BlindShuffler: Universal and Trustless Mixing for Confidential Transactions

Chenke Wang
Shanghai Jiao Tong University
Shanghai, China
w_chenke@sjtu.edu.cn

Zhonghui Ge
Shanghai Jiao Tong University
Shanghai, China
zhonghui.ge@sjtu.edu.cn

Yu Long*
Shanghai Jiao Tong University
Shanghai, China
longyu@sjtu.edu.cn

Xian Xu*
East China University of Science and Technology
Shanghai, China
xuxian@ecust.edu.cn

Shi-Feng Sun†
Shanghai Jiao Tong University
Shanghai, China
shifeng.sun@sjtu.edu.cn

Dawu Gu*†
Shanghai Jiao Tong University
Shanghai, China
dwgu@sjtu.edu.cn

ABSTRACT

Mixing services provide unlinkability for blockchains by breaking the link between sender/receiver identities and are highly appreciated for their compatibility with the underlying blockchains. Many efforts have been made to provide mixing services for either non-confidential or confidential payments. For confidential payments, all the known mixing protocols are designed for confidential blockchains using homomorphic commitment. There is, however, no satisfactory solution for confidential blockchains using public key encryption (PKE), such as PGC and Zether.

This work proposes the first universal mixing protocol for confidential blockchains built upon homomorphic PKE. We provide formal security definitions, and prove that the mixing protocol satisfies all the security requirements. Moreover, we present BlindShuffler, an instantiation of the universal mixing protocol. The evaluation shows that when applying BlindShuffler to PGC, notably lower cost is needed to achieve the same level of anonymity compared with other solutions, and it becomes more advantageous with the increase of the anonymous level. As an independent contribution in the building of BlindShuffer, we improve the state-of-the-art standard one-out-of-many proof to support heterogeneous public keys and construct an enhancement with logarithmic membership proof length.

CCS CONCEPTS

• **Theory of computation** → **Cryptographic protocols**; *Cryptographic primitives*; • **Security and privacy** → **Privacy-preserving protocols**; **Pseudonymity, anonymity and untraceability**.

*Corresponding authors.
†Also with Shanghai Jiao Tong University (Wuxi) Blockchain Advanced Research Center.

ASIA CCS '24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07
https://doi.org/10.1145/3634737.3637669

KEYWORDS

Blockchain, Mixing, Confidential transaction, Smart contract

ACM Reference Format:
Chenke Wang, Zhonghui Ge, Yu Long, Xian Xu, Shi-Feng Sun, and Dawu Gu. 2024. BlindShuffler: Universal and Trustless Mixing for Confidential Transactions. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3634737.3637669

1 INTRODUCTION

Cryptocurrencies [20, 22] are publicly accessible payment systems exploiting blockchain techniques, allowing for trustless and decentralized payment. To prevent malicious observers from tracking the transaction flow and monitoring exchanges between participants, *privacy* has become an important concern [3, 24]. In particularly, privacy for cryptocurrencies consists of two aspects. 1) *Anonymity/Unlinkability*: hiding the identities of the participants of a transaction or the link between transactions, and 2) *Confidentiality*: hiding the transaction value. To promote the privacy of cryptocurrencies, many privacy-preserving cryptocurrencies have been proposed, such as ZCash [2], Monero [30] and others [11, 17]. However, these methods are not compatible with any existing blockchains, because they need to either start up a new chain or hard-fork an existing one.

In contrast, mixing services can provide unlinkability for existing blockchains without compromising compatibility. In a nutshell, mixing services allow the link between a pair of sender and receiver to be hidden among many, while keeping full compatibility with the underlying blockchain by working as add-on components. Depending on whether the underlying chain is confidential or not, mixing services can be categorized into those for *non-confidential* transactions and those for *confidential* transactions.

Mixing services for non-confidential transactions. Most mixing protocols work on non-confidential blockchains [13, 16, 19, 26–28, 31] and only support fixed transaction values to avoid trivial identity links via transaction values. The fixed-amount requirement not only limits the potential anonymity set but also causes the mixing overhead to grow linearly (or logarithmically) with the increase in payment value. To support the mixing for variable payment amounts, Tornado Cash Nova [21] suggests that users use small withdraw amounts after mixing, and BlindHub [23] utilizes the

Table 1: Comparison with previous mixing solutions for confidential transactions.

Schemes	Coins Type *	Universal	Compatible Instance★	Non-interactive	# On-chain Txs Per Mixer	Support Payment	Model	Dos Attack Resistance
ValueShuffle [32]	Commitment	○	●	○	1	●	UTXO	◑‡
𝕍eksel [7]	Commitment	○	○	●	2	●	Account	●
MixCT [10]	Commitment	●	◑†	●	2	○	Account	●
BlindShuffler	PKE Ciphertext	●	●	●	2	●	Account	●

● Support ○ No support ◑Partial support
* Both commitment and public key encryption (PKE) has the homomorphic property.
★ "Compatible instance" means that there are pre-existing confidential blockchain instances which are compatible with the mixing solution.
† It can be easily instantiated by migrating the existing protocols (e.g., MimbleWimble) to the account-based model.
‡ Prevented by locating and excluding the malicious participants.

off-chain payment channel with non-interactive zero-knowledge proof. However, as discussed in the most recent work [12], all the mixing services for non-confidential blockchains suffer from the so-called "natural leakage". That is, since the transaction value on the non-confidential blockchain is visible, people can always come up with counterexamples that break unlinkability.

In contrast, in confidential blockchains, since the transaction amounts are invisible to the blockchain observers, the natural leakage disappears. So it is possible to achieve full-fledged privacy, i.e., the identity hiding of the mixing participants as well as the confidentiality of the payment values. In this work, we focus on mixing services for confidential transactions.

Mixing services for confidential transactions. Mixing services for *confidential transactions* can achieve the mixing of arbitrary transaction values in a direct fashion. Technically, confidential transactions [18] are first introduced by Maxwell, in which every payment value involved is hidden by using a cryptographic *commitment* to the value. For confidential transactions, ValueShuffle [25] provides the P2P mixing service utilizing stealth address, which would suffer from high communication overheads. For instance, every user is required to communicate with the other $n-1$ participants to achieve an n-size anonymity set. To address the efficiency issue, *non-interactive* mixing, 𝕍eksel [7] and MixCT [10], are proposed. In this kind of mixing, participants are mutually independent and do not need to communicate with others. A key component in these works is the non-interactive zero-knowledge proof (NIZK), which is used to justify the claim that a coin (i.e., a commitment) received by the payee is a rerandomization of one coin from a set.

Concretely, 𝕍eksel [7] utilizes the commitment to achieve confidentiality and provides the first non-interactive coin mixing service for confidential transactions in the account-based model.

However, 𝕍eksel is not compatible with any existing chains and works as a stand-alone solution. MixCT [10] breaks this limitation and is fully compatible with *any* confidential blockchains utilizing *homomorphic commitment*, which is defined as a *universal* solution. Nevertheless, MixCT does not *support payment*, i.e., the mixing users send their coins to a fresh address of their own. Afterward, senders can spend coins with the fresh addresses. This two-step process leads to the waste of the transaction fee and waiting time.

Considering accounts vulnerability caused by the failure of receiving one transaction in the setting of homomorphic commitment, there is another (arguably) more preferable approach to hide payment values, i.e., homomorphic public key encryption (PKE). Especially, *homomorphic PKE* has been successfully used in the account-based model with supporting to the smart contract, such as Zether [6] and PGC [8]. In this regard, coins belonging to different users are ciphertexts under different public keys, and we refer to this as the *heterogeneous public key* issue. Due to the heterogeneity of public keys, a coin of one user cannot be re-randomized as a new coin of another user by simply changing the random factor. Consequently, mixing service built on *confidential blockchains utilizing PKE* cannot be trivially realized using similar techniques in [7, 10]. To the best of our knowledge, no existing work implements the mixing for confidential transactions based on homomorphic PKE.

The foregoing discussion leads to the following question.

Can we design a mixing service that is non-interactive, universal, and supports payment, on top of all the existing confidential blockchains utilizing homomorphic PKE?

1.1 Our Contributions and Roadmap

In this work, we answer this question affirmatively by proposing a universal mixing protocol. Compared with the state-of-the-art works, this is the first protocol supporting mixing for confidential transactions based on homomorphic PKE and achieves competitive off-chain and on-chain performance. Specifically, we present:

- **The framework of a universal mixing service**. This framework is compatible with confidential payment systems built from the homomorphic PKE in the account-based model, such as Zether and PGC. The detailed comparison of our work with other mixing solutions for confidential transactions is summarized in Table 1. We describe the mixing service model in Section 3 and provide the construction and formal security proof of the framework in Section 4.
- **Logarithmic membership proof supporting heterogeneous public keys.** We design a novel membership proof for heterogeneous public keys, with a transparent setup and logarithmic proof size. Concretely, given a list of key/ciphertexts pairs (i.e., each pair is composed of a public key and a ciphertext under this key), a target public key, and a ciphertext under this target key, this membership proof is to prove that there is at least one record in the list that encrypts the same value as the target ciphertext. Compared with the "re-randomization" technique used in the standard one-out-of-many proof, the main challenge is to deal with the ciphertexts under different public keys.
  The resulting membership proof protocol, together with the optimization and the security proof, is presented in Section 5.2. Our membership proof can be readily used to realize mixing on virtually all confidential blockchain of today and we believe it will have more applications in achieving privacy.

- **An efficient and secure instantiation.** In Section 5, we present BlindShuffler, an instantiation of the universal mixing protocol with lightweight components. Our experiment in Section 6 shows the efficiency of BlindShuffler. To achieve the typical anonymous set of size 16, Blindshuffler consumes only 37% more gas than a plain confidential transfer. Besides, the transaction size complexity in BlindShuffler grows logarithmically in size of the anonymous set, rather than linearly as in Anonymous Zether.

1.2 Other Related Works

In addition to the mixing service, there are two other methods that provide anonymity for confidential blockchains.

Anonymous Zether and Many-out-of-Many Proof. Unlike the mixing service, users of the Anonymous Zether [6] can create an anonymous transaction by themselves. The main idea is that an anonymous transaction from the sender to the receiver contains N ($N \geq 3$, instead of 2 in Zether) ciphertexts $c_0, \cdots, c_{N-1}$ under keys $\mathsf{pk}_0, \cdots, \mathsf{pk}_{N-1}$ respectively, and the public keys of the sender and the receiver are hidden among them. To guarantee balance, only the sender and the receiver's ciphertexts contain opposite transaction values, and all the other ciphertexts contain 0. Thus, a complex NIZK proof is necessary to show that all ciphertexts are well-formed. Subsequently, [9] instantiates Anonymous Zether using many-out-of-many proof more efficiently.

Unfortunately, as a built-in solution, neither of the above mentioned solutions can be applied to the existing confidential blockchain without hard-forking or starting a new one, not to mention the increased transaction size which grows linearly with respect to N. Moreover, a necessary trick to design this proof is that the N ciphertexts should be encrypted using the *same* randomness, which precludes the many-out-of-many proof technique from migrating to the design of a mixing service. The point is that mixing users can create confidential transactions locally and thus independent users may *not* share the same randomness.

Naturally, the fact that N coins (ciphertexts) coming from N mixing users with different randomness motivates a novel membership proof for a more general relation. We emphasize that N ciphertexts under N public keys are allowed to use different randomnesses, and duplicates among keys or even ciphertexts are permitted.

Any-out-of-Many Proof. Most recently, a new technique, named any-out-of-many [33], aiming to solve the knowledge proof of arbitrarily many secrets out of a public list is proposed to improve the anonymity and efficiency of RingCT. This solution is also suggested to be applicable in providing mixing services for blockchain. However, such coin mixing requires users' cooperation and several rounds of off-chain communications. Thus, secure multi-party computation is required to achieve anonymity among the mixing participants, which limits the application of this solution.

1.3 Technical Overview

Basic setting. In the account-based model, several existing confidential blockchains (e.g., Zether and PGC) represent the account balance using the PKE ciphertext. In contrast to the commitment method, this avoids the out-of-band transfer between participants.

In line with the PKE-based account representation, the confidential transaction can be uniformly expressed as $\mathsf{ctx} = (\mathsf{pk}_s, \mathsf{pk}_r, c_s, c_r, \mathsf{info})$, where the sender (with the public key) pk_s transfers a hidden value v to the receiver (with the public key) pk_r by transforming the sender's coin c_s to the receiver's coin c_r. Here c_s and c_r encrypt the same value v under pk_s and pk_r respectively. To guarantee the system security, info is generated by the sender to prove that c_s and c_r are well-formed and the sender has authorized the generation (i.e., with the sender's signature).

Our work is the first mixing service that overlays above confidential blockchains to provide anonymity for blockchain users. Roughly, in our protocol, the mixing service is governed by the *tumbler*, which is a smart contract deployed on the blockchain. The tumbler with the public address pk_t divides the time into epochs, and one epoch consists of two phases: escrow and redemption.

- In the *escrow phase*, each escrow user pk_e can escrow a coin to the tumbler pk_t by generating a transaction $\mathsf{ctx}_{\mathsf{esc}} = (\mathsf{pk}_e, \mathsf{pk}_t, c_e, c_t, \mathsf{info})$. Then, the tumbler collects valid escrow transactions and publicly updates the tumbler's public state.
- In the *redemption phase*, each redemption user pk_r redeems a coin from the tumbler. It is ensured that each redemption operation will redeem the same value as the corresponding transaction in the escrow phase. Then, the tumbler funds the receivers who can provide valid redemption requests in this epoch.

It is worth noting that, as a smart contract, the tumbler is publicly deployed, automatically executed and has *no* secret key corresponding to pk_t. So the tumbler cannot derive the transfer value by decrypting c_t. Furthermore, our construction goal is that the tumbler is prevented from linking one escrow user with the corresponding redemption user, via any method.

