Hindawi
Security and Communication Networks
Volume 2022, Article ID 7746341, 15 pages
https://doi.org/10.1155/2022/7746341

Research Article

Accountable Monero System with Privacy Protection

Yifan Zhang [1,2,3] **and Haixia Xu** [1,2,3]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, Beijing, China*
[2]*Data Assurance and Communication Security Research Center, Beijing, China*
[3]*School of Cyber Security, University of Chinese Academy of Science, Beijing, China*

Correspondence should be addressed to Haixia Xu; xuhaixia@iie.ac.cn

Received 7 December 2021; Revised 30 January 2022; Accepted 10 February 2022; Published 23 April 2022

Academic Editor: Jie Cui

Copyright © 2022 Yifan Zhang and Haixia Xu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Monero is one of the prominent cryptocurrencies bringing robust privacy safeguard levels. However, for Monero lacking accountability, it is easy to use anonymity and privacy for committing the crime without the leakage of identity. And some exchanges are not very receptive to Monero. The purpose of this study is to balance privacy and accountability in Monero. Specifically, we studied the way to provide accountability while keeping privacy. We develop an accountable Monero model. Our model isolates three kinds of roles, users, the trusted registration authority, and a trusted regulator. Accountability enables the trusted regulator to reveal the signer's identity. Only the trusted regulator can trace users' public keys as needed. We give a construction for the accountable Monero system by combining CryptoNote protocol with ElGamal encryption. Our instantiation is with a marginal influence on efficiency. The security of our scheme is based on the discrete logarithm and decisional Diffie–Hellman assumption.

1. Introduction

Cryptocurrencies are digital currencies using cryptographic primitives to ensure transaction security, control the supply of currency, and verify the ownership of coins. Quite a few cryptocurrencies build decentralized blockchain-based transaction ledgers and run on a peer-to-peer network. Such cryptocurrencies have the benefit of direct transactions among users. Issuing currency and managing transactions are completed by collaborative efforts between peers in the network. Among the decentralized cryptocurrencies, Bitcoin accounts for more than 50% of the entire cryptocurrency market capitalization. Its use of pseudonyms corresponds to Bitcoin address towards protecting users' privacy. Nevertheless, Bitcoin relies on a shared transaction ledger and proof-of-work (PoW) consensus protocol to prevent double-spending. This requires all transactions to be public, linkable, and stored in the append-only ledger. Therefore, once a pseudonym is connected to a user's true identity, all associated transactions undertaken by the pseudonym are revealed. Bitcoin's reliance on pseudonyms to provide anonymity has been severely constrained. Different blockchain analysis techniques such as multisignature addressing techniques and network traffic analyzing techniques can break down user anonymity status [1].

Many researchers show that Bitcoin has various privacy-related weaknesses. A multitude of privacy-enhancing technologies can be classified into two major categories, namely, mixing services and altcoins. In the first, mixing protocols consist of two modes: centralized mixing protocols and distributed mixing networks. The former, such as [2], are to hide the relationship between the user and the coins by a mixer that randomly mixes users' coins. The latter, like [3], enable a set of untrusted users to mix their coins without being reliant on support from a third party. The second category is based on cryptographic technologies, such as Monero [4] using ring signatures and ZeroCash [5] using noninteractive zero-knowledge proofs.

The two types of methods can be used in the construction of anonymous cryptocurrencies to meet the users' demand for privacy protection. However, anonymous cryptocurrencies

have become powerful weapons that help cyber-criminals perform various illicit activities, such as ransomware, tax evasion, black market trade, and money laundering while evading prosecution. As a typical member of the second privacy protection category, Monero was launched in 2014. Privacy and scalability are the main focus of Monero. In privacy protection aspects, the key innovation of Monero is the use of the ring confidential transaction (RingCT) protocol to hide the sender's address and transaction amount, and the employment of stealth address to conceal the receiver's address. Monero has become a medium of exchange on the black market. Three of the top five black markets accept Monero as a means of payment. In addition, Monero is used for paying off ransom. In 2017, the organization behind the WannaCry ransomware tried to convert the bitcoin ransoms received into Monero [6]. Cybercriminals have realized that using bitcoin allows blockchain transactions to expose their identity, so they increasingly demand ransom payments in Monero coins. It also affects the plans of some exchanges to list Monero, increasing the difficulty of exchange between Monero and legal tender and other cryptocurrencies. Moreover, adding accountability will help Monero to be acceptable to law enforcement agencies [7]. We aim to achieve a balance between accountability and privacy in Monero.

1.1. Our Contribution. We propose a new accountable Monero system based on the RingCT protocol. In the system, we add two roles: trusted registration authority and trusted regulator. The trusted registration authority interacts with users to generate certificates and keys for them. The trusted regulator can gain the secret index selected by the transaction payer and can expose the long-term public key of offenders. Anonymity can still be guaranteed except for the tracing of the trusted regulator. We take advantage of cryptographic techniques to design a scheme with privacy and accountability assurance. Specifically, our contributions are as follows:

(1) We propose a new accountable Monero system, which includes users, the trusted registration authority, and the trusted regulator. To make the tracing for a user in the system possible, we employ the certified public key as the user's long-term public key. Our approach combines signature with rerandomizable keys, the signature of knowledge, and the RingCT protocol. The payer encrypts a randomization factor during spending, and the trusted regulator can obtain the payer's one-time public key by decryption. Long-term public keys can be further traced.

(2) The combination with Monero's RingCT protocol has produced the application of a more efficient accountable Monero system. We compare the efficiency and communication in our accountable Monero system, Monero, and the accountable Monero system in [8]. The performance analysis shows that our system has better computational efficiency in the most frequently used settings.

1.2. Related Work. Ring signature schemes generally include one-out-of-many proofs or membership proofs. The RingCT protocol used by Monero [9] is based on the discrete logarithm version of one-out-of-many signature in [10], which convinces a verifier that the signature is from one of n possible independent signers without revealing which one. In addition, an accumulator is a construction tool for ring signatures. Accumulators allow representing a set of elements $Y = \{y_1, y_2, \ldots, y_n\}$ as a single value v whose size is independent of Y's cardinality. For each set element y, there exists an efficiently computable witness w to prove that y is accumulated in v. In 2009, Au et al. [11] proposed the first dynamic universal accumulator. Dynamic accumulators can efficiently add and delete elements to the accumulated value. As the value is renewed, the witness w for each set element can be efficiently updated. The universality means that, for any element $\overline{y}$ outside the set Y, there exists an efficiently computable witness $\overline{w}$ to prove that $\overline{y}$ is not accumulated in v. In 2004, Dodis et al. [12] proposed constant-size ring signatures based on an accumulator of the one-way domain.

In 2004, Liu et al. [13] defined the linkability of ring signatures and proposed the first linkable spontaneous group signature (linkable ring signature) scheme. The linkability refers to the link between two signatures of the same signer. In 2005, Tsang and Wei [14] constructed the first short linkable ring signature scheme extended from a short ring signature scheme proposed by Dodis et al. [12]. In 2019, Wang et al. [15] proposed a general construction method to add linkability to arbitrary ring signature schemes with one-time signatures.

In 2007, Fujisaki and Suzuki [16] proposed a traceable ring signature. The traceable ring signature contains a tag composed of a list of ring members and an issue that refers to an affair. Anyone who signs the same message again with the same tag can be linked, whereas anyone who signs two different messages with the same tag can be traced. In 2013, Au et al. [17] proposed the first ID-based event-oriented linkable ring signature, which provided a revocable option. Two signatures with the same secret key in the same event can reveal the signer's identity. The traceability of these schemes is under linkable conditions. In these schemes, there is no way to revoke the anonymity of signers whose transactions are related to criminal behavior but are consistent with transaction generation rules. Bootle et al. [18] take care of this type of requirement in their work. They proposed an accountable ring signature scheme, which is based on the efficient one-out-of-many knowledge proof proposed in [19] and improved it. The signer can specify a public key, allowing the owner of it to revoke the signer's anonymity.

Using ring signature and one-time keys, CryptoNote for Monero conceals the signers and receivers of transactions. Extended by CryptoNote, RingCT protocol can allow the hiding of the amounts sent in a transaction. In 2017, Sun et al. [20] gave a rigorous security definition for RingCT protocol and proposed a new RingCT protocol (RingCT 2.0). Based on homomorphic commitment schemes, accumulators with one-way domain, and the signature of knowledge, RingCT 2.0 protocol constructs a constant size signature

independent of the number of input accounts. It requires trusted public parameters. In 2020, Yuen et al. [21] proposed RingCT 3.0, which does not require trusted settings. The ring signature size is the logarithm of the input account number.

In the matter of Monero's accountability, there are not many studies currently. In 2019, Li et al. [8] proposed a traceable Monero system based on RingCT 2.0 and a variation of the ElGamal public key encryption scheme. The scheme allows a trusted regulator to trace the user's one-time public key and long-term public key. Our solution is based on the RingCT protocol in Monero, certified key scheme [22] and the variant ElGamal encryption scheme. Consider that, since version v8 (2018-10), Monero uses a ring size of 11, and the latest recommended ring size is 15. Our scheme's computation efficiency is independent of the length of the element in $\mathbb{Z}_q$, which is 255 in CryptoNote 2.0. Therefore, when the ring size is less than the length of the element, our scheme is more efficient.

In terms of certified keys used in signature schemes, in 2007, Groth [23] constructed a group signature scheme with a certified signature scheme, which allows users to pick signature verification keys. The verification keys can be certified by a certification authority. In 2014, Ateniese et al. [22] proposed certified Bitcoin. By using a self-certified public key, a user can efficiently generate the certified address. The scheme is fully compatible with the Bitcoin system.

2. Definition of Accountable Monero System

An accountable Monero system is a tuple of algorithms (Setup, TKGen, RKGen, UKGen, Mint, Spend, Verify, BlockGen, BlockVer, Trace, Judge).

$(pp) \leftarrow$ Setup (1^λ): Given the security parameter λ, it produces the public parameter pp.

$(x_T, y_T) \leftarrow$ TKGen (pp): Given the public parameter pp, it produces a public-private key pair (x_T, y_T) for the trusted registration authority.

$(x_R, y_R) \leftarrow$ RKGen (pp): Given the public parameter pp, it produces a public-private key pair (x_R, y_R) for the trusted regulator.

$(sk, P) \leftarrow$ UKGen (pp, y_T): Given the public parameter pp and the public key of the trusted registration authority, it produces a signing key sk and a verification key P for the user.

$(act, ask) \leftarrow$ Mint (P, a): Given a public key and an amount a, it produces a coin cn with amount a for a public key P. The corresponding coin key is ck. The coin cn and the public key P form an account denoted by $act = (pk, cn)$, and the corresponding account secret key is defined as $ask = (sk, ck)$.

$(tx, I, CT, \pi, aux) \leftarrow$ Spend $(m, K_{in}, A_{in}, A, P_{out}, y_R)$: The inputs include a description $m \in \{0,1\}^*$, a set of account private keys K_{in}, and input account set A_{in}, accounts A including A_{in}, the output stealth public keys P_{out}, and the trusted regulator's public key y_R. This algorithm outputs a transaction message tx, a set of serial numbers I, a proof π, a ciphertext CT, and some additional information aux.

$(0/1) \leftarrow$ Verify (tx, I, CT, π, aux): Given the transaction message tx, serial numbers I, the proof π, the ciphertext CT, and additional information aux, this algorithm checks, the transaction is valid or not and outputs 1 or 0, respectively.

$(non, blk_n) \leftarrow$ BlockGen (TX, blk_{n-1}): Given a set of transactions $\{TX = (tx, I, CT, \pi, aux)\}$ and the previous block blk_{n-1}, this algorithm outputs a nonce non and a block blk_n.

