generate the corresponding evidence z .

$$(x, X) \leftarrow \text{KeyGen}(x), \sigma \leftarrow \text{Sign}(x, m) \quad (12)$$

$$z = \{X, m, \sigma\} \quad (13)$$

- $\{0, 1\} \leftarrow \text{zkpVer}(z)$. For the input z , if $1 \leftarrow \text{Ver}(\sigma, X, m)$, then the verification is passed, this algorithm outputs 1. Otherwise it outputs 0.

3. Ciphertext Equivalent Signature Scheme

In the existing blockchain system, each account contains a pair of keys (sk, pk) . It can construct any number of addresses for asset circulation based on its public key pk , but its available asset is only bound to its private key sk . Every time the user initiates a transaction, he needs to sign the transaction with his sk to ensure its non-repudiation. In RZcash, the privacy service is optional. To ensure the security of secret assets, RZcash requires that each user who uses this service must create a secret account to store encrypted assets and construct secret (or semi-secret) transactions. Correspondingly, the key pair bound to this new account should also be regenerated. It can isolate the user's open transactions from secret transactions, thereby avoiding the privacy disclosure caused by the same public key. However, it will quickly increase the storage space required for user keys. The cost of key management will also increase. The probability of the disclosure of private keys will increase accordingly.

To solve this problem, we attempt to improve the existing ciphertext equivalence test scheme and construct a new public key signature scheme, which is called ciphertext equivalence signature scheme (CEs). The so-called ciphertext equivalence test is to determine whether two different ciphertexts contain the same plaintext information without decryption. The ciphertext equivalent test scheme we used was proposed by Zhu et al. in 2018. This work improves the PKEwET scheme proposed by Lin et al. in 2016, and uses the straight line to construct equivalent tests to obtain better performance and lower ciphertext space. The specific details of the scheme called PKEwET-L are shown in Figure 2. This figure only introduces the core algorithm of PKEwET-L. Its concept definition and implementation details can be found in the original paper [14]. In simple terms, this scheme uses the straight line constructed by the message to implement the ciphertext authorization and equivalent test functions. The principle is: the straight line constructed by the same message must be the same. The scheme performs four authorizations based on four different scenarios, and each authorization performs different equivalent test functions. Its application scenarios are very comprehensive. Zhu et al. conducted a detailed analysis of its correctness and security. The results show that it can achieve OW-CCA security based on CDH and IND-CCA security based on DDH respectively for two different

types of attackers under the random prediction model. The specific analysis process can be found in Section 3.2.5 of [14].

Let $\Pi_{\text{PKEwET-L}} = \{\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Auth}, \text{Test}\}$ be the ciphertext equivalent test scheme in [14]:

| | |
|---|---|
| $\text{Setup}(1^\lambda)$ | $\{G, g, q, H_1, H_2, H_3, H_4, H_5, H_6\} \leftarrow \text{Setup}(1^\lambda); H_1: G \rightarrow \{0,1\}^3; H_2: G^4 \rightarrow Z_q^4; H_3, H_4, H_5, H_6: \{0,1\}^\lambda \rightarrow Z_q.$ |
| $\text{KeyGen}(\text{sp}, i)$ | $(pk_i, sk_i) \leftarrow \text{KeyGen}(\text{sp}, i); x_i, y_i \leftarrow Z_q, X_i = g^{x_i}, Y_i = g^{y_i}; (pk_i, sk_i) = ((X_i, Y_i), (x_i, y_i)).$ |
| $\text{Enc}(M, pk_i)$ | $C_i \leftarrow \text{Enc}(M, pk_i); p_1 = (H_3(M), H_4(M), p_2 = (H_5(M), H_6(M)); \text{construct } f(x) \text{ using } p_1, p_2; x_{i,3}, x_{i,2} \xleftarrow{\$} \{0,1\}^\lambda; y_{i,1} = f(x_{i,1}), y_{i,2} = f(x_{i,2}); r \in Z_q^*; C_{i,1} = g^r, C_{i,2} = M \oplus H_1(Y_i^r), C_{i,3} = (x_{i,1} x_{i,2} y_{i,1} y_{i,2}) \oplus H_2(X_i^r, C_{i,1}, C_{i,2}); C_i = (C_{i,1}, C_{i,2}, C_{i,3}).$ |
| $\text{Dec}(C_i, sk_i)$ | $\{M, \perp\} \leftarrow \text{Dec}(C_i, sk_i); M = C_{i,2} \oplus H_1(C_{i,1}^y); x_{i,1} x_{i,2} y_{i,1} y_{i,2} = C_{i,3} \oplus H_2(C_{i,1}^y, C_{i,1}, C_{i,2}); \text{if } f(x_{i,1}) = y_{i,1} \wedge f(x_{i,2}) = y_{i,2}, \text{return } M; \text{ else return } \perp.$ |
| $\text{Auth}(sk_i, C_i, \text{type})$ | $\{td_{1,i}, td_{2,i}, td_{3,i}, td_{4,i}\} \leftarrow \text{Auth}(sk_i, C_i, \text{type}); \text{if type} = 1, td_{2,i} = sk_i; \text{else if type} = 2, td_{2,i} = H_2(C_{i,1}^x, C_{i,3}, C_{i,2}); \text{else if type} = 3, td_{3,i} = (z_i, V_{i,1}, V_{i,2}) = \left(\left[H_2(C_{i,1}^x, C_{i,1}, C_{i,2}) \right]_{2^l}^{q^t}, Z_i^{y_{i,1}}, Z_i^{y_{i,2}} \right), Z_i = C_{i,1} C_{i,1}; \text{else if type} = 4, td_{4,i} = H_2(c_{i,1}^x, C_{i,1}, C_{i,2}).$ |
| $\text{Test}(C_i, td_{\text{type},i}, C_j, td_{\text{type},j})$ | $\{0,1\} \leftarrow \text{Test}(C_i, td_{\text{type},i}, C_j, td_{\text{type},j}); \text{type} = 1: x_{i,1} x_{i,2} y_{i,1} y_{i,2} = C_{i,3} \oplus H_2\left(C_{i,1}^{td_{1,i}}, C_{i,1}, C_{i,2}\right), x_{j,1} x_{j,2} y_{j,1} y_{j,2} = C_{j,3} \oplus H_2\left(C_{j,1}^{td_{1,j}}, C_{j,1}, C_{j,2}\right), f_i(x) = (q, (x_{i,1}, y_{i,1}), (x_{j,1}, y_{j,1})), f_j(x) = (q, (x_{i,2}, y_{i,2}), (x_{j,2}, y_{j,2})); \text{if } f_i(x) = f_j(x), \text{return 1; else return 0.} \\ \text{type} = 2: x_{i,1} x_{i,2} y_{i,1} y_{i,2} = C_{i,3} \oplus \text{td}_{2,i}, x_{j,1} x_{j,2} y_{j,1} y_{j,2} = C_{j,3} \oplus \text{td}_{2,j}, f_i(x) = (q, (x_{i,1}, y_{i,1}), (x_{j,1}, y_{j,1})), f_j(x) = (q, (x_{i,2}, y_{i,2}), (x_{j,2}, y_{j,2})); \text{if } f_i(x) = f_j(x), \text{return 1; else return 0.} \\ \text{type} = 3: x_{i,1} x_{i,2} = C_{i,3} \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix} \right]_{2^l}^{q^t} \oplus Z_i, x_{j,1} x_{j,2} = [C_{j,1}]_{2^l}^{q^t} \oplus Z_j, \Delta_{i,1} = \frac{x_{i,1}}{x_{i,1}-x_{i,2}}, \Delta_{i,2} = \frac{x_{i,2}}{x_{i,1}-x_{i,2}}, \Delta_{j,1} = \frac{x_{j,1}}{x_{j,2}-x_{j,1}}, \Delta_{j,2} = \frac{x_{j,2}}{x_{j,1}-x_{j,2}}, \text{if } V_{i,1}^{\Delta_{i,1}} V_{j,1}^{\Delta_{j,1}} = V_{j,2}^{\Delta_{j,2}} V_{i,2}^{\Delta_{i,2}}, \text{return 1; else return 0.} \\ \text{type} = 4: x_{i,1} x_{i,2} y_{i,1} y_{i,2} = C_{i,3} \oplus \text{td}_{4,i}, x_{j,1} x_{j,2} y_{j,1} y_{j,2} = C_{j,3} \oplus H_2\left(C_{j,1}^{td_{4,j}}, C_{j,1}, C_{j,2}\right), f_i(x) = (q, (x_{i,1}, y_{i,1}), (x_{j,1}, y_{j,1})), f_j(x) = (q, (x_{i,2}, y_{i,2}), (x_{j,2}, y_{j,2})); \text{if } f_i(x) = f_j(x), \text{return 1; else return 0.}$ |

Figure 2. Algorithmic Composition of PKEwET-L in [14].

Inspired by the straight line construction of this scheme, we further optimize the design of its equivalent test and propose a public key signature mechanism called CEs suitable for the blockchain system. Based on the original private key of users, CEs generates a new public key by constructing the ciphertext equivalent commitment. The public key is bound to the secret account and is used to sign and verify secret transactions. The address for receiving secret transactions is also generated based on it. CEs does not need to generate a new private key, so it can reduce the cost of key management while ensuring the non-repudiation of signatures.

Definition 8. The ciphertext equivalence signature scheme CEs is defined as the following four algorithms:

- $pp \leftarrow \text{Setup}(1^\lambda)$. For security parameter 1^λ , it outputs public parameters pp which will exist as implicit input in the following algorithms.
- $(\tilde{sk}, \tilde{pk}) \leftarrow \text{KGen}(sk)$. Let G be the base point of the elliptic curve $E(F_q)$. Hash₁ and Hash₂ are collision-resistant hash functions. The key pair (\tilde{sk}, \tilde{pk}) is generated based on the original private key sk :

$$\tilde{sk} = sk \quad (14)$$

$$\tilde{b}_1 = \text{Hash}_1(\tilde{sk}), \tilde{b}_2 = \text{Hash}_2(\tilde{sk}) \quad (15)$$

$$\tilde{pk} = \{\tilde{b}_1 \cdot G \parallel \tilde{b}_2 \cdot G\} \quad (16)$$

In this paper, we implement Hash1 and Hash2 based on SHA256. Their construction methods are: $\text{Hash}_1 = \text{SHA256}(0||x)$, $\text{Hash}_2 = \text{SHA256}(1||x)$. That is, we construct these two different hash functions by making simple changes to the input x .