Protocol overview. The core idea of the universal mixing scheme for confidential transactions (UMCT) is constructing an invisible permutation between the set of escrowed transactions and the set of successful redemptions for each epoch. Figure 1 shows the mixing procedure for one payment. Specifically, in one epoch:

- To escrow a coin in the escrow phase, each escrow user generates a unique token together with the escrow transaction $\mathsf{ctx}_{\mathsf{esc}}$, where token is a commitment to the redemption user's public key pk_r together with a random redemption tag t. Intuitively, after the escrow request has been collected by the tumbler, pk_r and t are both implied by token.
  Assume that the tumbler collects N pairs of $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token})$ in the escrow phase, then N is the anonymity set size. Note that the tumbler's collections are publicly accessible.
- To redeem a coin in the redemption phase, a redemption user pk_r submits three relevant pieces of information to the tumbler.
  - A unique tag t: To prevent double-redemption.
  - A new coin c_r: The ciphertext under pk_r (with the same value as the correspondingly escrowed one).
  - A valid proof of the redemption: This allows the tumbler to confirm the validity of this redemption.

  These pieces must be carefully designed to ensure that the tumbler will neither get linkable information nor accept invalid ones. To achieve this goal, we devise a NIZK proof. A valid proof would reveal that there exists an escrowed coin encrypting the same value as c_r, and more importantly, this coin's corresponding token encodes both pk_r and the unique tag t. Upon collecting valid redemption requests, the tumbler adds t to its public state and refunds c_r to the redemption user pk_r.

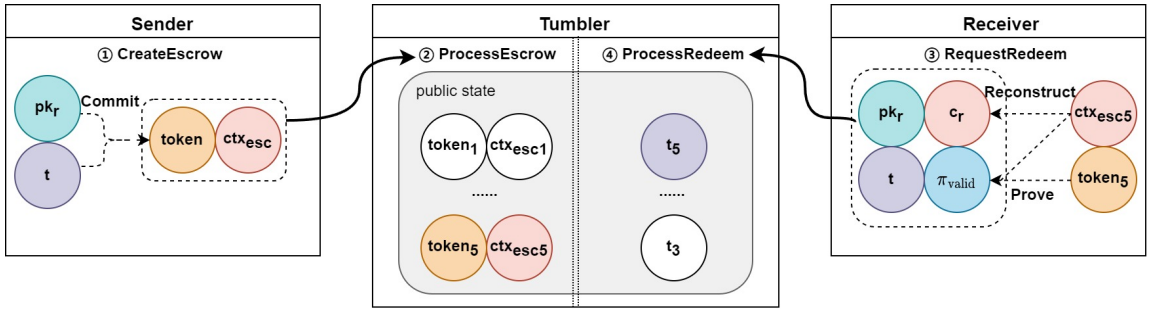

Figure 1: Overview of UMCT. This figure shows a mixing procedure for payments in one epoch. Black solid arrows represent the interactions between the sender, the tumbler, and the receiver. Black dashed arrows represent the data dependencies. ①② are algorithms in the escrow phase and ③④ are algorithms in the redemption phase. As a result, the accounts of the sender and the receiver will be updated accordingly (not shown in the figure). In this instance, the anonymity set size is 5.

Instantiation. To provide an efficient instantiation of the above universal protocol, we realize the NIZK proof with transparent setup. A key point to use the one-out-of-many proof is that: Since a redeemed coin c_r and its corresponding escrowed coin c_e are ciphertexts under pk_r and pk_e respectively, the standard way of operating c_r and c_e to obtain a ciphertext encrypting 0 no longer works. To resolve the heterogeneous public key issue, we propose a method that adapts the one-out-of-many proof in [4] to support the membership proof with respect to ciphertexts under heterogeneous public keys. Moreover, to optimize the proof size, our membership proof exploits the inner product argument technique in Bulletproofs[5], so as to obtain a logarithmic proof size.

Details of the membership proof supporting heterogeneous public keys are given in Section 5.2.

2 PRELIMINARIES

Basic notations. We denote the security parameter by 1^λ ($\lambda \in \mathbb{N}$). Given a space $\mathcal{K}$, uniform sampling is denoted by $k \leftarrow\!\$\ \mathcal{K}$. We use $y \leftarrow \mathsf{A}(x;k)$ to express that the probabilistic polynomial time (PPT) algorithm A outputs y with the input x and a random tape k. For $n \in \mathbb{N}$, we denote the set $\{0,1,\ldots,n-1\}$ by $[n]$.

Let $\mathcal{G}$ be a PPT algorithm that, on input 1^λ, outputs the description of a cyclic group $\mathbb{G}$ of prime order $p \geq 2^\lambda$, together with a random generator $g \leftarrow\!\$\ \mathbb{G}$. $\mathbb{Z}_p$ denotes the ring of integers modulo p. Throughout the paper, we use vector notations defined below. The bold font letter denotes vectors, e.g., $\mathbf{z} \in \mathbb{Z}_p^n$ is a verctor with elements $z_0, z_1, \cdots z_{n-1} \in \mathbb{Z}_p$. For a vector $\mathbf{g} = (g_0, g_1, \cdots, g_{n-1}) \in \mathbb{G}^n$ and $\mathbf{z} \in \mathbb{Z}_p^n$, we denote $\mathbf{g}^\mathbf{z} = \prod_{i=0}^{n-1} g_i^{z_i} \in \mathbb{G}$ and $\mathbf{g}^{z_0} = (g_0^{z_0}, \cdots, g_{n-1}^{z_0}) \in \mathbb{G}^n$. For $0 \leq l \leq n-1$, we denote slices of vectors by

$$\mathbf{z}_{[:l]} = (z_0, \cdots, z_{l-1}) \in \mathbb{Z}_p^l, \quad \mathbf{z}_{[l:]} = (z_l, \cdots, z_{n-1}) \in \mathbb{Z}_p^{n-l}.$$

The Hadamard product of two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ is denoted by $\mathbf{a} \circ \mathbf{b} = (a_0 \cdot b_0, \cdots, a_{n-1} \cdot b_{n-1}) \in \mathbb{Z}_p^n$. For convenience, we sometimes use $\{b_{j,i}\}_{j,i=0}^{m-1,n-1}$ to denote the sequence $(b_{0,0}, b_{0,1}, \cdots, b_{m-1,n-1})$ for $m, n \in \mathbb{N}$, with no ambiguity.

2.1 Assumptions

In what follows, we describe the discrete-logarithm based assumptions related to $(\mathbb{G}, p) \leftarrow \mathcal{G}(1^\lambda)$.

Definition 2.1 (Discrete Logarithm Relation Assumption). The discrete logarithm relation assumption holds if for any PPT adversary $\mathcal{A}$, for all $n \geq 2$ and $\mathbf{g} \leftarrow\!\$\ \mathbb{G}^n$,

$$\Pr\left[\exists i \in [n] :\ a_i \neq 0 \wedge \prod_{i=0}^{n-1} g_i^{a_i} = 1 \;\middle|\; \mathbf{a} \in \mathbb{Z}_p^n \leftarrow \mathcal{A}(\mathbb{G}, \mathbf{g})\right] \leq \mathsf{negl}(\lambda).$$

It is worth noting that this assumption is equivalent to the discrete logarithm (DLOG) assumption.

We also assume that the standard decisional Diffie-Hellman (DDH) assumption holds in $\mathbb{G}$ [6, 8].

2.2 Homomorphic Public Key Encryption

A Public Key Encryption scheme PKE with message space $\mathcal{M}$ and ciphertext space C consists of the following algorithms.

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, output the public parameters pp used implicitly in the following algorithms.
- $\mathsf{Gen}(\mathsf{pp})$: A probabilistic algorithm that on input the public parameters, outputs a key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: A probabilistic algorithm that on input the public key pk and a message $m \in \mathcal{M}$, outputs a ciphertext $c \in C$. It is also denoted as $\mathsf{Enc}_{\mathsf{pk}}(m;k)$ where k is the random tape.
- $\mathsf{Dec}(\mathsf{sk}, c)$: A deterministic algorithm that on input the secret key sk and the ciphertext $c \in C$, outputs a message $m \in \mathcal{M}$.

Definition 2.2 (Secure Homomorphic PKE). A public key encryption scheme $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as defined above is a secure homomorphic scheme, if it satisfies the following properties.

- Decryption Uniqueness. For any message $m \in \mathcal{M}$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{pp})$, it holds that $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.
- IND-CPA Security. PKE is IND-CPA secure if, for a given ciphertext, a CPA adversary cannot learn any information about the underlying plaintext [8].
- Additive Homomorphism. PKE satisfies the additive homomorphism if, for every pair of $m_0, m_1 \in \mathcal{M}$ and every public key pk generated from $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{pp})$, it holds that $\mathsf{Enc}(\mathsf{pk}, m_0) \cdot \mathsf{Enc}(\mathsf{pk}, m_1) = \mathsf{Enc}(\mathsf{pk}, m_0 + m_1)$, where $\cdot, +$ are operations in the ciphertext space C and the message space $\mathcal{M}$ respectively.

In this work, we focus on the underlying confidential payment systems exploiting a secure homomorphic PKE to achieve confidentiality. Several well-known payment systems meet this requirement in the literature. Typically, Zether [6] utilizes ElGamal, and PGC [8] proposes and applies the Twisted ElGamal, which are both homomorphic PKE. In this paper, we will take Twisted ElGamal.

Twisted ElGamal. The twisted ElGamal scheme TElG = (Setup, Gen, Enc, Dec) consists of four algorithms.

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter, run $(\mathbb{G}, p) \leftarrow \mathcal{G}(1^\lambda)$, pick $g, h \leftarrow\!\!\$\ \mathbb{G}$, output $\mathsf{pp} := (\mathbb{G}, g, h, p)$.
- $\mathsf{Gen}(\mathsf{pp})$: On input pp, choose $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$, set $\mathsf{pk} = g^{\mathsf{sk}}$.
- $\mathsf{Enc}_{\mathsf{pk}}(v; k)$: Compute $X = \mathsf{pk}^k$, $Y = g^k h^v$, output $c = (X, Y)$.
- $\mathsf{Dec}(c, \mathsf{sk})$: Parse $c = (X, Y)$, compute $h^v = Y/X^{\mathsf{sk}^{-1}}$ and recover v from h^v.

Theorem 2.3. *If the DDH assumption holds, the Twisted ElGamal scheme* TElG *is a secure homomorphic PKE [8].*

2.3 Commitments

A commitment scheme C consists of the following algorithms.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. Output the public parameters pp, including the commitment key, used implicitly to commit to messages.
- $c \leftarrow \mathsf{Com}(m; r)$. On input m and a masking factor r, output the commitment c.

In specific, when the message m is a vector, the commitment scheme is named as the *vector commitment*.

The commitment is required to be (computationally/perfectly) hiding and (perfectly/computationally) binding. In addition, we may require a commitment to be homomorphic.

Definition 2.4 (Secure Homomorphic Commitment). A commitment scheme $\mathsf{C} = (\mathsf{Setup}, \mathsf{Com})$ as defined above is a secure homomorphic scheme if it satisfies the following properties.

- (Computationally/Perfectly) Hiding. The distribution of $\mathsf{Com}(m; r)$ is (computationally/statistically) identical to the uniform distribution over the commitment space.
- (Perfectly/Computationally) Binding. There is no (all powerful/efficient) adversary that can produce two pairs $(m, r) \neq (m', r')$ such that $\mathsf{Com}(m; r) = \mathsf{Com}(m', r')$.
- Additive Homomorphism. For any $m_0, m_1 \in \mathcal{M}$, it holds that $\mathsf{Com}(m_0) \cdot \mathsf{Com}(m_1) = \mathsf{Com}(m_0 + m_1)$ where $\cdot, +$ are operations in the commitment space and the message space respectively.

2.4 Non-Interactive Zero-Knowledge Arguments of Knowledge

A non-interactive zero-knowledge (NIZK) argument for $(x; w) \in \mathsf{R}_{\mathcal{L}}$, where x is the public statement belonging to an NP language $\mathcal{L}$ and w is the witness, consists of three algorithms.

- $(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda)$. On input the security parameter, output a common reference string σ and a simulation trapdoor τ.
- $\pi \leftarrow \mathsf{Prove}(\sigma, x, w)$. On input a common reference string σ and $(x, w) \in \mathsf{R}_{\mathcal{L}}$, output an argument π. This algorithm is executed by the prover $\mathcal{P}$.
- $0/1 \leftarrow \mathsf{Verify}(\sigma, x, \pi)$. On input a common reference string σ, a statement x and an argument π, output 1 if accept and 0 otherwise. This algorithm is executed by the verifier $\mathcal{V}$.

We require that a NIZK protocol NIZK should have perfect completeness, knowledge soundness, and zero-knowledge. The formal definitions[14] are given in Appendix A. Roughly,

- *Completeness.* NIZK is complete if the prover who knows a witness of a statement $x \in \mathcal{L}$ can always output a proof to convince the verifier to accept.
- *Knowledge Soundness.* NIZK is knowledge sound if, for a valid proof π of an instance x, there exists a PPT extractor Ext who can get full access to the prover's state and extract the witness w from π, where $(x, w) \in \mathsf{R}_{\mathcal{L}}$.
- *Zero-knowledge.* NIZK is zero-knowledge if there exists a PPT algorithm Sim, who takes the simulation trapdoor τ as an extra input, can simulate the proof π for $x \in \mathcal{L}$ without knowledge of the witness w.

In this work, we construct a new NIZK argument of knowledge by proposing an interactive ZK argument of knowledge first, and then convert it into a non-interactive zero-knowledge proof protocol in the random oracle model using the Fiat-Shamir heuristic. We introduce ZK arguments of knowledge in Appendix A.

Standard one-out-of-many proof. A standard one-out-of-many proof (OOOM) is a proof protocol to prove that n ciphertexts $(c_0, c_1, \cdots, c_{n-1})$ includes an encryption of 0 under one public key pk [1]. Formally, it is defined by the following relation,

$$\begin{aligned} &\mathsf{R}(\{c_i\}_{i\in[n]}, \mathsf{pk}; l, k) = 1 \\ \Leftrightarrow &\forall i \in [n] : c_i \in C \ \wedge\ l \in [n] \ \wedge\ k \in \mathcal{K} \wedge\ c_l = \mathsf{Enc}_{\mathsf{pk}}(0; k), \end{aligned} \tag{1}$$

where C and $\mathcal{K}$ denote the ciphertext space and the randomness space, respectively. Groth and Kohlweiss [15] first introduced this protocol, and an optimized and generalized construction was given in [4]. In the standard way, Fiat-Shamir transform [1] can be used to turn the protocol into a NIZK proof scheme, which requires *no trusted setup*. Precisely, OOOM consists of three algorithms.