$(0/1) \leftarrow$ BlockVer (blk_n): Given the current block blk_n, this algorithm verifies whether the block is valid or not and outputs 1 or 0, respectively.

$(P, \psi) \leftarrow$ Trace (x_R, TX, CT): Given the trusted regulator's secret key x_R, the transaction TX, and the ciphertext CT, it outputs the payer's public key P and a trace proof ψ.

$(0/1) \leftarrow$ Judge (P, ψ, y_R, CT): Given the payer's public key P, the trace proof ψ, trusted regulator's public key y_R, and the ciphertext CT, this algorithm checks whether the trace proof is valid or not and outputs 1 or 0, respectively.

An accountable Monero system should be correct, balanced, anonymous, nonslanderous, and traceable as in [8, 20].

Definition 1 (Perfect Correctness). *All transactions generated by the Spend algorithm can pass the Verify algorithm. All traced public keys and corresponding proofs generated by the Trace algorithm can be verified by the Judge algorithm. Specifically, an accountable Monero system is perfectly correct if for all PPT adversaries $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l} Verify(tx, I, \\ CT, \pi, aux) \\ = 1 \\ Judge(P, \psi, \\ y_R) = 1 \end{array} : \begin{array}{c} (pp) \leftarrow Setup\left(1^\lambda\right); \\ (x_T, y_T) \leftarrow TKGen(pp); \\ (x_R, y_R) \leftarrow RKGen(pp); \\ (m, A, P_{\text{out}}) \leftarrow \mathcal{A}\left(pp, A_{in}, K_{in}\right) \\ \text{where } (A_{in}, K_{in}) = (act, ask) s.t. \\ (sk, P) \leftarrow UKGen\left(pp, y_T\right); \\ (act, ask) \leftarrow Mint\,(P, a); \\ (tx, I, CT, \pi, aux) \leftarrow Spend \\ \left(m, K_{in}, A_{in}, A, P_{\text{out}}, y_R\right); \\ (P, \psi) \leftarrow Trace\left(sk_R, tx, CT\right) \end{array}\right] = 1. \tag{1}$$

Definition 2 (Balance). *This property requires that any malicious payers cannot (1) spend by using accounts that are not their own; (2) spend more than the amount in their accounts. Specifically, an accountable Monero system is balanced if for all PPT adversaries $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} pp \leftarrow Setup\left(1^{\lambda}\right); \\ \mathscr{A}\text{wins: } \left(\{act_i'\}_{i=1}^{\mu}, \{\mathcal{S}_i\}_{i=1}^{\nu}\right) \leftarrow \\ \mathscr{A}^{\text{AddGen,ActGen,Spend,Corrupt}}(pp) \end{array}\right] \leq negl(\lambda). \quad (2)$$

where the oracles are defined as follows:

(i) *AddGen* (i)*: Given a query number i, if algorithm TKGen has not been run, runs* $(x_T, y_T) \leftarrow$ *TKGen* (pp)*. Then, it runs algorithm* $(sk_i, P_i) \leftarrow$ *UKGen* (pp, y_T) *and returns a public key* P_i*.*

(ii) *ActGen* (i, a_i)*: Given the index i and an amount* a_i*, it runs algorithm* $(act, ask) \leftarrow$ *Mint* (P_i, a_i)*, adds i,* $act_i = (P_i, cn_i)$ *to the lists* $\mathcal{I}$*,* $\mathcal{G}$*, respectively and returns* (cn_i, ck_i) *for* P_i *generated by AddGen. The corresponding private key is* $ask_i = (sk_i, ck_i)$*. The oracle uses* ask_i *to compute the serial number* I_i *of the account* act_i *and adds it to the list* $\mathcal{S}$*.*

(iii) *Spend* $(m, A_{in}, A, P_{\text{out}})$*: Given the inputs, if algorithm RKGen has not been run, it runs algorithm* $(x_R, y_R) \leftarrow$ *RKGen* (pp)*. Then, it runs algorithm* $(tx, I, CT, \pi, aux) \leftarrow$ *Spend* $(m, K_{in}, A_{in}, A, P_{\text{out}}, y_R)$*. Assume that at least one account in* A_{in} *has not been corrupted, where* $A_{in} \subset \mathcal{G}$*. It adds* (tx, I, CT, π, aux) *to the list* $\mathcal{T}$ *and returns it.*

(iv) *Corrupt* (i)*: Given the index i, it returns* sk_i *and uses* ask_i *to determine the serial number* I_i *of account* act_i*, adds* I_i*,* (I_i, a_i) *to the lists* $\mathcal{C}$ *and* $\mathcal{B}$ *respectively.*

Finally, $\mathscr{A}$ *outputs all his spending and some new accounts he created* $(act_1', \ldots, act_{\mu'}', \mathcal{S}_1, \ldots, \mathcal{S}_\nu)$*, where* $\mathcal{S}_i = (tx_i, \mathcal{I}_i, CT_i, \pi_i, aux_i), i \in [\nu]$ *and all transactions are paid to the challenger. Specifically, we call* $\mathscr{A}$ *wins if the following conditions are met:*

(1) *Verify* $(tx_i, I_i, CT_i, \pi_i, aux_i) = 1$*, where* $i \in [\nu]$*.*

(2) $\mathcal{S}_i \notin \mathcal{T} \wedge I_i \subset \mathcal{S}, i \in [\nu]$*, and* $I_j \cap I_k = \varnothing$ *for any different* $j, k \in [\nu]$*.*

(3) *Let* $I_i = \{I_{i,j}\}$ *and* $E = \cup_{i=1}^{\nu}\{a_{i,j}: (I_{i,j}, a_{i,j}) \in \mathcal{B} \wedge I_{i,j} \in (I_i \cap \mathcal{C})\}$*, it holds that* $\sum_{a_{i,j} \in E} a_{i,j} < \sum_{k=1}^{\nu} a_{\text{out}}^i$*, where* a_{out}^i *denotes the balance of the output accounts in* $\mathcal{S}_i$*.*

Definition 3 (Anonymity). *The property requires that the accounts used by the payer cannot be distinguished. Specifically, an accountable Monero system is anonymous if for all PPT adversaries* $\mathscr{A}$*,*

$$\left|\Pr\left[b' = b: \begin{array}{c} pp \leftarrow \text{Setup}\left(1^{\lambda}\right); \\ \left(m, A_{s_0}, A_{s_1}, A, P_{\text{out}}\right) \leftarrow \\ \mathscr{A}_1^{\text{AddGen,ActGen,Spend,Corrupt,Trace}}(pp); \\ b \leftarrow \{0,1\}, \\ \left(tx^*, I^*, CT^*, \pi^*, aux^*\right) \leftarrow \\ \text{Spend}\left(m, K_{s_b}, A_{s_b}, A, P_{out}, y_R\right); \\ b' \leftarrow \mathscr{A}_2^{\text{Spend,Corrupt,Trace}} \\ \left(pp, \left(tx^*, \pi^*, I^*, CT^*, aux^*\right)\right) \end{array}\right] - \frac{1}{2}\right| \leq negl(\lambda). \quad (3)$$

The oracles are defined as before, $A_{s_i} \in A$ *and* $A_{s_i} \subset \mathcal{G}$*, where* $i \in \{0, 1\}$*. Trace oracle is defined as follows:*

(i) *Trace* (tx, π)*. Given the queried transaction* tx *and the proof* π*, it returns a public key P. If the input is* (tx^*, π^*)*, returns* $\perp$*.*

Besides, the following conditions should be satisfied:

(1) *None of the accounts in* A_{s_0} *and* A_{s_1} *have been corrupted.*

(2) *No query in the form of* $(\cdot, A_s, \cdot, \cdot) s.t. A_s \cap A_{s_i} \neq \varnothing$ *has ever been queried to Spend oracle.*

Definition 4 (Nonslanderability). *The property requires that a malicious user cannot slander any honest user. Specifically, an accountable Monero system is nonslanderous if for all PPT adversaries* $\mathscr{A}$*,*

$$Pr\left[\mathscr{A}\text{ Wins: } \begin{array}{c} pp \leftarrow Setup\left(1^{\lambda}\right); ((\hat{tx}, \hat{I}, \widehat{CT}, \\ \hat{\pi} \\ \mathscr{A}^{\text{AddGen,ActGen,Spend,Corrupt}}(pp) \end{array} , \widehat{aux}), (\tilde{tx}, \tilde{I}, \widetilde{CT}, \tilde{\pi}, \widetilde{aux})) \leftarrow\right] \leq negl(\lambda). \quad (4)$$

where all oracles are defined as before, and $(\widehat{tx}, \widehat{I}, \widehat{CT}, \widehat{\pi}, \widehat{aux})$ *is one of the outputs of Spend oracle. Specifically,* $\mathcal{A}$ *wins if the following conditions are met: (1)Verify* $(\widetilde{tx}, \widetilde{I}, \widetilde{CT}, \widetilde{\pi}, \widetilde{aux}) = 1$; (2) $(\widetilde{tx}, \widetilde{I}, \widetilde{CT}, \widetilde{\pi}, \widetilde{aux}) \notin \mathcal{T}$; *(3)* $\widehat{I} \cap \mathcal{C} = \emptyset$ *but* $\widehat{I} \cap \widetilde{I} \neq \emptyset$.

The nonslanderability property already covers the linkability property [20].

Definition 5 (Traceability). *The property requires that the trusted regulator can trace the transaction payer and can prove the validity of tracing. Specifically, an accountable Monero system is traceable if for all PPT adversaries* $\mathcal{A}$,

$$Pr\left[\begin{array}{l} Trace(sk_R, tx, \\ CT) = (P, \psi) \\ Judge(P, \psi, \\ pk_R, CT) = 1 \end{array} : \begin{array}{c} (pp) \leftarrow \text{Setup}(1^\lambda); \\ (tx, I, CT, \pi, aux) \leftarrow \\ \mathcal{A}^{\text{AddGen,ActGen,Spend,Corrupt}}(pp); \\ Verify(tx, I, CT, \pi, aux) = 1 \end{array}\right] = 1. \tag{5}$$

3. Preliminaries

Define $[z] = \{1, 2, \ldots, z\}$. We use PPT, which stands for "probabilistic polynomial-time." "‖" denotes the concatenation of strings. Besides, we recall here the tools and assumptions used throughout this paper.

3.1. Terminology. Monero uses terms according to [24] as follows:

(i) q is a prime number and the order of the underlying finite field;

(ii) $\mathbb{Z}_q$ is the underlying finite filed;

(iii) E is an elliptic curve defined on $\mathbb{Z}_q$;

(iv) $\mathbb{G} \subseteq E(\mathbb{Z}_q)$ is a subgroup of large prime order of the elliptic curve E over a finite field $\mathbb{Z}_q$.

(v) G is a generator of $\mathbb{G}$;

(vi) l is a prime order of G.

Private ec-key is a standard elliptic curve private key, such as $a \in [1, l-1]$;

Public ec-key is a standard elliptic curve public key, such that $A = aG$.