- $Sig \leftarrow \text{SGen}(T, \tilde{sk})$. Generate a signature for transaction T by Equation (17). Here, \tilde{b}_1 and \tilde{b}_2 must be positive numbers that are not zero, and $\tilde{b}_1 / \tilde{b}_2 = \tilde{b}_1 (\tilde{b}_2^{-1}) \bmod q$.

$$Sig = \text{Hash}_1(T)(\tilde{b}_1 / \tilde{b}_2) \cdot G \quad (17)$$

- $\{0,1\} \leftarrow \text{SVer}(Sig, \tilde{pk})$. For bilinear pairing e , verify:

$$e(Sig, \tilde{b}_2 \cdot G) = e(\tilde{b}_1 \cdot G, \text{Hash}_1(T) \cdot G) \quad (18)$$

If Equation (18) holds, the verification of Sig is passed.

Like the other general public key signature scheme, our scheme is unforgeable and non-repudiation. Among them, the definition of unforgeability is as follows:

Theorem 3 (Unforgeability). *A adversary cannot successfully forge a valid signature without knowing the signer's private key. That is, for any PPT adversary \mathcal{A} , there is a negligible function $\text{negl}(\lambda)$ that makes his advantage satisfy:*

$$\text{Adv}_{\mathcal{A}}(\lambda) = P \left[b = 1 \left| \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda); \\ \widetilde{sk}', \widetilde{pk}' \leftarrow \mathcal{A}(pp), \widetilde{sk}' \neq \widetilde{sk}; \\ \text{Sig}' \leftarrow \text{SigGen}(T, \widetilde{sk}'); \\ b \leftarrow \text{SigVer}(\text{Sig}', \widetilde{pk}'). \end{array} \right. \right] \leq \text{negl}(\lambda) \quad (19)$$

Proof of Theorem 3. As shown in Equation (17), the generation process of the signature Sig can be divided into three parts: the hash value of the transaction $\text{Hash}_1(T) \in Z_q^*$, the scalar $(\tilde{b}_1/b_2) \in Z_q^*$ and the base point G . Since the transaction T and the point G are publicly known throughout the network, the scalar (\tilde{b}_1/b_2) is the main target of attacks. In other words, there are two main methods for signature forgery attacks on CEs:

- (1) Attack on \tilde{b}_1 and \tilde{b}_2 . For the hash function Hash_1 , the adversary \mathcal{A} needs to find a pair of collisions with the same hash value \tilde{b}_1 . Similarly, \mathcal{A} also needs to find a pair of collisions with the same hash value \tilde{b}_2 for Hash_2 . That is, \mathcal{A} needs to launch the strong collision attack on $\text{Hash}_1(T), \text{Hash}_2(T)$ at the same time and all succeed.
- (2) Attack on (\tilde{b}_1/b_2) . \mathcal{A} needs to launch an attack on ECDLP to Sig .

For the first attack mentioned above, we use the hash function with strong collision to achieve defense. In the implementation of CEs, we construct these two hash functions based on SHA256. By adding randomness to the input of SHA256, we implement Hash_1 and Hash_2 . It was proved that the SHA256 algorithm is more secure than MD5 and SHA-1 in defending against birthday attacks and known differential attacks. In other words, SHA256 has good resistance to strong collision. Therefore, CEs can resist this attack. For the second attack, since ECDLP is a difficult problem, there is currently no effective solution to the problem. As a result, CEs can also resist such attack. \square

Because CEs satisfy the unforgeability, a valid signature can only be generated by the signer himself. Therefore, CEs is non-repudiation.

4. Overview of RZcoin

4.1. System Model

RZcoin is a decentralized payment system that provides optional privacy services based on the account model blockchain. In this paper, we choose Ethereum as the underlying blockchain system to build it. Each user needs to register an Ethereum account for subsequent transactions when entering RZcoin for the first time. The user can choose to mortgage a certain amount of assets to become an accounting node (i.e., full node) to participate in mining, or can only perform daily transactions as lightweight nodes. When a transaction is initiated, the complete information of it will be broadcast across the entire network through P2P communication. Some full nodes collect these transactions and verify their validity. Valid transactions will be integrated into blocks and recorded in the blockchain through a consensus mechanism (e.g., PoW).

RZcoin mainly contains three roles, namely payer, payee and verifier. When the transaction requires privacy services, its payer and payee need to first create a new account to store and transfer secret assets. We call this account a secret account. After the creation is completed, the payer and the payee can use their secret accounts for the next secret transfer without regenerating. In RZcoin

we implemented, the verifier is mainly played by the full node. When the full node receives the transaction, it will verify the evidence, and the verification. In addition, only if the verification is passed, the transaction can be uploaded to the chain. Subsequently, the payee updates its balance according to the transaction data recorded in the chain. Assuming that the node *A* is the transaction payer and the node *B* is the transaction payee, the basic workflow of RZcoin is shown in Figure 3. To highlight the subject, the figure omits the description of the process of open transactions in RZcoin.

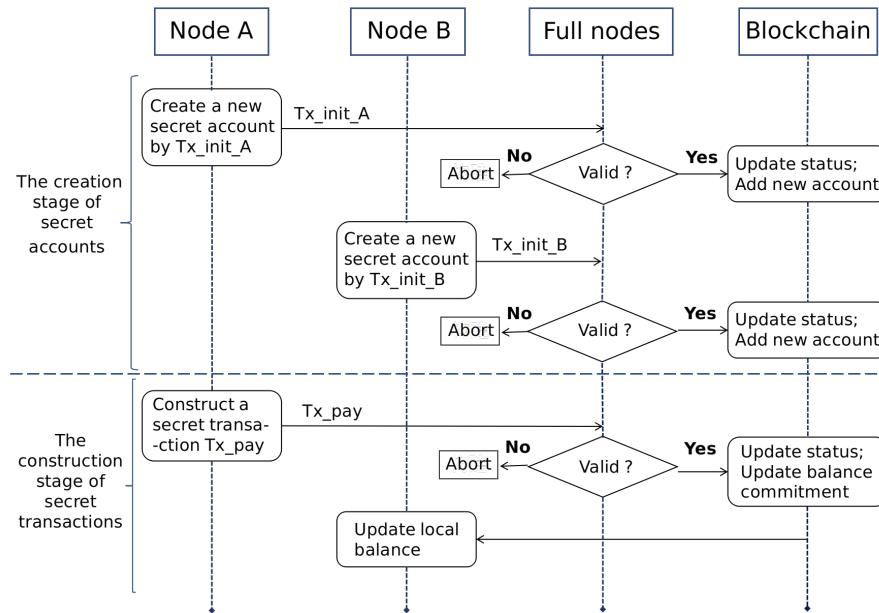


Figure 3. The Workflow of RZcoin.

Since the privacy service provided by RZcoin is optional, users can have both open accounts and secret accounts. The composition of transactions between different accounts is also different. Transactions between public accounts do not involve privacy services, so we do not describe them in this paper. Next, we will introduce the work that each role is responsible for in secret transactions.

- Payer: When the payer wants to initiate a secret transaction, he needs to update the current commitment of secret assets (i.e., its balance commitment) based on the amount spent in this transaction. He also generates the commitment corresponding to the transaction amount. For these commitments, he needs to provide corresponding evidence to prove its legitimacy, such as range proofs. At the same time, he will use the public key of payee to encrypt the transaction amount and corresponding random number. These ciphertexts will be sent to the payee so that he can verify the correctness of the amount. The payer packages the above data into a transaction, signs it and sends it to the network.
- Verifier: For the received secret transaction, the full node will verify the validity of the transaction signature, range proofs and the updated balance commitment. If the verification is passed, the transaction will be packaged into a block and recorded in the blockchain through distributed consensus.
- Payee: For the secret transaction related to him in the blockchain, the payee uses his private key to decrypt the ciphertext in it and verifies the corresponding commitment. If the verification fails, the payee will publish the evidence to the network and declare the secret transaction invalid. If the verification is successful, the payee updates his locally stored balance and random numbers.

To ensure security, the secret information owned by payer and payee must be stored in their respective TEEs, and calculations involving this information are also performed in it.

4.2. Adversarial Model

We define some types of attacks that RZcoin may face to lay the foundation for subsequent security analysis. RZcoin mainly faces the following three attack modes, namely balance forgery attack, signature forgery attack and over-spending attack. Next, we will introduce the principles and operations of these attacks.

4.2.1. Balance Forgery Attack

When the user wants to use the privacy service, he needs to create a secret account bound to his private key and publishes the initial information of it, such as its initial balance and account address, in the form of secret initial transaction on the blockchain network. The initial balance in such transaction is expressed in the form of commitment. That is, the balance of secret account is hidden from the beginning. Since then, the balance was updated in a hidden state. No one except the account owner can know the secret balance at any time.

Since RZcoin is built on Ethereum, the setting requirements of the initial balance are the same as that of Ethereum, i.e., the initial balance must be set to zero. Since the account creation is a local operation of the user, an adversary may want to violate the above system settings to forge the account balance commitment. When the adversary sets an account balance, he may want to use any known positive integer to replace zero to generate the commitment, thereby fabricating an available asset with no legal source for himself.

4.2.2. Signature Forgery Attack

For each secret transaction in the system, its initiator needs to sign it to ensure the traceability of the transaction source. This also realizes the non-repudiation of transactions. RZcoin uses CEs to generate the public key of secret accounts and sign secret transactions. In this regard, malicious adversaries may want to obtain key parameters of the signature by analyzing some public information in the Internet (e.g., a message signed with the same private key). The parameter can be used to forge the signer's signature, thereby further forging his transaction. Using this method, the adversary may want to transfer the victim's secret assets to his account or spend them. This is the so-called signature forgery attack.