- $\sigma \leftarrow \mathsf{OOOM.Setup}(1^\lambda)$. On input the security parameter 1^λ, output the common reference string σ used implicitly in all the following algorithms.
- $\pi \leftarrow \mathsf{OOOM.Prove}(\{c_i\}_{i\in[n]}, \mathsf{pk}), (l, k))$. On input the public key pk, the statement $(c_0, \cdots, c_{n-1})$, and the witness (l, k), output the proof π.
- $0/1 \leftarrow \mathsf{OOOM.Verify}(\mathsf{pk}, (c_0, \cdots, c_{n-1}), \pi)$. On input the public key pk, the statement $(c_0, \cdots, c_{n-1})$, and the proof π, output 1 if accept and 0 otherwise.

3 UNIVERSAL MIXING SCHEME FOR CONFIDENTIAL TRANSACTIONS

In this section, we formalize the universal mixing scheme for confidential transactions (UMCT) in the account-based model. We also describe the security goals to achieve.

3.1 Data Structures

Without loss of generality, the confidential blockchain system that our UMCT underlies could be described as follows.

Confidential payment system utilizing homomorphic PKE. A confidential payment system is virtually an append-only public

[1] OOOM can generalize easily to additive homomorphic encryption and commitment schemes, such as (Twisted) ElGamal and (vector) Pedersen commitment.

ledger $\mathbb{L}$. In the account-based model, this ledger contains two lists: 1) A list of registered user accounts, each of which is an ordered pair $(\mathsf{pk}, \mathsf{bal}[\mathsf{pk}])$ where $\mathsf{bal}[\mathsf{pk}]$ encodes the confidential balance of a user with the public key pk. 2) A list of all publicly verified transactions. The generation and verification of confidential transactions are defined as follows.

- $\mathsf{pp} \leftarrow \mathsf{CT.Setup}(1^\lambda)$. On input the security parameter λ, output public parameters pp used implicitly in the following algorithms.
- $\mathsf{ctx} \leftarrow \mathsf{CT.Create}(\mathsf{sk}_s, \mathsf{pk}_r, v)$. A sender with the key pair $(\mathsf{pk}_s, \mathsf{sk}_s)$ runs this algorithm to generate the confidential transaction ctx, which transfers v coin to a receiver with public key pk_r. The resulting ctx can be abstracted as the following quintuple,

$$\mathsf{ctx} = (\mathsf{pk}_s, \mathsf{pk}_r, c_s, c_r, \mathsf{info}),$$

  where c_s is the ciphertext under pk_s encrypting transaction value v, c_r is the ciphertext under pk_r encrypting the same value and info is the auxiliary information including, e.g., the signature and the proof for well-formed ciphertexts.
- $1/0 \leftarrow \mathsf{CT.Verify}(\mathsf{ctx})$. On input a transaction ctx, this algorithm outputs 1 if ctx passes the verification and 0 otherwise. Each transaction can be publicly verified by any user.

Once a ctx gets confirmed by $\mathsf{CT.Verify}(\mathsf{ctx})$, it will be recorded on the ledger. Thereafter, $\mathsf{bal}[\mathsf{pk}_s]$ (respectively $\mathsf{bal}[\mathsf{pk}_r]$) will be updated by substracting c_s (Resp., adding c_r)[2] homomorphically.

Smart contracts. We leverage a smart contract as the tumbler for mixing. *Smart contracts* are public and immutable codes deployed on blockchains. Each smart contract has a specific public key address with *no* corresponding private key. A user pk_u on the blockchain can trigger automatically executed functions in the smart contract pk_t by generating a transaction $\mathsf{ctx} = (\mathsf{pk}_u, \mathsf{pk}_t, c_u, c_t, \mathsf{info})$.

3.2 Entities

UMCT consists of three kinds of participants.

- **Escrow users** are a set of underlying blockchain users. Utilizing UMCT, they transfer coins to someone but want to hide the link between the source and the target identities of the transmission.
- **Tumbler** is a trustless medium providing the mixing service. Specifically, senders escrow their coins to the tumbler, and receivers redeem coins from it. In UMCT, the tumbler is a smart contract that is publicly executed by all blockchain miners. We emphasize again that a tumbler does not have the secret key corresponding to the public key used in the mixing.
- **Redemption users** are also a set of underlying blockchain users[3]. Each redemption user receives coins from some escrow user through the tumbler.

3.3 Definition of the Universal Mixing Scheme for Confidential Transactions

UMCT consists of two phases in each epoch: the escrow phase and the redemption phase. In the escrow phase, escrow users pay coins with arbitrary values to the tumbler by creating the underlying confidential transactions, which triggers the mixing service. In the redemption phase, each redemption user can redeem a coin with the same value as the corresponding escrowed coin. Before the redemption phase, only one interaction between a pair of escrow & redemption users is required, to notify the latter that a coin is ready to be redeemed.

Definition 3.1 (Syntax of UMCT). A UMCT scheme consists of a tuple of polynomial-time algorithms UMCT = (Setup, CreateEscrow, ProcessEscrow, RequestRedeem, ProcessRedeem), as below.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. On input the security parameter λ, output all public parameters, which are assumed to be an implicit input to all the remaining algorithms.
- $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathsf{aux}) \leftarrow \mathsf{CreateEscrow}(\mathsf{sk}_e, v, \mathsf{pk}_t, \mathsf{pk}_r, t, \mathbb{L})$. This algorithm is executed by the escrow user. First, on input the escrow user's secret key sk_e, the transaction value v, and the tumbler's public key pk_t, generate a confidential transaction $\mathsf{ctx}_{\mathsf{esc}}$. Secondly, on input the real receiver's public key pk_r and a unique tag t which is randomly selected, generate a token to imply the redemption information. Besides, some secrets used in this algorithm are packaged into aux and sent privately to pk_r.
- $(\mathbb{L}', 1/0) \leftarrow \mathsf{ProcessEscrow}(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathbb{L})$. This algorithm is executed by the tumbler. On input the confidential transaction $\mathsf{ctx}_{\mathsf{esc}}$ and the identifier token, output the updated ledger. In particular, the tumbler outputs 1 if it confirms the $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token})$ tuple, and 0 otherwise.
- $(t, c_r, \pi_{\mathsf{valid}}) \leftarrow \mathsf{RequestRedeem}(\mathsf{pk}_r, \mathsf{aux}, \mathbb{L})$. This algorithm is executed by the redemption user. On input the redemption user's public key pk_r and the auxiliary information aux, output a new coin c_r, disclosed redemption tag t, and a validity proof π_{valid}.
- $(\mathbb{L}', 1/0) \leftarrow \mathsf{ProcessRedeem}(\mathsf{pk}_r, t, c_r, \pi_{\mathsf{valid}}, \mathbb{L})$. This algorithm is executed by the tumbler. On input the prepared coin c_r, redemption tag t and its valid proof π_{valid} from pk_r, output the updated ledger $\mathbb{L}'$. In particular, the tumbler outputs 1 if it confirms the $(\mathsf{pk}_r, t, c_r, \pi_{\mathsf{valid}})$ tuple, and 0 otherwise.

3.4 Security Definition

A UMCT scheme should satisfy the requirements of correctness, security, and availiablity.

Correctness. The correctness of UMCT means that in each epoch, every escrowed coin conducted in the escrow phase can be redeemed successfully using the corresponding aux generated in the escrow phase. Formally, for a valid escrow $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathsf{aux})$,

$$\Pr\left[\begin{array}{c}(\mathbb{L}', 1) \leftarrow \\ \mathsf{ProcessRedeem}(\mathsf{pk}_r, t, \\ c_r, \pi_{\mathsf{valid}}, \mathbb{L})\end{array} \middle| \begin{array}{c}(t, c_r, \pi_{\mathsf{valid}}) \leftarrow \\ \mathsf{RequestRedeem}(\mathsf{pk}_r, \mathsf{aux}, \mathbb{L}) \\ \wedge \mathsf{ProcessRedeem\ is\ running} \\ \text{with } t \text{ for the first time}\end{array}\right] = 1.$$

The security of UMCT is much more involved, which consists of *balance*, *unlinkability*, and *confidentiality*. Intuitively,

- **Balance** has two levels. At the user level, each redemption user is able to receive the deserved value of coin. At the system level, the total escrow values should equal the total redemption values for each epoch. The two-level balance gives rise to the following specific security requirement.

(1) *Balanced escrow and redemption values*: for each escrow/ redemption pair, the redemption value is the same as the corresponding escrow value.

[2]Here the adding (or substracting) is used in a general sense. Concrete operations depends on the selected ciphertext space.

[3]Redemption users set is allowed to have an overlap with the escrow users set.

Table 2: Structure of L_e and L_r used in the security model

L_e :	pk_e	IsCrpt	$\mathsf{ctx}_{\mathsf{esc}}$	token	v_e	pk_r	t

L_r :	pk_r	IsCrpt	c_r	t	v_r	token

(2) *User theft prevention*: any PPT adversaries cannot redeem coins belonging to honest users. That is, once an escrow is conducted aiming at an honest receiver, the adversary cannot redeem this redemption to another public key.
(3) *System theft prevention*: any PPT adversaries cannot redeem coins belonging to nobody. In other words, the adversary is unable to create a valid redemption request without a corresponding escrow accepted by the tumbler.
(4) *Double-redemption prevention*: each escrow transaction can be redeemed no more than once.

The balanced escrow and redemption and the user theft prevention guarantee balance in the user level, while the other two guarantee balance in the system level.

- **Unlinkability** means that the tumbler cannot connect a valid redemption request to any valid escrow request, despite the tumbler providing the mixing service for users.
- **Confidentiality** stipulates that our universal mixing protocol cannot leak the transaction value. That is, the mixing procedure keeps the confidentiality of the underlying payment system.

To provide the formal definitions of the above properties, we need to define the oracles that are accessible by the adversary. To facilitate the description of the interaction between the challenger and the adversary, we introduce the data structure maintained by the challenger.

Specifically, the challenger C maintains the escrow request list L_e and the redemption request list L_r, as shown in Table 2. The two lists can be viewed as two databases. Retrieving some rows from L_e is denoted by, for example, $L_e[\mathsf{pk}_e]$ for some pk_e. Retrieving a column from L_e is denoted by L_e.item. For example, $L_e.v_e$ is the coin value column in L_e. The assembled notation $L_e[\mathsf{pk}_e].v_e$ is the coin value column of $L_e[\mathsf{pk}_e]$. Notations in L_r are just like L_e.

Oracles. Apart from that the adversary $\mathcal{A}$ can honestly execute algorithms in UMCT and ask the challenger C to process $\mathcal{A}$'s escrow and redemption requests, $\mathcal{A}$ can also induce honest users to escrow or redeem. In the following, we define oracles that $\mathcal{A}$ can access.

- $\mathsf{sk} \leftarrow O_{\mathrm{crpt}}(\mathsf{pk})$. The adversary can corrupt honest users via this query. Challenger C can retrieve the corresponding secret key from the underlying system and send it to $\mathcal{A}$. Then, once pk exists in L_e (or L_r) , C sets $L_e[\mathsf{pk}].\mathsf{IsCrpt} = 1$ (or $L_r[\mathsf{pk}].\mathsf{IsCrpt} = 1$).
- $O_{\mathrm{UMCT}}((\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token})/(\mathsf{pk}_r, t, c_r, \pi_{\mathsf{valid}}))$. This captures that $\mathcal{A}$ can control corrupted user $\mathsf{pk}_e/\mathsf{pk}_r$ to ask the tumbler to escrow/redeem, by submitting the escrow/redemption requests. To answer this kind of queries, C executes as follows,
  - If it receives $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token})$ from $\mathcal{A}$, then runs

$$(\mathbb{L}', \delta) \leftarrow \mathsf{ProcessEscrow}(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathbb{L}),$$

    and adds $(\mathsf{pk}_e, 1, \mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \perp, \perp, \perp)$ to the list L_e if $\delta = 1$.
  - If it receives $(\mathsf{pk}_r, t, c_r, \pi_{\mathsf{valid}})$ from $\mathcal{A}$, then runs

$$(\mathbb{L}', \delta) \leftarrow \mathsf{ProcessRedeem}(\mathsf{pk}_r, t, c_r, \pi_{\mathsf{valid}}, \mathbb{L}),$$

    and adds $(\mathsf{pk}_r, 1, c_r, t, \perp, \perp)$ to the list L_r if $\delta = 1$.
- $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathbb{L}') \leftarrow O_{\mathsf{esc}}(\mathsf{pk}_e, \mathsf{pk}_t, \mathsf{pk}_r, v, \mathbb{L})$. This captures that $\mathcal{A}$ can induce honest user pk_e to escrow a coin of value v for pk_r. To answer this kind of queries, C executes

$$(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathsf{aux}) \leftarrow \mathsf{CreateEscrow}(\mathsf{sk}_e, v, \mathsf{pk}_t, \mathsf{pk}_r, t, \mathbb{L}),$$
$$(\mathbb{L}', \delta) \leftarrow \mathsf{ProcessEscrow}(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathbb{L}).$$

  Then, C returns $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathbb{L}')$ to adversary $\mathcal{A}$ and adds $(\mathsf{pk}_e, 0, \mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, v, \mathsf{pk}_r, t)$ to the list L_e, if $\delta = 1$.
- $(c_r, \pi_{\mathsf{valid}}, \mathbb{L}') \leftarrow O_{\mathrm{red}}(\mathsf{pk}_r)$. This oracle indicates that $\mathcal{A}$ may induce honest user pk_r to redeem coins. The challenger C, on behalf of the honest pk_r, receives aux from one escrow user[4]. Then C executes

$$(t, c_r, \pi_{\mathsf{valid}}) \leftarrow \mathsf{RequestRedeem}(\mathsf{pk}_r, \mathsf{aux}, \mathbb{L}),$$
$$(\mathbb{L}', \delta) \leftarrow \mathsf{ProcessRedeem}(\mathsf{pk}_r, t, c_r, \pi_{\mathsf{valid}}, \mathbb{L}).$$

  Finally, C outputs $(c_r, \pi_{\mathsf{valid}}, \mathbb{L}')$ to adversary $\mathcal{A}$ and adds $(\mathsf{pk}_r, 0, c_r, t, v_r, \mathsf{token})$ to list L_r if $\delta = 1$, where token can be reconstructed by $t, \mathsf{pk}_r, \mathsf{aux}$.