3.2. Hardness Assumptions

Discrete logarithm (DL) Assumption Let $(l, \mathbb{G}, G)$ be a tuple, where $|\mathbb{G}| = l$ and $G \in \mathbb{G}$ such that $<G> = \mathbb{G}$. Given a tuple $(G, aG) \in G^2$, where $a \in \mathbb{Z}_q^*$, for all PPT adversaries $\mathcal{A}$, the probability of computing a is negligible in security parameter λ, which can be expressed as $Adv_{\mathcal{A}}^{DL}(\lambda) = P[\mathcal{A}(G, aG) = a] \leq negl(\lambda)$

Decisional Diffie–Hellman (DDH) Assumption Let $(l, \mathbb{G}, G)$, where $|\mathbb{G}| = l$ and $G \in \mathbb{G}$ such that $<G> = \mathbb{G}$. The DDH assumption is as follows: given a tuple $(G, aG, bG, ((1-x)ab + xc)G)$, where $a, b, c \in \mathbb{Z}_q^*$, $x \in \{0, 1\}$, for all PPT adversaries $\mathcal{A}$,

$$Adv_{\mathcal{A}}^{DDH}(\lambda) = |P[\mathcal{A}(G, aG, bG, abG)] - P[\mathcal{A}(G, aG, bG, cG)]| \leq negl(\lambda). \tag{6}$$

3.3. Homomorphic Commitment Scheme [19]. A noninteractive commitment scheme includes a pair of polynomial-time algorithms $(CGen, Com_{ck})$: on inputting a security parameter λ, $CGen(1^\lambda)$ outputs a commitment key ck. $Com_{ck}(m, r)$ generates a commitment C on the input message m and a randomness r, where ck is often omitted when it is clear in context.

Security of HCom A noninteractive homomorphic commitment scheme $H\text{Com} = (C\text{Gen}, \text{Com})$ is secure if it satisfies the following properties:

Binding This property means that a committer cannot generate a commitment C, which can be opened as two different messages. More precisely, for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[m_0 \neq m_1 \wedge Com(m_0; r_0) = Com(m_1; r_1) : ck \leftarrow CGen(1^\lambda); (m_0, r_0, m_1, r_1) \longleftarrow \mathcal{A}(ck)\right] \leq negl(\lambda). \tag{7}$$

Hiding This property means that the commitment reveals nothing about m. More precisely, for all PPT adversaries $\mathcal{A}$,

$$\left|\Pr\left[\begin{array}{c} ck \leftarrow CGen\left(1^{\lambda}\right); \\ \mathcal{A}(C) = b\text{: } (m_0, m_1) \leftarrow \mathcal{A}(ck); \\ b \leftarrow \{0,1\}; C \leftarrow \mathrm{Com}_{ck}(m_b) \end{array}\right] - \frac{1}{2}\right| \leq negl(\lambda). \tag{8}$$

Homomorphic This property means that for all $\lambda \in \mathbb{N}, ck \leftarrow CGen(1^{\lambda}), m_0, r_0, m_1, r_1 \in \mathbb{Z}_q$, it holds that

$$\mathrm{Com}(m_0; r_0) \cdot \mathrm{Com}(m_1; r_1) = \mathrm{Com}(m_0 + m_1; r_0 + r_1). \tag{9}$$

3.4. Elliptic Curve ElGamal Encryption [25]. The EC-ElGamal encryption scheme consists of three algorithms.

$(x, P) \leftarrow$ KeyGen (1^{λ}): Given a security parameter λ, this algorithm generates $gk = (l, \mathbb{G}, G)$, where $|\mathbb{G}| = l$ and $G \in \mathbb{G}$ such that $<G> = \mathbb{G}$. The private key $x \in [0, l-1]$ is chosen uniformly at random. The public key is computed as $P = xG$. It outputs the key pair (x, P).

$C \leftarrow$ Enc (P, m): Given a public key P and a message m, which has been embedded as a point M on the elliptic curve E over $\mathbb{Z}_q$, it chooses a random α and outputs the ciphertext $C = (C_1, C_2) = (\alpha G, \alpha(xG) + M)$.

$M \leftarrow$ Dec (x, C): Given the private key x and the ciphertext $C = (C_1, C_2)$, it outputs $M = C_2 - xC_1$.

The EC-ElGamal encryption is IND-CPA secure under the DDH assumption. And the EC-ElGamal ciphertexts are homomorphic.

3.5. Signatures with Rerandomizable Keys [26]. This primitive can consistently rerandomize the private and public keys of a signature scheme.

3.5.1. Signatures with Rerandomizable Keys. A digital signature scheme with rerandomizable keys contains the following PPT algorithms:

$(sk, vk) \leftarrow$ SGen (1^{λ}): Given the security parameter λ, this algorithm generates a public-private key pair (sk, vk).

$\sigma \leftarrow$ Sig (sk, m): Given a signing key sk and a message m, this algorithm generates a signature σ.

$b \leftarrow$ Ver (vk, m, σ): Given a verification key vk, a message m, and a signature σ, this algorithm outputs a bit b.

$sk' \leftarrow$ RndSK (sk, ρ): Given a signing key sk and a random number ρ, this algorithm outputs a new signing key sk'.

$vk' \leftarrow$ RndVK (vk, ρ): Given a verification key vk and a random number ρ, this algorithm outputs a new verification key vk'.

If for all $\lambda \in \mathbb{N}$, all key-pairs $(sk, vk) \leftarrow SGen(1^{\lambda})$, all messages m, $\mathrm{Ver}(vk, m, Sig(sk, m)) = 1$ and for all ρ, $\mathrm{Ver}(RndVK(vk, \rho), m, Sig(RndSK(sk, \rho), m)) = 1$, this scheme is complete. The formal definition is in the following.

3.5.2. Rerandomizable Keys. A signature scheme has perfectly rerandomizable keys if for all $\lambda \in \mathbb{N}$, all key-pairs $(sk, vk)) \leftarrow SGen(1^{\lambda})$ and a uniformly chosen random number ρ, the following distributions are identical:

$$(sk, vk, sk', vk') = (sk, vk, sk'', vk'') \tag{10}$$

where $(sk', vk') \leftarrow SGen(1^{\lambda})$, $sk'' \leftarrow RndSK(sk, \rho)$, and $vk'' \leftarrow RndVK(vk, \rho)$.

3.5.3. Unforgeability under Rerandomizable Keys. We say that a signature scheme achieves unforgeability under rerandomizable keys if all PPT adversary $\mathcal{A}$ cannot be able to win the following game with nonnegligible probability:

(1) The challenger runs $(sk, vk) \leftarrow SGen(1^{\lambda})$ and sends vk to $\mathcal{A}$.

(2) $\mathcal{A}$ is allowed to access the signing oracle and randomized signing oracle to make signing queries. A signing query is for the input $(m, \perp)$, where m is the message. The challenger responds with $Sig(sk, m)$. A randomized signing query is for the input (m, ρ), where m is the message, and ρ is randomness. The challenger responds with $Sig(RndSK(sk, \rho), m)$.

(3) $\mathcal{A}$ outputs a tuple (m^*, σ^*, ρ^*).

We say that $\mathcal{A}$ wins the game if:

(i) For all queries $(m, \cdot)$ to the signing oracle or randomized signing oracle, it holds that $m \neq m^*$;

(ii) Either $\mathrm{Ver}(vk, m^*, \sigma^*) = 1$ or $\mathrm{Ver}(RndVK(vk, \rho^*), m^*, \sigma^*) = 1$.

3.6. Signature of Knowledge [27]. A Signature of Knowledge (SoK) for an NP-relation $\mathcal{R}$ is a tuple of algorithms (Setup, Sign, Verify) as follows:

$par \leftarrow$ Setup (1^λ): Given a security parameter λ, it outputs public parameters par.

$\sigma_{SoK} \leftarrow$ Sign (par, s, w, m): Given the public parameter par, a statement s, a witness w, and a message m, where $(s, w) \in \mathscr{R}$, it outputs a signature σ_{SoK} on m.

$(0/1) \leftarrow$ Verf $(par, s, m, \sigma_{SoK})$: Given the public parameter par, a statement s, a message m, and a signature σ_{SoK}, it outputs 1 if σ_{SoK} is a valid signature or 0 otherwise.

The security definition for SoK (SimExt-secure) requires correctness, simulatability, and extractability.

3.6.1. Correctness. For any message m and a pair $(s, w) \in \mathscr{R}$, such that

$$Pr\left[\begin{matrix} Verf(par, s, \\ m, \sigma_{SoK}) = 1 \end{matrix} : \begin{matrix} par \leftarrow Setup\left(1^\lambda\right); \sigma_{SoK} \\ \leftarrow Sign(par, s, w, m) \end{matrix}\right] \geq 1 - negl(\lambda). \tag{11}$$

3.6.2. Simulatability. There exists PPT simulator Sim=(SimGen, SimSign), where SimGen (1^λ) outputs public parameters par and a trapdoor τ and SimSign (par, τ, s, m) outputs a signature σ_{SoK}, such that for all PPT adversaries $\mathscr{A}$,

$$\begin{aligned} &\left|\Pr\left[b = 1 : (\text{par}, \tau) \leftarrow SimGen\left(1^\lambda\right); b \leftarrow \mathscr{A}^{Sim}(\text{par})\right] - \right. \\ &\left.\Pr\left[b = 1 : \text{par} \leftarrow Setup\left(1^\lambda\right); b \leftarrow \mathscr{A}^{Sign}(\text{par})\right]\right| \leq negl(\lambda) \end{aligned}. \tag{12}$$

3.6.3. Extractability. For all PPT adversaries $\mathscr{A}$, there exists a polynomial-time extractor Ext, such that

$$\Pr\left[\begin{matrix} (s, w) \in \mathscr{R} \vee (s, m) \\ \in Q \vee Verf(\text{par}, \\ s, m, \sigma_{SoK}) = 0 \end{matrix} : \begin{matrix} (par, \tau) \leftarrow SimGen\left(1^\lambda\right) \\ (m, s, \sigma_{SoK}) \leftarrow \mathscr{A}^{\text{Sim}}(\text{par}) \\ w \leftarrow \text{Ext}\left(par, \tau, m, s, \sigma_{SoK}\right) \end{matrix}\right] \geq \begin{matrix} 1 - \\ negl(\lambda) \end{matrix}. \tag{13}$$

where Q denotes the queries to the SimSign oracle that $\mathscr{A}$ made.

4. Accountable Monero System

In this section, we present the procedure of the accountable Monero and describe the concrete scheme. Then, we give an instantiation of the SoK used in the system and the security analysis.

4.1. Accountable Monero System Workflow. Our accountable Monero system adds accountability to the original Monero system. It includes three types of subjects, namely, the trusted registration authority, denoted by T, the trusted regulator, denoted by R, and users. As shown in Figure 1, the main processes are *Registration*, *Transaction*, *Transaction on the chain*, and *Tracing*.

Registration: To meet the demand of the trusted regulator tracing users' long-term public keys, a user has to cooperate with the trusted registration authority to create the certificate and key pair. The user's signing public key and certificate can be accessed in the system. It allows others to make a transaction with him.

Transaction: The payer wants to transfer coins to others. Firstly, the payer produces the one-time public key for each payee. At the same time, the payer encrypts the necessary message with the trusted regulator's public key to support the tracing of payees' long-term public keys. Secondly, the payer mints coins for each payee. Thirdly, the payer randomizes his one-time keys, uses the trusted regulator's public key to encrypt the random number used in randomization, and signs the transaction. Finally, he broadcasts the transaction to the P2P network.

Transactions on the chain: Miners collect and verify new transactions in the P2P network. Then, they compete for a proof-of-work to add their block to the blockchain.

Tracing: When tracing the one-time public key, the trusted regulator decrypts the ciphertext in the transaction to obtain the random number. Then, the regulator can recover the one-time public key from the randomized public key. Obtaining the one-time public key, the trusted regulator can decrypt the ciphertext generated along with the one-time public key to reveal the long-term public key.