4.2.3. Over-Spending Attack

In RZcoin, The asset information involved in each secret transaction is hidden in the commitment. Except for its payer and payee, no one can know the amount of this transaction. In addition, the secret balance of the transaction participants cannot be known by other nodes in the network. Malicious adversaries may want to spend more than their account balance in a single secret transaction and forge the legal range proof to disguise the unreasonable consumption behavior. This is the so-called signature over-spending attack.

5. Description of RZcoin

In this section, we will systematically introduce the transaction details in RZcoin. Because the focus of our work is on privacy services, we will focus on the secret and semi-secret transactions in RZcoin to introduce the details of their construction process and entire transaction cycle. Since the use of secret services requires the construction of a new secret account, the secret initial transaction is also the focus of our description. The specific process of open transactions is omitted. We start with the definition of some notations involved in RZcoin.

5.1. Notations

Table 2 shows some symbols used in the description of RZcoin and their meanings. For example, the symbols G and H represent two different points of the elliptic curve group $E(F_p)$, where G is the

base point of the curve and H is any point other than the base point. It requires that the discrete logarithm \log_G^H of H relative to G must be unknown. (\tilde{sk}, \tilde{pk}) is a key pair generated by the user calling CEs when creating a secret account, where \tilde{sk} represents the private key of the secret account, which is the same as the original private key of the user's open account; \tilde{pk} represents the public key of the secret account, which can be used to generate the address of secret account. ID and \widetilde{ID} respectively represent the addresses of open account and secret account, which are used to receive and initiate transactions. Their subscript $role$ indicates the role played by the owner of address in this transaction, such as payer and payee. \tilde{B}_{role} represents the secret balance corresponding to \widetilde{ID}_{role} and B_{role} is the open balance. \tilde{A}_{pay} means the transaction amount involving secret assets, A_{pay} is the opposite.

For the secret balance and transaction amount, RZcoin can hide them in the commitment. \widetilde{CM}_{role} represents the commitment corresponding to \tilde{B}_{role} . \tilde{r}_{role} represents the random number required to generate \widetilde{CM}_{role} . \widetilde{CM}_{pay} represents the commitment corresponding to \tilde{A}_{pay} . \tilde{r}_{pay} represents the random number required to generate \widetilde{CM}_{pay} . For the legality of these commitments, we use bulletproof and Szkp to generate corresponding evidence. \tilde{R}_{cm} represents the range proof corresponding to cm and \tilde{Z}_{cm} represents the zero-knowledge proof corresponding to the cm . Some of the above variables will be stored in the transaction Tx . $\tilde{\sigma}$ and σ represent the signature of transactions initiated by the secret account and open account respectively.

Table 2. System Symbols and Their Meaning.

| Notation | Description |
|----------------------------|--|
| F_p | Finite field with order p |
| G, H | Two points of $E(F_p)$, where G is the base point |
| (sk, pk) | Key pair of open accounts |
| (\tilde{sk}, \tilde{pk}) | Key pair of secret accounts |
| ID_{role} | Address of open account of $role$ |
| \widetilde{ID}_{role} | Address of secret account of $role$ |
| B_{role} | Open balance of $role$ |
| \tilde{B}_{role} | Secret balance of $role$ |
| \tilde{r}_{role} | Random number required for \tilde{B}_{role} |
| \widetilde{CM}_{role} | Commitment of \tilde{B}_{role} |
| A_{pay} | Transaction amount involving open assets |
| \tilde{A}_{pay} | Transaction amount involving secret assets |
| \tilde{r}_{pay} | Random number required for \tilde{A}_{pay} |
| \widetilde{CM}_{pay} | Commitment of \tilde{A}_{pay} |
| \tilde{R}_{cm} | Range proof of cm |
| \tilde{Z}_{cm} | Zero-knowledge proof of cm |
| $\tilde{\sigma}$ | Signature generated by secret accounts |
| σ | Signature generated by open accounts |

5.2. Secret Initial Transaction

The secret initial transaction is the first transaction generated by users after calling the privacy service. Its construction depends on the algorithm called *AccountGen*. Its input is the user's private key sk . Its output includes the secret initial transaction Tx_{init} , the secret balance \tilde{B} and corresponding \tilde{r} which need to be stored locally. It can be written as $\{Tx_{init}, \tilde{B}, \tilde{r}\} \leftarrow AccountGen(sk)$. The steps of this algorithm are as follows:

- (1) Call the key generation algorithm *KGen* in CEs to generate the key pair (\tilde{sk}, \tilde{pk}) . At the same time, \widetilde{ID} will be generated based on \tilde{pk} . Its generation method is consistent with the same functional in Ethereum.

$$(\tilde{sk}, \tilde{pk}) \leftarrow KGen(sk) \quad (20)$$

- (2) Set the initial balance $\tilde{B} = 0$ and randomly select \tilde{r} from Z_q for subsequent commitment construction.

- (3) Calculate the commitment \widetilde{CM} of \widetilde{B} . To prove that \widetilde{B} is 0, the proof generation algorithm $zkpGen$ in Szkp needs to be called to generate the corresponding evidence $\widetilde{Z}_{\widetilde{CM}}$. m here is set by the system and its value will not affect the generation of evidence.

$$\widetilde{CM} = \widetilde{B} \cdot G + \widetilde{r} \cdot H = \widetilde{r} \cdot H \quad (21)$$

$$\widetilde{Z}_{\widetilde{CM}} \leftarrow zkGen(\widetilde{r}, m) \quad (22)$$

- (4) Integrate the data generated in the above steps into Tx_{init} .

$$Tx_{init} = \{\widetilde{ID}, \widetilde{pk}, \widetilde{CM}, \widetilde{Z}_{\widetilde{CM}}\} \quad (23)$$

- (5) Use \widetilde{sk} to sign this transaction, i.e., call the signature generation algorithm $SGen$ in CEs to generate the signature $\widetilde{\sigma}_{init}$. Attach the signature to Tx_{init} and broadcast Tx_{init} through the network.

$$\widetilde{\sigma}_{init} \leftarrow SGen(Tx_{init}, \widetilde{sk}) \quad (24)$$

$$Tx_{init} \leftarrow Tx_{init} || \widetilde{\sigma}_{init} \quad (25)$$

In the above process, \widetilde{sk} is bound to all assets in the secret account, namely \widetilde{B} . The randomness of \widetilde{r} determines the security of secret assets. Therefore, \widetilde{sk} , \widetilde{B} and \widetilde{r} must be stored in the TEE hardware environment of the user's local machine to ensure security. Whenever there is a transfer of secret assets, the user needs to update the locally stored \widetilde{B} and \widetilde{r} to ensure data consistency across the network.

When Tx_{init} is broadcast through the entire network, some full nodes will receive it and call the algorithm $AccountVer$ to verify its validity. Its input is Tx_{init} and its output is 1 or 0. It can be written as $\{0, 1\} \leftarrow AccountVer(Tx_{init})$. If the output is 1, it means that all verifications were passed. The full node will then package it together with other valid transactions into a new block, participate in the consensus process of the entire network and wait to be recorded in the blockchain. The steps of this algorithm are as follows:

- (1) Call the signature verification algorithm $SVer$ in CEs to verify the legality of $\widetilde{\sigma}_{init}$. If it returns 1, the verification is passed. Otherwise, the algorithm is aborted and Tx_{init} is discarded.
- (2) Call the proof verification algorithm $zkpVer$ in Szkp to verify $\widetilde{Z}_{\widetilde{CM}}$. If it returns 1, the verification is passed. Otherwise, the algorithm is aborted and Tx_{init} is discarded.
- (3) If all above verifications are passed, the algorithm returns 1, otherwise 0.

The process of the secret initial transaction is shown in Figure 3. All users in the system can construct a new secret account in this way.

5.3. Secret Transaction

In the previous section, we briefly introduced the concept of secret transactions. The entire process of secret transactions does not involve open assets. All transaction information is hidden through privacy services. Next, we will take a specific transaction scenario as an example to introduce the specific process of secret transactions. In this scenario, node \mathcal{A} is the payer of the transaction. It uses its secret account $\widetilde{ID}_{\mathcal{A}}$ to initiate this transaction. Node \mathcal{B} is the payee of the transaction. It receives the transfer from payer through its secret account $\widetilde{ID}_{\mathcal{B}}$. The full node plays the role of verifier. The overall process of the transaction is shown in Figure 4. Next, we will explain in detail according to different roles.

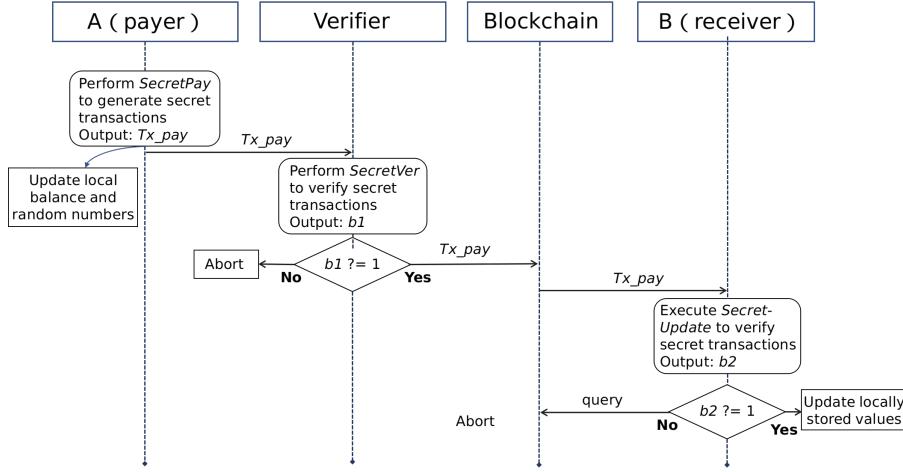


Figure 4. The Interactive Process of Secret Transactions.