To be concise, we denote the set of aforementioned oracles by Orc. Now, we give the formal security definitions of balance, unlinkability, and confidentiality.

Definition 3.2 (Balance). A UMCT scheme is balanced if for any PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{balance}}_{\mathcal{A},\mathrm{UMCT}}(\lambda) = \Pr[\mathsf{Balance}^{\mathcal{A}}_{\mathrm{UMCT}}(1^\lambda) = 1] \le \mathsf{negl}(\lambda),$$

where $\mathsf{Balance}^{\mathcal{A}}_{\mathrm{UMCT}}$ is defined as follows.

$\mathsf{Balance}^{\mathcal{A}}_{\mathrm{UMCT}}(1^\lambda)$

1 : $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2 : $(\mathsf{pk}_{\mathcal{A}}, t_{\mathcal{A}}, c_{\mathcal{A}}, \pi_{\mathsf{valid}}, \mathbb{L}) \leftarrow \mathcal{A}^{\mathrm{Orc}}(\mathsf{pp})$
3 : $(\mathbb{L}', b) \leftarrow \mathsf{ProcessRedeem}(\mathsf{pk}_{\mathcal{A}}, t_{\mathcal{A}}, c_{\mathcal{A}}, \pi_{\mathsf{valid}}, \mathbb{L})$
4 : $b_0 := (L_e[t_{\mathcal{A}}].v_e \neq L_r[t_{\mathcal{A}}].v_r)$
/ *Unbalanced escrow & redemption value. */
5 : $b_1 := (L_e[t_{\mathcal{A}}].\mathsf{pk}_r \neq \mathsf{pk}_{\mathcal{A}}) \wedge (L_r[(L_e[t_{\mathcal{A}}].\mathsf{pk}_r)].\mathsf{IsCrpt} = 0)$
/ *User $\mathsf{pk}_{\mathcal{A}}$ redeems a coin targeting to an honest public key $L_e[t_{\mathcal{A}}].\mathsf{pk}_r$ *./
6 : $b_2 := (\nexists \mathsf{token}_{\mathcal{A}}$ s.t. $L_e[\mathsf{token}_{\mathcal{A}}].\mathsf{pk}_r = \mathsf{pk}_{\mathcal{A}}$ or $L_e[\mathsf{token}_{\mathcal{A}}].t = t_{\mathcal{A}})$
/ *$\mathsf{pk}_{\mathcal{A}}$ redeems a coin from nothing. * /
7 : $b_3 := (\exists t', L_r[t'].\mathsf{token} = L_r[t_{\mathcal{A}}].\mathsf{token})$
/ *The same escrowed coin is redeemed twice. */
8 : **return** $b \wedge (b_0 \vee b_1 \vee b_2 \vee b_3)$

Definition 3.3 (Unlinkability). A UMCT scheme is unlinkable if for any PPT adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{unlink}}_{\mathcal{A},\mathrm{UMCT}}(\lambda) = \left|\Pr[\mathsf{Unlinkability}^{\mathcal{A}}_{\mathrm{UMCT}}(1^\lambda) = 1] - \frac{1}{2}\right| \le \mathsf{negl}(\lambda),$$

[4]If there are more than one coin escrowed to pk_r, C randomly chooses one coin together with its aux to redeem.

where $\text{Unlinkability}^{\mathcal{A}}_{\text{UMCT}}$ is defined as follows.

$\text{Unlinkability}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda)$

$\text{pp} \leftarrow \text{Setup}(1^\lambda)$

$\{(\text{ctx}^{(i)}_{\text{esc}}, \text{token}^{(i)})\}_{i\in\{0,1\}} \leftarrow \mathcal{A}^{\text{Orc}}(\text{pp})$

$b_0 := (\text{token}^{(i)} \text{ is not generated from } O_{\text{esc}})$

$b \leftarrow\$ \{0,1\}$

$(t^{(b)}, c_r^{(b)}, \pi^{(b)}_{\text{valid}}) \leftarrow \text{RequestRedeem}(\text{pk}_r^{(b)}, \text{aux}^{(b)}, \mathbb{L})$

$(\mathbb{L}', 1) \leftarrow \text{ProcessRedeem}(\text{pk}_r^{(b)}, t^{(b)}, c_r^{(b)}, \pi^{(b)}_{\text{valid}})$

$b' \leftarrow \mathcal{A}^{\text{Orc}}(\text{pk}_r^{(b)}, t^{(b)}, c_r^{(b)}, \pi^{(b)}_{\text{valid}}, \mathbb{L})$

$b_1 := (L_e[\text{pk}_e^{(i)}].\text{IsCrpt} = 0) \wedge (L_r[\text{pk}_r^{(i)}].\text{IsCrpt} = 0)$

/ *Public keys involved are uncorrupted. */

return $b_0 \wedge b_1 \wedge (b = b')$

Definition 3.4 (Confidentiality). A UMCT scheme has confidentiality if for any PPT adversary $\mathcal{A}$,

$$\text{Adv}^{\text{confidential}}_{\mathcal{A},\text{UMCT}}(\lambda) = \left|\Pr[\text{Confidentiality}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda) = 1] - \frac{1}{2}\right| \leq \text{negl}(\lambda),$$

where $\text{Confidentiality}^{\mathcal{A}}_{\text{UMCT}}$ is defined as follows.

$\text{Confidentiality}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda)$

$\text{pp} \leftarrow \text{Setup}(1^\lambda)$

$\text{pk}_e, \text{pk}_r, v_0, v_1 \leftarrow \mathcal{A}^{\text{Orc}}(\text{pp})$

$b \leftarrow\$ \{0,1\}$

$(\text{ctx}^{(b)}_{\text{esc}}, \text{token}, \text{aux}^{(b)}) \leftarrow \text{CreateEscrow}(\text{sk}_e, v_b, \text{pk}_t, \text{pk}_r, t, \mathbb{L})$

$\mathbb{L}' \leftarrow \text{ProcessEscrow}(\text{ctx}^{(b)}_{\text{esc}}, \text{token}, \mathbb{L})$

$(t, c_r^{(b)}, \pi^{(b)}_{\text{valid}}) \leftarrow \text{RequestRedeem}(\text{pk}_r, \text{aux}^{(b)}, \mathbb{L})$

$(\mathbb{L}', b_0) \leftarrow \text{ProcessRedeem}(\text{pk}_r, t, c_r^{(b)}, \pi^{(b)}_{\text{valid}}, \mathbb{L})$

$b' \leftarrow \mathcal{A}^{\text{Orc}}(\text{ctx}^{(b)}_{\text{esc}}, \text{token}, \text{pk}_r, t, c_r^{(b)}, \pi^{(b)}_{\text{valid}}, \mathbb{L})$

$b_1 := (L_e[\text{pk}_e].\text{IsCrpt} = 0) \wedge (L_r[\text{pk}_r].\text{IsCrpt} = 0)$

/ *Public keys involved are uncorrupted. */

return $b_0 \wedge b_1 \wedge (b = b')$

Besides, Considering the in-practice vulnerability to the deny-of-service (DoS) attack, we also consider of availability for our mixing protocol. See Appendix B for further discussion.

4 A UMCT SCHEME

In this section, we construct the UMCT scheme using commitment and NIZK, and provide the formal security proof.

4.1 Building Blocks for Our Construction

Commitment. We assume a vector commitment scheme $\mathsf{C} = (\text{Setup}, \text{Com})$ with hiding and binding properties[5]. In the escrow phase, to imply the unique redemption user's information, we define $\text{token} = \text{Com}(t, h(\text{pk}_r); r_t)$, where t is a tag, pk_r is the receiver's identity, and r_t is a random number. To prevent double redemption attack, t and pk_r will be revealed in the redemption phase, though they will remain secret in the escrow phase due to the hiding property of C.

Notably, we assume that $h(\text{pk}_r)$ is always in the message space of C. Specifically, we require $h(\cdot)$ to be a permutation mapping the space of pk to the message space of C.

NIZK. We recall that in UMCT, in order to redeem a coin without leaking any linkable information, the redemption user should provide a zero-knowledge proof π_{valid} indicating that this user redeems a valid coin escrowed before. For this purpose, we use the following zero-knowledge proof.

- **Membership proof**. To hide the corresponding escrowed coin, we prove that a re-encryption ciphertext c_r (which is revealed publicly) actually refers to "one-out-of-many" in the set of escrowed coins (in one epoch). Formally,

$$\text{Mem} = (\text{Setup}, \text{Prove}, \text{Verify})$$

is a complete, knowledge sound and zero-knowledge NIZK for the relation:

$$\begin{aligned} &\mathsf{R}_{\text{mem}}(S, t, \text{pk}_r, c_r; l, v, r_l, r_{\text{red}}, r_t) = 1 \\ \Leftrightarrow &l \in [N] \wedge c_e^{(l)} = \text{Enc}_{\text{pk}_l}(v; r_l) \wedge c_r = \text{Enc}_{\text{pk}_r}(v; r_{\text{red}}) \\ &\wedge \text{token}_l = \text{Com}(t, h(\text{pk}_r); r_t), \end{aligned} \tag{2}$$

where N is the number of the escrow transactions collected in the escrow phase, $S = \{(\text{pk}_i, c_e^{(i)}, \text{token}_i)\}_{i\in[N]}$ is the key information about the N escrow transactions, pk_i is the public key used to generate the i_{th} coin $c_e^{(i)}$ (our UMCT scheme allows duplicate public keys in S). Besides, l is the unique index of the escrowed coin, which is redeemed by c_r, in the set S.

Our UMCT scheme merits the following property.

- **No (additional) range proof is required**. In the mixing procedure, the redemption users *don't* need to prove that the coin value v (which is encrypted to c_r) is in the valid range. Intuitively, in our membership proof, we have claimed that c_r encrypts the *same* value as $c_e^{(l)}$. Moreover, the fact that v in $c_e^{(l)}$ is in the valid range is stipulated by the underlying confidential blockchain. Therefore, no more range-proof (than what could be used to form ctx_{esc}) in UMCT is required.

4.2 Construction Description

The UMCT starts with a global setup phase and the following epochs. Each epoch consists of an escrow phase and a redemption phase. Roughly, in each epoch, The tumbler is responsible for cutting off the connection between the escrowed coins and redeemed coins, which leads to the term "mixing". Figure 1 provides the algorithms overview of UMCT, and the detailed construction of UMCT is given as follows.

Setup phase. In the global setup phase, all public parameters used in the mixing protocol are generated and a smart contract is deployed. These public parameters are the implicit input for the following algorithms.

- $\text{pp} \leftarrow \text{Setup}(1^\lambda)$. On input security parameter 1^λ, runs $\text{pp}_{\text{ctx}} \leftarrow \text{CT.Setup}(1^\lambda), \text{pp}_{\text{com}} \leftarrow \text{C.Setup}(1^\lambda), \text{pp}_{\text{mem}} \leftarrow \text{Mem.Setup}(1^\lambda)$,

[5]The homomorphic property is not necessary for the general mixing construction but is required in Section 5 for the efficient instantiating.

deploys the smart contract (tumbler) to get pk_t, outputs $\mathsf{pp} = (\mathsf{pp}_{\mathsf{ctx}}, \mathsf{pp}_{\mathsf{com}}, \mathsf{pp}_{\mathsf{mem}}, \mathsf{pk}_t)$.

Escrow phase. During the escrow phase, escrow users create confidential transactions which transfer coins to the tumbler, triggering the mixing service. Concretely,

- $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathsf{aux}) \leftarrow \mathsf{CreateEscrow}(\mathsf{sk}_e, v, \mathsf{pk}_t, \mathsf{pk}_r, t, \mathbb{L})$. The escrow user with key pair $(\mathsf{pk}_e, \mathsf{sk}_e)$ runs this algorithm to create an escrow transaction. This algorithm can be executed by the following procedures.
  (1) On input the escrow user's secret key sk_e, the transaction value v, and the tumbler's public key pk_t, as introduced in Section 3.1, create a confidential transaction $\mathsf{ctx}_{\mathsf{esc}} := (\mathsf{pk}_e, \mathsf{pk}_t, c_e, c_t, \mathsf{info})$ by executing $\mathsf{CT.Create}\ (\mathsf{sk}_e, \mathsf{pk}_t, v)$. Namely, the user with pk_e escrows a confidential coin c_e to the tumbler.
  (2) Randomly choose a tag t and run Com to generate an identifier $\mathsf{token} = \mathsf{Com}(t, h(\mathsf{pk}_r); r_t)$.
  (3) Package secret information into $\mathsf{aux} = (v, r_{\mathsf{enc}}, t, \mathsf{pk}_r, r_t)$, where r_{enc} is the randomness used in c_e's generation.
  This algorithm outputs the resulting $\mathsf{ctx}_{\mathsf{esc}}$, token, aux. Specifically, the escrow user sends $(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token})$ to the tumbler as the function call, and sends aux to the receiver pk_r with the anonymous and confidential communication channels. Otherwise, the unlinkability between pk_e and pk_r is broken trivially.
- $(\mathbb{L}', 1/0) \leftarrow \mathsf{ProcessEscrow}(\mathsf{ctx}_{\mathsf{esc}}, \mathsf{token}, \mathbb{L})$. Upon the ProcessEscrow function being called by an escrow user, the tumbler executes the following checks.
  (1) $\mathsf{CT.Verify}(\mathsf{ctx}_{\mathsf{esc}}) = 1$.
  (2) token is unique in this epoch.
  If all checks pass, the tumbler adds $(\mathsf{pk}_e, c_e, \mathsf{token})$ to its public state, c_t to $\mathsf{bal}[\mathsf{pk}_t]$ and subtracts c_e from $\mathsf{bal}[\mathsf{pk}_e]$. Otherwise, the tumbler keeps its public state unchanged.

At the end of the escrow phase, the tumbler adds N $(\mathsf{pk}_e, c_e, \mathsf{token})$ triples to its public state S, or $S = \{(\mathsf{pk}_i, c_e^{(i)}, \mathsf{token}_i)\}_{i\in[N]}$, and then the redemption phase starts.

Redemption phase. In this phase, redemption users can generate redemption requests with the (secretly received) aux. After the tumbler verifies the validity of a redemption request (which means there exists one correspondingly escrowed coin), the redemption user can redeem a coin with the same value as the escrowed one. Meanwhile, other users cannot link this escrow/redemption pair.