4.2. Accountable Monero System Instantiation. Our accountable Monero system is designed as follows:

Setup (1^λ): This algorithm generates the public parameters of the system. On inputting a security parameter λ, $H, \tilde{H}, U$ are generators in $\mathbb{G}$ and their discrete logarithms regarding G are unknown. We denote par as par = SoK.Setup (1^λ). It outputs $pp = (1^\lambda, par, G, H, \tilde{H}, U, \mathscr{H}_s, \mathscr{H}_p, \xi)$, where $\mathscr{H}_s$: $\{0,1\}^* \longrightarrow \mathbb{Z}_q$ and $\mathscr{H}_p$: $\{0,1\}^* \longrightarrow \{0,1\}^*$ denote two cryptographic secure hash functions, and ξ is a fixed function from $\mathbb{G}$ to $\mathbb{Z}_q$.

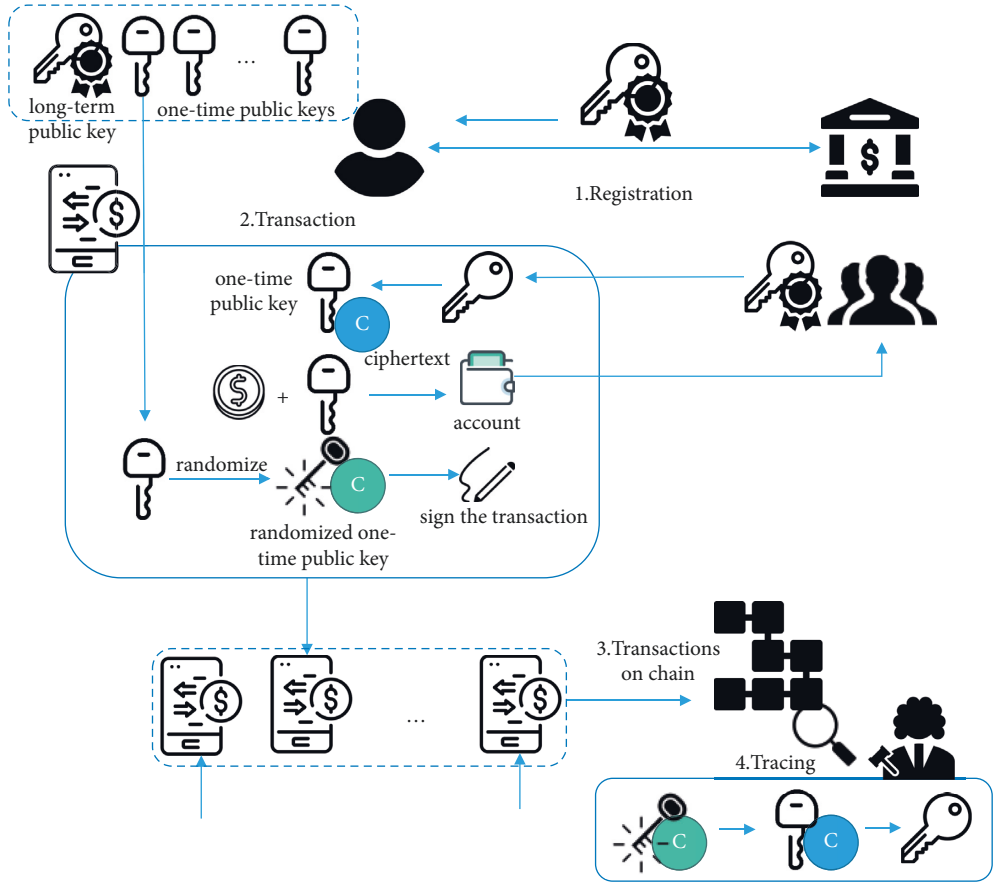

Figure 1: Workflow of our accountable Monero system.

TKGen (pp): This algorithm generates the trusted registration authority's key pair. Given input pp, it outputs (x_T, Y_T), where $x_T \in [0, l-1]$ and $Y_T = x_T G$.

RKGen (pp): This algorithm generates the trusted regulator's key pair. Given input pp, it outputs (x_R, Y_R), where $x_R \in [0, l-1]$ and $Y_R = x_R U$.

UKGen (pp, Y_T): This algorithm generates the user's public key. We use the certified key [22] generation method.

A certified key is certified by the trusted registration authority. The Certified Key Generation protocol is executed between user U and the registration authority T as follows:

(1) U picks uniformly at random k in $[0, l-1]$, computes $Z = kG$ and sends Z to T.

(2) On receiving Z from user U, T picks uniformly at random k' in $[0, l-1]$ and computes $C_u = Z + k'G$, $e = \xi(C_u), \overline{x} = k' + e \cdot x_T$. Finally, T sends C_u, $\overline{x}$ to user U.

(3) On receiving $C_u, \overline{x}$ from T, U computes $x = \overline{x} + k$ as his private key corresponding to his certified public key $Y_u = C_u + \xi(C_u) Y_T$ and verifies that

$$xG = C_u + \xi(C_u) Y_T. \tag{14}$$

The user's public key Y_u and certificate C_u are publicly accessed. A payer obtains the payee's long-term public key $Y_u = xG$, where x is the payee's long-term private key. The payer generates a one-time public key as a receiving public key for the payee, which can hide the payee's real public key. In the process of generating a one-time public key, to simplify, we use one of the methods in [28]. The payer firstly chooses uniformly at random $r \in [0, l-1]$ and computes $R = rG$. The payee's one-time public key is computed as $P = \mathcal{H}_s(rY_u)G + Y_u$, and the corresponding one-time private key is $x + \mathcal{H}_s(xR)$. The payer picks uniformly at random $\tilde{r} \in [0, l-1]$ and then encrypts $\mathcal{H}_s(rY_u)G$ with the trusted regulator's public key Y_R as $Ct_1 = \tilde{r}Y_R, Ct_2 = \mathcal{H}_s(rY_u)G + \tilde{r}U$. The ciphertext is denoted as $CT_1 = (Ct_1, Ct_2)$.

And it produces SoK as follows:

$$SoK\{(C_u, Y_u, r, \tilde{r}): Y_u = C_u + \xi(C_u) Y_T \wedge P = Y_u + \mathcal{H}_s(rY_u)G \wedge R = rG \wedge Ct_1 = \tilde{r}Y_R \wedge Ct_2 = \mathcal{H}_s(rY_u)G + \tilde{r}U\}. \tag{15}$$

Mint (P, a): This algorithm mints a coin for the public key address P with an amount a as follows. It chooses uniformly at random $r \in [0, l-1]$ and computes the commitment $C = \mathrm{Com}(ck, a; r) = rG + aH$ and then outputs $(Cn, ck) = (C, (r, a))$. The account is denoted

as $act = (P, C)$ and the corresponding account private key is denoted as $ask = (sk, ck) = (sk, (r, a))$.

Spend(m $, K_{in}, A_{in}, A, P_{out}, Y_R$): This algorithm allows users to spend their coins while maintaining privacy. in is a secret index corresponding to the signer of the ring. Given the transaction message $\mathbf{m} \in \in \{0,1\}^*$, account private keys K_{in} and the corresponding account set A_{in}, accounts A which contain A_{in}, a set of output public key P_{out} and the trusted regulator's public key Y_R, it outputs a transaction, which contains a proof π, randomized spending public keys P^* and serial numbers I associated with A_{in}. Without loss of generality, we denote $K_{in} = ask_{in,j} = \left\{sk_{in,j}, (r_{in,j}, a_{in,j})\right\}_{j\in[m]}$, $A_{in} = act_{in,j} = \left\{P_{in,j}, Cn_{in,j}\right\}_{j\in[m]}$, $A = act_{i,j} = \left\{P_{i,j}, Cn_{i,j}\right\}_{i\in[n], j\in[m]}$ and the output public key set $P_{\text{out}} = \left\{P_{\text{out},k}\right\}_{k\in[t]}$. The process of spending coins is as follows:

(1) The payer sets the output amount $a_{out,k}\,(k \in [t])$ for each output public key $P_{\text{out},k}\,(k \in [t])$ such that the input sum is equal to the output sum, i.e., $\sum_{j=1}^{m} a_{in,j} = \sum_{k=1}^{t} a_{out,k}$. Then, the payer chooses uniformly at random $r_{out,k} \in [0, l-1]\,(k \in [t])$ and mints coins $Cn_{\text{out},k} = C_{\text{out},k} = \text{Com}\,(a_{\text{out},k}; r_{\text{out},k}) = r_{\text{out},k}G + a_{\text{out},k}H$ for the output public keys. Afterward, the payer adds the output accounts $act_{\text{out},k} = \left\{P_{\text{out},k}, Cn_{\text{out},k}\right\}_{k\in[t]}$ to receivers' account set A_R and sends $ck_{\text{out},k} = (r_{\text{out},k}, a_{\text{out},k})$ to the owner of a public key $P_{\text{out},k}\,(k \in [t])$ through secret channels.

(2) The payer computes $\widetilde{P}_i = \sum_{j=1}^{m} P_{i,j} + \sum_{j=1}^{m} C\ n_{i,j} - \sum_{k=1}^{t} Cn_{\text{out},k}$, where $i \in [n]$. If $\sum_{j=1}^{m} a_{in,j} = \sum_{k=1}^{t} a_{\text{out},k}$, $\widetilde{P}_{in}$ is a commitment to 0. $\widetilde{P}_{in} = (\sum_{j=1}^{m} (sk_{in,j} + r_{in,j}) - \sum_{k=1}^{t} r_{\text{out},k})G = \widetilde{sk}_{in}G$, where $\widetilde{sk}_{in} = \sum_{j=1}^{m} (sk_{in,j} + r_{in,j}) - \sum_{k=1}^{t} r_{out,k}$.

(3) The payer chooses uniformly at random $\rho \leftarrow [0, l-1]$ and computes the randomized public keys $P_j^* \leftarrow P_{in,j} + \rho G$ and the randomized private keys $sk_j^* \leftarrow sk_{in,j} + \rho$, where $j \in [m]$. We denote $P^* = (P_1^*, \ldots, P_m^*)$. The payer computes $\underline{I} \leftarrow \rho \widetilde{H}$.

(4) The payer computes $I_j = sk_j \widetilde{H}$ and $I_j^* = I_j + \underline{I} = sk_j^* \widetilde{H}$, where $j \in [m]$. Let I denote $(I_1, \ldots, I_m)$.

(5) The payer picks uniformly at random $\widetilde{a} \in [0, l-1]$ and then encrypts ρ with the trusted regulator's public key Y_R as $Ct_1' = \widetilde{a} Y_R$, $Ct_2' = \rho G + \widetilde{a} U$. The ciphertext is denoted as $CT_2 = (Ct_1', Ct_2')$. The SoK proof π is as follows:

$$SoK\left\{\begin{array}{ll} \left(\left\{sk_j\right\}_{j=1}^{m}, in, \rho, \widetilde{sk}_{in}, \widetilde{\alpha}\right) & : Ct_1' = \widetilde{\alpha} Y_R \wedge \\ Ct_2' = \rho G + \widetilde{\alpha} U & \wedge \widetilde{P}_{in} = \widetilde{sk}_{in} G \wedge \\ P_1^* = (sk_1 + \rho) G & \wedge I_1 = sk_1 \widetilde{H} \wedge \\ \vdots & \\ P_m^* = (sk_m + \rho) G & \wedge I_m = sk_m \widetilde{H} \end{array}\right\} (tx). \tag{16}$$

Finally, the payer outputs $TX = (tx, P^*, \underline{I}, I, CT_2, \pi)$, where $tx = (m, A, A_R)$.