5.3.1. Payer \mathcal{A}

The construction of secret transactions depends on the algorithm *SecretPay*. The payer \mathcal{A} will call it to generate the key parameters required in the secret transaction. Its input includes $\tilde{s}\mathbf{k}_{\mathcal{A}}$, \mathcal{A} 's current balance $\tilde{B}_{\mathcal{A}}^{old}$, the corresponding $\tilde{r}_{\mathcal{A}}^{old}$, the amount \tilde{A}_{pay} , $\tilde{p}\mathbf{k}_{\mathcal{B}}$, \mathcal{B} 's current balance commitment $\widetilde{CM}_{\mathcal{B}}^{old}$, $\widetilde{ID}_{\mathcal{A}}$ and $\widetilde{ID}_{\mathcal{B}}$. Its output includes the secret transaction $Tx_{\overline{pay}}$, \mathcal{A} 's updated balance $\tilde{B}_{\mathcal{A}}^{new}$ and the corresponding $\tilde{r}_{\mathcal{A}}^{new}$. Among them, $\tilde{B}_{\mathcal{A}}^{new}$ and $\tilde{r}_{\mathcal{A}}^{new}$ are used to update the corresponding variables stored in the local TEE. $Tx_{\overline{pay}}$ is released to the blockchain network. It can be written as $\{Tx_{\overline{pay}}, \tilde{B}_{\mathcal{A}}^{new}, \tilde{r}_{\mathcal{A}}^{new}\} \leftarrow SecretPay(\widetilde{ID}_{\mathcal{A}}, \widetilde{ID}_{\mathcal{B}}, \tilde{s}\mathbf{k}_{\mathcal{A}}, \tilde{B}_{\mathcal{A}}^{old}, \tilde{r}_{\mathcal{A}}^{old}, \tilde{A}_{pay}, \tilde{p}\mathbf{k}_{\mathcal{B}}, \widetilde{CM}_{\mathcal{B}}^{old})$. The steps of this algorithm are as follows:

- (1) Randomly select $\tilde{r}_{pay} \in Z_q^*$ to generate the commitment of \tilde{A}_{pay} , and update the locally saved variables;

$$\tilde{B}_{\mathcal{A}}^{new} = \tilde{B}_{\mathcal{A}}^{old} - \tilde{A}_{pay} \quad (26)$$

$$\tilde{r}_{\mathcal{A}}^{new} = \tilde{r}_{\mathcal{A}}^{old} - \tilde{r}_{pay} \quad (27)$$

- (2) For the updated balance $\tilde{B}_{\mathcal{A}}^{new}$, $\tilde{r}_{\mathcal{A}}^{new}$ is used to generate $\widetilde{CM}_{\mathcal{A}}^{new}$. At the same time, \tilde{r}_{pay} is used to generate \widetilde{CM}_{pay} .

$$\widetilde{CM}_{\mathcal{A}}^{new} = \tilde{B}_{\mathcal{A}}^{new} \cdot G + \tilde{r}_{\mathcal{A}}^{new} \cdot H \quad (28)$$

$$\widetilde{CM}_{pay} = \tilde{A}_{pay} \cdot G + \tilde{r}_{pay} \cdot H \quad (29)$$

- (3) For the two commitments generated in the previous step, the proof generation algorithm *ProofGen* in bulletproof is used to generate the corresponding evidence.

$$\widetilde{R}_{\widetilde{CM}_{\mathcal{A}}^{new}} \leftarrow ProofGen(\tilde{B}_{\mathcal{A}}^{new}, \tilde{r}_{\mathcal{A}}^{new}) \quad (30)$$

$$\widetilde{R}_{\widetilde{CM}_{pay}} \leftarrow ProofGen(\tilde{A}_{pay}, \tilde{r}_{pay}) \quad (31)$$

- (4) Use $\tilde{p}\mathbf{k}_{\mathcal{B}}$ to encrypt \tilde{A}_{pay} and \tilde{r}_{pay} . This operation allows \mathcal{B} to confirm the transaction amount and verify it with the corresponding commitment to ensure that \mathcal{A} has not cheated. The encryption algorithm here is not limited.

$$C \leftarrow Enc_{\tilde{p}\mathbf{k}_{\mathcal{B}}}(\tilde{A}_{pay}, \tilde{r}_{pay}) \quad (32)$$

- (5) Update the balance commitment of \mathcal{B} and attach it to the transaction.

$$\widetilde{CM}_{\mathcal{B}}^{new} = \widetilde{CM}_{\mathcal{B}}^{old} + \widetilde{CM}_{pay} \quad (33)$$

- (6) Construct the secret transaction $Tx_{\widetilde{pay}}$ as follows, use $\tilde{sk}_{\mathcal{A}}$ to generate its signature by $SGen$ in CEs, and finally broadcast it through the network.

$$Tx_{\widetilde{pay}} = \{\widetilde{ID}_{\mathcal{A}}, \widetilde{ID}_{\mathcal{B}}, \widetilde{CM}_{\mathcal{A}}^{new}, \widetilde{CM}_{pay}, \widetilde{CM}_{\mathcal{B}}^{new}, \widetilde{R}_{\widetilde{CM}_{\mathcal{A}}^{new}}, \widetilde{R}_{\widetilde{CM}_{pay}}, C\} \quad (34)$$

$$\sigma_{\widetilde{pay}} \leftarrow SGen(Tx_{\widetilde{pay}}, \tilde{sk}_{\mathcal{A}}) \quad (35)$$

$$Tx_{\widetilde{pay}} \leftarrow Tx_{\widetilde{pay}} || \sigma_{\widetilde{pay}} \quad (36)$$

5.3.2. Verifier

After receiving the transaction, the full node needs to call the algorithm $SecretVer$ to verify the validity of $Tx_{\widetilde{pay}}$. Its input includes $Tx_{\widetilde{pay}}, \widetilde{CM}_{\mathcal{A}}^{old}, \widetilde{CM}_{\mathcal{B}}^{old}$ and $\widetilde{pk}_{\mathcal{A}}$. Its output is 0 or 1. It can be written as $\{0, 1\} \leftarrow SecretVer(Tx_{\widetilde{pay}}, \widetilde{CM}_{\mathcal{A}}^{old}, \widetilde{CM}_{\mathcal{B}}^{old}, \widetilde{pk}_{\mathcal{A}})$. Only when all the following verifications are passed, the output will be 1. The full node will package $Tx_{\widetilde{pay}}$ into blocks and record it in the blockchain. The steps of this algorithm are as follows:

- (1) Call $SVer$ in CEs to verify the validity of $\sigma_{\widetilde{pay}}$. If it returns 1, the verification is passed and the following steps can be continued. Otherwise the transaction is discarded.
- (2) Call the proof verification algorithm $ProofVer$ in bulletproof to verify $\widetilde{R}_{\widetilde{CM}_{\mathcal{A}}^{new}}$ and $\widetilde{R}_{\widetilde{CM}_{pay}}$. If they all return 1, the verification is passed and the following steps can be continued. Otherwise the transaction is discarded.
- (3) Respectively verify the algebraic relationship between these balance commitments. If Equations (37) and (38) hold, the verification is passed and the following steps can be continued. Otherwise the transaction is discarded.

$$\widetilde{CM}_{\mathcal{A}}^{new} = \widetilde{CM}_{\mathcal{A}}^{old} - \widetilde{CM}_{pay} \quad (37)$$

$$\widetilde{CM}_{\mathcal{B}}^{new} = \widetilde{CM}_{\mathcal{B}}^{old} + \widetilde{CM}_{pay} \quad (38)$$

- (4) If all above verifications are passed, the algorithm returns 1. Otherwise it returns 0.

5.3.3. Payee \mathcal{B}

When the payee \mathcal{B} finds $Tx_{\widetilde{pay}}$ in the blockchain, he needs to call the algorithm $SecretUpdate$ to perform the final verification process shown below. Its input is $Tx_{\widetilde{pay}}$ and $\tilde{sk}_{\mathcal{B}}$. Its output is the verification result. 1 means pass and 0 means failure. It can be written as $\{0, 1\} \leftarrow SecretUpdate(Tx_{\widetilde{pay}}, \tilde{sk}_{\mathcal{B}})$. If the verification is passed, the locally stored variables will be updated. If the verification fails, \mathcal{B} will cancel the transaction and broadcast the corresponding evidence through the entire network, namely \widetilde{A}_{pay} and \widetilde{r}_{pay} . The steps of this algorithm are as follows:

- (1) Use $\tilde{sk}_{\mathcal{B}}$ stored locally to decrypt the ciphertext C .

$$\widetilde{A}_{pay}, \widetilde{r}_{pay} \leftarrow Dec_{\tilde{sk}_{\mathcal{B}}}(C) \quad (39)$$

- (2) Use the decrypted \widetilde{A}_{pay} and \widetilde{r}_{pay} to verify the legitimacy of \widetilde{CM}_{pay} . If the equation holds, the algorithm returns 1. Otherwise it returns 0.

$$\widetilde{CM}_{pay} = \widetilde{A}_{pay} \cdot G + \widetilde{r}_{pay} \cdot H \quad (40)$$

5.4. Semi-Secret Transaction

The semi-secret transactions refers to transactions involving one of the payer and payee using public accounts. It realizes the conversion between secret assets and open assets. When the payer

and payee are the same user, this conversion realizes the "cash-out" and "recharge" operations of the user's secret assets. Next, we will use two specific scenarios as examples to show the interactive process of semi-secret transactions, which are called Scene 1 and Scene 2 respectively. Scene 1 is used to describe it initiated by the open account and Scene 2 is used to describe it initiated by the secret account. There are three nodes involved in these two scenarios: \mathcal{A} , \mathcal{B} and \mathcal{C} . \mathcal{A} plays the role of payer in Scene 1. It uses the open account to initiate the semi-secret transaction Tx_{pay1} . \mathcal{B} plays the role of payee in Scene 1. Subsequently, \mathcal{B} plays the role of payer in Scene 2. It uses the secret account to initiate the semi-secret transaction Tx_{pay2} . \mathcal{C} plays the role of payee in Scene 2. In these two scenarios, the full node is responsible for verifying transactions. The interactive process of these scenarios is shown in Figure 5. Next, we will first introduce Scene 1 according to different roles.

