This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3251735

Monero with Multi-Grained Redaction

Ke Huang, *Member, IEEE,* Yi Mu, *Senior Member, IEEE,* Fatemeh Rezaeibagha, *Member, IEEE,* Xiaosong Zhang, *Senior Member, IEEE*, Xiong Li, Sheng Cao

Abstract—Monero is a privacy-centric cryptocurrency that allows users to obscure their transactions with multiple input and output addresses. Current research on Monero mainly focuses on identifying design vulnerabilities or optimizing towards stronger privacy, security, etc. For example, improving the design of ring confidential transaction (RingCT) protocol proposed by Noether et al. As revealed by Ali et al. in USENIX 2016, new blockchains have inadequate nodes and network computing resources to resist powerful attack (e.g. 51% attack). Obviously, Monero blockchain is not an exception. Ateniese et al. proposed the notion of redactable blockchain in EuroS&P 2017, which begins the trend of formalizing blockchain with extra cryptographic primitives. The motivation is to turn an immutable blockchain into a mutable ledger by adapting the blockchain design and integrating with new cryptographic schemes. In such a setting, users could use their private keys to perform the secure multi-party computation to reverse blockchain history. The idea of redactable blockchain has attracted many researchers to pursuit this topic. However, few works have considered the privacy-preserving setting. Even fewer have practised their designs in an actual cryptocurrency. In this paper, we seek to adapt the RingCT protocol with several building blocks. Our proposal achieves most of the desired properties for blockchain redaction. It allows multiple tracing authorities to collaboratively trace users' identities, and a system manager to perform multi-grained (including block-level, transaction-level, accumulator-level and commitment-level) redaction on block contents. Our proposal can be seen as an extension of RingCT protocol. We give rigorous security requirements and comprehensive analysis of our scheme. The performance evaluation suggested that our scheme suffers from some unscalabilities in large-scale implementations. A more elegant design to achieve stronger security and ideal scalability is deemed as a challenging and interesting future work.

Index Terms—Blockchain, privacy-preservation, Monero, ring confidential transaction

1 Introduction

Cryptocurrency is commonly known as a decentralized digital currency based on public-key cryptography (PKC) to address the central shortcoming of well-established currencies. As the first and most successful cryptocurrency, the original Bitcoin [1] does not offer a meaningful level of anonymity. Though privacy is not defined as an inherent property in Bitcoin's initial design, it is strongly associated with the system. In recent years, an array of privacy-preserving cryptocurrencies have emerged, successful cases are: Monero [2], Litecoin, Dash coin, Zerocoin, Zerocash, etc.

Monero [2] is a leading privacy-centric cryptocurrency. The original Monero was not an extension of Bitcoin, but it was based on the CryptoNote protocol [3]. It exceeds other privacy-preserving cryptocurrencies in terms of market capitalization and promises privacy and anonymity through unlinkable and untraceable transactions. Monero protects payee's identity based on CryptoNote protocol [3] and utilizes one-time ring signature provided by CryptoNote and ring confidential transaction (RingCT) [4] to achieve full privacies for the payer and the payee. It allows the signer to randomly choose a set of users' public keys to form a ring in which his public key is hidden. Since such a ring is formed spontaneously, other users are unaware that they are summoned for generating the signature. Specifically, RingCT adopts linkable ring signature where the same signer's linkable ring signatures can be detected through linkability of tags via computations. This is an ideal method to detect double-spending in cryptocurrencies.

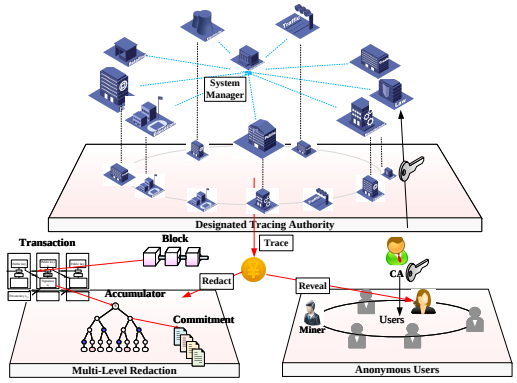

Fig. 1: The system model of redactable Monero

As the main research line, current studies on Monero [5]–[9] are mainly based on RingCT protocol to achieve better

This work was supported in part by the National Key R&D Program of China under Grant No.2017YFB0802300, Grant No.2018YFB08040505 and Grant No.2018YFB0804100; National Natural Science Foundation of China under Grants 62002048, 61872087, U19A2066, 62072078, U22B2029; University Startup Grant under Grant No.Y030202059018061; Blockchain Research Lab of UESTC, Chengdu Jiaozi Financial Holding Group Company Ltd; China Mobile Information Communication Technology Co., Ltd (Chengdu) 2020 UAV Operation Management Platform Phase II (Package 2: Safety Subsystem)(No. CMCMII-202001245); Key Research and Development Program of Sichuan Province under Grants 2021YFG0310 and 2021ZHCG0001.

K. Huang, X. Zhang, X. Li and S. Cao are with the College for Cyber Security, University of Electronic Science and Technology of China (UESTC), Chengdu, 611731, China; Blockchain Research Lab of UESTC-Chengdu Jiaozi Financial Holding Group Co.Ltd, Chengdu, 610042, China. (e-mail: huangke@uestc.edu.cn,johnsonzxs@uestc.edu.cn,lixiong@uestc.edu.cn, caosheng@uestc.edu.cn)

Y. Mu is with the Institute of Data Science, City University of Macau, Macao, 999078, China. (e-mail: ymu.ieee@gmail.com)

F. Rezaeibagha is with the Discipline of Information Technology, Murdoch University, Perth, WA 6150, Australia (e-mail: fatemeh.rezaeibagha@murdoch.edu.au)

privacy, efficiency, etc. Sun et al. [10] utilized a cryptographic accumulator to reduce the storage space of the original RingCT, and proposed RingCT 2.0 as an enhancement. As another research direction, some works focused on revealing the traceability of Monero by both theoretical analysis and practical approaches [11]. The vulnerabilities found in Monero and its relevant aspects impede researchers to improve the RingCT protocol or to build different cryptocurrencies with native untraceable transaction support, like Zerocoin, Zerocash, Stealthcoin, etc.

Despite the traceability problem revealed by recent analytic techniques [11], the vulnerabilities in Monero's immutable blockchain are more critical. Here, blockchain refers to the public distributed trust ledger used to record transactions in Bitcoin-like cryptocurrencies. The immutability property of Bitcoin-alike blockchain remains the cornerstone of blockchain's security. It plays an indispensable role in most of the coin design and blockchain-driven applications. Simply, it means the blockchain used to record transactions is generally append-only and tamper-resistant. However, this also brings difficulties to undo any errors or misbehaviors committed to the blockchain. According to current surveys [12], [13], we point out two cases which are difficult for immutable blockchain to avoid:

- **Unwanted contents [14].** According to Matzutt et al. [15], 1.4% of about 251 million transactions in Bitcoin blockchain carry some arbitrary data. These data could make the blockchain unscalable.
- **Consequences of powerful and practical attacks.** As reported by Ali et al. [13], many newly launched blockchains are vulnerable to 51% attacks due to inadequate network computing power. In addition to powerful attacks, sophisticated attacks may arise from various aspects of the practical environment.
- **General concerns.** Permanent and public storage may rise concerns from various aspects. It could lead to malware distribution platform [16], abuse of storage contents [17], invalidation of Bitcoin 2.0 [17], failure in scaling smart contract [17], etc.

To address the above listed problems in the Monero system practically, in this work, we propose redactable Monero (RM). The system framework of our scheme is given in Figure 1. As it is shown in Figure 1, all users and payments are anonymous in our setting. A group of tracing authorities can be summoned to trace payer's real identity collectively while a system manager is enabled to perform multi-grained redaction on Monero's blockchain. Our main contributions can be summarized as follows:

1) We systematically review the relevant works. We provide extensive comparison and review of related works based on the desired redaction properties. System model, threat model and rigorous security model for dominant securities are provided.
2) We give a concrete construction of redactable Monero (RM). Our scheme mainly presents four building blocks to take effect. To our knowledge, our proposal is the first to introduce redaction to actual privacy-preserving cryptocurrencies. User anonymity is achieved. What is more, our proposal achieves most of the desired properties for redacting blockchain in comparison with recent works.
3) We prove the security of our scheme based on rigorous security models. Through theoretical and experimental evaluations, we found some minor unscalabilities in large-scale implementations of our scheme. The main scheme is acceptably efficient.

2 Related Works

In EuroS&P2017, Ateniese et al. [17] proposed the notion of the redactable blockchain (also known as mutable blockchain [12]), which is based on the chameleon hash to build the editable blockchain. The idea of utilizing multi-party computation to achieve decentralized redaction is popular, and has been followed by many recent works. However, there are still some unresolved problems, for example anonymity, fine-grained redaction, practical application, etc. To solve these problems, various schemes have been proposed [14], [18]–[24].

We summarize the desired properties in recent works and compare them with ours in Table 1. The most recent work which caught our attention is Xu et al.'s work [25], in which they proposed a generic construction for their redactable blockchain with formal proofs, superior functionalities and performances. For a systematic review, Politou et al. [18] proposed a review article to list challenges and developments for redactable blockchain.

As it is shown in Table 1, majority of the listed works adopted chameleon hash to build the redactable blockchain. These works mainly target fine-grained redaction [20], [21], [25] or decentralized redaction [18]–[20], [22] via a unique design or smart application of chameleon hash function. Another popular method is the adoption of secure multi-party computation protocol [19] or threshold cryptography [21] to achieve redaction. However, few works have applied their designs with an actual cryptocurrency [14], [17], [18], [23]. Also, few of them have achieved anonymity [14], [17], [18], [21] (regardless of user anonymity or payment anonymity). Although it is rather simple to integrate the Bitcoin protocol with other cryptographic primitives, few existing works have done it.

In fact, researchers are more interested in dealing with privacy-related [26], [27] and security-related topics [28], [29] topics rather than applying cryptographic techniques directly to actual cryptocurrencies. For example, Guo et al. [21] considered anonymous outsourced computation. Matzutt et al. [14] considered that objectional contents were anonymously inserted and distributed in blockchain. Differently, Tian et al. [22] considered accountable transactions, which are supposed to make all redaction operations transparent to the verifier. Deuber et al. [18] recently have achieved a formal design and rigorous analysis on redactable blockchain. Among the above works, to note, these two [18], [25] works are currently the most well designed and analyzed works on this topic. However, except for our work, none of the above works has achieved all the desired properties listed in Table 1, particularly instantiating with an actual and anonymous cryptocurrency.

TABLE 1: Review of relevant works

Type	Scheme	Contribution	Key component	Properties of redaction (mutability)					
				Security analysis	Anonymity	Efficiency	Fine-grained	Decentralization	Applying with coin
CS	Ateniese et al. [17]	Conceptual design, security model, generic transformation	Chameleon Hash	✓	✓	Ideal	✗	✓	Bitcoin
	Deuber et al. [18]	Protocol design, formal analysis and implementation	Consensus-based voting	✓	✓	Ideal	✗	✓	Bitcoin
	Ashritha et al. [19]	Scheme design and technical review	Chameleon hash multi-party computations	✗	✗	✗	✗	✓	✗
	Derler et al. [20]	Generic construction and rigorous analysis	Chameleon hash	✓	✗	Ideal	✓	✓	✗
	Guo et al. [21]	Schematic design, security and efficiency analysis	Chameleon hash and ring signature	✓	✓	Optimized to be ideal	✓	✗	✗
	Tian et al. [22]	Schematic design, security and efficiency analysis	ABE and chameleon hash	✓	✗	Acceptable	✗	✓	✗
	Xu et al. [25]	Formal proofs, generic construction	Chameleon hash, signatures	✓	✗	Ideal	✓	✗	✗
	Our scheme	Formal modelling, security analysis, instantiation	Multiple cryptographic modules	✓	✓	Acceptable	✓	✓	Monero
NCS	Matzutt et al. [14]	Conceptual countermeasures and technical discussion	Content detector	✗	✓	Ideal	✗	✗	Bitcoin
	Puddu et al. [23]	Design, instantiation, and implementation	Structure design of mutable transaction	✗	✗	Not ideal	✓	✗	Hyperledger Fabric
	Xu et al. [24]	Generic Design, formal modelling, security analysis	Generic design of protocol	✓	✗	Ideal	✗	✗	✗

Denote ✓ as satisfied; ✗ as not satisfied; CS as cryptographic scheme; NCS as non-cryptographic scheme; ABE as attribute-based encryption.