Verify $(tx, P^*, \underline{I}, I, CT, \pi)$: Receiving A and A_R, the verifiers can compute $\widetilde{P}_i = \sum_{j=1}^{m} P_{i,j} + \sum_{j=1}^{m} Cn_{i,j} - \sum_{k=1}^{t} Cn_{\text{out},k}$, where $i \in [n]$. Output 1 if π passes the verification of the SoK and the serial numbers of the transaction have never appeared on the blockchain. Output 0 otherwise.

BlockGen $(\{TX\}, blk_{n-1})$: Monero uses the Proof-of-Work (PoW) consensus protocol. Firstly, miners collect some Monero transactions from the network and verify them. Validated transactions can be used to generate a Merkle tree. Information such as the root of the Merkle Tree and the hash value of the previous block are contained in the block. Finally, miners try different nonce to make the hash value of the new block meet the mining condition. It outputs the nonce and the block blk_n.

BlockVer (blk_n): Given the current block blk_n, the miners in the network verify whether the hash value of the current block meets the mining condition and whether the transactions in the block are valid; then output 1 or 0, respectively.

Trace (x_R, TX, CT_2): There are two ways to trace user public keys. One is to reveal the user's one-time public key, and the other is to reveal the user's long-term public key.

(1) Given the ciphertext $CT_2 = (Ct_1', Ct_2')$ in the transaction TX, the method to reveal the user's one-time public key is as follows. The trusted regulator decrypts the ciphertext with his private key x_R by computing $P = \rho G = Ct_2' - sk_R^{-1} Ct_1'$. Then, he looks up the list $(P_{1,j}, \ldots, P_{n,j})$ and verifies if $P + P_{i,j} = P_j^*$ is satisfied. If the condition is satisfied, then i is the index of the payer in the ring and $\left\{P_{i,j}\right\}$, where $j \in [m]$ are the one-time public keys of the user being tracked. The trusted regulator can prove to others that $\left\{P_{i,j}\right\}$ are the one-time public keys of the payer by generating the proof $\psi = SoK\{(x_R)\colon Y_R = x_R U \wedge Ct_1' = x_R (Ct_2' - \rho G)\}$. He outputs $\left\{P_{i,j}\right\}$ and ψ.

(2) After getting a one-time public key $P_{i,j}$ of the user, the trusted regulator can reveal the user's long-term public key. He finds where the one-time public key is generated on the blockchain and extracts the ciphertext $CT_1 = (Ct_1, Ct_2)$ from it. Then, he decrypts the ciphertext with his private key x_R by computing $\mathscr{H}_s(rY)G = Ct_2 - sk_R^{-1} Ct_1$ and gets the user's long-term public key by calculating $Y = P_{i,j} - \mathscr{H}_s(rY)G$. The trusted regulator proves to others that the public key Y is the long-term public key of the user by generating the proof $\psi' = SoK\{(x_R)\colon Y_R = x_R U \wedge Ct_1 = x_R (Ct_2 - \mathscr{H}_s(rY)G)\}$. He outputs Y and ψ'.

Judge $(Pk, \psi(\psi'), Y_R, CT_1\,(CT_2))$: Given the user's public key, the proof is generated by the trusted

regulator, the public key of the regulator, and the ciphertext. The verifier can verify the correctness of trace results by checking whether the proof is valid or not and outputs 1 or 0, respectively.

Figure 2 shows the transaction flow in our system. Similar to transaction from Payer to Payee-Payer, that from Payee-Payer to Payee is omitted. Figure 3 shows the transaction verification flow in our system. Figure 4 shows the tracing flow in our system. Figure 5 shows the judging flow in our system. In these figures, we only consider the communications relevant operations. Besides, except that $ck_{out,k}$ is sent through secret channels, all other messages are public. The transactions are assumed verified in Figures 4 and 5.

4.3. Instantiation of SoK. The instantiation of SoK in **UKGen** uses ZKSnark. The SoK described in **Spend** can be instantiated as follows. We can divide it into three parts, and the first part is as follows:

$$PoK_1\left\{\left(in, \rho, \tilde{sk}_{in}\right): P_j^* - P_{in,j} = \rho G, \text{ where } j \in [m] \wedge \underline{I} = \rho \tilde{H} \wedge \tilde{P}_{in} = \tilde{sk}_{in} G\right\}. \tag{17}$$

Pick uniformly at random $s_{i,j} \in [0, l-1], i \in [n], j \in [m+1]$ and select uniformly at random $\alpha_{in,j} \in [0, l-1]$, where $j \in [m+1]$, $in \in [n]$ and compute

$$\begin{aligned} M &\leftarrow \mathscr{H}_p\left(P_{\text{out},1} \| Cn_{\text{out},1} \| \cdots \| P_{\text{out},t} \| Cn_{\text{out},t}\right), \\ L_{in,j} &= \alpha_{in,j} G, j \in [m] \\ R_{in,j} &= \alpha_{in,j} \tilde{H}, j \in [m] \\ L_{in,m+1} &= \alpha_{in,m+1} G. \end{aligned} \tag{18}$$

then set $c_{in+1} = \mathscr{H}_s(m, M, L_{in,1}, R_{in,1}, \ldots, L_{in,m}, R_{in,m}, L_{in,m+1})$.

$$\begin{aligned} L_{in+1,j} &= s_{in+1,j} G + c_{in+1}\left(P_j^* - P_{in+1,j}\right), j \in [m] \\ R_{in+1,j} &= s_{in+1,j} \tilde{H} + c_{in+1} \underline{I}, j \in [m] \\ L_{in+1,m+1} &= s_{in+1,m+1} G + c_{in+1} \tilde{P}_{in+1}. \end{aligned} \tag{19}$$

and repeat this, incrementing in mod n up to

$$\begin{aligned} L_{in-1,j} &= s_{in-1,j} G + c_{in-1}\left(P_j^* - P_{in-1,j}\right), j \in [m] \\ R_{in-1,j} &= s_{in-1,j} \tilde{H} + c_{in-1} \underline{I}, j \in [m] \\ L_{in-1,m+1} &= s_{in-1,m+1} G + c_{in-1} \tilde{P}_{in-1} \\ c_{in} &= \mathscr{H}_s\left(m, M, L_{in-1,1}, R_{in-1,1}, \ldots, \cdot L_{in-1,m}, R_{in-1,m}, L_{in-1,m+1}\right). \end{aligned} \tag{20}$$

Solve for $s_{in,j}$ by $\alpha_{in,j} = s_{in,j} + c_{in} \cdot \rho \bmod q, j \in [m]$ and solve for $s_{in,m+1}$ by $\alpha_{in,m+1} = s_{in,m+1} + c_{in} \cdot \tilde{sk}_{in} \bmod q$. Finally, get $\pi_1 = (c_1, s_{1,1}, \ldots, s_{1,m}, s_{1,m+1}, s_{2,1}, \ldots, s_{2,m}, s_{2,m+1}, s_{n,1}, \ldots, s_{n,m}, s_{n,m+1})$.

The second part of SoK is as follows:

$$PoK_2\left\{\left(\left\{sk_j^*\right\}_{j=1}^m\right): \left(I_j^* = sk_j^* \tilde{H} \wedge P_j^* = sk_j^* G\right), j \in [m]\right\}. \tag{21}$$

Pick uniformly at random $a_j \in [0, l-1]$, and compute the commitment $A_j^1 = a_j G, A_j^2 = a_j \tilde{H}, j \in [m]$, $c_2 = \mathscr{H}_s(I_1^*, \ldots, I_m^*, P_1^*, \ldots, P_m^*, A_1^1, \ldots, A_m^1, A_1^2, \ldots, A_m^2)$ and $z_j = a_j + c_2 \cdot sk_j^*$, where $j \in [m]$. Finally, get $\pi_2 = (A_1^1, \ldots, A_m^1, A_1^2, \ldots, A_m^2, z_1, \ldots, z_m)$.

The third part of SoK is as follows:

$$PoK_3\{(\rho, \tilde{a}): Ct_1 = \tilde{a} Y_R \wedge Ct_2 = \rho G + \tilde{a} U \wedge \underline{I} = \rho \tilde{H}\}. \tag{22}$$

Pick uniformly at random $a, b \in [0, l-1]$, and compute the commitment $A_1' = aY_R$, $A_2' = aU + bG, B = b\tilde{H}, c_3 = \mathscr{H}_s(Ct_1, Ct_2, \underline{I}, A_1', A_2', B)$ and $z_1' = a + c_3 \cdot \tilde{a}$, $z_2' = b + c_3 \cdot \rho$. Finally, get $\pi_3 = (A_1', A_2', B, z_1', z_2')$.

In conclusion, $\pi = (\pi_1, \pi_2, \pi_3)$.

The verification of the SoK is as follows. Parse π as π_1, π_2, π_3.

(1) Verify π_1 as follows:

$$\begin{aligned} M &\leftarrow \mathscr{H}_p\left(P_{out,1} \| Cn_{out,1} \| \cdots \| P_{out,t} \| Cn_{out,t}\right), \\ L_{1,j} &= s_{1,j} G + c_1\left(P_j^* - P_{1,j}\right), j \in [m] \\ R_{1,j} &= s_{1,j} \tilde{H} + c_1 \underline{I}, j \in [m] \\ L_{1,m+1} &= s_{1,m+1} G + c_1 \tilde{P}_1 \\ c_2 &= \mathscr{H}_s\left(m, M, L_{1,1}, R_{1,1}, \ldots, L_{1,m}, R_{1,m}, L_{1,m+1}\right). \end{aligned} \tag{23}$$

And repeat this, incrementing $i \bmod n$ until they figure out all $L_{i,j}$, $R_{i,j}$, c_i, where $j \in [m], i \in [n]$ and $L_{i,m+1}$, where $i \in [n]$, and then verify if $c_{n+1} = c_1$.

(2) Compute $I_j^* = I_j \cdot \underline{I}$, where $j \in [m]$. Verify $\pi_2 = (A_1^1, \ldots, A_m^1, \ldots, A_1^2, \ldots, A_m^2, z_1, \ldots, z_m)$ by checking whether $\text{Verf}(I_j^*, P_j^*, \pi_2) = 1$, where $j \in [m]$. The details are as follows:

Compute $c_2' = \mathscr{H}_s(I_1^*, \ldots, I_m^*, P_1^*, \ldots, P_m^*, A_1^1, \ldots, A_m^1, \cdots, A_1^2, \ldots, A_m^2)$. Check whether the equations are satisfied, including $A_j^1 = z_j G - c_2' P_j^*, A_j^2 = z_j \tilde{H} - c_2' I_j^*$, where $j \in [m]$.

(3) Verify $\pi_3 = (A_1', A_2', B, z_1', z_2')$ by checking whether $\text{Verf}(\underline{I}, Ct_1, Ct_2, Y_R, \pi_3) = 1$. The details are as follows:

Compute $c_3' = \mathscr{H}_s(Ct_1, Ct_2, \underline{I}, A_1', A_2', B)$, and check whether the following equations are satisfied: $A_1' =$

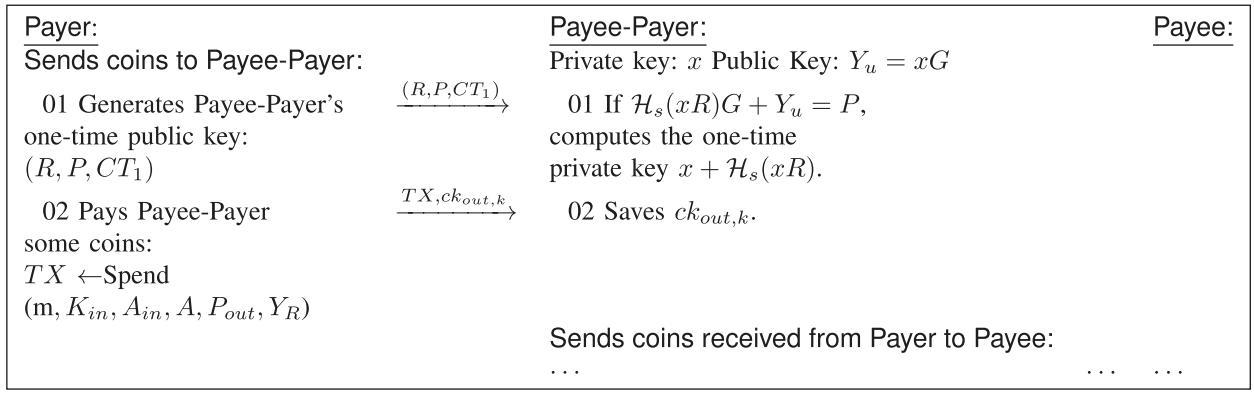

Figure 2: Transaction flow of our system.