- $(t, c_r, \pi_{\mathsf{mem}}) \leftarrow \mathsf{RequestRedeem}(\mathsf{pk}_r, \mathsf{aux}, \mathbb{L})$. The redemption user with public key pk_r executes this algorithm. With the received secret information $\mathsf{aux} := (v, r_{\mathsf{enc}}, t, \mathsf{pk}_r, r_t)$, the redemption user does the following procedures.
  (1) Encrypt the same value v under pk_r to get c_r, that is, $c_r = \mathsf{Enc}_{\mathsf{pk}_r}(v; r_{\mathsf{red}})$.
  (2) Generate a proof π_{mem} to justify the validity of the redemption request. That is,

$$\pi_{\mathsf{mem}} \leftarrow \mathsf{Mem.Prove}(S, t, \mathsf{pk}_r, c_r; l, v, r_{\mathsf{enc}}, r_{\mathsf{red}}, r_t),$$

  where Mem is a NIZK for $\mathsf{R}_{\mathsf{mem}}$ in equation (2).
  (3) Expose the tag t to prevent double redemption.
  Further, the redemption user sends $(\mathsf{pk}_r, t, c_r, \pi_{\mathsf{mem}})$ to the tumbler as the call to ProcessRedeem.
- $(\mathbb{L}', 1/0) \leftarrow \mathsf{ProcessRedeem}(\mathsf{pk}_r, t, c_r, \pi_{\mathsf{mem}}, \mathbb{L})$. Upon this function being called by a redemption user, the tumbler executing the following checks.
  (1) $\mathsf{Mem.Verify}(S, t, \mathsf{pk}_r, c_r, \pi_{\mathsf{mem}}) = 1$.
  (2) t is unique in this epoch till now, i.e., this coin has not been redeemed yet.
  If the check passes, output 1 and update the public state by adding t to the redeemed tags set and c_r to $\mathsf{bal}[\mathsf{pk}_r]$. Otherwise, output 0 to reject this request and do not change the public state.

4.3 Security Proof

This section gives the formal security proof for the UMCT construction. The correctness is straightforward from the correctness of the building blocks. In what follows, we focus on the other properties.

Theorem 4.1. *As designed in section 4.2,* UMCT=(Setup, CreateEscrow, ProcessEscrow, RequestRedeem, ProcessRedeem)*, is a balanced, unlinkable, and confidential mixing scheme for confidential blockchains.*

Proof. Corresponding to the detailed construction, the aux in the UMCT scheme can be parsed as $(v, r_{\mathsf{enc}}, t, \mathsf{pk}_r, r_t)$. We prove the theorem via three lemmas below, i.e., Lemma 4.2, 4.3 and 4.4. □

Lemma 4.2. UMCT*, as stated in Theorem 4.1, is a balance scheme based on the* binding *of commitment, and the* soundness *of π_{mem} for R_{mem} described in equation* (2).

Lemma 4.3. UMCT*, as stated in Theorem 4.1, is unlinkable based on the* zero-knowledge *property of π_{mem} for R_{mem} described in* (2)*, the* hiding *property of commitment and the* CPA security *of PKE.*

Lemma 4.4. UMCT*, as stated in Theorem 4.1, is confidential based on the* zero-knowledge *property of* Mem *described in* (2)*, the* confidentiality *of the underlying confidential payment system and the* CPA *security of PKE.*

The detailed proofs of these three lemmas are shown in Appendix C.1, C.2, and C.3, respectively.

5 BLINDSHUFFLER: AN EFFICIENT INSTANTIATION

In this section, we present BlindShuffler, an efficient instantiation of the UMCT construction proposed in Section 4. We emphasis that BlindShuffler is free of a trusted setup.

Specifically, for the underlying blockchain, we utilize the PGC construction [8], due to its competitive performance. To be fully compatible with PGC, we realize the membership proof in BlindShuffler by proposing a new zero knowledge proof, which supports heterogeneous public keys. Furthermore, we optimize the new protocol to reduce the proof size. Notably, the new zero knowledge proof can be directly applied to Zether by replacing the Twisted ElGamal ciphertext with the ElGamal ciphertext in Mem-proof, or to a broader range of confidential blockchains smoothly.

5.1 Instantiating of the Confidential Blockchain and the Token Commitment

Confidential blockchain. The PGC construction [8] is chosen for our instantiating. Roughly, PGC utilizes the Twisted-ElGamal to hide the coin value and thus integrates more effectively with Bulletproofs than other solutions. Refer to [8] for more discussions.

Token Commitment. The commitment used to instantiate our token construction is described in Figure 2, and denoted as C°. Here We use $\mathcal{G}$ to denote a group generation function which returns a cyclic group of order p.

$\mathsf{C}^\circ.\mathsf{Setup}(1^\lambda)$	$\mathsf{C}^\circ.\mathsf{Com}(t, h(\mathsf{pk}_r); r_t)$
$(\mathbb{G}, q) \leftarrow \mathcal{G}(1^\lambda)$	Commit $(t, h(\mathsf{pk}_r)) \in \mathbb{Z}_p^2$
$f_0, f_1, f_2 \leftarrow\!\$\ \mathbb{G}$	with $r_t \leftarrow\!\$\ \mathbb{Z}_p$
return $\mathbb{G}, q, f_0, f_1, f_2$	**return** $f_0^t f_1^{h(\mathsf{pk}_r)} f_2^{r_t} \in \mathbb{G}$

Figure 2: Commitment instantiation with C°.

C° is a standard 2-dimension vector Pedersen commitment [5]. In general, analogously, we can define d-dimension vector Pedersen commitment (with $d+1$ generators), notation C^d. So C° is actually C^2. The following theorem is due to [5].

Theorem 5.1. *If the discrete logarithm relation assumption holds for $\mathbb{G}$, the commitment scheme C° is computationally binding and perfectly hiding.*

5.2 Instantiating of the Membership Proof

Here we describe our efficient membership proof Mem *without* a trusted setup for the relation $\mathsf{R}_{\mathsf{mem}}$. Let us recall the proof setting in more detail.

The prover (i.e., a redemption user) and the verifier (i.e., the tumbler) hold the escrow transactions list, abstracted as $S = \{(\mathsf{pk}_i, c_e^{(i)}, \mathsf{token}_i)\}_{i\in[N]}$, which can be derived from the tumbler's public state. Then the prover, who has a coin c_r under pk_r together with a unique tag t, wishes to convince the verifier that the prover requests to redeem a unique coin in S. To achieve this, we assume that the prover takes the witness $(l, v, r_l, r_{\mathsf{red}}, r_t)$ as the input (e.g., the prover has received the witness from the escrow user privately). With this witness, the prover can convince the verifier that,
(1) c_r and $c_e^{(l)}$ encrypts the same value v, and
(2) $(t, h(\mathsf{pk}_r), r_t)$ is the opening of the l-th token (i.e., token_l),
via the membership proof Mem.

In what follows, we use three steps (which share the same setup phase) to provide an efficient instantiating of the above-mentioned membership proof. In the first step, we recall the *commitment to bits* technique, which is utilized in the next step to construct a *membership proof* Mem *supporting heterogeneous public keys*. In the last step, we reduce the Mem-proof size from the linearity to the *logarithm* w.r.t. the number of members. To begin, a unique setup phase is required for all the following constructions.

Setup. We assume that the number of the escrow transactions, i.e., N, equals n^m ($n \geq 2, m \geq 0$). Thus, any $u \in [N]$ can be expressed as the n-digit number of the m-bit. For instance, $u = \sum_{j=0}^{m-1} u_j n^j := \langle u_0, u_1, ..., u_{m-1}\rangle$, where $u_j \in [n]$ is the j-th digit of u. We will use mn-dimension vector Pedersen commitment C^{nm}, with its $mn+1$ generators denoted by the vector $\mathbf{ck}$. For the sake of convenience, in the construction of the protocols, we take $\mathsf{C}^\triangle$ as an alias for C^{nm}.

Specifically, run the same group generation function $\mathcal{G}$ together with the setup algorithms for the vector Pedersen commitment schemes (with dimension mn and 2 respectively), and the Twisted ElGamal scheme. Thus, the three algorithms, including $(\mathbb{G}, p, \mathbf{ck}) \leftarrow \mathsf{C}^\triangle.\mathsf{Setup}$, $(\mathbb{G}, p, f_0, f_1, f_2) \leftarrow \mathsf{C}^\circ.\mathsf{Setup}$, $(\mathbb{G}, p, g, h) \leftarrow \mathsf{TElG.Setup}$, have the same $\mathbb{G}$ and p.

STEP 1: Commitment to bits. Here we recall the zero knowledge proof introduced by Bootle *et al.*, [4], which proves that a commitment B can be opened to m sequences of bits. Especially, each sequence consists of n ordered 0/1 bits and exactly only one of them is 1. More formally, the resulting protocol, defined as Π^{bin}, is a protocol for the relation

$$\begin{aligned} &\mathsf{R}_{\mathsf{bin}}(B; b_{0,0}, b_{0,1}, \cdots, b_{m-1,n-1}, r_B) = 1 \\ \Leftrightarrow &\forall i \in [n], j \in [m] : b_{j,i} \in \{0,1\} \wedge \forall j \in [m], \Sigma_{i=0}^{n-1} b_{j,i} = 1 \\ &\wedge B = \mathsf{C}^\triangle.\mathsf{Com}(\{b_{j,i}\}_{j,i=0}^{m-1,n-1}; r_B). \end{aligned} \tag{3}$$

Recall that $\{b_{j,i}\}_{j,i=0}^{m-1,n-1}$ denotes the sequence $(b_{0,0}, \cdots, b_{m-1,n-1})$.

The description of Π^{bin} is given in Appendix F. Then we have,

Lemma 5.2. *If the discrete logarithm relation assumption holds for $\mathbb{G}$, Π^{bin}, as described in Figure 5, is perfectly complete, knowledge sound and perfectly SHVZK.*

Proof. The proof could be found in Appendix B of [4]. □

STEP 2: Mem-proof supporting heterogeneous public keys. To start with, since we instantiate the underlying confidential coin as the Twisted ElGamal ciphertext (introduced in Section 2.2), we can rewrite $\mathsf{R}_{\mathsf{mem}}$ as follows,

$$\begin{aligned} &\mathsf{R}_{\mathsf{mem}}(S, t, \mathsf{pk}_r, c_r; l, r_l, r_{\mathsf{red}}, r_t) = 1 \\ \Leftrightarrow &l \in [N] \wedge c_e^{(l)}/c_r = \mathsf{Enc}_{\mathsf{pk}_l}(0; r_l) \cdot \mathsf{Enc}_{\mathsf{pk}_r}(0; -r_{\mathsf{red}}) \\ &\wedge \mathsf{token}_l = \mathsf{C}^\circ.\mathsf{Com}(t, h(\mathsf{pk}_r); r_t). \end{aligned} \tag{4}$$

To construct the Mem-proof for above $\mathsf{R}_{\mathsf{mem}}$ with the witness tuple $(l, r_l, r_{\mathsf{red}}, r_t)$, we can split our task into two-fold.

- First, we need to prove that the l_{th} token (i.e., token_l) is a commitment that opens to $t, h(\mathsf{pk}_r)$.
- Secondly, we need to prove that the l_{th} ciphertext $c_e^{(l)}$ encrypts the same value as c_r (i.e., $c_e^{(l)}/c_r = \mathsf{Enc}_{\mathsf{pk}_l}(0; r_l) \cdot \mathsf{Enc}_{\mathsf{pk}_r}(0; -r_{\mathsf{red}})$[6]). Inspired by [4], our core idea lies in proving the knowledge of l for which $\prod_{u=0}^{N-1}(c_e^{(u)}/c_r)^{\delta_{l,u}}$ is the product of two ciphertexts encrypting 0 under pk_l and pk_r respectively, where $\delta_{l,u}$ is defined to be 1 if $u = l$ and 0 otherwise. More concretely, writing $u = \sum_{j=0}^{m-1} u_j n^j$ and $l = \sum_{j=0}^{m-1} l_j n^j$, where u_j and $l_j \in [n]$, we have $\delta_{l,u} = \prod_{j=0}^{m-1} \delta_{l_j,u_j}$[7]. Thereafter, we can reformulate what we want to prove as that $\prod_{u=0}^{N-1}(c_e^{(u)}/c_r)^{\prod_{j=0}^{m-1}\delta_{l_j,u_j}}$ is the product of two ciphertexts encrypting 0 under pk_l and pk_r respectively. Note that pk_l cannot be leaked during the proof.

The first part can be easily achieved by exploiting the standard one-out-of-many proof technique in [4]. Specifically, instantiate the equation (1) with $\{c_i\}_{i\in[N]}$, which are the 2-dimension vector Pedersen commitments shown in Figure 2.

However, to achieve the second part of our proof, a direct imitating of [4] will result in a complete loss of unlinkability. The

[6]This formula holds based on the discrete logarithm assumption.
[7]Encoding of l using $\delta_{l,u}$ requires a vector of length n^m, while encoding l using δ_{l_j,u_j} requires a vector of mn-length only.

key observation of equation (1) is that [4] achieves the standard one-out-of-many proof by taking a *single* public key as part of the *statement*. As a result, using the standard one-out-of-many proof for our relation $\mathsf{R}_{\mathsf{mem}}$ would inevitably expose the l_{th} public key (i.e., pk_l) in the statement, which breaks the unlinkability. Thus, more sophisticated considerations are necessary to hide pk_l.

To bypass this obstacle, we pull all the heterogeneous public keys (i.e., $\{\mathsf{pk}_u\}_{u\in[N]}$) into our Mem-proof's verification phase. In other word, $\{\mathsf{pk}_u\}_{u\in[N],u\neq l}$ work as the "blindage" of pk_l. We provide our zero knowledge protocol Mem for the mentioned $\mathsf{R}_{\mathsf{mem}}$ in Figure 3, and highights the difference to [4] in lightgray.