3 DEFINITIONS

In this section, we give the system components, threat model and security goals, intractable assumptions and definitions of building blocks.

3.1 System Components

- **Certificate authority (CA):** A trusted entity to assign public and private keys to each user.
- **Tracing authority:** Trusted entity. No less than k_0 out of n_0 members of tracing authorities can collectively trace user's identity.
- **User:** Untrusted and anonymous entity who wishes to spend, receive or mint Monero coins. User's identity is concealed but can be revealed by tracing authority.
- **System manager:** Trusted entity that initiates the whole system. Simply, we assume it controls the private key to perform all redaction operations.

3.2 Threat Model and Security Goals

Similar to Monero [10], security threats to redactable Monero include double-spending, anonymity attack, etc [9]. We employ the P2P network of Monero for our redactable Monero. So, we mainly focus on two types of active attacks. (1). **Tracing Attack.** Several tracing authorities are corrupted to reveal the public key of an actual payer in an anonymous payment. (2). **Redaction Attack.** Someone without private key may try to forge valid redaction proof in order to manipulate blockchain. In addition, redaction result cannot be accepted by the majority unless it has been properly implemented to all level of data (from fine-grained to coarse-grained) according to redaction policy. Following security goals should be satisfied.

- **Threshold Tracing Protection:** Even if at most $(k_0 - 1)$-out-of-n_0 tracing authorities (corresponding private keys) have been corrupted, they cannot collectively reveal the true identity of a spender. If k_0 is large enough (e.g. close to the total number of nodes in our P2P network) and the majority is honest, tracing attack is hard succeed.
- **Multi-Grained Redaction Protection:** Redaction should be performed with permission and guidelines (i.e. using master key and following a redaction policy), and from fine-grained to coarse-grained manner. The first condition requires a secure redaction protocol for validation. The second condition considers the basic correctness property of blockchain mining. If a Monero chain is supposed to be altered, its underlying block (transaction, commitment and accumulator) should be altered as well, otherwise, it cannot be authenticated and accepted by the majority nodes. So, adversary cannot successfully alter a chain without controlling all trapdoor keys or breaking every level of redaction procedure.

3.3 Group, Pairing and Intractability

Given two multiplicative groups $\mathbb{G}$, $\mathbb{G}_T$ with the same order p. Denote g as a generator of $\mathbb{G}$. Let $\hat{e}$ be a symmetric bilinear pairing $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ s.t. (1) Bilinearity: $\forall X, Y \in \mathbb{G}$, $a, b \in \mathbb{Z}_p^*$ s.t. $\hat{e}(X^a, Y^b) = \hat{e}(X, Y)^{ab}$; (2) Non-degeneracy: $\exists X, Y \in \mathbb{G}$ s.t. $\hat{e}(X, Y) \neq 1$; (3) Computability: $\forall X, Y \in \mathbb{G}$, to compute $\hat{e}(X, Y)$ is efficient.

Definition 1. **Discrete Logarithm Problem (DLP).** Given $g^a \in \mathbb{G}$ where $a \in \mathbb{Z}_p^*$, to compute a is hard.

Definition 2. **Decisional Diffie-Hellman Problem (DDHP).** Given $< g, g^a, g^b, g^c \in \mathbb{G}^4 >$ where $a, b, c \in \mathbb{Z}_p^*$, to compute $c = ab \mod p$ is hard.

3.4 Building Blocks

We briefly formalize some notions for our $\mathsf{R}^2\mathsf{CT}$. Instantiation for each building block is discussed in section 5.3.

3.4.1 Signature of Knowledge

Signature of Knowledge for an NP-relation $\mathcal{R}$ with the corresponding language $L = \{y : \forall x_{\mathsf{SoK}}, s.t.(x_{\mathsf{SoK}}, y_{\mathsf{SoK}}) \in \mathcal{R}\}$ consists of the following algorithms. Important securities include SimExt-security (simulatability and extraction), correctness, simulatability and extraction properties. Refer to [30] for more details.

- **SoKGen**. On input a security parameter λ, output public parameter par.
- **SoKSign**. On input a message m and a pair $(x_{\mathsf{SoK}}, y_{\mathsf{SoK}}) \in \mathcal{R}$, output a SoK η.
- **SoKVerf**. On input a message m, a SoK η and a statement y_{SoK}, output 0 or 1.

3.4.2 Trapdoor Commitment

Trapdoor commitment (TC) [31] is a trapdoor analogue of commitment scheme, it allows the holder of private key to produce a decommitment to an arbitrary message. Specifically, in our scheme, this allows the system manager to alter an anonymous payment without asking the actual payer to open corresponding coin key or to alter SoK. A secure TC should satisfy simulation-sound binding and hiding properties [31].

- **TC-Setup**. On input a security parameter λ, output a public key pk_{TC} and private key sk_{TC}.
- **TC-Com**. On input pk_{TC}, a message m and a tag $\overline{tag}$, output a commitment c and a decommitment Dec.
- **TC-Ver**. On input c, m, $\overline{tag}$ and d, output 0 or 1.
- **TC-Fake**. On input pk_{TC}, sk_{TC} and $\overline{tag}$, output (c', ξ). c' is a commitment and ξ is auxiliary information.
- **TC-Decom**. On input auxiliary information ξ and message m, output a decommitment d'.

3.4.3 Trapdoor Accumulator

Trapdoor accumulator (TA) [32] is a trapdoor analogue of cryptographic accumulator. It allows the holder of private key to alter an accumulated value without invalidating its proof. Consequently, this enables system manager to manipulate input accounts and alter a transaction while still letting it to pass verification and maintain anonymity. Trapdoor accumulator can be viewed as a relaxed and flexible version of dynamic accumulator as it does not require handling each witness for changed element [32]. A a secure TA should satisfy indistinguishability and collision resistance [32].

- **TA-Gen.** On input the security parameter λ, it outputs the key pair $(sk_{\mathsf{TA}}, pk_{\mathsf{TA}})$.
- **TA-Dig**. On input the set $\mathcal{S}$ to accumulate, the public parameters pk_{TA}, it outputs accumulator value acc.
- **TA-Proof**. On input the secret key sk_{TA}, acc, and an element v within acc, it returns a witness $\mathbf{p}$ for v. Output $\mathbf{p}$.
- **TA-Verf**. On input the public key pk_{TA}, acc, a witness $\mathbf{p}$, and a value v, it outputs 0 or 1, indicating whether $\mathbf{p}$ is a valid witness for v and acc.

3.4.4 Threshold Public-Key Encryption

A k_0-out-of-n_0 threshold public-key encryption [33] (TPKE) is a threshold analogue of public-key encryption. It allows no less than k_0 out of n_0 designated parties to collectively decrypt a message. A secure k_0-out-of-n_0 TPKE should satisfy the properties of confidentiality and decryption consistency [33].

- **TPKE-Setup**. On input a positive number n_0 and threshold number $k_0 < n_0$ and a security parameter λ, output a tuple $(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, sk_{\mathsf{TPKE}})$ where pk_{TPKE} is the public key, vk_{TPKE} is the verification key and $sk_{\mathsf{TPKE}} = \{sk_1, \cdots, sk_{n_0}\}$ is a vector of n_0 private key slices.
- **TPKE-Enc**. On input pk_{TPKE} and a message M, output a TPKE ciphertext C.
- **TPKE-Dec**. On input pk_{TPKE}, index i, private key slice $sk_{i,\mathsf{TPKE}}$ and ciphertext C, output a decryption partition (share) C_i with its index i. If $C_i = \perp$, it signifies invalid decryption partition.
- **TPKE-Ver**. On input a public key pk_{TPKE}, verification key vk_{TPKE}, ciphertext C and decryption partition (i, C_i), output 0 or 1.
- **TPKE-Comb** On input pk_{TPKE}, vk_{TPKE}, ciphertext C and a set of decryption partitions $\{(1, C_1), \cdots, (k_0, C_{k_0})\}$, output a plaintext M or $\perp$.

3.4.5 Collision-Resilient Chameleon Hash

Collision-resilient chameleon hash CH is a trapdoor analogue of cryptographic hash function (e.g. SHA-256 used in blockchain). It allows the holder of trapdoor key to produce a hash collision under a customized identity [34]. In the spirit of Ateniese et al.'s work [17], this allows the system manager to efficiently redact the block content without causing any hashing inconsistency. A secure CH should satisfy collision resistance and indistinguishability [17], [35].

- **CH-Setup**. On input a security parameter λ, output system parameter $\mathsf{param}_{\mathsf{CH}}$.
- **CH-KeyGen**. On input $\mathsf{param}_{\mathsf{CH}}$, output a public key pk_{CH} and private key sk_{CH}.
- **CH-Hash**. On input pk_{CH}, a customized identity CID and message m, output chameleon hash $\hbar$ and chameleon randomness $\tilde{r}$.
- **CH-Ver**. On input pk_{CH}, CID and a tuple $(m, \hbar, \tilde{r})$, output 0 or 1.
- **CH-Forge**. On input sk_{CH}, CID, a tuple $(m, \hbar, \tilde{r})$, run $\mathsf{CH\text{-}Verify}(pk_{\mathsf{CH}}, \mathsf{CID}, (m, \hbar, \tilde{r})$ to check validity. If output is 0, return $\perp$ to signify an error; otherwise, output a new chameleon randomness $\tilde{r}'$.

4 Security Requirements

In this section, we give an overview of our redactable RingCT protocol ($\mathsf{R^2CT}$) and dominant security requirements.

4.1 Framework of $\mathsf{R^2CT}$

A redactable RingCT protocol is specified by the following algorithms. Refer section 5.2 for more details.

- **$\mathsf{R^2CT}$-Setup**. On input a security parameter λ, output system parameter $pp_{\mathsf{R^2CT}}$ and master key $mk_{\mathsf{R^2CT}}$.

- **R²CT-KeyGen**. On input $pp_{\mathsf{R^2CT}}$, output public key pk and corresponding private key sk for user.
- **R²CT-Mint**. On input $pp_{\mathsf{R^2CT}}$ and pk, output a coin cn and coin key ck.
- **R²CT-Spend**. On input a transaction string m, a set of secret keys K_s, a set A of groups of input accounts containing input accounts A_s, a set of target output addresses R and and public key of TPKE pk_{TPKE}, output a transaction tx, a signature of knowledge SoK η, a set of serial numbers S and a TPKE ciphertext C.
- **R²CT-Verify**. On input a transaction tx, A and A_R, a SoK η for tx, a set of serial numbers S and TPKE ciphertext C, output 0 or 1.
- **R²CT-Trace**. On input a private key $sk_{i,\mathsf{TPKE}}$ of tracing authority AU_i, a verification key vk_{TPKE}, transaction tx and TPKE ciphertext C, output $\perp$ to indicate failure or (i, C_i) as a tracing proof.
- **R²CT-Judge**. On input a public key pk_{TPKE}, a verification key vk_{TPKE}, a transaction tx and k_0 decryption partitions $\{(1, C_1), \cdots, (k_0, C_{k_0})\}$, return $\perp$ or payer's public key pk_s.
- **BlockGen**. On input $pp_{\mathsf{R^2CT}}$, some authenticated transactions $\{\hat{\sigma}(tx)\}$ and a block blk_N, output new nonce and new block (non_{N+1}, blk_{N+1}).
- **BlockVer**. On given blk_N and non_N, return 0 to reject or 1 to accept.
- **ChainRedact**. On input a chain $\mathcal{C}_N$ of N blocks, $pp_{\mathsf{R^2CT}}$ and private key sk_{CH} of CH, output $\perp$ or a new chain $\mathcal{C}'$. Sub-algorithm for block insertion, modification and deletion are defined as follows:

  - **Append**. On input $\mathcal{C}_N$, a new block blk', position $N-i$, output the new chain $\mathcal{C}'_{N+1}$.