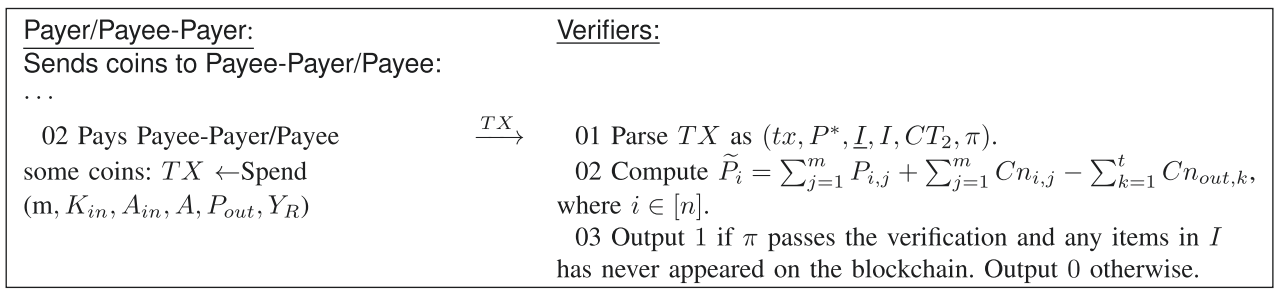

Figure 3: Transaction verification flow of our system.

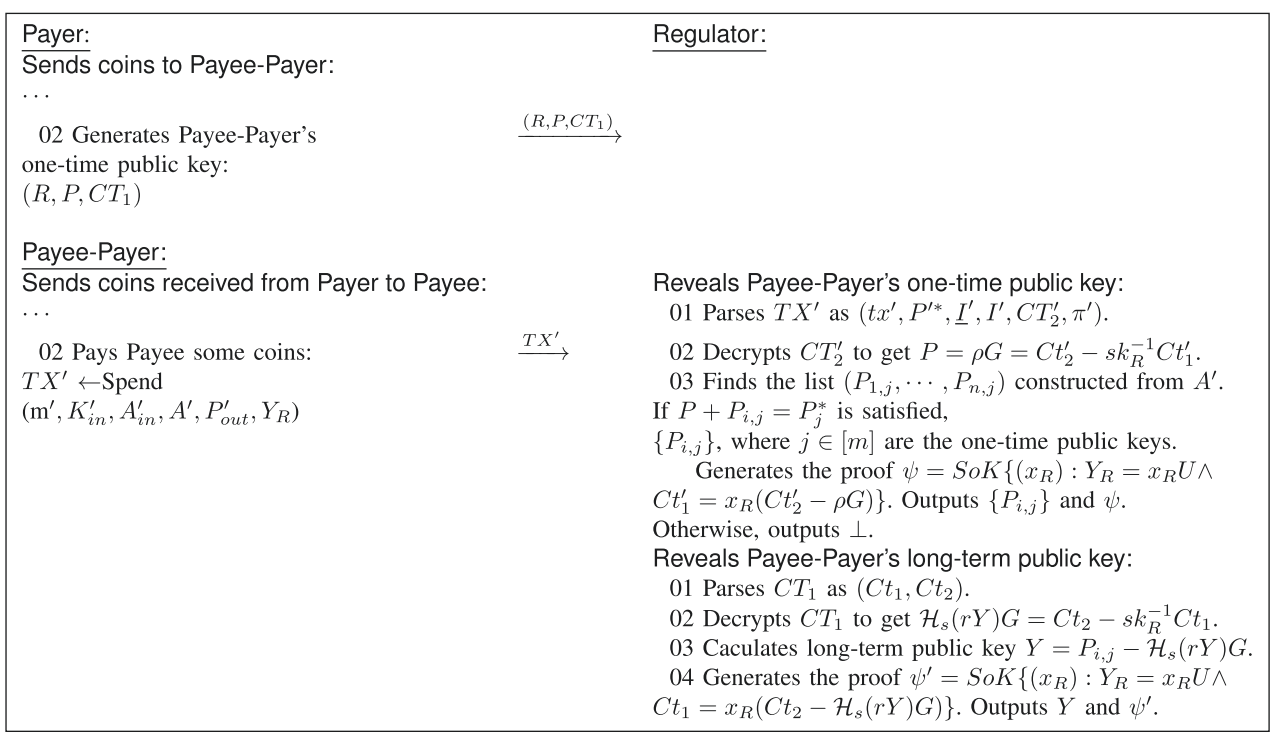

Figure 4: Tracing flow of our system.

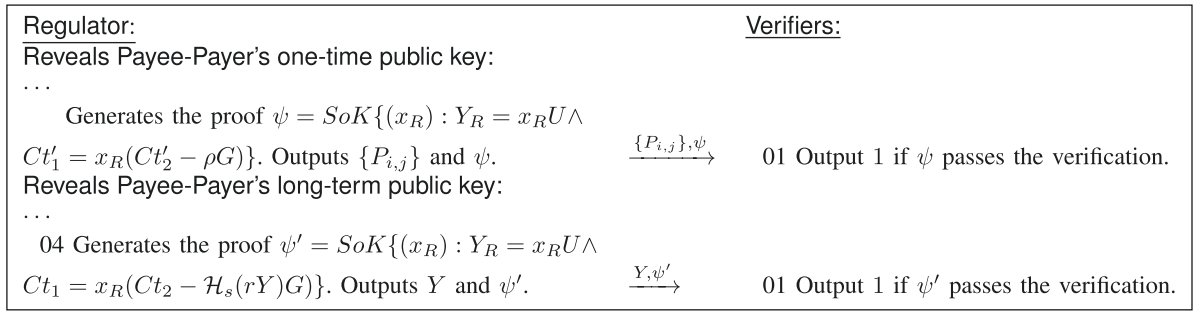

Figure 5: Judging flow of our system.

$z_1'Y_R - c_3'Ct_1$, $A_2' = z_1'U + z_2'G - c_3'Ct_2, B = z_2'\tilde{H} - c_3'\underline{I}$.

The instantiation of SoK in **Trace** is similar to PoK_2 in **Spend**.

4.4. Security Analysis. In this section, the security of our accountable Monero scheme is analyzed.

Theorem 1. *Let SoK be a SimExt-secure signature of knowledge. The proposed accountable Monero scheme is balanced under the discrete logarithm (DL) assumption.*

Theorem 2. *Assume the SoK is SimExt-secure. The homomorphic commitment is perfectly hiding, and the encryption scheme is IND-CPA secure. The proposed accountable Monero scheme satisfies anonymity under the decisional Diffie–Hellman (DDH) assumption.*

Theorem 3. *Assume the SoK is a SimExt-secure. The proposed accountable Monero scheme satisfies nonslanderability under the discrete logarithm (DL) assumption.*

Theorem 4. *Assume the SoK and public key encryption are with perfect correctness. The SoK is sound and the certified address scheme satisfies unforgeability. The proposed accountable Monero scheme achieves traceability.*

5. Efficiency Analysis

In this section, we analyze the performance of our accountable Monero system and compare it with that of Monero. Then, we compare our scheme with the accountable Monero system of [8] in the practical efficiency. Our system is constructed directly on RingCT protocol, the linkable ring signature used in Monero, while the scheme of [8] depends on the accumulator with a one-way domain, which compresses the ring signature size, so that it is independent of the number of groups. However, in practice, Monero's ring size is small. Our scheme will obtain better efficiency. The comparison is given in Table 1.

"n": the number of input account groups; "m": the number of input accounts in each group; "l": the length of the element in $\mathbb{Z}_q$; "$\exp_q$": an exponentiation operation in the group $\mathbb{G}$; "$\exp_1$": an exponentiation operation in group $\mathbb{G}_1$ used in the accumulator of [8]; "$\exp_T$": an exponentiation operation in the group $\mathbb{G}_T$ used in the accumulator of [8]; "p": a bilinear pairing operation; $|\mathbb{G}|$, $|\mathbb{G}_1|$, $|\mathbb{G}_T|$, $|\mathbb{Z}_q|$ and $|\mathbb{Z}_p|$ are the lengths of the element in group $\mathbb{G}$, $\mathbb{G}_1$, $\mathbb{G}_T$, $\mathbb{Z}_q$ and $\mathbb{Z}_p$ respectively.

In the efficiency analysis, we only consider the expensive operations, including exponentiation and multi-exponentiation, and ignore other lightweight calculations. We use the same optimization as in [8], where the double-exponentiation calculation costs 1.2 times as many as that of a single exponentiation, and the triple-exponentiation calculation cost is 1.5 times that of a single exponentiation, and we neglect the time consumption of operations that can be precomputed.

The details of the analysis are as follows. For a payer, $(2m+1)(n-1)\cdot \exp_q$ is required in PoK_1. All the other computations in SoK can be precomputed. For a verifier, it needs $1.2\cdot(2m+1)\cdot n\cdot \exp_q$ in verifying PoK_1, $2.4m\cdot \exp_q$ in verifying PoK_2 and $3.9\cdot \exp_q$ in verifying PoK_3. For the communication cost, as in [8], we consider the parameters to be stored and transferred to a verifier in SoK. In addition, we count the parameters P^* and $\underline{I}$ in our scheme. $(m+1)|\mathbb{G}|$ is required in P^* and $\underline{I}$, $((m+1)\cdot n+1)|\mathbb{Z}_q|$ is required in PoK_1, $2m|\mathbb{G}| + m|\mathbb{Z}_q|$ is required in PoK_2 and $3|\mathbb{G}| + 2|\mathbb{Z}_q|$ is required in PoK_3.

6. Proof of Theorems

6.1. Proof of Theorem 1

Proof. Our proof is consistent with [8, 20], with only minor differences. If there exists an efficient adversary $\mathcal{A}$ that breaks the property of balance with nonnegligible probability ε, the DL problem can be solved by constructing an efficient algorithm $\mathcal{A}'$.

Having DL instance $(G, \alpha G)$, the algorithm $\mathcal{A}'$ runs SoK.Gen (1^λ) to generate par. Then, it lets $H = \alpha G$ and picks generators uniformly at random $\tilde{H}, U \in \mathbb{G}$. The public parameter is $pp = (1^\lambda, par, G, H, \tilde{H}, U, \mathcal{H}_s, \mathcal{H}_p, \xi)$, where $\mathcal{H}_s$: $\{0,1\}^* \longrightarrow \mathbb{Z}_q$ and $\mathcal{H}_p$: $\{0,1\}^* \longrightarrow \{0,1\}^*$ denote two cryptographic secure hash functions, and ξ is a function from $\mathbb{G}$ to $\mathbb{Z}_q$. $\mathcal{A}$ can make queries to AddGen, ActGen, Spend and Corrupt Oracles at most q_{ad}, q_{ac}, q_{sp} and q_{co} times, respectively. First of all, $\mathcal{A}'$ chooses a $j^* \in [q_{ad}]$ and picks uniformly at random $sk_i \in [0, l-1]$, for all $i \in [q_{ad}]$ as the private keys. The queried public keys are calculated as follows:

Table 1: Comparison of Monero systems.