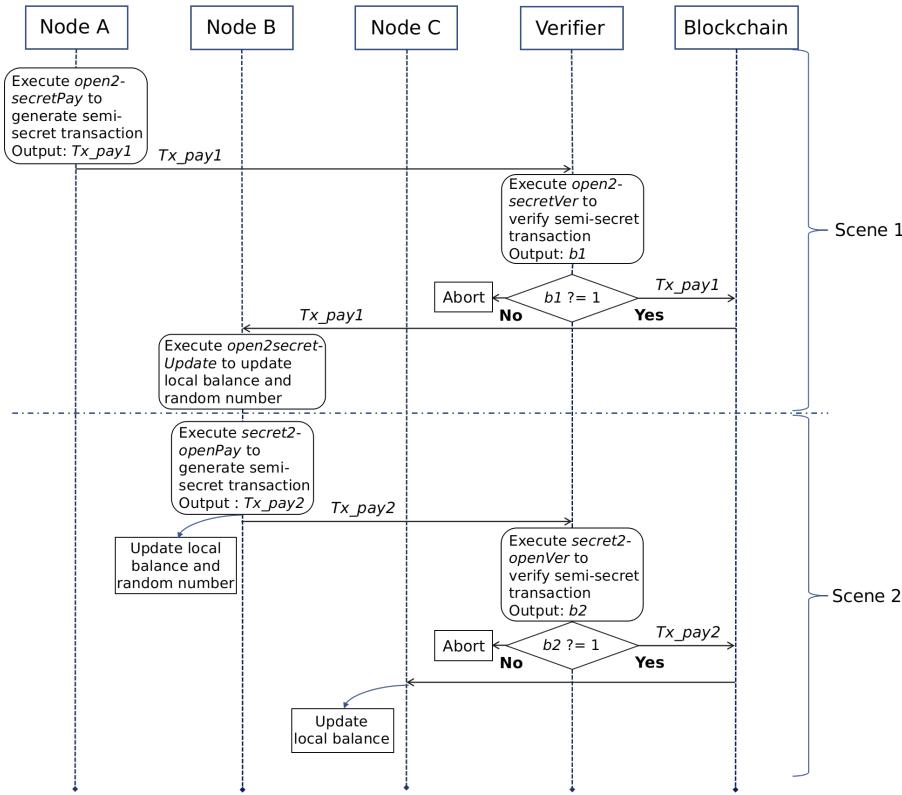


Figure 5. The Interactive Process of Semi-Secret Transactions.

5.4.1. Scene 1

Payer \mathcal{A} :

The construction of Tx_{pay1} depends on the algorithm $open2secretPay$. The payer \mathcal{A} will call it to generate the key parameters required in this transaction. It can be written as $Tx_{pay1} \leftarrow open2secretPay(A_{pay1}, sk_{\mathcal{A}}, B_{\mathcal{A}}^{old}, ID_{\mathcal{A}}, \tilde{pk}_{\mathcal{B}}, \widetilde{CM}_{\mathcal{B}}^{old}, \widetilde{ID}_{\mathcal{B}})$. The steps of this algorithm are as follows:

- (1) Randomly select $\tilde{r}_{pay1} \in Z_q^*$ for the update operation of the random number saved by \mathcal{B} . Since the transaction amount is a public value, only the commitment corresponding to \tilde{r}_{pay1} and its zero-knowledge proof need be generated.

$$\widetilde{CM}_{\tilde{r}_{pay1}} = \tilde{r}_{pay1} \cdot H \quad (41)$$

$$\widetilde{Z}_{\tilde{r}_{pay1}} \leftarrow zkpGen(\tilde{r}_{pay1}, m) \quad (42)$$

- (2) Use $\widetilde{pk}_{\mathcal{B}}$ to encrypt \widetilde{r}_{pay1} to get the ciphertext C_1 . The ciphertext is sent to the \mathcal{B} together with Tx_{pay1} , assisting \mathcal{B} to update the local random number.

$$C_1 \leftarrow Enc_{\widetilde{pk}_{\mathcal{B}}}(\widetilde{r}_{pay1}) \quad (43)$$

- (3) Update $\widetilde{CM}_{\mathcal{B}}^{old}$ based on A_{pay1} .

$$\widetilde{CM}_{\mathcal{B}}^{new} = \widetilde{CM}_{\mathcal{B}}^{old} + A_{pay1} \cdot G + \widetilde{r}_{pay1} \cdot H \quad (44)$$

- (4) \mathcal{A} update his open balance $B_{\mathcal{A}}^{old}$.

$$B_{\mathcal{A}}^{new} = B_{\mathcal{A}}^{old} - A_{pay1} \quad (45)$$

- (5) Construct Tx_{pay1} as follows, use $sk_{\mathcal{A}}$ to generate the corresponding transaction signature, and finally broadcast Tx_{pay1} through the entire network. $SigGen$ is the original signature algorithm of the blockchain system (generally ECDSA).

$$Tx_{pay1} = \{ID_{\mathcal{A}}, ID_{\mathcal{B}}, B_{\mathcal{A}}^{new}, A_{pay1}, \widetilde{CM}_{\mathcal{B}}^{new}, \widetilde{CM}_{\widetilde{r}_{pay1}}, \widetilde{Z}_{\widetilde{r}_{pay1}}, C_1\} \quad (46)$$

$$\sigma_{pay1} = SigGen(Tx_{pay1}, sk_{\mathcal{A}}) \quad (47)$$

$$Tx_{pay1} \leftarrow Tx_{pay1} || \sigma_{pay1} \quad (48)$$

Verifier:

After receiving the transaction Tx_{pay1} , the full node needs to call the algorithm $open2secretVer$ to verify the validity of the transaction. It can be written as $\{0,1\} \leftarrow open2secretVer(Tx_{pay1}, pk_{\mathcal{A}}, B_{\mathcal{A}}^{old}, \widetilde{CM}_{\mathcal{B}}^{old})$. Only when its output is 1, can Tx_{pay1} be finally recorded in the blockchain. The steps of this algorithm are as follows:

- (1) Call $SigVer$ to verify the legality of σ_{pay1} , if it returns 1, the verification is passed. Otherwise the transaction is discarded.
- (2) Call $zkpVer$ to verify the legality of $\widetilde{Z}_{\widetilde{r}_{pay1}}$. If it returns 1, the verification is passed. Otherwise the transaction is discarded;
- (3) Verify the correctness of the balance updates of \mathcal{A} and \mathcal{B} . If Equations (49) and (50) hold, the verification is passed. Otherwise the transaction is discarded.

$$B_{\mathcal{A}}^{new} = B_{\mathcal{A}}^{old} - A_{pay1} \quad (49)$$

$$\widetilde{CM}_{\mathcal{B}}^{new} = \widetilde{CM}_{\mathcal{B}}^{old} + A_{pay1} \cdot G + \widetilde{CM}_{\widetilde{r}_{pay1}} \quad (50)$$

Payee \mathcal{B} :

When \mathcal{B} finds that Tx_{pay1} is recorded in the blockchain, it calls the algorithm $open2secretUpdate$ to update the locally stored variables. It can be written as $\{\widetilde{B}_{\mathcal{B}}^{new}, \widetilde{r}_{\mathcal{B}}^{new}\} \leftarrow open2secretUpdate(Tx_{pay1}, \widetilde{sk}_{\mathcal{B}}, \widetilde{B}_{\mathcal{B}}^{old}, \widetilde{r}_{\mathcal{B}}^{old})$. The steps of this algorithm are as follows:

- (1) Use the locally saved $\widetilde{sk}_{\mathcal{B}}$ to decrypt the ciphertext C_1 to obtain the random number corresponding to the transaction.

$$\widetilde{r}_{pay1} \leftarrow Dec_{\widetilde{sk}_{\mathcal{B}}}(C_1) \quad (51)$$

- (2) Use the decrypted \widetilde{r}_{pay1} to update the locally stored variables.

$$\widetilde{r}_{\mathcal{B}}^{new} = \widetilde{r}_{\mathcal{B}}^{old} + \widetilde{r}_{pay1} \quad (52)$$

$$\tilde{B}_{\mathcal{B}}^{new} = \tilde{B}_{\mathcal{B}}^{old} + A_{pay1} \quad (53)$$

5.4.2. Scene 2

In Scene 1, we describe the conversion from open assets to secret assets. Next, we will introduce Scene 2 according to different roles.

Payer \mathcal{B} :

The construction of Tx_{pay2} depends on the algorithm $secret2openPay$. The payer \mathcal{B} will call it to generate the key parameters required in this transaction. It can be written as $\{Tx_{pay2}, \tilde{B}_{\mathcal{B}}^{new}, \tilde{r}_{\mathcal{B}}^{new}\} \leftarrow secret2openPay(\widetilde{ID}_{\mathcal{B}}, \widetilde{ID}_{\mathcal{C}}, A_{pay2}, \tilde{s}\mathbf{k}_{\mathcal{B}}, \tilde{B}_{\mathcal{B}}^{old}, \tilde{r}_{\mathcal{B}}^{old}, pk_{recC}, B_{\mathcal{C}}^{old})$. The steps of this algorithm are as follows:

- (1) Randomly select $\tilde{r}_{pay2} \in Z_q^*$ to update the balance and random number of \mathcal{B} 's secret account stored locally.

$$\tilde{B}_{\mathcal{B}}^{new} = \tilde{B}_{\mathcal{B}}^{old} - A_{pay2} \quad (54)$$

$$\tilde{r}_{\mathcal{B}}^{new} = \tilde{r}_{\mathcal{B}}^{old} - \tilde{r}_{pay2} \quad (55)$$

- (2) Generate the commitment of \tilde{r}_{pay2} and its corresponding zero-knowledge proof.

$$\widetilde{CM}_{\tilde{r}_{pay2}} = \tilde{r}_{pay2} \cdot H \quad (56)$$

$$\widetilde{Z}_{\tilde{r}_{pay2}} \leftarrow zkpGen(\tilde{r}_{pay2}, m) \quad (57)$$