- Prover
  - Encodes l to a mn-dimentsional vector $(\delta_{l_0,0},\cdots,\delta_{l_{m-1},n-1})$
  - Picks $r_B, \{\rho_k^{(u)}\}_{k,u=0}^{m-1,N-1}, \{\eta_k,\gamma_k\}_{k=0}^{m-1} \leftarrow\$ \mathbb{Z}_p$
  - Computes $B = \mathsf{C}^{\triangle}.\mathsf{Com}(\delta_{l_0,0},\delta_{l_0,1},\cdots,\delta_{l_{m-1},n-1};r_B)$
  - $A,C,D \leftarrow \Pi^{\text{bin}}.\mathsf{Prove}(B,(\{\delta_{l_j,i}\}_{j,i=0}^{m-1,n-1},r_B))$
  - $\mathsf{token}_{\mathsf{tmp}} = \mathsf{C}^{\circ}.\mathsf{Com}(t,h(\mathsf{pk}_r);0)$
  - $\forall k\in[m]$, computes
    - $G_k = \left(\prod_{u=0}^{N-1}(c_e^{(u)}/c_r)^{p_{u,k}}\mathsf{Enc}_{\mathsf{pk}_u}(0;\rho_k^{(u)})\right)\mathsf{Enc}_{\mathsf{pk}_r}(0;\eta_k)$
      / *$p_{u,k}$ is defined in equation (5).* /
    - $H_k = \prod_{u=0}^{N-1}(\mathsf{token}_u/\mathsf{token}_{\mathsf{tmp}})^{p_{u,k}}\mathsf{C}^{\circ}.\mathsf{Com}(0,0;\gamma_k)$
  - Sends: $A,B,C,D,\{G_k,H_k\}_{k\in[m]}$
- Verifier
  - Sends $x \leftarrow\$ \mathbb{Z}_p$
- Prover sends
  - $\{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}, z_A, z_C \leftarrow \Pi^{\text{bin}}.\mathsf{Prove}(x)$
  - $\forall u\neq l, z_u = -\sum_{k=0}^{m-1}\rho_k^{(u)}x^k$
  - $z_l = r_l\cdot x^m - \sum_{k=0}^{m-1}\rho_k^{(l)}x^k$
  - $z_{\text{red}} = -r_{\text{red}}\cdot x^m - \sum_{k=0}^{m-1}\eta_k x^k$
  - $z_t = r_t\cdot x^m - \sum_{k=0}^{m-1}\gamma_k x^k$
- Verifier
  - checks $\Pi^{\text{bin}}.\mathsf{Verify}(B,A,C,D,\{f_{j,i}\}_{j=0,i=1}^{m-1,n-1},z_A,z_C) \stackrel{?}{=} 1$
  - computes $\forall j\in[m], f_{j,0} = x - \Sigma_{i=1}^{n-1}f_{j,i}$
  - computes $c = \prod_{u=0}^{N-1}(c_e^{(u)}/c_r)^{\prod_{j=0}^{m-1}f_{j,u_j}}\prod_{k=0}^{m-1}G_k^{-x^k}$
  - checks $c \stackrel{?}{=} \prod_{u=0}^{N-1}\mathsf{Enc}_{\mathsf{pk}_u}(0;z_u)\mathsf{Enc}_{\mathsf{pk}_r}(0;z_{\text{red}})$
  - $\prod_{u=0}^{N-1}(\mathsf{token}_u/\mathsf{token}_{\mathsf{tmp}})^{\prod_{j=0}^{m-1}f_{j,u_j}}\prod_{k=0}^{m-1}H_k^{-x^k}$ $\stackrel{?}{=} \mathsf{C}^{\circ}.\mathsf{Com}(0,0;z_t)$, where $\mathsf{token}_{\mathsf{tmp}} = \mathsf{C}^{\circ}.\mathsf{Com}(t,h(\mathsf{pk}_r);0)$

Figure 3: The description of zero knowledge proof Mem for $\mathsf{R}_{\mathsf{mem}}$, where we highlight our modification in lightgray compared with the standard one-out-of-many proof in [4].

Intuitively, in Mem-proof's "commit phase", the prover starts by commiting to an mn-dimension vector $\delta_{l_0,0},\cdots,\delta_{l_{m-1},n-1}$ and derives B in R_{bin}. Then, it runs $\Pi^{\text{bin}}.\mathsf{Prove}$ to claim that B commits to binary bits and $\forall j\in[m], \sum_{i=0}^{n-1}\delta_{l_j,i}=1$. Upon receiving the challenge x from the verifier, the prover reveals $f_{j,i} = \delta_{l_j,i}x + a_{j,i}$ for all $j\in[m], i\in[n]$. For each $u\in[N]$, define a polynomial $p_u(x) = \prod_{j=0}^{m-1}f_{j,u_j} = \prod_{j=0}^{m-1}(\delta_{l_j,u_j}x + a_{j,u_j})$. For each $u\in[N]$, we rephrase the polynomial $p_u(x)$ as below.

$$p_u(x) = \prod_{j=0}^{m-1}(\delta_{l_j,u_j}x) + \sum_{k=0}^{m-1}p_{u,k}x^k = \delta_{l,u}x^m + \sum_{k=0}^{m-1}p_{u,k}x^k, \quad (5)$$

where coefficients $p_{u,k}$ depends on l, a_{j,u_j} and can be computed by the prover before receiving x. The key observation is that $p_l(x)$ is the only polynomial with degree m. That is, $p_u(x)$ with $u\neq l$ is at most $(m-1)$-degree. Exploiting this property, for $\forall k\in[m]$, define

$$G_k = \prod_{u=0}^{N-1}(c_e^{(u)}/c_r)^{p_{u,k}}\mathsf{Enc}_{\mathsf{pk}_u}(0;\rho_k^{(u)})\mathsf{Enc}_{\mathsf{pk}_r}(0;\eta_k), \quad (6)$$

where $\mathsf{Enc}_{\mathsf{pk}_u}(0;\rho_k^{(u)})$ for $u\in[N]$ and $\mathsf{Enc}_{\mathsf{pk}_r}(0;\eta_k)$ are necessary to hide all the witnesses in equation (4). $\{G_k\}_{k\in[m]}$ are sent by the prover before seeing x (i.e., also in the commit phase) to "cancel out" these lower-order terms in the verifier's final verification. Therefore, the verifier can check if

$$\prod_{u=0}^{N-1}(c_e^{(u)}/c_r)^{\prod_{j=0}^{m-1}f_{j,u_j}}\prod_{k=0}^{m-1}G_k^{-x^k} = \prod_{u=0}^{N-1}\mathsf{Enc}_{\mathsf{pk}_u}(0;z_u)\mathsf{Enc}_{\mathsf{pk}_r}(0;z_{\text{red}}). \quad (7)$$

During the verification, all z_u shares the same distribution and all pk_u is used, so the real escrow user pk_l is hidden among all public keys from the view of verifier. Compared to [4], the construction in Figure 3 supports the Mem-proof for heterogeneous public keys. However, this proof suffers from the linearly proof size. Fortunately, we can reduce the cost to a logarithmic of N, in the next step.

STEP 3: The logarithmic membership proof. Utilizing a variant of the inner product argument construction proposed in Bulletproof [5], we can optimize the previous step.

Recall that in the membership proof under heterogeneous public keys, i.e., $\mathsf{pk}_0,\cdots,\mathsf{pk}_{N-1}$, the prover sends $\mathbf{z} = (z_0,\cdots,z_{N-1})$, z_{red} to the verfier. Obviously, the transcripts size grows linearly with N. Now, our goal is to optimize our Mem-proof to achieve logarithmic size in N.

Considering the equation (7), instead of transferring $\mathbf{z}, z_{\text{red}}$, we wish to give a logarithmic proof that claims

$$c = \prod_{u=0}^{N-1}\mathsf{Enc}_{\mathsf{pk}_u}(0;z_u)\mathsf{Enc}_{\mathsf{pk}_r}(0;z_{\text{red}}), \quad (8)$$

where $\mathbf{z}, z_{\text{red}}$ are witnesses and the leftside c holds to be

$$\prod_{u=0}^{N-1}(c_e^{(u)}/c_r)^{\prod_{j=0}^{m-1}f_{j,u_j}}\prod_{k=0}^{m-1}G_k^{-x^k},$$

which can be computed by both the prover and the verifier.

To solve this problem, we define Π^{dlr} as a zero-knowledge protocol for the following discrete logarithm relation,

$$\mathsf{R}_{\text{dlr}}(c,g_0,\cdots,g_{N-1},g_{\text{red}};\mathbf{z},z_{\text{red}}) = 1 \Leftrightarrow c = \prod_{i=0}^{N-1}g_i^{z_i}g_{\text{red}}^{z_{\text{red}}}, \quad (9)$$

where g_i is a tuple (pk_i,g), g_{red} is a tuple (pk_r,g), and we denote $(g_0,\cdots,g_{N-1})$ by $\mathbf{g}$. Figure 4 presents Π^{dlr}.

If $N=1$,
- Prover
  - Picks $\alpha_0,\alpha_1 \leftarrow\$ \mathbb{Z}_p$
  - Computes and sends $d = g_0^{\alpha_0}g_{\text{red}}^{\alpha_1}$
- Verifier sends $x \leftarrow\$ \mathbb{Z}_p$
- Prover computes and sends $\theta_0 = \alpha_0 - xz_0$, $\theta_1 = \alpha_1 - xz_{\text{red}}$

- Verifier checks $c^x g_0^{\theta_0} g_{\text{red}}^{\theta_1} = d$

If $N > 1$, $N' = N/2$

- Prover
  - Picks $r_0, r_1 \leftarrow\$ \mathbb{Z}_p$
  - Computes and sends $L = \mathbf{g}_{[N':]}^{\mathbf{z}_{[:N']}} \cdot g_{\text{red}}^{r_0}$, $R = \mathbf{g}_{[:N']}^{\mathbf{z}_{[N':]}} \cdot g_{\text{red}}^{r_1}$
- Verifier sends $x \leftarrow\$ \mathbb{Z}_p$
- Prover and verifier computes
  - $\mathbf{g}' = \mathbf{g}_{[:N']}^{x^{-1}} \circ \mathbf{g}_{[N':]}^{x}$, $c' = c \cdot L^{x^2} R^{x^{-2}}$
- Prover computes
  - $\mathbf{z}' = x\mathbf{z}_{[:N']} + x^{-1}\mathbf{z}_{[N':]}$, $z'_{\text{red}} = z_{\text{red}} + x^2 r_0 + x^{-2} r_1$

Recursively run Π^{dlr} on $\mathsf{R}_{\text{dlr}}(c', \mathbf{g}', g_{\text{red}}; \mathbf{z}', z'_{\text{red}})$

Figure 4: The description of protocol Π^{dlr} for R_{dlr}.

Lemma 5.3. *If the discrete logarithm relation assumption holds for $\mathbb{G}$, Π^{dlr}, as described in Figure 4, is perfectly complete, knowledge sound, and perfectly SHVZK.*

Theorem 5.4. *If the discrete logarithm relation assumption holds for $\mathbb{G}$, combined with Π^{dlr}, the* Mem*-proof with logarithmic size is perfectly complete, knowledge sound and perfectly SHVZK.*

The formal proofs of Lemma 5.3 and Theorem 5.4 are shown in Appendix D and Appendix E, respectively.

Non-interactive proof through Fiat-Shamir heuristic. So far the presented Mem-proof works as an interactive protocol with a logarithmic number of rounds. Similar to [5], this protocol can be converted into a non-interactive protocol (with logarithmic proof length) using the Fiat-Shamir heuristic [1], i.e., by replacing all random challenges with hashes of the transcript up to the point.

6 IMPLEMENTATION AND COMPARISON

We implement BlindShuffler as an add-on for PGC. The mixing service works as a smart contract interacting with PGC. For PGC, we use the version instantiated with the Ethereum smart contract.

We use Truffle Suite [29] as the development environment. The tumbler is implemented with the Solidity programming language, as a smart contract running on the top of the Ethereum virtual machine (EVM). Escrow/redemption users are implemented with Javascript. The performance is evaluated on a laptop with a 2.3GHz Intel Core i5-8300H processor and the source code is available at https://github.com/boing-lee/MixSC.

Table 3: Communication and computation overhead of one escrow/redemption transaction in BlindShuffler (N is the anonymity size).

Tx Type	PGC transfer [8]	Escrow	Redeem				
			$N=4$	$N=8$	$N=16$	$N=32$	$N=64$
Tx Size (KB)	1.41	1.73	2.98	3.62	4.26	4.90	5.54
Gas (Units)	7,188K	7,484K	939K	1,456K	2,352K	4,022K	7,303K

Evaluation. To demonstrate the efficiency of BlindShuffler, we change the number N of mixing users in each epoch from 4 to 64, and record the communication and computation costs in the escrow and the redemption procedures. Communication and computation costs are measured by the on-chain transaction size and transaction *gas* paid to the blockchain miners respectively. The results are collected in Table 3, where we use the confidential transaction in PGC as the baseline. When the achieved anonymity level is $N = 64$, the computation cost of BlindShuffler only doubles that of PGC.

Cost comparison. Using a plain transfer (confidential but no anonymity) in the confidential blockchains as a comparison basis, we contrast the cost of BlindShuffler with Anonymous Zether [9]. We make this choice of comparison since BlindShuffler and Anonymous Zether are the only two works known to achieve anonymity on PKE-based confidential transactions. See the results in Table 4.

Table 4: Communication and computation overhead of one payment procedure in a transfer.

Schemes	An Original Transfer* [6, 8]	Anonymous Zether [9]	BlindShuffler
Tx Size (KB)	≥ 1.31	6.15	5.99
Gas (Units)	$\geq 7,188$K	10,890K	9,836K
Gas Cost Percentage of Original Transfer	100%	152%	137%
Tx Size Complexity	$\geq 32\mathbb{G} + 11\mathbb{Z}_p$	$(2N + 8\log N + 20)\mathbb{G}$ $+(2\log N + 10)\mathbb{Z}_p$	$(7\log N + 14)\mathbb{G}$ $+(2\log N + 9)\mathbb{Z}_p$

* We use the smaller values from the original transactions as the baseline.

- For a comparison, let's fix the anonymity set size N to be 16 (a most frequent size). In BlindShuffler, for an escrow/redemption user pair, transferring a PGC coin via the tumbler costs 2648K gas, which accounts for 37% of the original transfer. In other words, we achieve an anonymous PGC transaction with 137% of the original cost. In contrast, Anonymous Zether [9] costs 152% of the oroginal transfer to achieve the same (i.e., $N = 16$) level of anonymity. Therefore, BlindShuffler is noticeably cheaper than Anonymous Zether. Moreover, Blindshuffler is fully compatible with PGC whereas Anoymous Zether works as a new blockchain.
- *As the anonymity set size N increases*, the advantages of our solution will become more obvious. Concretely, our solution's transaction size complexity only grows logarithmically with respect to N, while in contrast, this complexity grows linearly in Anonymous Zether.