  - **Modify**. On input $\mathcal{C}_N$, blk' and a position $(N-i)$, output the new chain $\mathcal{C}'_{N+1}$.

  - **Shrink** On input a chain $\mathcal{C}_N$ and position $N-i$, output $\mathcal{C}'_{N-1}$.
- **TranRedact**. On input a signature $\hat{\sigma}(tx)$ on a transaction tx, a private key sk_{CH}, output $\perp$ or redacted signature $\hat{\sigma}(tx')$.
- **AccRedact**. On input an old accumulator and some auxiliary information, return a new witness for validation.
- **CommitRedact**. On input an old coin and some auxiliary information, return the decommitment used for spending the coin.

A basic security requirement for our R²CT is correctness, that is, if every party follows the scheme as negotiated, then any failure would not occur during the scheme. Formally, for any $(pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}}) \leftarrow \mathsf{R^2CT\text{-}Setup}(\lambda)$, the following three cases should be satisfied:

- **Case1: Spending Correctness:** For any key pair $(pk, sk) \leftarrow \mathsf{R^2CT\text{-}KeyGen}(pp_{\mathsf{R^2CT}})$, any coin $(cn, ck) \leftarrow \mathsf{R^2CT\text{-}Mint}(pp_{\mathsf{R^2CT}}, pk)$ and any output from spending algorithm $(tx, \eta, S, C) \leftarrow \mathsf{R^2CT\text{-}Spend}(\cdot, \cdot, \cdot, \cdot, \cdot, pk_{\mathsf{TPKE}})$, if $(tx', \eta', S', C') = (tx, \eta, S, C)$, then the verification algorithm $\mathsf{R^2CT\text{-}Verify}(tx', \eta', S', C')$ always outputs 1.
- **Case2: Tracing Correctness:** Following Case1, for fixed k_0, n_0 and any set of no less than k_0 tracing proofs: $\{(1, C_1), \cdots, (k_0, C_{k_0}), \cdots\}$. If each (j, C_j) is honestly generated, i.e. $\mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C, (j, C_j)) = 1$ and no two decryption shares (j, C_j) and (k, C_k) bear the same tracing authority's index i, then the judging algorithm $\mathsf{R^2CT\text{-}Judge}(\cdot, \cdot, \cdot, \{(1, C_1), \cdots, (k_0, C_{k_0})\})$ should always output actual spender's public key pk_s.
- **Case3: Redaction Correctness:** Following Case1 and Case2, for any old chain $\mathcal{C}$ where $(non, blk) \leftarrow \mathsf{BlockGen}((pp_{\mathsf{R^2CT}}, \{\hat{\sigma}(tx)\}, blk))$, if $\mathcal{C}'$ is redacted from $\mathcal{C}$ according to algorithms ChainRedact, TranRedact, AccRedact, CommitRedact, then both $\mathsf{BlockVer}(blk, non)$ and $\mathsf{BlockVer}(blk', non')$ should always output 1.

Case1 is straightforward. It indicates that any R²CT payment in (tx, η, S, C) format made by an anonymous payer can be publicly verified and successfully spent. Case2 suggests that no less than k_0 out of n_0 tracing authorities together can always trace the payer's public key in an anonymous payment. Case3 means that any old chain $\mathcal{C}$ generated by normal mining procedure and its redacted result $\mathcal{C}'$ should both pass verification and maintain hashing consistency. We let a redaction policy to dictate which chain is to be finally accepted by the majority miners.

4.2 Formal Security Definitions

Definition 3. **(Security of R²CT).** In addition to correctness, our R²CT should satisfy balance, anonymity, traceability and redaction soundness. We only formalize traceability in this part, the rest is straightforward from [9], [10].

Traceability. Traceability asks that: (1). $k_0 > 0$ designated authorities together can efficiently trace the payer's public key for a transaction. (2). The corresponding tracing proofs are hard to forge even if the adversary has corrupted $k_0 - 1$ authorities. The definition of traceability is based on experiment $\mathbf{Exp}^{\mathsf{Trace}}_{\mathsf{R^2CT}}(\lambda)$. Traceability holds for R²CT if for any PPT adversary $\mathcal{A}$, his winning advantage in experiment $\mathbf{Exp}^{\mathsf{Trace}}_{\mathsf{R^2CT}}(\lambda)$ is negligible, i.e.

$$\mathbf{Adv}^{\mathsf{Trace}}_{\mathcal{A},\mathsf{R^2CT}}(\lambda) = \left|\mathsf{Pr}[\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{A},\mathsf{R^2CT}}(\lambda) = 1]\right| \leq negl(\lambda).$$

We proceed to define the traceability security as follows.

Setup: With the security parameter λ, k_0 as the minimum number of tracing authorities and n_0 as the maximum number of tracing authorities, the challenger runs the R²CT-Setup algorithm to generate system parameters $pp_{\mathsf{R^2CT}}$ and master key $mk_{\mathsf{R^2CT}}$. It gives $pp_{\mathsf{R^2CT}}$ to adversary $\mathcal{A}$ and keeps $mk_{\mathsf{R^2CT}}$ secret. $\mathcal{A}$ picks a set of k_0-1 tracing authorities to corrupt. Let $\tilde{S}' = \{j_1, \cdots, j_{k_0-1}\} \subset \{1, \cdots, n_0\}$ be a set of indexes for corrupted authorities. $\mathcal{A}$ announces $\tilde{S}'$.

Query Phase 1: The adversary $\mathcal{A}$ issues queries to oracles $\mathcal{O}_{\mathsf{AddGen}}$, $\mathcal{O}_{\mathsf{ActGen}}$, $\mathcal{O}_{\mathsf{Spend}}$, $\mathcal{O}_{\mathsf{Corrupt}}$ as defined in experiment $\mathbf{Exp}^{\mathsf{Trace}}_{\mathsf{R^2CT}}(\lambda)$. To respond, the challenger runs $\mathsf{R^2CT\text{-}Setup}(\lambda, k_0, n_0)$ to initiate TPKE and

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3251735

$\mathbf{Exp}^{\mathsf{Trace}}_{\mathsf{R^2CT}}(\lambda)$
$(pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}}) \leftarrow \mathsf{R^2CT\text{-}Setup}(\lambda, k_0, n_0)$
$\tilde{S}' \leftarrow \mathcal{A},\ \tilde{S}' \subset [n_0], |\tilde{S}'| = k_0 - 1;$
$\mathcal{L}_{\mathsf{AddGen}}, \mathcal{L}_{\mathsf{ActGen}}, \mathcal{L}_{\mathsf{Spend}}, \mathcal{L}_{\mathsf{Corrupt}}, \mathcal{L}_{\mathsf{Trace}} \leftarrow \emptyset;$
$((tx', \eta', S', C'), \{i, C'_i\}_{i\in\tilde{S}''}, \tilde{S}'') \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AddGen}}, \mathcal{O}_{\mathsf{ActGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Corrupt}}, \mathcal{O}_{\mathsf{Trace}}}(pp_{\mathsf{R^2CT}}, \{(i, sk_{i,\mathsf{TPKE}}, C'_i)\}_{i\in\tilde{S}'});$
return 1 if: $\mathsf{R^2CT\text{-}Verify}(tx', \eta', S', C') = 1 \wedge$
$\forall i \in \tilde{S}', i' \in \tilde{S}'' :$
$\mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C', (i, C'_i)) =$
$\mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C', (i', C'_{i'})) = 1 \wedge$
$\mathsf{R^2CT\text{-}Judge}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx', \{(i, C'_i)\}_{i\in\tilde{S}'}) \neq$
$\mathsf{R^2CT\text{-}Judge}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx', \{(i', C'_{i'})\}_{i'\in\tilde{S}''}) \wedge$
$\forall i, j \in \tilde{S}', i \neq j : C'_i \neq C'_j \wedge$
$\forall i', j' \in \tilde{S}'', i' \neq j' : C'_{i'} \neq C'_{j'} \wedge$
$\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C', \{i, C'_i\}_{i\in\tilde{S}'}) \neq$
$\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C', \{i', C'_{i'}\}_{i'\in\tilde{S}''}) \wedge$
$C' \notin \mathcal{L}_{\mathsf{Trace}} \wedge (\mathsf{m}', A'_s, A') \notin \mathcal{L}_{\mathsf{Spend}},$
else, return 0.

$\mathbf{Oracle}\ \mathcal{O}_{\mathsf{AddGen}}(i)$
$\tau_i \xleftarrow{R} \mathbb{Z}_p, (sk_i, pk_i) \leftarrow$
$\mathsf{R^2CT\text{-}KeyGen}(pp_{\mathsf{R^2CT}}, \tau_i);$
$\mathcal{L}_{\mathsf{AddGen}} \leftarrow \mathcal{L}_{\mathsf{AddGen}} \cup \{i\};$
return pk_i.

$\mathbf{Oracle}\ \mathcal{O}_{\mathsf{ActGen}}(i, a_i)$
$(cn_i, ck_i) \leftarrow$
$\mathsf{R^2CT\text{-}Mint}(pp_{\mathsf{R^2CT}}, pk_i);$
$\mathcal{L}_{\mathsf{AddGen}} \leftarrow \mathcal{L}_{\mathsf{AddGen}} \cup (i);$
$\mathcal{L}_G \leftarrow \mathcal{L}_G \cup (act_i = (pk_i, cn_i));$
return (act_i, ck_i).

$\mathbf{Oracle}\ \mathcal{O}_{\mathsf{Corrupt}}(i)$
$\forall i \in \mathcal{L}_{\mathsf{AddGen}}, s_i \leftarrow f(ask_i, act_i);$
$\mathcal{L}_C \leftarrow \mathcal{L}_C \cup (s_i);$
$\mathcal{L}_B \leftarrow \mathcal{L}_B \cup (s_i, a_i);$
$\mathcal{L}_{\mathsf{Corrupt}} \leftarrow \mathcal{L}_{\mathsf{Corrupt}} \cup \mathcal{L}_C \cup \mathcal{L}_B;$
return τ_i.

$\mathbf{Oracle}\ \mathcal{O}_{\mathsf{Spend}}(\mathsf{m}, A_s, A)$
$(tx, \eta, S, C) \leftarrow$
$\mathsf{R^2CT\text{-}Spend}(\mathsf{m}, K_s, A_s, A, R);$
$\mathcal{L}_{\mathsf{Spend}} \leftarrow \mathcal{L}_{\mathsf{Spend}} \cup (\mathsf{m}, A_s, A);$
return (tx, η, S, C).