Ref.	Payer	Verifier	Communication
[9]	$(2.2\,m\ n-1.2\,m+n)..\ \mathrm{Exp_q}$	$2.4mn\ \mathrm{exp_q}$	$(m\ n+1)\ \lvert\mathbb{Z}_q\rvert$
[8]	$((n-1)+1)\ (m+1)\ \mathrm{exp}\ 1+(m+1)\ \mathrm{exp}\ T+(m+1)\ p+1.2\,l(m+1)\ \mathrm{exp}$	$5.6l(m+1)\ \mathrm{exp\ q}+(4.2+n+1.2l)\ (m+1)\ \mathrm{exp1}+4(m+1)\ \mathrm{exp}\ T+3(m+1)\ \mathrm{p}$	$(l+4)\ (m+1)\ \lvert\mathbb{G}_1\rvert+(5l+8)\ (m+1)\ \lvert\mathbb{Z}_q\rvert$ p+$5l(m+1)+(m+1)\ \lvert\mathbb{G}_T\rvert$
Ours	$(2m+1)\ (n-1)\ \mathrm{exp_q}$	$(2.4m\ n+1.2n+2.4\,\mathrm{m}+3.9)\ \mathrm{exp_q}$	$(3m+4)\ \lvert\mathbb{G}\rvert+((m+1)\ (n+1)+3)\ \lvert\mathbb{Z}_q\rvert$

$$P_i=\begin{cases} sk_iG, & i\neq j^*,\\ sk_iH, & i=j^*,\end{cases} \tag{24}$$

$\mathscr{A}'$ simulates the oracles as follows:

(i) AddGen (i): Return P_i for the $i-$ th query.

(ii) ActGen (i,a_i): Given i and an amount a_i, $\mathscr{A}'$ chooses uniformly at random $r_i\in[0,l-1]$ and sets $Cn_i=C_i=r_iG+a_iH$. i, $act_i=(P_i,Cn_i)$, and $I_i=sk_i\widetilde{H}$ are added to the lists $\mathscr{I}$, $\mathscr{G}$, and $\mathscr{S}$, respectively.

(iii) Spend $(m,A_{in},A,P_{\mathrm{out}})$: Given the inputs, if $act_{j^*}\notin A_{in}$, $\mathscr{A}'$ generates $(tx,P^*,\underline{I},I,CT,\pi)$ according to the scheme. Otherwise, $\mathscr{A}'$ produces π by simulating SoK. Without witnesses, it is indistinguishable from $\mathscr{A}$'s view. $(tx,P^*,\underline{I},I,CT,\pi)$ is added to the list $\mathscr{T}$.

(iv) Corrupt (i): Given i, if $i=j^*$, $\mathscr{A}'$ aborts. Otherwise, $\mathscr{A}'$ returns sk_i and adds I_i, (I_i,a_i) to the list $\mathscr{C}$ and the list $\mathscr{B}$, respectively.

Finally, $\mathscr{A}$ outputs some new accounts he created and all his expenses $(act_1',\ldots,act_{\mu'}',S_1,\ldots,S_\nu)$, where $S_i=(tx_i,\mathscr{P}_i^*,\underline{I}_i,I_i,\pi_i,CT_i)$, for all $i\in[\nu]$. All transactions are paid to $\mathscr{A}'$.

Let $E_i=\{a_{i,j}\colon (I_{i,j},a_{i,j})\in\mathscr{B}\wedge I_{i,j}\in(I_i\cap\mathscr{C})\}$, where $i\in[\nu]$, $j\in[m]$, $E=E_1\cup E_2\cup\cdots\cup E_\nu$.

(i) Case 1: $\forall i\in[\nu], I_i\backslash\mathscr{C}=\phi$. According to the condition of a successful attack $\sum_{a_{i,j}\in E}a_{i,j}<\sum_{k=1}^{\nu}a_{\mathrm{out}}^i$, where $a_{\mathrm{out}}^i=\sum_{k=1}^{t}a_{out,k}^i$, there exists some $i^*\in[\nu]$, which have $\sum_{a_{i^*,j}\in E}a_{i^*,j}<a_{out}^{i^*}$. From the serial numbers, $\mathscr{A}'$ can find the set $\{act_{i^*,j}\}$ of accounts spent in A_{i^*} from the list $\mathscr{G}$, where $act_{i^*,j}=(pk_{i^*,j},Cn_{i^*,j})$ and the private key is $ask_{i^*,j}=(sk_{i^*,j},(r_{i^*,j},a_{i^*,j}))$. Then, $\mathscr{A}'$ can compute $\widetilde{P}_{i^*}=\sum_{j=1}^{m}P_{i^*,j}+\sum_{j=1}^{m}Cn_{i^*,j}-Cn_{\mathrm{out}}^{i^*}=(\sum_{j=1}^{m}sk_{i^*,j}+\sum_{j=1}^{m}r_{i^*,j}-r_{out}^{i^*})G+(\alpha\cdot(\sum_{j=1}^{m}a_{i^*,j}-a_{\mathrm{out}}^{i^*}))G$. $S_i\notin\mathscr{T}$ for all $i\in[\nu]$; thus, $\mathscr{A}'$ can use Ext to extract the witness $\widetilde{sk}_{i^*}$. The DL instance can be solved by calculating

$$\alpha=\frac{\widetilde{sk}_{i^*}-\left(\sum_{j=1}^{m}sk_{i^*,j}+\sum_{j=1}^{m}r_{i^*,j}-r_{\mathrm{out}}^{i^*}\right)}{\sum_{j=1}^{m}a_{i^*,j}-a_{\mathrm{out}}^{i^*}},\quad \text{where } a_{\mathrm{out}}^{i^*}=\sum_{k=1}^{t}a_{\mathrm{out},k}^{i^*},\ r_{\mathrm{out}}^{i^*}=\sum_{k=1}^{t}r_{\mathrm{out},k}^{i^*}. \tag{25}$$

(ii) Case 2: $\exists i\in[\nu], I_i\backslash\mathscr{C}\neq\phi$, Let $\mathscr{J}=\{i\in[\nu]\colon I_i\backslash\mathscr{C}\neq\phi\}$, $I_{\mathscr{J}}=\cup_{i\in\mathscr{J}}(I_i/\mathscr{C})$. If $I_{j^*}\in I_{\mathscr{J}}$, then $\exists i^*\in\mathscr{J}$, $I_{j^*}\in I_{i^*}$. $\mathscr{A}'$ can extract a piece of witness that contains sk_{j^*}' and ρ_{j^*}', namely, $P_{j^*}=g^{sk_{j^*}'}=g^{\alpha\cdot sk_{j^*}+\rho_{j^*}'}$. DL instance can be solved by calculating $\alpha=((sk_{j^*}'-\rho_{j^*}')/sk_{j^*})$. □

6.2. Proof of Theorem 2

Proof. We define $\Pr[Succ_i]$ as the probability that adversary $\mathscr{A}$ succeeds in $Game_i$.

Game 0. Game 0 is the challenge game defined in anonymity. The challenger generates public parameters $pp=(1^\lambda,par,G,H,\widetilde{H},U,\mathscr{H}_s,\mathscr{H}_p,\xi)$ by executing $Setup(1^\lambda)$ and responds to all queries to AddGen, ActGen, Spend, Corrupt Oracles as in the real game. When receiving the challenge query $(m,A_{s_0},A_{s_1},A,P_{\mathrm{out}})$, where $A=\{(P_{i,j},Cn_{i,j})\}_{i\in[n],j\in[m]}$ are the input accounts, $A_{s_b}=\{(P_{s_b,j},Cn_{s_b,j})\}_{j\in[m]}$ is the s_b-th group of A, and $P_{\mathrm{out}}=\{P_{out,k}\}_{k\in[t]}$, the challenger chooses uniformly at random $b\in\{0,1\}$ and calculates according to the algorithm to get a response $(tx,P^*,\underline{I},I,CT)$. Finally, A outputs a guess b'. According to the definition of anonymity, we have

$$\Pr[\mathrm{Succ}_0]=\Pr[b'=b]. \tag{26}$$

Game 1. Game 1 and Game 0 are the same, except that the algorithm Gen is replaced by SimGen, and all SoK are generated by calling the simulator Sim = (SimGen, SimSign). That means no witness is needed in the calculation during Spend Oracle queries and the challenge query. Specifically, for a challenge query $(m,A_{s_0},A_{s_1},A,P_{out})$ and allowed Spend queries $(m,A_s,A,P_{\mathrm{out}})$, it has $A_s\cap A_{s_b}=\varnothing$. For Spend queries, the challenger computes the statement $(M,P^*,\underline{I},I,CT)$ and generates $\pi=\mathrm{Sim}(M,P^*,\underline{I},I,CT)$. For the challenge query $(m,A_{s_0},A_{s_1},A,P_{out})$, the proof is generated in the same way. If the difference between the adversary's success probability of **Game 0** and in **Game 1** is not negligible, we can use the adversary to construct an algorithm to break the SimExt property of SoK.

$$\left|\Pr[\mathrm{Succ}_1]-\Pr[\mathrm{Succ}_0]\right|\leq negl(\lambda). \tag{27}$$

Game 2. The difference from Game 1 is that, in the challenge phase of Game 2, the challenger uniformly and randomly selects the transaction output addresses $P_{\mathrm{out},k}$, where $k\in[t]$. Specifically, the challenger picks uniformly at random $\alpha\leftarrow[0,l-1]$, sets $H=\alpha G$ and sets output coins $Cn_{out,k}=\hat{r}_kG$ for addresses $P_{out,k}\in P_{out}$ by choosing uniformly at random $\hat{r}_k\in[0,l-1]$. The corresponding coin private keys satisfy $ck_{\mathrm{out},k}=(\hat{r}_k-\alpha\cdot a_{\mathrm{out},k}^{(b)},a_{\mathrm{out},k}^{(b)})$, where $k\in[t]$. Thus, the output accounts $A_R=\{(P_{\mathrm{out},k},Cn_{\mathrm{out},k})\}_{k\in[t]}$ are independent of the choice of b. In this case, $\widetilde{P}_i=\sum_{j=1}^{m}P_i^j+\sum_{j=1}^{m}Cn_i^j-\sum_{k=1}^{t}Cn_{\mathrm{out},k}$, where $i\in[n]$ is independent of b. If the adversary's success probability in **Game 1** and **Game 2** is

different, we can use the adversary to construct an algorithm to break the perfect hiding property of homomorphic commitment.

$$\Pr[Succ_2] = \Pr[Succ_1]. \tag{28}$$

Game 3. The difference between Game 3 and Game 2 is in the generation of serial numbers. When answering the challenge query, the challenger chooses uniformly at random $I_j \in \mathbb{G}$, as the value of the serial numbers, where $j \in [m]$. Assume that no account in A_{s_b} is corrupted and has been asked in the Spend Oracle queries. Therefore, the serial numbers $\widetilde{I} = (I_1, \ldots, I_m)$ are fresh to the adversary. If DDH assumption holds, the serial numbers are uniformly distributed. If the difference between the adversary's success probability in **Game 3** and **Game 2** is not negligible, we can use the adversary to construct an algorithm to break DDH assumption. The adversary's success probability satisfies

$$\left|\Pr[Succ_3] - \Pr[Succ_2]\right| \leq negl(\lambda). \tag{29}$$

Game 4. When replying Spend Oracle queries, the challenger encrypts a random number μ to replace the original ciphertext. When replying Trace Oracle, if tx being queried is in the list $\mathcal{T}$, the challenger returns ρG through searching the list. Otherwise, the simulator can obtain a witness through the SimExt-secure of SoK. And according to the IND-CPA security of the encryption scheme, the challenger gets a unique ρG. If the difference between the adversary's success probability in **Game 4** and **Game 3** is not negligible, we can use the adversary to construct an algorithm to break the SimExt-secure of SoK and the IND-CPA of the encryption scheme. The adversary's success probability satisfies

$$\left|\Pr[Succ_4] - \Pr[Succ_3]\right| \leq negl(\lambda). \tag{30}$$

In **Game 4**, the challenge transaction $(\widetilde{tx}, \widetilde{P}^*, \widetilde{\underline{I}}, \widetilde{I}, \widetilde{\pi}, \widetilde{CT})$ is independent of b. Hence, the probability of the adversary's success in **Game 4** is $(1/2)$. In summary, the probability of success is $(1/2) + negl(\lambda)$. □

6.3. Proof of Theorem 3

Proof. Suppose that there is an adversary $\mathcal{A}$ that can break the nonslanderability of the protocol with a nonnegligible probability. We can construct a polynomial-time algorithm $\mathcal{A}'$ to solve the DL problem in $\mathbb{G}$ by calling $\mathcal{A}$.