In summary, BlindShuffler provides a more practical method to achieve the same anonymous set size for confidential transactions.

7 CONCLUSION

In this paper, we present a universal mixing service protocol for confidential blockchains utilizing homomorphic PKE. To design this protocol, we propose a new NIZK argument of knowledge, called Mem-proof. We also instantiate this mixing protocol upon PGC, named BlindShuffler, to justify the efficiency of our solution.

ACKNOWLEDGMENTS

We thank Yu Chen and the anonymous reviewers for their help and constructive suggestions. We also thank Yiqi Liu and Boyang Li for their effort in the implementation of BlindShuffler. This work has been supported in part by the National Key Research and Development Project 2020YFA0712300, Shanghai Science and Technology Innovation Action Plan (Grant No. 23511101100), National Natural Science Foundation of China (Grant No. 62272294), and the Key Research and Development Plan of Shandong Province (No. 2021CXGC010105).

REFERENCES

[1] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. 2002. From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security. In EUROCRYPT. Springer, 418–433.

[2] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In S&P. IEEE, 459–474.

[3] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. 2014. Deanonymisation of Clients in Bitcoin P2P Network. In ACM CCS. 15–29.

[4] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. 2015. Short accountable ring signatures based on DDH. In ESORICS. Springer, 243–265.

[5] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In 2018 IEEE symposium on security and privacy (SP). IEEE, 315–334.

[6] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards Privacy in a Smart Contract World. In FC. Springer, 423–443.

[7] Matteo Campanelli and Mathias Hall-Andersen. 2022. Veksel: Simple, Efficient, Anonymous Payments with Large Anonymity Sets from Well-Studied Assumptions. In Asia CCS. 652–666.

[8] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Au. 2020. PGC: Decentralized Confidential Payment System with Auditability. In ESORICS. Springer, 591–610.

[9] Benjamin E. Diamond. 2021. Many-out-of-Many Proofs and Applications to Anonymous Zether. In S&P. IEEE, 1800–1817.

[10] Jiajun Du, Zhonghui Ge, Yu Long, Zhen Liu, Shifeng Sun, Xian Xu, and Dawu Gu. 2022. MixCT: Mixing Confidential Transactions from Homomorphic Commitment. In ESORICS. Springer, 763–769.

[11] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. 2019. Quisquis: A New Design for Anonymous Cryptocurrencies. In ASIACRYPT. Springer, 649–678.

[12] Zhonghui Ge, Jiayuan Gu, Chenke Wang, Yu Long, Xian Xu, and Dawu Gu. 2023. Accio: Variable-Amount, Optimized-Unlinkable and NIZK-Free Off-Chain Payments via Hubs. In ACM CCS. 1541–1555.

[13] Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. 2022. Foundations of Coin Mixing Services. In ACM CCS. 1259–1273.

[14] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In Advances in Cryptology–EUROCRYPT 2016. Springer, 305–326.

[15] Jens Groth and Markulf Kohlweiss. 2015. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Advances in Cryptology - EUROCRYPT. Springer, 253–280.

[16] Ethan Heilman, Leen AlShenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2017. TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub. In NDSS.

[17] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. 2019. Omniring: Scaling Private Payments Without Trusted Setup. In ACM CCS. 31–48.

[18] Gregory Maxwell. 2015. Confidential transactions. https://people.xiph.org/~greg/confidential_values.txt

[19] S Meiklejohn and R Mercer. 2018. Möbius: Trustless Tumbling for Transaction Privacy. PoPETs 2018, 2 (2018), 105–121.

[20] Satoshi Nakamoto. 2008. A peer-to-peer electronic cash system. (2008), 15.

[21] Tornado Cash Nova. 2021. https://github.com/tornadocash/tornado-nova

[22] Ethereum Project: Blockchain App Platform. 2023. https://www.ethereum.org/

[23] Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, Oğuzhan Ersoy, Amin Sakzad, Muhammed F Esgin, Joseph K Liu, Jiangshan Yu, and Tsz Hon Yuen. 2023. Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts. In S& P. IEEE, 2462–2480.

[24] Dorit Ron and Adi Shamir. 2013. Quantitative Analysis of the Full Bitcoin Transaction Graph. In FC. Springer, 6–24.

[25] Tim Ruffing and Pedro Moreno-Sanchez. 2017. ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin. In FC. Springer, 133–154.

[26] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2014. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In ESORICS. Springer, 345–364.

[27] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2017. P2P Mixing and Unlinkable Bitcoin Transactions. In NDSS.

[28] István András Seres, Dániel A Nagy, Chris Buckland, and Péter Burcsi. 2019. Mixeth: efficient, trustless coin mixing service for ethereum. Cryptology ePrint Archive (2019).

[29] Truffle Suite. 2022. https://trufflesuite.com/

[30] Shi-Feng Sun, Man Ho Au, Joseph K. Liu, and Tsz Hon Yuen. 2017. RingCT 2.0: A Compact Accumulator-Based Protocol for Blockchain Cryptocurrency Monero. In ESORICS. Springer Verlag, 456–474.

[31] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. 2021. A^2L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs. In S&P. IEEE, 1834–1851.

[32] Luke Valenta and Brendan Rowan. 2015. Blindcoin: Blinded, Accountable Mixes for Bitcoin. In Financial Cryptography and Data Security. Springer, 112–126.

[33] Tianyu Zheng, Shang Gao, Yubo Song, and Bin Xiao. 2023. Leaking Arbitrarily Many Secrets: Any-out-of-Many Proofs and Applications to RingCT Protocols. In 2023 IEEE Symposium on Security and Privacy (SP). IEEE, 2533–2550.

A DEFINITIONS OF ZERO-KNOWLEDGE ARGUMENTS

Definition A.1 (NIZK). The triple $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a NIZK for relation R if it satisfies the following three definitions.

Definition A.2 (Perfect Completeness). A non-interactive zero-knowledge argument protocol NIZK is perfectly complete if the prover who knows a witness of a statement could always convince the verifier to accept. Formally,

$$\Pr\left[\mathsf{Verify}(\sigma, x, \pi) = 1 \vee (x, w) \notin \mathsf{R}_{\mathcal{L}} \;\middle|\; \begin{matrix}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda);\\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w)\end{matrix}\right] = 1.$$

Definition A.3 (Knowledge Soundness). For a non-interactive zero-knowledge argument protocol NIZK, knowledge soundness holds if the prover "knows" a witness for the given accepted instance. Namely, for all non-uniform polynomial time prover $\mathcal{P}^*$, there exists a PPT extractor Ext such that,

$$\Pr\left[\begin{matrix}(x, w) \notin \mathsf{R}_{\mathcal{L}} \wedge\\ \mathsf{Verify}(\sigma, x, \pi) = 1\end{matrix} \;\middle|\; \begin{matrix}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda);\\ (x, \pi; w) \leftarrow \mathsf{Ext}^{\mathcal{P}^*}(\sigma, x);\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

where $\mathsf{Ext}^{\mathcal{P}^*}$ means the extractor gets full access to the prover's state, including any random coins.

Definition A.4 (Zero-knowledge). A non-interactive zero-knowledge argument protocol NIZK is zero-knowledge if the verifier cannot get additional information except that the statement is true. Formally, for $x \in \mathcal{L}$, there exists a PPT algorithm Sim such that for all adversaries $\mathcal{A}$,

$$\Pr\left[\mathcal{A}(\sigma, \tau, x, \pi) = 1 \;\middle|\; \begin{matrix}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda);\\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w)\end{matrix}\right] \approx$$
$$\Pr\left[\mathcal{A}(\sigma, \tau, x, \pi) = 1 \;\middle|\; \begin{matrix}(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda);\\ \pi \leftarrow \mathsf{Sim}(\tau, x)\end{matrix}\right].$$

A zero-knowledge arguments of knowledge for $(x, w) \in \mathsf{R}_{\mathcal{L}}$ consists of three interactive algorithms (Setup, Prove, Verify) running in probabilistic polynomial time. On input the security parameter 1^λ, Setup outputs a common reference string σ. The transcripts generated by the interaction between Prover and Verifier is denoted by $tr \leftarrow \langle \mathsf{Prove}(\sigma, x, w), \mathsf{Verify}(\sigma, x)\rangle$. We write the output of interactive protocol as $b = \langle \mathsf{Prove}(\sigma, x, w), \mathsf{Verify}(\sigma, x)\rangle$. If the verifier accepts $b = 1$, otherwise $b = 0$.

We require that the zero-knowledge arguments of knowledge for $\mathsf{R}_{\mathcal{L}}$ have perfect completeness, knowledge soundness, and perfect special honest verifier zero knowledge (SHVZK). We refer readers to [5] for the detailed definitions of these three properties.

B AVALIABILITY OF UMCT

Avaliability. Availability means that the UMCT is DoS resistant, which requires the UMCT scheme to resist DoS attacks, and any honest users can terminate without losing money.

Avaliability analysis. It is irrational for the adversary to launch a DoS attack since users pay for the mixing service before redeeming

it. We emphasize that this is *not* the *rational people hypothesis* used in blockchain since an irrational adversary towards our protocol can only hurt its benefit rather than the security of the mixing service. So our universal mixing service satisfies availability.

C SECURITY PROOF FOR UMCT SCHEME

C.1 Proof of Lemma 4.2

As described in Section 3.4, we prove the balance property in 4 cases, where we consider a challenger C that runs an adversary $\mathcal{A}$ in the Balance experiment. Note that, to redeem coins, every redemption user generates a valid zero-knowledge proof π_{mem}. Therefore, exploiting the extractor $\text{Ext}^{\mathcal{P}}$ for the Mem, C extracts $(l, v, r_l, r_{\text{red}}, r_t)$, where $l \in [N]$, with the following properties,

$$c_e^{(l)} = \text{Enc}_{\text{pk}_l}(v; r_l) \wedge c_r = \text{Enc}_{\text{pk}_r}(v; r_{\text{red}}), \tag{10}$$

$$\text{token}_l = \text{Com}(t, h(\text{pk}_r); r_t). \tag{11}$$

If π_{mem} is valid, we first prove that the equation $(10, 11)$ holds with all but a negligible probability. Otherwise, if the equation $(10, 11)$ don't hold, we can construct an adversary $\mathcal{B}$ to break the soundness of π_{mem} for R_{mem}. Specifically, the algorithm $\mathcal{B}$ proceeds as follows. Receiving σ generated by $(\sigma, \tau) \leftarrow \text{Mem.Setup}(1^\lambda)$ from $\mathcal{B}$'s soundness challenger, $\mathcal{B}$ simulates the Balance experiment. Upon $\mathcal{B}$ (which is on behalf of the balance adversary's challenger C), executing $\text{Ext}^{\mathcal{P}}(S, t, \text{pk}_r, c_r)$ to derive witnesses $(l, v, r_l, r_{\text{red}}, r_t)$ which makes the equation $(10, 11)$ not valid, $\mathcal{B}$ sends (S, t, pk_r, c_r), π_{mem} and the extracted witnesses to $\mathcal{B}$'s challenger and wins. Therefore, the probability that $(10, 11)$ don't hold is $\text{Adv}^{\text{soundness}}_{\mathcal{A},\text{Mem}}(\lambda)$, which is negligible.

Therefore, in the following discussion, we use $\text{Adv}^{\text{soundness}}_{\mathcal{A},\text{Mem}}(\lambda)$ to bound the errors introduced by the calling of $\text{Ext}^{\mathcal{P}}$.

Case 1: user theft. Let $\mathcal{E}_{\text{utheft}}$ be the event that $\mathcal{A}$ wins in the following way.

For the valid challenge $(\text{pk}_{\mathcal{A}}, t_{\mathcal{A}}, c_{\mathcal{A}}, \pi_{\text{mem}})$ from $\mathcal{A}$, the event that $L_e[t_{\mathcal{A}}].\text{pk}_r \neq \text{pk}_{\mathcal{A}}$ and $L_e[t_{\mathcal{A}}].\text{pk}_r$ is an honest user occurs. That is, $\text{pk}_{\mathcal{A}}$ steals the coin targeting honest $L_e[t_{\mathcal{A}}].\text{pk}_r$. In this case, upon receiving the challenge $(\text{pk}_{\mathcal{A}}, c_{\mathcal{A}}, t_{\mathcal{A}}, \pi_{\text{mem}})$ from $\mathcal{A}$ and this request is confirmed, C runs the extractor $\text{Ext}^{\mathcal{P}}(S, t_{\mathcal{A}}, \text{pk}_{\mathcal{A}}, c_{\mathcal{A}})$ of Mem, then derives the witnesses $(l, v, r_l, r_{\text{red}}, r_t)$ satisfying

$$\text{token}_l = \text{Com}(t_{\mathcal{A}}, h(\text{pk}_{\mathcal{A}}); r_t). \tag{12}$$

On the other hand, the fact that the coin targets $L_e[t_{\mathcal{A}}].\text{pk}_r$ implies

$$\text{token}_l = \text{Com}(t_{\mathcal{A}}, h(L_e[t_{\mathcal{A}}].\text{pk}_r); r'_t), \tag{13}$$

where the challenger knows $r'_t \in \text{aux}'$, since the challenger C can receive aux' on behalf of the honest $L_e[t_{\mathcal{A}}].\text{pk}_r$.

Since $(t_{\mathcal{A}}, h(L_e[t_{\mathcal{A}}].\text{pk}_r))$ is different from $(t_{\mathcal{A}}, h(\text{pk}_{\mathcal{A}}))$, we can design an algorithm $\mathcal{B}_1$ to break the binding property of the commitment C. Given $\text{pp} \leftarrow \text{C.Setup}(1^\lambda)$, $\mathcal{B}_1$ simulates experiment $\text{Balance}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda)$. Once the event above occurs, i.e., $L_e[t_{\mathcal{A}}].\text{pk}_r \neq \text{pk}_{\mathcal{A}}$, $\mathcal{B}_1$ can forward $(t_{\mathcal{A}}, h(\text{pk}_{\mathcal{A}}), r_t)$ and $(t_{\mathcal{A}}, h(L_e[t_{\mathcal{A}}].\text{pk}_r), r'_t)$ to $\mathcal{B}_1$'s challenger. $\mathcal{B}_1$ wins if and only if $L_e[t_{\mathcal{A}}].\text{pk}_r \neq \text{pk}_{\mathcal{A}}$.