$\mathbf{Oracle}\ \mathcal{O}_{\mathsf{Trace}}(tx, \eta, C)$
$(pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}}) \leftarrow \mathsf{R^2CT\text{-}Setup}(\lambda, k_0, n_0);$
$\forall i \in [k_0],\ (i, C_i) \leftarrow \mathcal{O}_{\mathsf{Dec}}(C, i);$
$pk_s \leftarrow \mathsf{R^2CT\text{-}Judge}(\cdots, tx, \{(1, C_1), \cdots, (k_0, C_{k_0})\});$
$\mathcal{L}_{\mathsf{Trace}} \leftarrow \mathcal{L}_{\mathsf{Trace}} \cup (tx, \eta, C);$
return pk_s if $pk_s \neq \perp \wedge \mathsf{SoKVerf}(\cdots, \eta) = 1.$

Notations: $\tilde{S}'$ is a set of public keys for designated tracing authorities. $\mathcal{O}_{\mathsf{Dec}}$ is the decryption oracle of TPKE as defined in [33]. i in $\mathcal{O}_{\mathsf{AddGen}}$ indicates a query number. a_i in $\mathcal{O}_{\mathsf{ActGen}}$ indicates an amount for an address.

get decryption share (i, C_i) from oracle $\mathcal{O}_{\mathsf{Dec}}$. $\mathcal{O}_{\mathsf{Dec}}$ is a decryption oracle for TPKE as defined in [33] (we will review security model of TPKE in section 6). With k_0 decryption shares, the challenger runs algorithm $\mathsf{R^2CT\text{-}Judge}(\cdots, tx, \{(1, C_1), \cdots, (k_0, C_{k_0})\})$ to reveal spender's public key pk_s in tx. Accordingly, the challenger records (tx, η, C) and return pk_s to $\mathcal{A}$ if it is a valid key and corresponding SoK can pass verification.

Challenge: For each $i \in \tilde{S}'$, the challenger generates $(i, sk_{i,\mathsf{TPKE}}, C'_i)$ by running TPKE-Setup, TPKE-Enc and querying $\mathcal{O}_{\mathsf{Dec}}$ as previously mentioned. It relays $\{(i, sk_{i,\mathsf{TPKE}}, C'_i)\}_{i\in\tilde{S}'}$ to $\mathcal{A}$ where $|\tilde{S}'| = k_0 - 1$ private keys have been corrupted. Apparently, C' can not been queried during Query Phase 1.

Query Phase 2: The same as Query Phase 1 except that C' should not be queried.

Guess: The adversary $\mathcal{A}$ outputs $(tx', \eta', S', C'), \{i, C'_i\}_{i\in\tilde{S}''}, \tilde{S}''$ where (tx', η', S', C') has not been queried and $\{i, C'_i\}_{i\in\tilde{S}''}$ are a new set of decryption shares under a new index set $\tilde{S}''$.

To win the experiment, two sets of decryption shares under $\tilde{S}'$ and $\tilde{S}''$ respectively should both pass verification, i.e. $\mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C', (i, C'_i)) = \mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C', (i', C'_{i'})) = 1$. In addition, some other restrictions should apply: (1) No identical decryption share should be found in $\tilde{S}'$ and $\tilde{S}''$. (2) The merged result from each different set of decryption shares should be different as well, this also implies the winning condition. (3) (tx', η', S', C') should not be queried to any oracles as mentioned above. If the adversary $\mathcal{A}$ managed to win the experiment, he basically found two different sets of decryption shares under same ciphertext which leads to different public key. This apparently violates the decryption consistency security of TPKE.

$\mathbf{Exp}^{\mathsf{BlockRedact}}_{\mathcal{A},\mathsf{BRP}}$
$(pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}}) \leftarrow \mathsf{R^2CT\text{-}Setup}(\lambda);$
$\mathcal{L}_{\mathsf{App}} \leftarrow \emptyset$; //Append query list
$\mathcal{L}_{\mathsf{Mod}} \leftarrow \emptyset$; //Modify query list
$\mathcal{L}_{\mathsf{Del}} \leftarrow \emptyset$; //Shrink query list
$\mathcal{C}'^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{App}}, \mathcal{O}_{\mathsf{Mod}}, \mathcal{O}_{\mathsf{Shr}}}(pp_{\mathsf{R^2CT}}, \mathcal{C}^*);$
return 1 if $\forall blk^*_j \in \mathcal{C}^* \cup \mathcal{C}'^*$:
$\mathsf{BlockVer}(blk^*_j, non^*_j) = 1 \wedge$
$blk^*_j \notin \mathcal{L}_{\mathsf{App}}, \mathcal{L}_{\mathsf{Mod}}, \mathcal{L}_{\mathsf{Shr}} \wedge$
$|\mathcal{C}^*| > 1 \wedge |\mathcal{C}'^*| > 1,$
else, return 0.

$\mathbf{Oracle}\ \mathcal{O}_{\mathsf{Shr}}(\mathcal{C}_N, i)$
$(pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}}) \leftarrow \mathsf{R^2CT\text{-}Setup}(\lambda)$
$\mathcal{C}'_{N-1} \leftarrow \mathsf{Modify}(\mathcal{C}_N, i, sk_{\mathsf{CH}});$
$\mathcal{L}_{\mathsf{Shr}} \leftarrow \mathcal{L}_{\mathsf{Shr}} \cup (\mathcal{C}_N, i);$
return $\mathcal{C}'_{N-1}$ *if* $\mathcal{C}'_{N-1} \neq \perp$.

$\mathbf{Oracle}\ \mathcal{O}_{\mathsf{App}}(\mathcal{C}_N, blk', i)$
$(pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}}) \leftarrow$
$\mathsf{R^2CT\text{-}Setup}(\lambda);$
$\mathcal{C}'_{N+1} \leftarrow$
$\mathsf{Append}(\mathcal{C}_N, blk', i, sk_{\mathsf{CH}});$
$\mathcal{L}_{\mathsf{App}} \leftarrow \mathcal{L}_{\mathsf{App}} \cup (\mathcal{C}_N, blk', i);$
return $\mathcal{C}'_{N+1}$ *if* $\mathcal{C}'_{N+1} \neq \perp$.

$\mathbf{Oracle}\ \mathcal{O}_{\mathsf{Mod}}(\mathcal{C}_N, blk', i)$
$(pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}}) \leftarrow$
$\mathsf{R^2CT\text{-}Setup}(\lambda);$
$\mathcal{C}'_{N+1} \leftarrow$
$\mathsf{Modify}(\mathcal{C}_N, blk', i, sk_{\mathsf{CH}});$
$\mathcal{L}_{\mathsf{Mod}} \leftarrow \mathcal{L}_{\mathsf{Mod}} \cup (\mathcal{C}_N, blk', i);$
return $\mathcal{C}'_N$ *if* $\mathcal{C}'_N \neq \perp$.

Definition 4. **Redaction Soundness**. All redactions should be performed without causing any hashing inconsistency or failure in validation. This means no malicious user can efficiently change the contents of blockchain at any level without causing invalidation or holding corresponding trapdoor key. Due to space limitation, we only formalize security model for block-level redaction soundness. Based on it, the rest is straightforward.

Each block-level redaction should be performed without causing any hashing inconsistency. The definition of block-level redaction soundness is based on experiment $\mathbf{Exp}^{\mathsf{BlockRedact}}_{\mathcal{A},\mathsf{BRP}}$. It holds for BRP if for any PPT adversary $\mathcal{A}$, his winning advantage in experiment $\mathbf{Exp}^{\mathsf{BlockRedact}}_{\mathcal{A},\mathsf{BRP}}$ is negligible, i.e.

$$\mathbf{Adv}^{\mathsf{BlockRedact}}_{\mathcal{A},\mathsf{BRP}}(\lambda) = \left|\mathsf{Pr}[\mathbf{Exp}^{\mathsf{BlockRedact}}_{\mathcal{A},\mathsf{BRP}}(\lambda) = 1]\right| \leq negl(\lambda).$$

5 Proposed Redactable Monero

In this section, we give the construction of redactable Monero with several building blocks. We give brief discussion on instantiation in section 5.3.

5.1 System Overview

A highlight of $\mathsf{R^2CT}$ is the multi-level redaction on Monero's ring confidential transaction (RingCT) protocol. The basic approach to achieve this is to introduce trapdoor cryptography so that designated entity can reverse Monero payment without invalidating the verification. From a technical point of view, we follow Ateniese et al.'s idea [17] to achieve redactable blockchain with chameleon hash, we denote block and transaction-level redaction as coarse-grained redaction. We also employ trapdoor commitment [31] and trapdoor accumulator [32] to guarantee the successful verification of each Monero payment after coarse-grained redaction, i.e. fine-grained redaction. To avoid abuse of anonymity property, we also introduce threshold public-key encryption TPKE to trace spender's public key in a completely anonymous payment. Ideally, threshold cryptography can also be applied to our trapdoor primitives to avoid abuse of redaction power, an example is [36]. The workflow of the system architecture is given as below.

- The system manager delegates n_0 tracing authorities during system setup and sets a threshold k_0 as minimum votes for tracing. Each designated tracing authority AU_i is assigned with a private key $sk_{i,\mathsf{TPKE}}$ for tracing. For ease of discussion, we assume master key $mk_{\mathsf{R^2CT}} = \{sk_{\mathsf{TC}}, sk_{\mathsf{TA}}, sk_{\mathsf{TPKE}}\}$ is controlled by system manager. But sk_{TPKE} is actually controlled by each tracing authority AU_i. On deriving user's key pair from $\mathsf{R^2CT\text{-}KeyGen}$, user can mine, spend and verify a Monero coin similarly as the original RingCT 2.0 protocol [10].
- For generality, we capture mining by $\mathsf{BlockGen}$ and $\mathsf{BlockVer}$. It replaces the original hash function (e.g. SHA-256) used in blockchain with a collision-resilient chameleon hash CH. To reverse a remote payment history on a chain, system manager should start from locating the target block first. Then proceed from fine-grained (accumulator, commitment) to coarse-grained (transaction and block) redaction. To unify, we let a global redaction policy to dictate the execution sequence and acceptance criteria of redaction, i.e. writing it into smart contract.
- During redaction, if the recipient is changed, corresponding commitment and address should also be changed; otherwise, it is neither traceable nor verifiable. If we abort the payment and ban the payer's future payment, then all later payment from this account should be aborted. To achieve this, we can summon a public account to collect the forthcoming money. We can also consider automatic function like smart contract which runs redaction as negotiated with redaction policy. Ideally, this also helps off-load most of the computation intensive operations to a third party as the user associated with payment maybe off-line and computationally-limited.

5.2 Construction

Due to sophistication of presentation, concrete construction of each building block is separately discussed in section 5.3 and not shown in this part.

Notions. Suppose $A = \{pk^k_{in,i}, cn^k_{in,i}\}_{i\in[n],k\in[l]}$ indicate n groups of input accounts (including the group of l accounts the payer wishes to spend). Set payer's group as $A_s = \{pk^k_{in,s}, cn^k_{in,s}\}_{k\in[l]}$, i.e. the s-th group of A. We can arrange every account groups key in A into a matrix so that each group corresponds to a column. The payer only needs to prove that he uses one account of the same column where elements are accumulated into one element [10].

5.2.1 System Setup

On input a security parameter λ, the system manager initiates each building block. Noticeably, we let system manager to control redaction key: $sk_{\mathsf{TC}}, sk_{\mathsf{TA}}$, and each tracing authority AU_i to control its own tracing key $sk_{i,\mathsf{TPKE}}$.