- (3) Generate the updated balance and commitment.

$$\widetilde{CM}_{\mathcal{B}}^{new} = \tilde{B}_{\mathcal{B}}^{new} \cdot G + \tilde{B}_{\mathcal{B}}^{new} \cdot H \quad (58)$$

$$B_{\mathcal{C}}^{new} = B_{\mathcal{C}}^{old} + A_{pay2} \quad (59)$$

- (4) Call bulletproof to generate the corresponding evidence of the updated commitment.

$$\widetilde{R}_{\widetilde{CM}_{\mathcal{B}}^{new}} \leftarrow ProofGen(\tilde{B}_{\mathcal{B}}^{new}, \tilde{r}_{\mathcal{B}}^{new}) \quad (60)$$

- (5) Construct Tx_{pay2} as follows, use $\tilde{s}\mathbf{k}_{\mathcal{B}}$ to generate the transaction signature, and finally broadcast Tx_{pay2} through the entire network.

$$Tx_{pay2} = \{\widetilde{ID}_{\mathcal{B}}, ID_{\mathcal{C}}, \widetilde{CM}_{\mathcal{B}}^{new}, A_{pay2}, B_{\mathcal{C}}^{new}, \widetilde{R}_{\widetilde{CM}_{\mathcal{B}}^{new}}, \widetilde{CM}_{\tilde{r}_{pay2}}, \widetilde{Z}_{\tilde{r}_{pay2}}\} \quad (61)$$

$$\sigma_{pay2} \leftarrow SGen(Tx_{pay2}, \tilde{s}\mathbf{k}_{\mathcal{B}}) \quad (62)$$

$$Tx_{pay2} \leftarrow Tx_{pay2} || \sigma_{pay2} \quad (63)$$

Verifier:

After receiving Tx_{pay2} , the full node needs to call the algorithm $secret2openVer$ to verify the validity of the transaction. It can be written as $\{0, 1\} \leftarrow secret2openVer(Tx_{pay2}, \widetilde{pk}_{\mathcal{B}}, \widetilde{CM}_{\mathcal{B}}^{old}, \widetilde{B}_{\mathcal{C}}^{old})$. Only when the output is 1, can Tx_{pay2} be finally recorded in the blockchain.

- (1) Call $SVer$ to verify the validity of σ_{pay2} . If it returns 1, the verification is passed. Otherwise the transaction is discarded.
- (2) Call $zkpVer$ to verify the validity of $\widetilde{Z}_{\tilde{r}_{pay2}}$. If it returns 1, the verification is passed. Otherwise the transaction is discarded.
- (3) Call $ProofVer$ to verify the validity of $\widetilde{R}_{\widetilde{CM}_{\mathcal{B}}^{new}}$. If it returns 1, the verification is passed. Otherwise the transaction is discarded.

- (4) Verify the correctness of the updated balance of \mathcal{B} and \mathcal{C} . If Equations (64) and (65) hold, the verification is passed. Otherwise the transaction is discarded.

$$B_{\mathcal{C}}^{new} = B_{\mathcal{C}}^{old} + A_{pay2} \quad (64)$$

$$\widetilde{CM}_{\mathcal{B}}^{new} = \widetilde{CM}_{\mathcal{B}}^{old} - A_{pay2} \cdot G - \widetilde{CM}_{\tilde{r}_{pay2}} \quad (65)$$

Payee \mathcal{C} :

When \mathcal{C} finds that Tx_{pay2} was recorded in the blockchain, it only needs to update the locally stored balance of its open account. This operation has nothing to do with the secret service and will not be described here.

6. Security Analysis

In the previous article, we defined three attack models that RZcoin may face. In this section, we will perform the heuristic analysis of security of RZcoin based on these attack models.

6.1. Balance Forgery Attack

This attack mainly refers to an attack against the initial balance. As long as the initial balance is legal, RZcoin will ensure the legality of balances updated subsequently. RZcoin chooses to use the non-interactive zero-knowledge proof scheme called Szkp to resist this attack. This scheme can prove that the user honestly calculated the initial balance commitment of the secret account without exposing the secret information $\widetilde{CM} = \tilde{r} \cdot H$. This solution is based on the schnorr signature, so its security is based on the security of schnorr signature. The schnorr signature scheme we used is the so-called "key-prefixed" variant. This variant is considered to have better multi-user security than the classic variant and is therefore widely used [40]. We chose this variant to make our solution more portable. The schnorr signature has higher security than ECDSA, so the security of our scheme is also improved.

Because of the schnorr signature, the proof generated by Szkp also satisfies unforgeability. This property is based on the ECDLP. Based on it, a malicious adversary cannot know the discrete logarithm $\log_H^{\widetilde{CM}'}$ of the forged commitment \widetilde{CM}' relative to H , where $\widetilde{CM}' = \tilde{B}' \cdot G + \tilde{r}' \cdot H$, without knowing the discrete logarithm \log_G^H . That is, he cannot transform \widetilde{CM}' into $\widetilde{CM}' = (\tilde{B}' \log_G^H + \tilde{r}') \cdot H$. Therefore, the adversary cannot construct a legal zero-knowledge proof. This shows that **RZcoin can resist the balance forgery attack**.

6.2. Signature Forgery Attack

The signature mechanism of RZcoin is implemented through CEs. In Section 3, We analyzed its unforgeability. We divided the attacks against this scheme into two categories, and gave heuristic security analysis separately. We use CEs to separate the signature mechanism of secret transactions from that of open transactions. In this case, even if the signature of open transactions is attacked, as long as the private key is not leaked, the security of the signature of secret transactions will not be affected. Therefore, **RZcoin can resist the signature forgery attacks**.

6.3. Over-Spending Attack

In RZcoin, the asset information involved in each secret transaction is hidden in the commitment. Except for payer and payee, no node can know the transaction amount. To ensure the validity of secret transactions, we require the payer to provide the legality proof for its hidden asset information. Therefore, a malicious adversary who wants to launch an over-spending attack must be able to forge a legal proof to disguise his unreasonable consumption behavior. The security proof of bulletproof used in this article was given in [36]. Therefore, **RZcoin can resist the over-spending attacks**.

7. Simulation and Performance Evaluation

We use python to simulate RZcoin and test its performance. We will first evaluate the underlying improvements, such as CEs and Szkp. Then we will evaluate the performance of each algorithm in RZcoin from the running time and memory consumption. We will start with the configuration of simulation environment.

7.1. Configuration of Simulation Environment

To facilitate testing, we chose to implement RZcoin's core algorithm on the laptop with Ubuntu 18.04.2 LTS operating system. The hardware and software configuration of the entire test environment is shown in Table 3. In addition, the elliptic curve point multiplication and point addition operations in RZcoin are implemented through the python version of ECDSA algorithm library. The type of elliptic curve we selected is secp256k1, and the parameter configuration of this curve is shown in Table 4. CEs is based on the pypbc library to achieve the pairing operation required for the generation and verification of signatures. For all basic operations in RZcoin, we choose to use the GNU multiple precision operation library (GMP) to ensure the accuracy of large number operations in this mechanism [41]. In addition, all large number operations are implemented in the prime field. For this field, we require the size of the prime to be 256 bits and the same order as the curve secp256k1. To ensure the safety of RZcoin, all random numbers are generated by calling the random algorithm library that comes with python and the precision is set to 256 bits.

Table 3. The Hardware and Software Configuration of RZcoin's Test Environment [41].

| Name | Configuration |
|--------|---|
| CPU | 2.60GHz Intel(R) Core (TM) i7-6500U CPU |
| OS | Ubuntu 18.04.2 LTS |
| RAM | 8 GB |
| python | 3.7.4 |
| pypbc | 2.1.0 |

Table 4. The Parameter Configuration of Curve secp256k1.

| Parameter | Recommended Value |
|-----------|---|
| p | FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFF FFFFFFFF FFFFFFFE FFFFEC2F |
| a | 00000000 00000000 00000000 00000000 |
| b | 00000000 00000000 00000000 00000000 |
| G | 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 |
| N | FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141 |
| h | 01 |

7.2. Evaluation of Improvements

In RZcoin, we mainly give three improvements: (1) We propose CEs to solve the key redundancy problem caused by secret accounts; (2) We use Schnorr to reconstruct the zero-knowledge proof scheme to reduce the size of proof; (3) We used bulletproof to further reduce the size of the transaction. In the previous section, we performed a heuristic analysis of their security. It shows that these improvements improved the security of RZcoin. In this section, we will further evaluate their performance. To facilitate evaluation, the accuracy of the running time of all algorithms is set to six decimal places.

7.2.1. Evaluation of CEs

The signature scheme mainly contains three algorithms, namely KeyGen, SigGen, and SigVer. We will evaluate the running time of these three algorithms and the size of the generated signature. The test results are shown in Table 5, where all the data are the average of the results obtained after 20 tests. Since the signature of CEs is a single value, its size is much smaller than that of ECDSA. Compared to ECDSA, its signature size is reduced by 80%. However, in terms of running time, CEs are not as good as ECDSA. However, the design goal of CEs is to solve the security risks caused by key redundancy. In RZcoin, two different types of accounts use different signature keys and mechanisms. Even if the signature mechanism of the one is attacked, the security of the other account can be guaranteed as long as the private key is not stolen. This setting will improve the security of the account to a certain extent. Since the running time is at the millisecond level, the difference in performance between the two algorithms is not very large. In addition, KeyGen only runs once when the account is created, so it has little impact on the system's daily performance.