Given a DL instance $(G, H = \alpha G)$, the algorithm $\mathcal{A}'$ runs Gen (1^λ) to produce par. $\mathcal{A}'$ uniformly chooses a random $\beta \in [0, l-1]$, computes $\widetilde{H} = \beta G$, selects a generator at random $U \in \mathbb{G}$ and sets $pp = (par, G, H, \widetilde{H}, U, \mathcal{H}_s, \mathcal{H}_p, \xi)$, where $\mathcal{H}_s$: $\{0,1\}^* \longrightarrow \mathbb{Z}_q$ and $\mathcal{H}_p$: $\{0,1\}^* \longrightarrow \{0,1\}^*$ denote two cryptographic secure hash functions, and ξ is a function from $\mathbb{G}$ to $\mathbb{Z}_q$. Assume that $\mathcal{A}$ can make queries to AddGen, ActGen, Spend and Corrupt Oracles at most q_{ad}, q_{ac}, q_{sp} and q_{co} times, respectively. First of all, $\mathcal{A}'$ chooses a $j^* \in [q_{ad}]$ and picks uniformly random $sk_i \in [0, l-1]$, as the private keys, where $i \in [q_{ad}]$. The queried public keys are calculated as follows:

$$P_i = \begin{cases} sk_i G, & i \neq j^*, \\ sk_i H, & i = j^*, \end{cases} \tag{31}$$

$\mathcal{A}'$ simulates AddGen, ActGen, Spend, and Corrupt Oracles, consistent with that in the proof of Theorem 1. Finally, $\mathcal{A}$ outputs $((\widehat{tx}, \widehat{P}^*, \widehat{\underline{I}}, \widehat{I}, \widehat{\pi}, \widehat{CT}), (\widetilde{tx}, \widetilde{P}^*, \widetilde{\underline{I}}, \widetilde{I}, \widetilde{\pi}, \widetilde{CT}))$. If $\mathcal{A}$ succeeds in the nonslanderability game, that means (1) $\text{Verify}(\widetilde{tx}, \widetilde{P}^*, \widetilde{\underline{I}}, \widetilde{I}, \widetilde{\pi}, \widetilde{CT}) = 1$; (2) $(\widetilde{tx}, \widetilde{P}^*, \widetilde{\underline{I}}, \widetilde{I}, \widetilde{\pi}, \widetilde{CT}) \notin \mathcal{T}$; (3) $\widehat{I} \cap \mathcal{C} = \varnothing$ but $\widehat{I} \cap \widetilde{I} \neq \varnothing$. Checking if $j^* \notin \widehat{I} \cap \widetilde{I}$, $\mathcal{A}'$ outputs a random bit guess. Otherwise, $\mathcal{A}'$ can extract the witness of $(\widetilde{tx}, \widetilde{P}^*, \widetilde{\underline{I}}, \widetilde{I}, \widetilde{\pi}, \widetilde{CT})$, which includes $\widetilde{sk}_{j^*}$ and $\widetilde{\rho}_{j^*}$, namely, $\widetilde{P^*_{j^*}} = \widetilde{sk^*_{j^*}} G = (\alpha \cdot sk_{j^*} + \widetilde{\rho_{j^*}})G$. The discrete logarithm of H concerning G can be calculated as $\alpha = ((\widetilde{sk^*_{j^*}} - \widetilde{\rho_{j^*}})/sk_{j^*})$. □

6.4. Proof of Theorem 4

Proof. If the SoK proof π in the transaction TX of **Spend** algorithm is correct and sound, Ct'_1 and Ct'_2 are in the correct form. The encrypted plaintext ρG is unique according to the perfect correctness of the public key encryption system. With ρG, P^* and the input accounts A, the one-time public key P can be determined. According to the extractability of SoK in the KeyGen algorithm, r can be extracted. Y, R, P, and CT_1 are all in the correct form. Because $\mathcal{H}_s$ is a cryptographic secure hash function, the probability of finding a Y' that satisfies $\log_G(P - Y') = H(rY)$ is negligible. Therefore, Y is the long-term public key of the real payee selected by the payer. Moreover, due to the unforgeability of the certified address and SoK's soundness, Y has been certified by the trusted registration authority. The correctness of the encryption scheme means that we can get $Y = P - (Ct_2 - sk_R^{-1} Ct_1)$. The other part of the tracing output is the SoK proof $\psi(\psi')$, which proves that the public key outputted is correct. The correctness and soundness of the SoK prove that it can be correctly verified. □

7. Conclusion

In this paper, an accountable Monero system based on the RingCT protocol is proposed. We utilize the signature with rerandomizable keys to randomize the one-time spend keys and exploit ElGamal public key encryption to encrypt the random number, which only enables the trusted regulator to reveal the public keys of the payers. And to enable the traceability of all users of the system, we replace the long-term public key with a certified long-term public key. The designed system maintains correctness, balance, nonslanderability, and traceability. Through security proof and analysis, we demonstrate the security and performance of the proposed system.

Data Availability

No data were used during this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the National Key R&D Program of China (2017YFB0802500) and Shandong Province Major Science and Technology Innovation Project (2019JZZY020129).

References

[1] M. Conti, E. Sandeep Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[2] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world.," 2013, https://bitcointalk.org/index.php?topic=279249.

[3] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: an untrusted bitcoin-compatible anonymous payment hub," in *Proceedings of the Network and Distributed System Security Symposium*, California, CA, USA, 2017.

[4] S. Nakamoto, "Monero project," 2021, https://www.getmonero.org/index.html.

[5] E. B. Sasson, A. Chiesa, C. Garman et al., "Zerocash: decentralized anonymous payments from bitcoin," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, pp. 459–474, IEEE, California, CA, USA, 2014.

[6] A. B. Turner, S. McCombie, and A. J. Uhlmann, "A target-centric intelligence approach to wannacry 2.0," *Journal of Money Laundering Control*, vol. 22, 2019.

[7] I. Damgård, C. Ganesh, H. Khoshakhlagh, C. Orlandi, and L. Siniscalchi, "Balancing privacy and accountability in blockchain identity management," in *Proceedings of the Topics in Cryptology - CT-RSA 2021*, pp. 552–576, San Francisco, CA, USA, 2021.

[8] Y. Li, G. Yang, W. Susilo, Y. Yu, M. H. Au, and D. Liu, "Traceable monero: anonymous cryptocurrency with enhanced accountability," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 2, pp. 679–691, 2021.

[9] S. Noether, A. Mackenzie, and T. M. Research Lab, "Ring confidential transactions," *Ledge*, vol. 1, pp. 1–18, 2016.

[10] M. Abe, M. Ohkubo, and K. Suzuki, "1-out-of-n signatures from a variety of keys," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 415–432, Springer, Singapore, 2002.

[11] M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu, "Dynamic universal accumulators for ddh groups and their application to attribute-based anonymous credential systems," in *Proceedings of the Cryptographers' Track at the RSA Conference*, pp. 295–308, Springer, San Francisco, CA, USA, 2009.

[12] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup, "Anonymous identification in ad hoc groups," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 609–626, Springer, Interlaken, Switzerland, 2004.

[13] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable spontaneous anonymous group signature for ad hoc groups," in *Proceedings of the Australasian Conference on Information Security and Privacy*, pp. 325–335, Springer, Sydney, Australia, 2004.

[14] P. P. Tsang and V. K. Wei, "Short linkable ring signatures for e-voting, e-cash and attestation," in *Proceedings of the International Conference on Information Security Practice and Experience*, pp. 48–60, Springer, Singapore, 2005.

[15] X. Wang, Y. Chen, and X. Ma, "Adding linkability to ring signatures with one-time signatures," in *Proceedings of the International Conference on Information Security*, pp. 445–464, Springer, New York City, NY, USA, 2019.

[16] E. Fujisaki and K. Suzuki, "Traceable ring signature," in *Proceedings of the International Workshop on Public Key Cryptography*, pp. 181–200, Springer, Beijing, China, 2007.

[17] M. H. Au, J. K. Liu, W. Susilo, and T. H. Yuen, "Secure id-based linkable and revocable-iff-linked ring signature with constant-size construction," *Theoretical Computer Science*, vol. 469, pp. 1–14, 2013.

[18] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, and C. Petit, "Short accountable ring signatures based on DDH," in *Proceedings of the Computer Security -- ESORICS 2015*, pp. 243–265, Springer, Vienna, Austria, 2015.

[19] J. Groth and M. Kohlweiss, "One-out-of-many proofs: or how to leak a secret and spend a coin," *Advances in Cryptology - EUROCRYPT 2015*, Springer, in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 253–280, 2015.

[20] S.-F. Sun, M. H. Au, J. K. Liu, and T. H. Yuen, "Ringct 2.0: a compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero," in *Proceedings of the European Symposium on Research in Computer Security*, pp. 456–474, Springer, Oslo, Norway, 2017.

[21] T. H. Yuen, S.-f. Sun, J. K. Liu et al., "Ringct 3.0 for blockchain confidential transaction: shorter size and stronger security," in *Proceedings of the International Conference on Financial Cryptography and Data Security*, pp. 464–483, Springer, Kota Kinabalu, Malaysia, 2020.

[22] G. Ateniese, A. Faonio, B. Magri, and B. De Medeiros, "Certified bitcoins," *Applied Cryptography and Network Security*, in *Proceedings of the International Conference on Applied Cryptography and Network Security*, pp. 80–96, Lausanne, Switzerland, 2014.

[23] J. Groth, "Fully anonymous group signatures without random oracles," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 164–180, Springer, Kuching, Malaysia, 2007.

[24] N. Van Saberhagen, "Cryptonote v 2.0," 2013, https://cryptonote.org/whitepaper.pdf.

[25] N. Koblitz, "A Course in Number Theory and Cryptography," Springer, USA, 1987.

[26] G. Malavolta and D. Schröder, "Efficient ring signatures in the standard model," *Advances in Cryptology - ASIACRYPT 2017*, Springer, in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 128–157, 2017.

[27] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *Proceedings of the Annual International Cryptology Conference*, pp. 78–96, Springer, California, CA, USA, 2006.

[28] N. T. Courtois and R. Mercer, "Stealth address and key management techniques in blockchain systems," in *Proceedings of the ICISSP*, vol. 2017, pp. 559–566, Porto, Portugal.