- **$\mathbf{RC^2T\text{-}Setup}(\lambda, k_0, n_0) \rightarrow (pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}})$.** Pick two multiplicative groups $\mathbb{G}$ and $\mathbb{G}_T$ of the same order p. Denote g as the generator of group $\mathbb{G}$. Set $H : \{0,1\}^* \rightarrow \mathbb{G}$. Set $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Run $\mathsf{SoKGen}(\lambda) \rightarrow (\mathsf{par})$ to setup SoK. Run $\mathsf{TC\text{-}Setup}(\lambda) \rightarrow (pk_{\mathsf{TC}}, sk_{\mathsf{TC}})$ to initiate TC. Run $\mathsf{TA\text{-}Gen}(\lambda) \rightarrow (sk_{\mathsf{TA}}, pk_{\mathsf{TA}})$ to initiate TA. Run $\mathsf{TPKE\text{-}Setup}(n_0, k_0, \lambda) \rightarrow (pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, sk_{\mathsf{TPKE}})$ to initiate TPKE. Parse $sk_{\mathsf{TPKE}} = \{sk_{1,\mathsf{TPKE}}, \cdots, sk_{n_0,\mathsf{TPKE}}\}$. Run $\mathsf{CH\text{-}Setup}(\lambda) \rightarrow (\mathsf{param}_{\mathsf{CH}})$ to setup CH. Run $\mathsf{CH\text{-}KeyGen}(\mathsf{param}_{\mathsf{CH}}) \rightarrow (pk_{\mathsf{CH}}, sk_{\mathsf{CH}})$ to get trapdoor key sk_{CH} for CH. Randomly pick $h_0, h_1, u, \hat{h} \xleftarrow{R} \mathbb{G}$. Denote system parameter and master key as

$$pp_{\mathsf{R^2CT}} = \{\mathsf{par}, \mathbb{G}, \mathbb{G}_T, p, g, p, H, \hat{e}, h_0, h_1, u, \hat{h}, pk_{\mathsf{TC}}, pk_{\mathsf{TA}}, pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}\}$$

$$mk_{\mathsf{R^2CT}} = \{sk_{\mathsf{TC}}, sk_{\mathsf{TA}}, sk_{\mathsf{TPKE}}\}$$

5.2.2 Key Generation

On input $pp_{\mathsf{R^2CT}}$, CA runs this algorithm to issue a pair of key (pk, sk) to each user.

- **$\mathbf{R^2CT\text{-}KeyGen} \rightarrow (pk, sk)$.** CA random picks $x \xleftarrow{R} \mathbb{Z}_p$, $h_0 \in \mathbb{G}$. It keeps $sk = x$ as private key and computes $pk = h_0^x$ as public key. sk is kept private by user, pk is open to the public.

5.2.3 Mint

On input system parameter $pp_{\mathsf{R^2CT}}$ and a payer's public key pk, the user runs this randomized algorithm to mint a coin for a public key address with an amount a. This algorithm generates a coin cn and the corresponding coin key ck. Different from the mint algorithm in [10], our coin key can be an account a chosen by the user or an output generated from $\mathsf{TC\text{-}Decom}$ (enjoyed by the trapdoor property of TC) by system manager. For simplicity, we refer cn_{TC} as cn, and ck_{TC} as ck hereafter.

- **R²CT-Mint**$(pp_{\mathsf{R^2CT}}, pk) \rightarrow (cn_{\mathsf{TC}}, ck_{\mathsf{TC}})$. Given the public key pk as address and an amount a, the user randomly picks a tag $\overline{tag}$ and runs $\mathsf{TC\text{-}Com}(pk_{\mathsf{TC}}, a, \overline{tag}) \rightarrow (cn_{\mathsf{TC}}, ck_{\mathsf{TC}})$. For ease of discussion, we instantiate TC with Pedersen commitment, i.e. $\mathsf{TC\text{-}Com}(pk_{\mathsf{TC}}, a, \overline{tag}) = h_0^{\overline{tag}} h_1^a$.

5.2.4 Spend

On input a transaction string m, a set of secret keys K_s associated with the group of input accounts $A_s = \{pk^k_{in,s}, cn^k_{in,s}\}_{k\in[l]}$, an arbitrary set $A = \{pk^k_{in,i}, cn^k_{in,i}\}_{i\in[n],k\in[l]}$ of groups of input accounts containing A_s, a set $R = \{pk_{out,j}\}_{j\in[t]}$ as intended output addresses and tracing authority's public key pk_{TPKE}, user runs this algorithm to spend a coin and outputs a set (tx, η, S, C) including transaction tx, a SoK η, a set of serial numbers and TPKE ciphertext C. We mainly focus on the adaptations made from the original scheme [10] as follows

- **R²CT-Spend**$(\mathsf{m}, K_s, A_s, A, R, \cdot) \rightarrow (tx, \eta, S, C)$. Based on point 1 and 2 in section 5.1 of RingCT 2.0 [10], for all $i \in [n]$, we have $\widetilde{pk_i} \doteq y_i^{(l+1)}$ and $\widetilde{sk_s} \doteq x_s^{(l+1)}$. For $k \in [l+1]$ and $i \in [n]$, run $\mathsf{TA\text{-}Dig}(\lambda, pk_{\mathsf{TA}}, \mathcal{X} = \{y_i^{(k)} \cdot u^i\})$ to derive the accumulated value v_k. Run $\mathsf{TA\text{-}Proof}(\lambda, sk_{\mathsf{TA}}, v_k, \mathsf{v}) \rightarrow \mathbf{p}$ to generate a witness $\mathbf{p}$. Thus, others can run TA-Verf to check the validity of the accumulated value v_k. Denote $\delta \in [n]$ as the index of the actual payer in a group of n members. Run $\mathsf{TPKE\text{-}Enc}(pk_{\mathsf{TPKE}}, \delta) \rightarrow (C)$ to encrypt δ and derive the ciphertext C.

5.2.5 Verify

On input a transaction tx (includes A and A_R), the SoK η' for tx, a set of serial numbers S and TPKE ciphertext C, user verifies whether the transaction tx is properly spent (from input accounts A to output addresses R). It outputs 0 to reject or 1 to accept. Adaptations are:

- **R²CT-Verify**$(tx, \eta', S, C) \rightarrow (0 \text{ or } 1)$. For all $k \in [l]$, run TA-Verf to calculate the accumulated values. To verify the validity of received SoK η', run TA-Verf.

5.2.6 Trace

On input a private key $sk_{i,\mathsf{TPKE}}$ of tracing authority AU_i, a verification key vk_{TPKE}, transaction tx and TPKE ciphertext C, each tracing authority AU_i runs this algorithm to get a decryption share. It outputs $\perp$ to indicate failure or (i, C_i) as a tracing proof. Adaptations are as follows:

- **R²CT-Trace**$(sk_{i,\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx, C) \rightarrow (\perp \text{or} (i, C_i))$. A tracing authority AU_i $(i \in [k_0])$ runs TPKE-Dec to derive the index and corresponding decryption partition, i.e. $\mathsf{TPKE\text{-}Dec}(pk_{\mathsf{TPKE}}, i, sk_{i,\mathsf{TPKE}}, C) \rightarrow (i, C_i)$. Then, run $\mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C, (i, C_i))$ to check the validity of (i, C_i). If the output is 0, output $\perp$; otherwise, output (i, C_i).

On input a public key pk_{TPKE}, a verification key vk_{TPKE}, a transaction tx and k_0 decryption partitions $\{(1, C_1), \cdots, (k_0, C_{k_0})\}$, return $\perp$ or the exact spender's public key pk_s.

- **R²CT-Judge**$(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx, \{(1, C_1), \cdots, (k_0, C_{k_0})\}) \rightarrow (\perp \text{ or } pk_s)$. Run $\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C, \{(1, C_1), \cdots, (k_0, C_{k_0})\})$ to merge all k_0 partitions to recover δ. If output is $\perp$, return $\perp$ and terminate; otherwise, use δ to locate the public key of the payer pk_s in transaction tx from a group of keys.

5.2.7 Block Generation and Verification

On input $pp_{\mathsf{R^2CT}}$, some authenticated transactions $\{\hat{\sigma}(tx)\}$ (denote $\hat{\sigma}()$ as a digital signing scheme) in the P2P network and a block blk_N in N^{th} position of a chain $\mathcal{C}$ (counting from the leftmost block), user (miner) runs this algorithm during a Proof-of-Work [1] based mining procedure to output new nonce and new block (non_{N+1}, blk_{N+1}).

- **BlockGen**$(\cdot, \{\hat{\sigma}(tx)\}, blk_N) \rightarrow (non_{N+1}, blk_{N+1})$. Run CH-Hash by taking the N^{th} block blk_N and a freshly chosen customized identity CID_{N+1} as inputs to derive the chameleon hash of the $N+1^{th}$ block, i.e. $(\hbar_{blk_N}, \tilde{\tilde{r}}_N)$. Compute a proper nonce non_{N+1} to make the hash of the block less than some target value. Denote SIGN as a strongly-unforgeable signing scheme. Each transaction tx in a block is authenticated via SIGN, i.e. $\hat{\sigma}(tx) = \mathsf{SIGN}(tx)$.

On given a block blk_N and nonce non_N, any user can run this algorithm to verify the validity of a block on a chain. It returns 0 to reject or 1 to accept. Basic ideas are as follows:

- **BlockVer**$(blk_N, non_N) \rightarrow (0 \text{ or } 1)$. Check whether the hash value of block blk_N is less than the target value non_N. Run CH-Verify to check the validity of chameleon hash $\hbar_{blk_N}$ with chameleon randomness $\tilde{\tilde{r}}_N$. Check the validity of each signature $\hat{\sigma}$ included in the block blk_N.

5.2.8 Block-Level Redaction

On input a chain $\mathcal{C}_N = \{blk_i\}_{i\in[N]}$ of N blocks, $pp_{\mathsf{R^2CT}}$ and private key sk_{CH} of CH, system manager runs this algorithm to generate block redaction proof. It outputs $\perp$ or a new chain $\mathcal{C}'$. We give detailed sub-algorithm for block insertion, modification and deletion as follows:

- **ChainRedact**$(\mathcal{C}_N, pp_{\mathsf{R^2CT}}, sk_{\mathsf{CH}}) \rightarrow (\perp \text{ or } \mathcal{C}')$.
  - **Append**$(\mathcal{C}_N, blk', i, sk_{\mathsf{CH}}) \rightarrow \mathcal{C}'_{N+1}$. On input $\mathcal{C}_N$, a new block blk', position i and sk_{CH}, insert block blk' in the $(N-i)^{th}$ position of $\mathcal{C}_N$. Then, pick $\mathsf{CID}'_{N-i} \xleftarrow{R} \{0,1\}^*$. For $j = N-i$, run $\mathsf{CH\text{-}Hash}(pk_{\mathsf{CH}}, \mathsf{CID}_j, blk_{j-1}) = (\hbar_{blk_j}, \tilde{\tilde{r}}_j)$ where $\hbar_{blk_j}$ is the the hash of blk_j. For $j = N-i+1$, run $\mathsf{CH\text{-}Forge}(sk_{\mathsf{CH}}, \mathsf{CID}_j, (blk_{j-2}, \tilde{\tilde{r}}_j, \hbar_{blk_j}), blk') \rightarrow \tilde{\tilde{r}}'_j$. Accordingly, update $\mathcal{C}_{N+1}$. Output new chain $\mathcal{C}'_{N+1}$.
  - **Modify**$(\mathcal{C}_N, blk', i, sk_{\mathsf{CH}}) \rightarrow \mathcal{C}'_N$. On input a chain $\mathcal{C}_N$, a new block blk' to be modified to in the $(N-i)^{th}$ position of $\mathcal{C}_N$, position i and sk_{CH}. For $j = N-i+1$, run $\mathsf{CH\text{-}Forge}(sk_{\mathsf{CH}}, \mathsf{CID}_j, (blk_{j-1}, \tilde{\tilde{r}}_j, \hbar_{blk_j}), blk') \rightarrow \tilde{\tilde{r}}'_j$. Then, pick $\mathsf{CID}'_{N-i} \neq \mathsf{CID}_{N-i}$ and set $\mathsf{CID}_{N-i} \leftarrow \mathsf{CID}'_{N-i}$. Next, modify block blk_{N-i} to blk'. For $j = N-i$, run $\mathsf{CH\text{-}Hash}(pk_{\mathsf{CH}}, \mathsf{CID}_j, blk_{j-1}) = (\hbar_{blk_j}, \tilde{\tilde{r}}_j)$

where $\hbar_{blk_j}$ is the hash of blk_j. Accordingly, update $\mathcal{C}_{N+1}$. Output the new chain $\mathcal{C}'_{N+1}$.