Table 5. Comparison of Performance between CEs and ECDSA.

| | CEs | ECDSA |
|---------------------------|----------|----------|
| Time of KeyGen (/s) | 0.010977 | 0.000087 |
| Time of SigGen (/s) | 0.001926 | 0.000452 |
| Time of SigVer (/s) | 0.004263 | 0.001930 |
| Size of Signature (/byte) | 56 | 296 |

7.2.2. Evaluation of Szkp

The purpose of using schnorr signatures to achieve improvements is mainly as follows: (1) Schnorr has higher security than ECDSA. Using it to achieve optimization can further enhance the security of the solution. (2) The size of the signature generated by schnorr is smaller than that of ECDSA. Therefore, this improvement can further reduce the size of transaction. (3) The schnorr signature is linear, so the zero-knowledge proof based on it also has this property. It provides the possibility for subsequent scalability improvements. We will test the proof generation and verification algorithm of Szkp and compare the results with the original scheme ZKP in RZcash. The test results are shown in Table 6, where all the data are the average of the results obtained after 20 tests. It can be seen from Table 6 that the size of proof generated by Szkp is about 2/3 of that of ZKP. In addition, the difference between the two schemes in verification time is not very large.

Table 6. Comparison of Performance between Szkp and ZKP.

| | Szkp(RZcoin) | ZKP(RZcash) |
|-------------------------|--------------|-------------|
| Time of proofGen (/s) | 0.008463 | 0.000532 |
| Time of proofVer (/s) | 0.005645 | 0.002015 |
| Size of zkproof (/byte) | 200 | 304 |

7.2.3. Evaluation of Bulletproof

We use bulletproof to reduce the size of range proofs. Therefore, we mainly focus on the comparison between it and the range proof scheme RP of RZcash in the size of the generated proof. We use N to denote the maximum bit length of the object (ie number) allowed by the range proof. In its performance test, we will gradually adjust N from 2 to 64 to evaluate the influence of N on the size of returned result. The test results are shown in Table 7, where all the data are the average of the results obtained after 20 tests. As shown in Table 7, the proof size of bulletproof is smaller than RP, and as N increases, the difference between them becomes larger and larger. Therefore, bulletproof effectively reduces the size of proof.

Table 7. Comparison of Performance between Bulletproof and RP.

| N | 2 | 4 | 8 | 16 | 32 | 64 |
|----------------------------|------|------|------|-------|-------|-------|
| Bulletproof (/byte) | 1694 | 2046 | 2279 | 2521 | 2760 | 3012 |
| RP (/byte) | 1604 | 3200 | 6376 | 12764 | 25598 | 51134 |

7.3. Performance Evaluation

The above evaluation results of these improvements show that although they increase the running time to a certain extent, the size of generated proof (or signature) is much smaller than that before optimization. In addition, the security of RZcoin has also been improved. Overall, these are some good improvements. However, by simplifying RZcoin's model algorithms, we achieved good performance in the operation of the overall system. The algorithm description for RZcoin is shown in Table 8. In Scane2, payee does not need to perform any operations related to privacy services. He only needs to update the public balance saved locally. Therefore, RZcoin does not contain related algorithms. In the model, we use N to denote the maximum bit length of the object (ie number) allowed by the range proof. In the performance test of RZcoin's algorithm, we will gradually adjust N from 2 to 64 to evaluate the influence of N on the algorithm running time and the size of returned result. These data are the average of the results obtained after 20 tests. To facilitate evaluation, the accuracy of the calculation time of all algorithms is set to six decimal places.

Table 8. The Algorithm Description for RZcoin.

| Name | Function | Caller |
|--------------------------|--|------------|
| <i>AccountGen</i> | used to build the secret account | users |
| <i>AccountVer</i> | used to verify Tx_{init} | full nodes |
| <i>SecretPay</i> | used to build secret transactions | payer |
| <i>SecretVer</i> | used to verify secret transactions | full nodes |
| <i>SecretUpdate</i> | used for local updates | payee |
| <i>open2secretPay</i> | used to construct semi-secret transactions in Scane1 | payer |
| <i>open2secretVer</i> | used to verify semi-secret transactions in Scane1 | full nodes |
| <i>open2secretUpdate</i> | used for local updates | payee |
| <i>secret2openPay</i> | used to construct semi-secret transactions in Scane2 | payer |
| <i>secret2openVer</i> | used to verify semi-secret transactions in Scane2 | full nodes |

Figure 6 shows the test results of the running time of *AccountGen* and *AccountVer* involved in the secret initial transaction. At the same time, for the results returned by *AccountGen* (specifically transactions), Table 9 also shows its memory usage. It can be seen that the running time of these two algorithms is not affected by the change of N . The size of secret initial transactions will also not change as N grows.

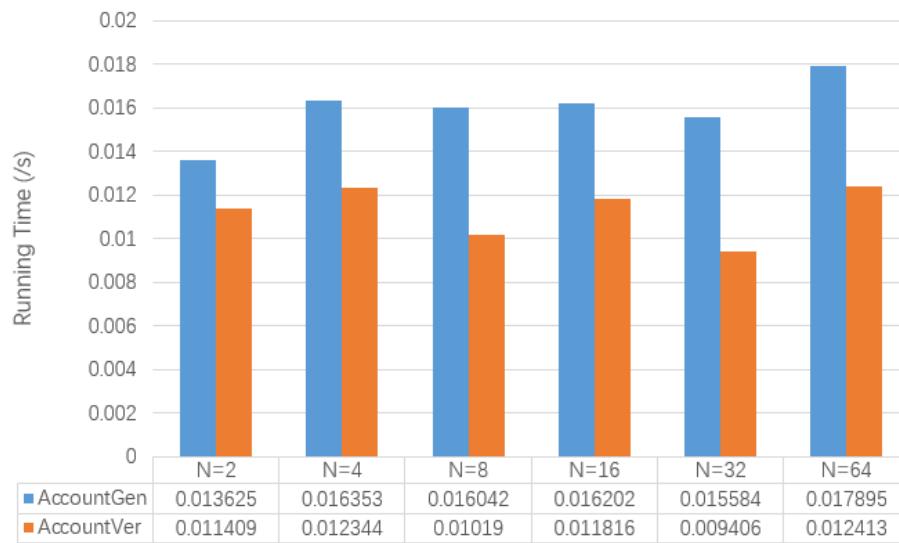


Figure 6. The Comparison of Algorithm Performance of Secret Initial Transactions.

Table 9. The Size of Secret Initial Transactions.

| N | 2 | 4 | 8 | 16 | 32 | 64 |
|--------------------|-----|-----|-----|-----|-----|-----|
| $Tx_{init}(/byte)$ | 450 | 449 | 452 | 450 | 454 | 452 |

Figure 7 shows the test results of the running time of *SecretPay*, *SecretVer* and *SecretUpdate* involved in the secret transaction. At the same time, Table 10 also shows the memory usage of Tx_{pay} . It can be seen that the running time of *SecretPay* and *SecretVer* increases with the increase of N . However, the running time of *SecretUpdate* is not affected by the change of N . The size of secret transactions will gradually increase as N grows.

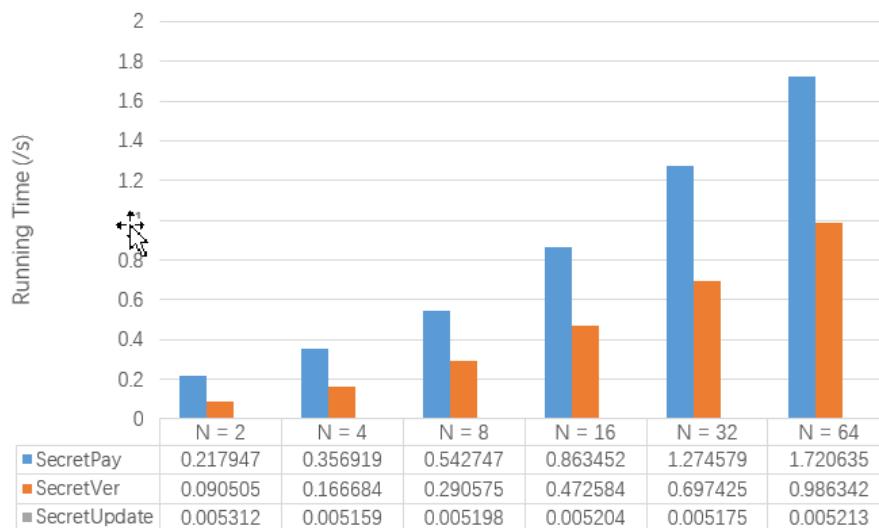


Figure 7. The Comparison of Algorithm Performance of Secret Transactions.

Table 10. The Size of Secret Transactions.

| N | 2 | 4 | 8 | 16 | 32 | 64 |
|-------------------|------|------|------|------|------|------|
| $Tx_{pay}(/byte)$ | 4036 | 4740 | 5206 | 5690 | 6168 | 6672 |

Figure 8 shows the test results of the running time of $open2secretPay$, $open2secretVer$, $open2secretUpdate$, $secret2openPay$ and $secret2openVer$ involved in the semi-secret transaction. At the same time, Table 11 also shows the memory usage of Tx_{pay1} and Tx_{pay2} . The running time of $secret2openPay$ and $secret2openVer$ increases with the increase of N . However, the running time of $open2secretPay$, $open2secretVer$ and $open2secretUpdate$ are not affected by the change of N . In addition, Tx_{pay2} will gradually increase as N grows, but Tx_{pay1} will not change with N .

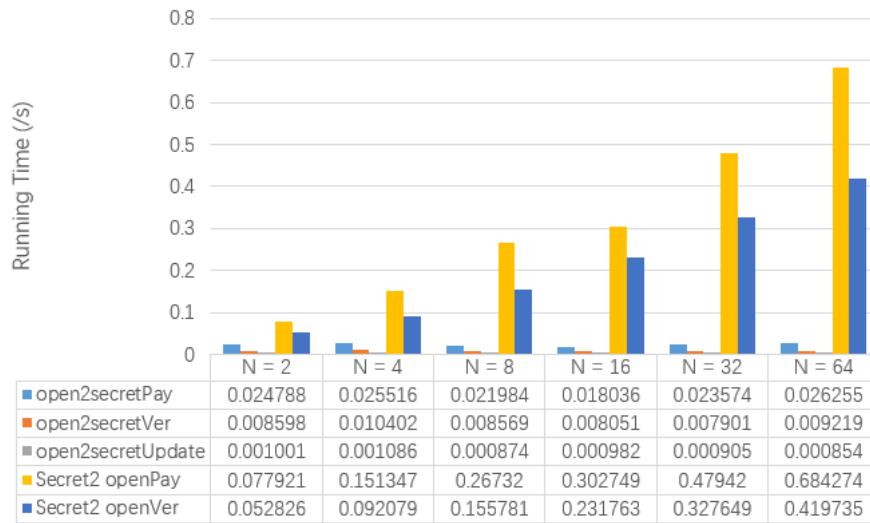


Figure 8. The Comparison of Algorithm Performance of Semi-secret Transactions.