- **Shrink**$(\mathcal{C}_N, i, sk_{\mathsf{CH}}) \rightarrow \mathcal{C}'_{N-1}$. On input a chain $\mathcal{C}_N$, position i and sk_{CH}. For $j = n - i + 1$, run $\mathsf{CH\text{-}Forge}(sk_{\mathsf{CH}}, \mathsf{CID}_j, (blk_{j-1}, \tilde{\tilde{r}}_j, \hbar_{blk_j}), blk_{j-2}) \rightarrow \tilde{\tilde{r}}'_j$. Then, delete the $(N-i)^{th}$ block blk_{N-i} from $\mathcal{C}_N$. Accordingly, update $\mathcal{C}_{N-1}$. Output new chain $\mathcal{C}'_{N-1}$.

5.2.9 Transaction-Level Redaction

On input a signature $\hat{\sigma}(tx) = \mathsf{SIGN}(\hbar) \parallel \tilde{\tilde{r}}$ on a transaction tx where $\mathsf{CH\text{-}Hash}(pk_{\mathsf{CH}}, \mathsf{CID}, tx) = (\hbar, \tilde{\tilde{r}})$, a private key sk_{CH} of CH and a new transaction tx', the system manager runs this algorithm to output $\perp$ or redacted signature $\hat{\sigma}(tx')$ as a proof to authenticate the redacted transaction.

- **TranRedact**$(\hat{\sigma}(tx), sk_{\mathsf{CH}}, tx') \rightarrow (\perp$ or $\hat{\sigma}(tx'))$. Run $\mathsf{CH\text{-}Forge}(sk_{\mathsf{CH}}, \mathsf{CID}, (tx, \tilde{\tilde{r}}, \hbar), tx') \rightarrow \tilde{\tilde{r}}'$ to generate a new chameleon randomness $\tilde{\tilde{r}}'$. Then, run CH-Verify to check the validity of $\hbar$. If output 0, output $\perp$ and terminate; else, output $\hat{\sigma}(tx') = \mathsf{SIGN}(\hbar) \parallel \tilde{\tilde{r}}'$. Clearly, $\hat{\sigma}(tx')$ can still pass authentication as the signature is signed on a chameleon hash $\hbar$ and $\hbar$ is not changed.

5.2.10 Accumulator-Level Redaction

On input an accumulated value v_k, private key sk_{TA} of TA, an element x to be changed and a witness π, the system manager runs this algorithm which outputs $\perp$ to terminate or $\mathbf{p}$ as a witness (i.e. proof) to validate a redacted accumulator.

- **AccRedact**$(v_k, sk_{\mathsf{TA}}, \mathrm{x}, \pi) \rightarrow (\perp$ or $\mathbf{p})$. Run $\mathsf{TA\text{-}Proof}(\lambda, sk_{\mathsf{TA}}, v_k, \mathrm{x}) \rightarrow \mathbf{p}$ to derive a witness $\mathbf{p}$ for x. Then, run $\mathsf{TA\text{-}Verf}(\lambda, pk_{\mathsf{TA}}, v_k, \mathrm{x}, \mathbf{p})$ to check the validity. If output is 0, return $\perp$ and terminate; else, return $\mathbf{p}$.

5.2.11 Commitment-Level Redaction

On input a coin cn_{TC}, corresponding coin key ck_{TC}, public key pk_{TC} of TC, private key sk_{TC} of TC and new tag $\overline{tag}'$, the system manager runs this algorithm to output $\perp$ or a decommitment dec' to validate a redacted commitment. With decommitment Dec′, the system manager can run $\mathsf{R^2CT\text{-}Spend}$ accordingly to spend an unspent coin or alter an spent payment as follows.

- **CommitRedact**$(cn_{\mathsf{TC}}, ck_{\mathsf{TC}}, pk_{\mathsf{TC}}, sk_{\mathsf{TC}}, \overline{tag}') \rightarrow (\perp$or Dec′). Run TC-Ver to check the validity of TC commitment cn_{TC}. If output 0, return $\perp$ and terminate; else, proceed. Pick a random $\overline{tag}' \neq \overline{tag}$ and run TC-Fake to derive a commitment c' and auxiliary information ξ. Then, run TC-Decom to derive the decommitment d'. Check whether $\mathsf{TC\text{-}Ver}(pk_{\mathsf{TC}}, c, a, \overline{tag}, d) = \mathsf{TC\text{-}Ver}(pk_{\mathsf{TC}}, c', a, \overline{tag}', d') = 1$ holds. If holds, output Dec′; otherwise, output $\perp$.

5.3 Instantiation with Minimizd Parameters

As several building blocks are involved in our $\mathsf{R^2CT}$ system, we next discuss about how to instantiate the construction and compress public parameters.

- **SoK:** The SoK can be instantiated by Fiat-Shamir paradigm [37] included in zero-knowledge protocols in [38].
- **Trapdoor schemes:** DL-based construction for TC and CH can be easily achieved by using some ideas related to Pedersen commitments [35]. However, major lines of accumulator schemes with concrete construction we found are based on RSA assumptions [39]. Au et al's proposal [38] is based on DDHP, but the construction is not trivial. Generally, implementing dynamism for accumulator leads to complex design, extra assumptions or low performance. This is even so for constructing trapdoor accumulator. A trivial idea is to treat each witness in the manner of chameleon hash so that validation is always guaranteed despite of any dynamism. However, this surely brings high costs during redaction as each witness should be enumerated and re-computed for validation. So, a compatible and trivial construction of TA for our scheme is left to be answered.
- **Encryption:** For encryption, Li et al. [9] employed a variant ElGamal encryption which is IND-CPA secure under DDHP and compatible with the original scheme [10]. Differently, we consider threshold encryption in this work. The prior choice is Shoup and Gennaro [40] scheme based on DDHP (with some minor ingredients). However, for practical security, we consider Beoneh et al's [33] scheme where construction is instantiated under decisional Bilinear Diffie-Hellman Problem (BDHP) and proved IND-CCA secure in standard model.
- **Other Required Properties:** Some other properties notified in [10] should be satisfied as well: (1). Homomorphism for user's public keys and commitments. (2) Well-formation of commitments and derivation of secret keys.

Since Monero's privacy and practicality are achieved by cryptographic accumulator, finding an ideal trapdoor accumulator is the key to achieve practical redactable Monero. To minimize parameters, we need to find a DL-based TA with simple design and ideal performance (this is hard). Alternatively, for practical confidentiality, we introduce some BDHP parameters for implementing IND-CCA secure TPKE. Due to sophistication of presentation, we did not give concrete construction in this work.

6 The Security Analysis

We only give security analysis for traceability and redaction soundness in this section. The rest is straightforward from [9], [10].

6.1 Proof of Traceability

Theorem 1. The proposed $\mathsf{R^2CT}$ achieves traceability if each tracing authority is honest and each TPKE ciphertext

can be decrypted, our R^{2}CT satisfies correctness and the underlying SoK, TC, TA and TPKE are secure as previously defined.

Before we proceed to prove traceability, we first review decryption consistency of TPKE formalized in [33] as our traceability relies on decryption consistency of TPKE.

Decryption Consistency. The definition of decryption consistency is based on experiment $\mathbf{Exp}^{\mathsf{DC}}_{\mathsf{TPKE}}$. Decryption consistency holds for TPKE if for any PPT adversary $\mathcal{B}$, his advantage in winning the experiment $\mathbf{Exp}^{\mathsf{DC}}_{\mathsf{TPKE}}$ is negligible, i.e.

$$\mathbf{Adv}^{\mathsf{DC}}_{\mathcal{B},\mathsf{TPKE}}(n_0,k_0,\lambda) = \Big|\mathsf{Pr}[\mathbf{Exp}^{\mathsf{DC}}_{\mathcal{B},\mathsf{TPKE}}(n_0,k_0,\lambda)=1]\Big| \leq negl(\lambda).$$

$\mathbf{Exp}^{\mathsf{DC}}_{\mathsf{TPKE}}$
$\tilde{S}^* \leftarrow \mathcal{B},\ \tilde{S}^* \subset [n_0], |\tilde{S}^*| = k_0 - 1;$
$\mathcal{L}_{\mathsf{Dec}} \leftarrow \emptyset$; // TPKE-Dec query list
$(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, sk_{\mathsf{TPKE}}) \leftarrow \mathsf{TPKE\text{-}Setup}(n_0, k_0, \lambda);$
$(C^*, \{(i, C_i^*)\}_{i\in\tilde{S}^*}, \tilde{S}^{\#}, \{(i', C_{i'}^*)\}_{i'\in\tilde{S}^{\#}}) \leftarrow$
$\mathcal{B}^{\mathcal{O}_{\mathsf{Dec}}(\cdot)}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, \{i, sk_{i,\mathsf{TPKE}}\}_{i\in\tilde{S}^*});$
return 1 if: $\tilde{S}^{\#} \subset [n_0] \ \wedge |\tilde{S}^{\#}| = k_0 - 1 \wedge$
$\forall i \in \tilde{S}^*, i' \in \tilde{S}^{\#}, \mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, (i, C_i^*))$
$= \mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, (i', C_{i'}^{\#})) = 1 \wedge$
$\forall i, j \in \tilde{S}^*, i \neq j : C_i^* \neq C_j^* \wedge$
$\forall i', j' \in \tilde{S}^{\#}, i' \neq j' : C_{i'}^* \neq C_{j'}^* \wedge$
$\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, \{i, C_i^*\}_{i\in\tilde{S}^*}) \neq$
$\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, \{i', C_{i'}^{\#}\}_{i'\in\tilde{S}^{\#}}) \wedge$
$C^* \notin \mathcal{L}_{\mathsf{Dec}}$,
else, return 0.

Denote $\tilde{S}^*$ as a set of $k_0 - 1$ public keys of tracing authorities.

Proof. Suppose $\mathcal{A}$ is an adversary against the traceability of R^{2}CT, $\mathcal{B}$ is an adversary against the decryption consistency of TPKE [33]. We briefly show how to transform $\mathcal{B}$'s attack against TPKE's decryption consistency to $\mathcal{A}$'s attack against R^{2}CT's traceaiblity. Here, a challenger controls oracles $\mathcal{O}_{\mathsf{AddGen}}, \mathcal{O}_{\mathsf{ActGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Corrupt}}, \mathcal{O}_{\mathsf{Trace}}$ and interacts with $\mathcal{A}$ during traceability experiment. $\mathcal{A}$ simulates the challenger of TPKE for $\mathcal{B}$ during decryption consistency experiment.

To start off, $\mathcal{B}$ outputs a set $\tilde{S}^* \subset [n_0]$ for the $k_0 - 1$ authorities to corrupt where $|\tilde{S}^*| = k_0 - 1$, and sends $\tilde{S}^*$ to $\mathcal{A}$. $\mathcal{A}$ sets $\tilde{S}' = \tilde{S}^*$ and sends it to the challenger. Then, the challenger runs $\mathsf{R^2CT\text{-}Setup}(\lambda) \rightarrow (pp_{\mathsf{R^2CT}}, mk_{\mathsf{R^2CT}})$ to derive $pp_{\mathsf{R^2CT}}$ and $mk_{\mathsf{R^2CT}}$. Then, the challenger runs $\mathsf{R^2CT\text{-}Spend}(\mathsf{m}', K_s', A_s', A', R', pk_{\mathsf{TPKE}}) \rightarrow (tx', \eta', S', C')$ to generate $(tx',\ \eta', S\ ', C')$, and sends $chal = (pp_{\mathsf{R^2CT}}, (tx', \eta', S', C'), \{i, sk_{i,\mathsf{TPKE}}, C_i'\}_{i\in\tilde{S}'}, \tilde{S}')$ to $\mathcal{A}$ where $(i, C_i) \leftarrow \mathsf{R^2CT\text{-}Trace}(sk_{i,\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx, C)$. Accordingly, $\mathcal{A}$ sends $chal = (pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, \{i, sk_{i,\mathsf{TPKE}}\}_{i\in\tilde{S}'})$ to $\mathcal{B}$ where $\tilde{S}' = \tilde{S}^*$. Before and after receiving the challenge, $\mathcal{B}$ can make adaptive queries about (i, C) to oracle $\mathcal{O}_{\mathsf{Dec}}$. To simulate the responses, $\mathcal{A}$ invokes oracle $\mathcal{O}_{\mathsf{Spend}}$ on input (m, A_s, A) to derive the corresponding tuple (tx, η, S, C) as a response. Then, $\mathcal{B}$ extracts (tx, η, C) from it to query oracle $\mathcal{O}_{\mathsf{Trace}}$ in order to get the decryption partition (i, C_i) as a response to $\mathcal{B}$'s query. Here, one restriction should apply: queries about C' are not allowed; otherwise, $\mathcal{A}$ can always win the traceability game (so is for $\mathcal{B}$ to win the decryption consistency game). Meanwhile, $\mathcal{A}$ can make queries to oracles $\mathcal{O}_{\mathsf{AddGen}}$, $\mathcal{O}_{\mathsf{ActGen}}$, $\mathcal{O}_{\mathsf{Spend}}$, $\mathcal{O}_{\mathsf{Corrupt}}$, $\mathcal{O}_{\mathsf{Trace}}$. The challenger of R^{2}CT will respond accordingly as previously defined (restriction also applies). Finally, $\mathcal{B}$ is supposed to output $(C^*, \tilde{S}^*, \{(i, C_i^*)\}_{i\in\tilde{S}^*}, \tilde{S}^{\#}, \{(i', C_{i'}^*)\}_{i'\in\tilde{S}^{\#}})$ at the end of decryption consistency game. Based on it, $\mathcal{A}$ sets $(\{i, C_i'\}_{i\in\tilde{S}''}, \tilde{S}'') = (\{(i', C_{i'}^*)\}_{i'\in\tilde{S}^{\#}}, \tilde{S}^{\#})$ and sends it to the challenger at the end of traceability game. To win the decryption consistency game of TPKE, $\mathcal{B}$'s output is supposed to hold Equations (1) and (2):

$$\begin{aligned}&\forall i \in \tilde{S}^*, i' \in \tilde{S}^{\#},\\ &\mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, (i, C_i^*))\\ &= \mathsf{TPKE\text{-}Ver}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, (i', C_{i'}^*))\\ &= 1\end{aligned} \tag{1}$$

$$\begin{aligned}&\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, \{i, C_i^*\}_{i\in\tilde{S}^*}) \neq\\ &\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, \{i', C_{i'}^*\}_{i\in\tilde{S}^{\#}})\end{aligned} \tag{2}$$

For clarity, parse $\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, \{i, C_i^*\}_{i\in\tilde{S}^*}) = \delta^*$ and $\mathsf{TPKE\text{-}Comb}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, C^*, \{i', C_{i'}^*\}_{i\in\tilde{S}^{\#}}) = \delta^{\#}$. So, $\delta^* \neq \delta^{\#}$. Accordingly, to win the traceability game, $\mathcal{A}$'s output is supposed to hold Equations (3) and (4):

$$\mathsf{R^2CT\text{-}Verify}(tx', \eta', S', C') = 1 \tag{3}$$

$$\begin{aligned}&\mathsf{R^2CT\text{-}Judge}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx', \{i, C_i^*\}_{i\in\tilde{S}'}) \neq\\ &\mathsf{R^2CT\text{-}Judge}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx', \{i', C_{i'}{}''\}_{i'\in\tilde{S}''}\end{aligned} \tag{4}$$

Apparently, any queries about (tx', η', C') and $(\mathsf{m}', K_s', A_s', A', R')$ will not be answered by the oracle $\mathcal{O}_{\mathsf{Trace}}$ and $\mathcal{O}_{\mathsf{Spend}}$ respectively; otherwise, $\mathcal{A}$ can always win the traceability game. For clarity, parse $\mathsf{R^2CT\text{-}Judge}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx', \{i, C_i^*\}_{i\in\tilde{S}'} = pk_s'$ and $\mathsf{R^2CT\text{-}Judge}(pk_{\mathsf{TPKE}}, vk_{\mathsf{TPKE}}, tx', \{i', C_{i'}{}''\}_{i'\in\tilde{S}''} = pk_s''$. So, $pk_s' \neq pk_s''$. According to Equation (2), we can infer that δ^* implies pk_s' and $\delta^{\#}$ implies pk_s''; otherwise, it breaks the correctness of R^{2}CT and leads to a worse breaking of decryption inconsistency (i.e. two valid decryption partitions of one ciphertext lead to more than two different plaintexts). If $\mathcal{B}$'s output is a successful attack against decryption consistency, $\mathcal{A}$ can then use it to launch a successful attack against the traceability. Thus, we successfully transform an attack against TPKE's decryption consistency to an attack against R^{2}CT's traceability. □

6.2 Analysis of Redaction

The proposed R^{2}CT is secure for multi-grained redaction if SIGN is a secure and unforgeable signing scheme, the underlying TC and TA are secure according to Section 3.4. Suppose $\mathcal{A}$ is an efficient adversary against the block-level redaction, he outputs a new chain (after being appended,

TABLE 2: Complexity comparison of R^{2}CT scheme with relevant schemes

Schemes	KeyGen	Mint	Spend	Verify	Trace	Judge
Our R^{2}CT	$\mathcal{O}(1)(T_e$ $+T_p)$	$\mathcal{O}(1)(T_e$ $+T_m)$	$\mathcal{O}(l^2+q)T_e+$ $\mathcal{O}(l)T_m+\mathcal{O}(1)T_p$	$\mathcal{O}(l^2+q)T_e+\mathcal{O}$ $(l)(T_m+T_i+T_p)$	$\mathcal{O}(k_0)(T_e+$ $T_m+T_p+T_{sig})$	n/a
RingCT2.0 [10]	$\mathcal{O}(1)T_e$	$\mathcal{O}(1)(T_e)$	$\mathcal{O}(q)(T_e+T_m)$	$\mathcal{O}(q)(T_e+T_m)$	n/a	n/a
Traceable Monero [9]	$\mathcal{O}(1)(T_e$ $+T_{\mathsf{SoK}})$	$\mathcal{O}(1)(T_e$ $+T_m)$	$\mathcal{O}(q)(T_e+T_m)$	$\mathcal{O}(q)(T_e+T_m)$	$\mathcal{O}(1)(T_e+T_m$ $+T_i+T_{\mathsf{SoK}})$	$\mathcal{O}(1)(T_e+$ $+T_m+T_{\mathsf{SoK}})$

Denote k_0 as the number of tracing authorities (threshold number); l as the number of input accounts of Monero user in each group; n as the number of group of input accounts; $\tilde{n}$ as the size of accumulated set; T_m as multiplication in group $\mathbb{G}$; T_e as exponentiation in group $\mathbb{G}$; T_i as inversion in group $\mathbb{G}$; T_p as pairing operation of $\hat{e}$; T_{sig} as the cost of running SIGN to compute a digital signature; n/a as not applicable; T_{SoK} as cost for producing a signature of knowledge; set $q=(n+l)\cdot\tilde{n}$.

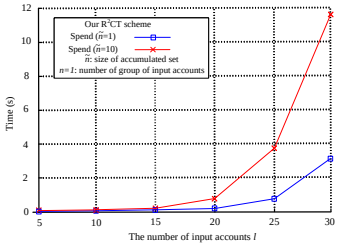

Fig. 2: Impact of accumulator size on spending

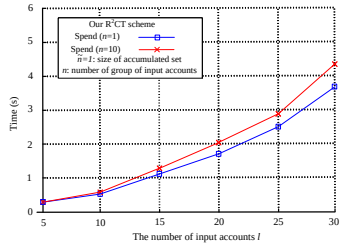

Fig. 3: Impact of input accounts on spending

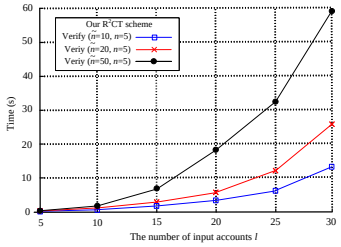

Fig. 4: Impact of accumulator size on verification

modified or shrunk), which can still pass verification of BlockVer. Before outputting a new chain $\mathcal{C}'$, $\mathcal{A}$ is allowed to make adaptive queries to relevant oracles.

One restriction should apply: any queries about the challenged operation (insertion, deletion or modification) should not be answered. Since we use an unforgeable signing scheme SIGN for authentication. It signs on an output of collision-resilient chameleon hash function CH.

The collision-resistance of CH guarantees that no PPT $\mathcal{A}$ is given non-negligible advantage in finding a new CH collision under a fresh customized identity CID* even after seeing arbitrary collisions. Besides, the unforgeability of SIGN ensures that no PPT $\mathcal{A}$ could efficiently forge a valid signature of SIGN for a freshly chosen message (i.e. block). The proof of transaction-level redaction soundness is similar to above, we omit details here.

Assume $\mathcal{A}$ is an efficient adversary against the accumulator-level redaction sourness, i.e. $\mathcal{A}$ is supposed to output a fresh accumulator where the elements within the accumulator have been modified. Assume $\mathcal{B}$ is an adversary against the collision resistance of TA, following above proof, we can also transform $\mathcal{B}$'s attack to TA to $\mathcal{A}$'s attack against the accumulator redaction soundness.

Similarly, we can show that commitment-level redaction soundness is reducible to simulation-sound binding of TC. Suppose $\mathcal{A}$ is an adversary against the commitment-level redaction soundness, and $\mathcal{B}$ is an adversary against the simulation-sound binding of TC. During a commitment-level redaction game, $\mathcal{A}$ can simulate the challenger for $\mathcal{B}$ and uses $\mathcal{B}$'s valid output to launch a successful attack against commitment-level redaction soundness. According to the definition of simulation-sound binding of TC [31], $\mathcal{B}$ cannot gain non-negligible advantage to win ever after seeing arbitrary openings of commitment in breaking the simulation-sound binding of TC. Accordingly, this gives $\mathcal{A}$ no useful information in winning the commitment-level redaction game.

To wrap up, redaction soundness holds if SIGN is secure and unforgeable, and TC, TA, CH are secure as previously defined. □

7 Performance Evaluations

7.1 Complexity Analysis

We give a complexity analysis for relevant works in Table 2 by applying each building block (TC, TA, TPKE, CH) with concrete construction according to section 5.3. Concretely, TC is instantiated with Pedersen commitment, TA is instantiated with [41], TPKE is instantiated with [33], CH is instantiated with [42].

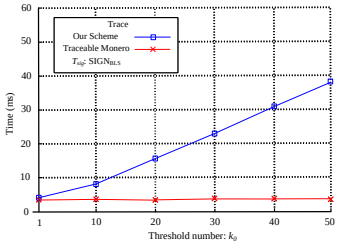

Fig. 5: Impact of threshold on tracing

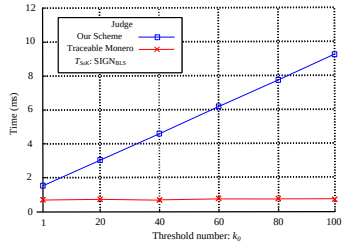

Fig. 6: Impact of threshold on judging

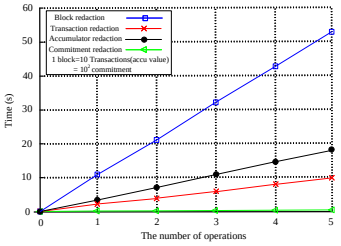

Fig. 7: Costs of dependent redaction

Based on the instantiation, we give a complexity analysis for relevant works in Table 2. As can be observed from Table 2, the costs of key generation algorithms for user are generally constant and small. Here, we did not list the costs of threshold key generation and distribution. According to [33], these costs are one-time expenditure and are generally efficient to execute. To note, our scheme is less scalable than other works in the Spend and the Verify. These costs are mainly factored by l and q. When spending or verifying a coin, intense computations are introduced to enumerate each accumulated element and corresponding witness for validation (also for update). Thanks to the elegant design of TPKE by Boneh et al. [33], the resulted threshold tracing only costs $\mathcal{O}(k_0)(T_e + T_m + T_p + T_{sig})$ which is acceptably efficient. Meanwhile, since T_{sig} is the most costly expenditure in tracing, a smart choice of SIGN is vital.