Table 11. The Size of Semi-secret Transactions.

| N | 2 | 4 | 8 | 16 | 32 | 64 |
|---------------------------|------|------|------|------|------|------|
| $Tx_{pay1}(\text{/byte})$ | 890 | 906 | 889 | 894 | 890 | 892 |
| $Tx_{pay2}(\text{/byte})$ | 2344 | 2696 | 2929 | 3171 | 3410 | 3662 |

By summarizing and comparing the above test results, we observe that as long as the payer uses his secret account to initiate transactions, the size of transactions will increase with the growth of N , regardless of the account used by the payee. The running time of the generation algorithm of secret transactions that the payer needs to call and the verification algorithm of secret transactions that the full node needs to call will also show a positive growth trend with the change of N . The running time of all algorithms is in the millisecond level. This shows that RZcoin has good performance.

8. Conclusions

Based on our previous work, called RZcash, we will further improve its core algorithm and propose a decentralized payment called RZcoin that provides the optional privacy protection service. For the key redundancy caused by the secret account, we propose a signature scheme called CEs based on the ciphertext equivalent test scheme. We also use Schnorr signature and bulletproof to reduce the size of proof required for services. We implement the above decentralized payment scheme based on the Ethereum test chain and evaluate its performance. At the same time, we also propose the adversarial model it faces and give the corresponding security analysis. The result shows that the scheme can provide the privacy service to users with good performance and lower communication cost while ensuring security. Next, we will further explore how the privacy protection scheme for smart contracts can be combined with RZcoin.

Author Contributions: X.B. wrote the first draft of the manuscript. All authors contributed equally to this work and approved the submission. Conceptualization, X.B.; Data curation, X.B. and L.W.; Formal analysis, H.Z., X.B. and S.Z.; Funding acquisition, H.Z.; Methodology, X.B.; Project administration, H.Z.; Resources, H.Z.; Software, X.B.; Validation, H.Z., S.Z. and L.W.; Writing—original draft, X.B.; Writing—review and editing, H.Z., S.Z. and L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This work was supported by the Shandong Provincial Key Research and Development Program of China (2018CXGC0701), the National Natural Science Foundation of China (NSFC) (No. 61972050), and in part by the 111 Project (No. B08004).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2019. Available online: <https://bitcoin.org/bitcoin.pdf> (accessed on 15 June 2020).
2. Wood, G. Ethereum: A Secure Decentralized Generalized Transaction Ledger. Ethereum project yellow paper, 151:1-32. 2014. Available online: <https://ljk.imag.fr/membres/Jean-Guillaume.Dumas/Enseignements/ProjetsCrypto/Ethereum/ethereum-yellowpaper.pdf> (accessed on 15 June 2020).
3. Libra Whitepaper. 2019. Available online: <https://libra.org/en-US/white-paper/> (accessed on 15 June 2020).
4. Chaum, D. Blind signatures for untraceable payments. In Proceedings of the Advances in Cryptology: Crypto 82, Santa Barbara, CA, USA, 23–25 August 1982.
5. Dai, W. b-Money. 1998. Available online: www.weidai.com/bmoney.txt (accessed on 15 June 2020).
6. Back, A. Hashcash: A Denial of Service Counter-Measure. In Proceedings of the 2002 USENIX Annual Technical Conference, Monterey, CA, USA, 10–15 June 2002.
7. Finney, H. Reusable Proofs of Work. 2004. Available online: <https://cryptome.org/rpow.htm> (accessed on 15 June 2020).
8. Massias, H.; Avila, X.S.; Quisquater, J.J. Design of a secure timestamping service with minimal trust requirements [C]. In Proceedings of the 20th Symposium on Information Theory in the Benelux, Haasrode, Belgium, 27–28 May 1999.
9. Haber, S.; Stornetta, W.S. How to time-stamp a digital document. *J. Cryptol.* **1991**, *3*, 99–111. [[CrossRef](#)]
10. Bayer, D.; Haber, S.; Stornetta, W.S. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security and Computer Science*; Springer: New York, NY, USA, 1993; pp. 329–334.
11. Haber, S.; Stornetta, W.S. Secure names for bit-strings. In Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland, 1–4 April 1997; pp. 28–35.
12. Merkle, R.C. Protocols for public key cryptosystems. In Proceedings of the 1980 Symposium on Security and Privacy, Oakland, CA, USA, 14–16 April 1980; pp. 122–133.
13. Bai, X.; Wang, L.C.; Zhou, L.J.; Yang, S.; Li, L. RZcash: A Privacy Protection Scheme for the Account-based Blockchain. In Proceeding of the 17th International Conference on Privacy, Security and Trust (PST), Fredericton, NB, Canada, 26–28 August 2019; pp. 1–9.
14. Zhu, H.J. Research on The Ciphertext-Based Equivalent Test Scheme. Ph.D. Thesis, Beijing University of Posts and Telecommunications, Beijing, China, 2018.
15. Luongo, M.; Pon, C.; Cardozo, A.S. Keep. Available online: <https://github.com/keep-network/whitepaper> (accessed on 15 June 2020).
16. Microsoft. CCF. Available online: <https://github.com/microsoft/CCF/blob/master/CCF-TECHNICAL-REPORT.pdf> (accessed on 15 June 2020).
17. Duffield, E.; Diaz, D. Dash: A Payments-Focused Cryptocurrency. Available online: <https://github.com/dashpay/dash/wiki/Whitepaper> (accessed on 15 June 2020).
18. Sun, S.F.; Au, M.H.; Liu, J.K.; Yuen, T.H. RingCT 2.0: A Compact Accumulator-based(Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero. *Eur. Symp. Res. Comput. Secur.* **2017**, *10493*, 456–474.
19. Sasson, E.B.; Chiesa, A.; Garman, C. Zerocash: Decentralized Anonymous Payments from Bitcoin. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 18–21 May 2014.
20. Rivest, R.L.; Shamir, A.; Tauman, Y. How to Leak a Secret: Theory and Applications of Ring Signatures. In *Theoretical Computer Science*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 167–186.

21. Back, A. Bitcointalk, Bitcoins with Homomorphic Value. 2014. Available online: <https://bitcointalk.org/index.php?topic=305791.0> (accessed on 15 June 2020).
22. Maxwell, G. Coinjoin: Bitcoin Privacy for the Real World. 2013. Available online: <https://bitcointalk.org/index.php?topic=279249.0.2013> (accessed on 15 June 2020).
23. Maxwell, G. Confidential Transactions. 2015. Available online: <https://people.xiph.org/~greg/confidential-values.txt.2015> (accessed on 15 June 2020).
24. Goldwasser, S.; Micali, S.; Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **1989**, *18*, 186–208. [CrossRef]
25. Jedusor, T.E. Mimblewimble. 2016. Available online: <https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt> (accessed on 15 June 2020).
26. Gabizon, A.; Zachary, J.W.; Ciobotaru, O. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.* **2019**, *2019*, 953. Available online: <https://eprint.iacr.org/2019/953> (accessed on 15 June 2020).
27. Halo. Available online: <https://github.com/haloplatform> (accessed on 15 June 2020).
28. Maller, M.; Bowe, S.; Kohlweiss, M. Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 2111–2128.
29. Meiklejohn, S.; Mercer, R. Mobius: Trustless Tumbling for Transaction Privacy. *Nephron Clin. Pract.* **2018**, *2018*, 105–121. [CrossRef]
30. Bnz, B.; Agrawal, S.; Zamani, M. Zether: Towards Privacy in a Smart Contract World. *IACR Cryptol. ePrint Arch.* **2019**, *2019*, 191.
31. Zachary, J.; Williamson. The AZTEC Protocol. 2018. Available online: www.aztecprotocol.com/ (accessed on 15 June 2020).
32. Yu, C.; Ma, X. PGC: Pretty Good Confidential Transaction System with Accountability. *IACR* **2019**, *2019*, 319.
33. Li, X. Public-Key Cryptography Based on Elliptic Curve Discrete Logarithm Problem. *Com. Eng. App.* **2002**, *38*, 20–22.
34. Barreto, P.S.L.M.; Naehrig, M. Pairing-friendly elliptic curves of prime order. In *International Workshop on Selected Areas in Cryptography*; Spring: Berlin/Heidelberg, Germany, 2005; pp. 319–331.
35. Torben, P. Pedersen: Non-interactive and information-theoretic secure verifiable secret sharing. In Proceedings of the Advances in Cryptology—CRYPTO 1991, Santa Barbara, CA, USA, 11–15 August 1991; pp. 129–140.
36. Bunz, B.; Bootle, J.; Boneh, D. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy (SP)*; IEEE: Piscataway, NJ, USA, 2018; pp. 315–334.
37. Blum, M.; Feldman, P.; Micali, S. Non-interactive zero-knowledge and its applications (extended abstract). In Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC 1988, Chicago, IL, USA, 2–4 May 1988; pp. 103–112.
38. Bootle, J.; Cerulli, A.; Chaidos, P. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, 8–12 May 2016; pp. 327–357.
39. Maxwell, G.; Poelstra, A.; Seurin, Y. Simple Schnorr multi-signatures with applications to Bitcoin. *Des. Codes Cryptogr.* **2019**, *87*, 2139–2164. [CrossRef]
40. Bernstein, D.J. Multi-User Schnorr Security, Revisited. IACR Cryptology ePrint Archive. Report. 2015. Available online: <http://eprint.iacr.org/2015/996> (accessed on 15 June 2020).
41. GNU Multiple Precision Arithmetic Library. Available online: <https://github.com/alexit/gmpy> (accessed on 15 June 2020).



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).