7.2 Experimental Analysis

To simulate, we implemented our scheme using C language on a laptop with a 3.5GHz 4-cores CPU, 8GB RAM and 256 SSD for storage. The operating system is 32 bits Windows 7 SP1. All algorithms are implemented using PBC (version-0.5.13) for all cryptographic operations. We choose a supersingular curve $y^2 = x^3 + x$ with an embedding degree of 2. Denote $|G| \approx 160$ and $|G_T| \approx 1024$ as the binary sizes of groups $|G|$ and $|G_T|$, respectively. We use OpenSSL as a means of secure communication. Each result is computed by a mean of 10 consecutive trials.

We range the the number of input accounts l, number of group of input accounts n, threshold number k_0 and size of accumulated set $\tilde{n}$ differently, and show the running time of relevant stages in Figure 2 to Figure 7. As it is shown in Figure 2 and 3, the running time of the Spend increases rapidly with l and $\tilde{n}$.

As it is shown in Figure 4, the cost of the Verify increases linearly with the growth of l and $\tilde{n}$. Next, we compare the running costs in Trace and Judge of our scheme with Li et al.'s Traceable Monero [9] in Figure 5 and 6. The results suggested that our costs increase linearly the threshold number k_0. This obviously indicates the threshold (distributed) feature of our work. Then, we test the operations of redaction with practical concerns. We assume each block contains 10 transactions and each transaction contains 10 commitments (i.e. $l = 10$). When increasingly ranging the number of operations, the costs grow differently for each type of redaction.

As it is shown in Figure 7, coarse-grained redaction (e.g. block-level redaction) increases dramatically with the number of rounds. This is obvious because redacting a block implies manipulating the underlying transaction, accumulated values and commitment contained in a block.

8 Conclusion

In this work, we designed a mutable blockchain for Monero to overcome the drawbacks brought by the immutability of blockchain. We proposed a concrete construction for addressing the challenges of re-writing multiple contents in the Monero blockchain without impairing its anonymity or causing significant contradiction to its original design. We gave security analysis based on a rigorously defined security model and evaluated theoretical and experimental perspectives. While security analysis guaranteed the security of our scheme under a pre-determined model, our evaluations caught some unscalabilities caused by our adaptations from the original design. To capture a safer and more practical use, a more elegant design is considered a challenging but interesting future work.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf, 2008.

[2] "Monero," https://www.monero.org/.

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

[4] S. Noether, A. Mackenzie, *et al.*, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, 2016.

[5] W. A. A. Torres, R. Steinfeld, A. Sakzad, J. K. Liu, V. Kuchta, N. Bhattacharjee, M. H. Au, and J. Cheng, "Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice ringct v1. 0)," in *Australasian Conference on Information Security and Privacy*, pp. 558–576, Springer, Wollongong, NSW, Australia, 2018.

[6] T. H. Yuen, S.-f. Sun, J. K. Liu, M. H. Au, M. F. Esgin, Q. Zhang, and D. Gu, "RingCT 3.0 for blockchain confidential transaction: Shorter size and stronger security," in *International Conference on Financial Cryptography and Data Security*, pp. 464–483, Springer, Kota Kinabalu, Malaysia, 2020.

[7] W. A. Torres, V. Kuchta, R. Steinfeld, A. Sakzad, J. K. Liu, and J. Cheng, "Lattice ringct v2. 0 with multiple input and multiple output wallets," in *Australasian Conference on Information Security and Privacy*, pp. 156–175, Springer, Christchurch, New Zealand, 2019.

[8] R. Morais, P. Crocker, and S. M. de Sousa, "Delegated ringct: faster anonymous transactions," *arXiv preprint arXiv:2011.14159*, 2020.

[9] 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*, 2019.

[10] 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 *European Symposium on Research in Computer Security*, pp. 456–474, Springer, Oslo, Norway, 2017.

[11] Z. Yu, M. H. Au, J. Yu, R. Yang, Q. Xu, and W. F. Lau, "New empirical traceability analysis of cryptonote-style blockchains," in *International Conference on Financial Cryptography and Data Security*, pp. 133–149, Springer, St. Kitts, Saint Kitts and Nevis, 2019.

[12] E. Politou, F. Casino, E. Alepis, and C. Patsakis, "Blockchain mutability: Challenges and proposed solutions," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[13] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16)*, pp. 181–194, USENIX, Santa Clara, CA, 2016.

[14] R. Matzutt, M. Henze, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Thwarting Unwanted Blockchain Content Insertion," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 364–370, IEEE, 2018.

[15] R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldorf, D. Müllmann, O. Hohlfeld, and K. Wehrle, "A quantitative analysis of the impact of arbitrary blockchain content on bitcoin," in *International Conference on Financial Cryptography and Data Security*, pp. 420–438, Springer, Nieuwpoort, Cura?ao, 2018.

[16] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pp. 491–500, ACM, 2011.

[17] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain–or–rewriting history in bitcoin and friends," in *2017 IEEE European symposium on security and privacy (EuroS&P)*, pp. 111–126, IEEE, Paris, France, 2017.

[18] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable blockchain in the permissionless setting," in *2019 IEEE Symposium on Security and Privacy (S&P)*, pp. 124–138, IEEE, San Francisco, CA, 2019.

[19] K. Ashritha, M. Sindhu, and K. Lakshmy, "Redactable blockchain using enhanced chameleon hash function," in *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, pp. 323–328, IEEE, Tamil Nadu, India, 2019.

[20] D. Derler, K. Samelin, D. Slamanig, and C. Striecks, "Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based.," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 406, 2019.

[21] L. Guo, Q. Wang, and W.-C. Yau, "Online/offline rewritable blockchain with auditable outsourced computation," *IEEE Transactions on Cloud Computing*, 2021.

[22] Y. Tian, N. Li, Y. Li, P. Szalachowski, and J. Zhou, "Policy-based chameleon hash for blockchain rewriting with black-box accountability," in *Annual Computer Security Applications Conference*, pp. 813–828, ACM, Austin USA, 2020.

[23] I. Puddu, A. Dmitrienko, and S. Capkun, "μchain: How to forget without hard forks.," *IACR Cryptology ePrint Archive*, vol. 2017, p. 106, 2017.

[24] J. Xu, X. Li, L. Yin, Y. Lu, Q. Tang, and Z. Zhang, "Redactable blockchain protocol with instant redaction.," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 223, 2021.

[25] S. Xu, J. Ning, J. Ma, X. Huang, and R. H. Deng, "K-time modifiable and epoch-based redactable blockchain," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4507–4520, 2021.

[26] K. Gai, Y. Wu, L. Zhu, M. Qiu, and M. Shen, "Privacy-preserving energy trading using consortium blockchain in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3548–3558, 2019.

[27] K. Gai, J. Guo, L. Zhu, and S. Yu, "Blockchain meets cloud computing: a survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2009–2030, 2020.

[28] K. Gai, Y. Wu, L. Zhu, K.-K. R. Choo, and B. Xiao, "Blockchain-enabled trustworthy group communications in uav networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4118–4130, 2020.

[29] K. Gai, Y. Zhang, M. Qiu, and B. Thuraisingham, "Blockchain-enabled service optimizations in supply chain digital twin," *IEEE Transactions on Services Computing*, 2022.

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

[31] P. MacKenzie and K. Yang, "On simulation-sound trapdoor commitments," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 382–400, Springer, Interlaken, Switzerland, 2004.

[32] H. C. Pöhls and K. Samelin, "On updatable redactable signatures," in *International Conference on Applied Cryptography and Network Security*, pp. 457–475, Springer, Lausanne, Switzerland, 2014.

[33] D. Boneh, X. Boyen, and S. Halevi, "Chosen ciphertext secure public key threshold encryption without random oracles," in *Cryptographers Track at the RSA Conference*, pp. 226–243, Springer, San Jose, CA, 2006.

[34] G. Ateniese and B. de Medeiros, "Identity-based chameleon hash and applications," in *International Conference on Financial Cryptography*, pp. 164–180, Springer, Key West, FL, USA, 2004.

[35] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors," in *IACR International Workshop on Public Key Cryptography*, pp. 152–182, Springer, Amsterdam, The Netherlands, 2017.

[36] R. Gennaro, "Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks," in *Annual International Cryptology Conference*, pp. 220–236, Springer, Santa Barbara, California, USA, 2004.

[37] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*, pp. 186–194, Springer, Linz, Austria, 1986.

[38] 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 *Cryptographers track at the RSA conference*, pp. 295–308, Springer, San Francisco, CA, USA, 2009.

[39] D. Derler, C. Hanser, and D. Slamanig, "Revisiting cryptographic accumulators, additional properties and relations to other primitives," in *Cryptographers track at the rsa conference*, pp. 127–144, Springer, CA, USA, 2015.

[40] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 1–16, Springer, Espoo, Finland, 1998.

[41] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *International workshop on public key cryptography*, pp. 481–500, Springer, Irvine, CA, USA, 2009.

[42] G. Ateniese and B. de Medeiros, "On the key exposure problem in chameleon hashes," in *International Conference on Security in Communication Networks*, pp. 165–179, Springer, Amalfi, Italy, 2004.

Ke Huang received his M.S. degree and Ph.D. degree in 2015 and 2019 respectively from University of Electronic Science and Technology of China (UESTC). He is currently an Associate Professor in College for Cyber Security, UESTC. He has published 10 journal papers and 1 book. His research interests are blockchain and applied cryptography.

Yi Mu received the Ph.D. degree from the Australian National University in 1994. He is currently a Professor with the Faculty of Data Science, City University of Macau, Macao, China. His current research interests include cybersecurity and cryptography. Prof. Mu was the Editor-in-Chief of the International Journal of Applied Cryptography. He serves as an associate editor for several other international journals.

Fatemeh Rezaeibagha received the M.S. degree in Information Security from Lulea University of Technology, Lulea, Sweden, in 2013, and the Ph.D. degree in Computer Science from the University of Wollongong (UOW), Wollongong, NSW, Australia, in 2017. She is now a Lecturer of Cyber Security with Murdoch University, Perth, Australia. She is a member of IEEE and Australian Computer Society (ACS).

Xiaosong Zhang received his M.S. and Ph.D. degree in 1999 and 2011 respectively from University of Electronic Science and Technology of China (UESTC). He is currently a Professor with the School of Computer Science and Engineering, UESTC, Chengdu, China. He is the Cheung Kong Scholar Distinguished Professor. He is also the Head of College for Cyber Security, UESTC. His research interests are blockchain, AI security, etc.

Xiong Li received the Ph.D. degree in computer science and technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2012. He is currently a Professor with the School of Computer Science and Engineering, UESTC, Chengdu, China. He was a recipient of the 2015 and 2020 Journal of Network and Computer Applications Best Research Paper Award and the 2020 IEEE SYSTEMS JOURNAL Best Paper Award.

Sheng Cao recieved his PhD degree from the University of Chinese Academy Sciences, Bei jing, in 2008. He is currently a professor with the University of Electronic Science and Technology of China, Chengdu. His research interests include network security, blockchain, service computing, and intelligent education.