In summary, we can derive

$$\Pr[\mathcal{E}_{\text{utheft}} \text{ occurs}] \leq \text{Adv}^{\text{binding}}_{\mathcal{A},\text{C}}(\lambda) + \text{Adv}^{\text{soundness}}_{\mathcal{A},\text{Mem}}(\lambda).$$

Case 2: system theft. Let $\mathcal{E}_{\text{stheft}}$ be the event that $\mathcal{A}$ wins in the following way.

$\text{pk}_{\mathcal{A}}$ successfully steals a coin from nothing, i.e., no such escrow coin corresponds to the redeemed coin. Upon receiving the valid challenge $(\text{pk}_{\mathcal{A}}, t_{\mathcal{A}}, c_{\mathcal{A}}, \pi_{\text{mem}})$ from $\mathcal{A}$, C runs the extractor $\text{Ext}^{\mathcal{P}}(t_{\mathcal{A}}, \text{pk}_{\mathcal{A}}, c_{\mathcal{A}})$ of Mem and derives the witnesses $(l, v, r_l, r_{\text{red}}, r_t)$ satisfying equation (10,11). This means that the $\text{pk}_{\mathcal{A}}$ steals a coin from the l_{th} escrowed one, which contradicts our case. Thus,

$$\Pr[\mathcal{E}_{\text{stheft}} \text{ occurs}] = \text{Adv}^{\text{soundness}}_{\mathcal{A},\text{Mem}}(\lambda).$$

Case 3: unbalanced amount. Let $\mathcal{E}_{\text{unbal}}$ be the event that $\mathcal{A}$ wins in the following way: the challenge $(\text{pk}_{\mathcal{A}}, t_{\mathcal{A}}, c_{\mathcal{A}}, \pi_{\text{mem}})$ provided by the adversary $\mathcal{A}$, but $L_e[t_{\mathcal{A}}].v_e \neq L_r[t_{\mathcal{A}}].v_r$. That is, $\text{pk}_{\mathcal{A}}$ redeems unequal (e.g., more) value of coin than being escrowed. Upon receiving the challenge $(\text{pk}_{\mathcal{A}}, t_{\mathcal{A}}, c_{\mathcal{A}}, \pi_{\text{mem}})$ from $\mathcal{A}$, C runs the extractor $\text{Ext}^{\mathcal{P}}(S, \text{pk}_{\mathcal{A}}, t_{\mathcal{A}}, c_{\mathcal{A}})$ of Mem. Then C derives the witnesses $(l, v, r_l, r_{\text{red}}, r_t)$ satisfying equation (10), which implies that c_l and c_r encrypt the same message v, i.e. $L_e[t_{\mathcal{A}}].v_e = L_r[t_{\mathcal{A}}].v_r = v$. This contradicts our case. Thus,

$$\Pr[\mathcal{E}_{\text{unbal}} \text{ occurs}] = \text{Adv}^{\text{soundness}}_{\mathcal{A},\text{Mem}}(\lambda).$$

Case 4: double-redemption. Define $\mathcal{E}_{\text{2x-redeem}}$ as the event that $\mathcal{A}$ wins the Balance experiment in the following way. For the valid challenge $(\text{pk}_{\mathcal{A}}, t_{\mathcal{A}}, c_{\mathcal{A}}, \pi_{\text{mem}})$ from $\mathcal{A}$, there exists another valid redemption request (pk'_r, t', c'_r) recorded in L_r. Then C runs extractor $\text{Ext}^{\mathcal{P}}(S, t_{\mathcal{A}}, \text{pk}_{\mathcal{A}}, c_{\mathcal{A}})$ and $\text{Ext}^{\mathcal{P}}(S, t', \text{pk}', c'_r)$ of Mem and derives witnesses $(l, v, r_l, r_{\text{red}}, r_t)$, $(l', v', r'_l, r'_{\text{red}}, r'_t)$ satisfying equation (11) and $\text{token}_l = \text{token}'_l$, i.e. the l_{th} escrowed coin is redeemed twice. Restate this fact as a formula, we derive that

$$\text{Com}(t_{\mathcal{A}}, h(\text{pk}_{\mathcal{A}}); r_t) = \text{Com}(t', h(\text{pk}'_r); r'_t) \tag{14}$$

Recall that in UMCT, the tumbler checks the uniqueness of tag t and then adds t to its public state to prevent double-redemption. Therefore, since the above two redemption requests are both confirmed by the tumbler, it must hold $t_{\mathcal{A}} \neq t'$. We can design an algorithm $\mathcal{B}_2$ to break the binding property of the commitment C. Given $\text{pp} \leftarrow \text{C.Setup}(1^\lambda)$, $\mathcal{B}_2$ simulates experiment $\text{Balance}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda)$. Once $\mathcal{E}_{\text{2x-redeem}}$ occurs, i.e., $\text{token}_l = \text{token}'_l$, $\mathcal{B}_2$ forwards $(t_{\mathcal{A}}, h(\text{pk}_{\mathcal{A}}), r_t)$ and $(t', h(\text{pk}'_r), r'_l)$ to $\mathcal{B}_2$'s challenger. $\mathcal{B}_2$ wins if and only if $\mathcal{E}_{\text{2x-redeem}}$ occurs. Thus,

$$\Pr[\mathcal{E}_{\text{2x-redeem}} \text{ occurs}] \leq \text{Adv}^{\text{binding}}_{\mathcal{A},\text{C}}(\lambda) + 2\text{Adv}^{\text{soundness}}_{\mathcal{A},\text{Mem}}(\lambda).$$

Combining above four cases, we can derive

$$\text{Adv}^{\text{balance}}_{\mathcal{A},\text{UMCT}}(\lambda) \leq 2\text{Adv}^{\text{binding}}_{\mathcal{A},\text{C}}(\lambda) + 5\text{Adv}^{\text{soundness}}_{\mathcal{A},\text{Mem}}(\lambda). \tag{15}$$

The proof of Lemma 4.2 is finished.

C.2 Proof of Lemma 4.3

We prove that $\text{Adv}^{\text{unlink}}_{\mathcal{A},\text{UMCT}}(\lambda)$ is negligible via a sequence of games.

Game 0 is the real experiment $\text{Unlinkability}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda)$ described in Definition 3.3.

Game 1 is the same as Game 0 except that C computes $\pi^{(b)}_{\text{mem}}$ with the simulator $\text{Sim}(\tau, S, t^{(b)}, \text{pk}^{(b)}_r, c^{(b)}_r)$ in the challenge phase. We claim that Game 0 is computationally indistinguishable from Game 1 via defining an algorithm $\mathcal{D}_1$ to break the zero-knowledge property of Mem. The algorithm $\mathcal{D}_1$ proceeds as follows. Given σ generated

by $(\sigma, \tau) \leftarrow \mathsf{Mem.Setup}(1^\lambda)$, $\mathcal{D}_1$ simulates either Game 0 or Game 1. Instead of computing $\pi_{\text{mem}}^{(b)}$ in the challenge phase, $\mathcal{D}_1$ sends $(S, t^{(b)}, \mathsf{pk}_r^{(b)}, c_r^{(b)}; l_b, v_b, r_l^{(b)}, r_{\text{red}}^{(b)}, r_t^{(b)})$ to his challenger. Then $\mathcal{D}_1$ receives

$$\pi_{\text{mem}} \leftarrow \mathsf{Mem.Prove}(S, t^{(b)}, \mathsf{pk}_r^{(b)}, c_r^{(b)}; l_b, v_b, r_l^{(b)}, r_{\text{red}}^{(b)}, r_t^{(b)}),$$

when $\beta = 0$, otherwise, $\pi_{\text{mem}} \leftarrow \mathsf{Sim}(\tau, S, t^{(b)}, \mathsf{pk}_r^{(b)}, c_r^{(b)})$. We can observe that if $\beta = 0$, $\mathcal{D}_1$ simulates Game 0, otherwise, $\beta = 1$, $\mathcal{D}_1$ simulates Game 1. Upon the adversary distinguishing Game 0 and Game 1 outputs β', $\mathcal{D}_1$ forwards β' to his own challenger. Thus,

$$\left|\mathsf{Adv}^{\text{game1}}_{\mathcal{A},\text{UMCT}}(\lambda) - \mathsf{Adv}^{\text{game0}}_{\mathcal{A},\text{UMCT}}(\lambda)\right| \le \mathsf{Adv}^{\text{zk}}_{\mathcal{A},\mathsf{Mem}}(\lambda).$$

Game 2 is the same as Game 1 except that C chooses random $\mathsf{pk}_r^{(b)}$ and $t^{(b)}$ from their spaces respectively, generates $c_r^{(b)} = \mathsf{Enc}_{\mathsf{pk}_r^{(b)}}(v_b; r_{\text{red}}^{(b)})$ and outputs $(\mathsf{pk}_r^{(b)}, t^{(b)}, c_r^{(b)}, \pi_{\text{mem}}^{(b)})$ to $\mathcal{A}$. We claim that Game 1 is computationally indistinguishable from Game 2 via defining an algorithm $\mathcal{D}_2$ to break the hiding property of C. The algorithm $\mathcal{D}_2$ proceeds as follows. Given $\mathsf{pp}_{\text{com}} \leftarrow \mathsf{C.Setup}(1^\lambda)$, $\mathcal{D}_2$ simulates either Game 1 or Game 2. $\mathcal{D}_2$ can generate Q escrow transactions on behalf of honest users before the challenge phase. Here Q ($2 \le Q \le N$) is the number of escrow transactions that neither be induced by nor target the adversary. In special, when $\mathcal{D}_2$ creates an honest escrow transaction, instead of computing $\mathsf{token} = \mathsf{Com}(t, h(pk_r), r_t)$, $\mathcal{D}_2$ chooses random (t', pk_r') and forwards $(t, h(pk_r))$ and $(t', h(pk_r'))$ to his own challenger. Then $\mathcal{D}_2$ receives $\mathsf{token}_\beta = \mathsf{Com}(t, h(pk_r), r_t)$ if $\beta = 0$, otherwise $\mathsf{token}_\beta = \mathsf{Com}(t', h(pk_r'), r_t')$. If $\beta = 0$, $\mathcal{D}_2$ simulates Game 1. If $\beta = 1$, $\mathcal{D}_2$ simulates Game 2 with $1/Q$ probability because token_β is chosen by the adversary $\mathcal{A}$ and the challenger $\mathcal{D}_2$ in Game 2 with $1/Q$ probability. Upon the adversary distinguishing Game 1 and Game 2 outputs β', $\mathcal{D}_2$ forwards β' to his own challenger. Thus,

$$\left|\mathsf{Adv}^{\text{game2}}_{\mathcal{A},\text{UMCT}}(\lambda) - \mathsf{Adv}^{\text{game1}}_{\mathcal{A},\text{UMCT}}(\lambda)\right| \le Q \cdot \mathsf{Adv}^{\text{hiding}}_{\mathcal{A},\text{C}}(\lambda),$$

Game 3 is the same as Game 2 except that C generates $c_r^{(b)} = \mathsf{Enc}_{\mathsf{pk}_r^{(b)}}(0; r_{\text{red}})$. We claim that Game 2 is computationally indistinguishable from Game 3 via defining an algorithm $\mathcal{D}_3$ to break the CPA security of PKE used in the underlying blockchain.

The algorithm $\mathcal{D}_3$ proceeds as follows. Given $\mathsf{pp}_{\text{ctx}} \leftarrow \mathsf{CT.Setup}(1^\lambda)$ and a public key pk_r from his challenger, $\mathcal{D}_3$ simulates either Game 2 or Game 3. Instead of computing $c_r^{(b)}$ under pk_r when executing RequestRedeem in the challenge phase, $\mathcal{D}_3$ sends $(v_b, 0)$ to his challenger. Here v_b can be derived from $\mathsf{aux}^{(b)}$. Then $\mathcal{D}_3$ receives $c_r^{(b)} = \mathsf{Enc}_{\mathsf{pk}_r}(v_b, r_{\text{red}})$ when $\beta = 0$, otherwise, $c_r^{(b)} = \mathsf{Enc}_{\mathsf{pk}_r}(0, r_{\text{red}})$. We can observe that if $\beta = 0$, $\mathcal{D}_3$ simulates Game 2, otherwise, $\beta = 1$, $\mathcal{D}_3$ simulates Game 3. Upon the adversary distinguishing Game 2 and Game 3 outputs β', $\mathcal{D}_3$ forwards β' to his own challenger. Thus,

$$\left|\mathsf{Adv}^{\text{game3}}_{\mathcal{A},\text{UMCT}}(\lambda) - \mathsf{Adv}^{\text{game2}}_{\mathcal{A},\text{UMCT}}(\lambda)\right| \le \mathsf{Adv}^{\text{cpa}}_{\mathcal{A},\text{PKE}}(\lambda).$$

Note that $(\mathsf{pk}_r, c_r, \pi_{\text{mem}}, t)$ in Game 3 is completely independent of $\mathcal{A}$'s challenge. Therefore, the output of $\mathcal{A}$ is also independent of b. Hence the advantage that $\mathcal{A}$ wins in Game 3 is 0.

In summary, we can derive

$$\mathsf{Adv}^{\text{unlink}}_{\mathcal{A},\text{UMCT}}(\lambda) = \left|\Pr[\text{Unlinkability}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda) = 1] - \frac{1}{2}\right| \le \mathsf{Adv}^{\text{zk}}_{\mathcal{A},\mathsf{Mem}}(\lambda) + Q \cdot \mathsf{Adv}^{\text{hiding}}_{\mathcal{A},\text{C}}(\lambda) + \mathsf{Adv}^{\text{cpa}}_{\mathcal{A},\text{PKE}}(\lambda).$$

The proof of Lemma 4.3 is finished.

C.3 Proof of Lemma 4.4

We prove that $\mathsf{Adv}^{\text{confidential}}_{\mathcal{A},\text{UMCT}}(\lambda)$ is negligible via a sequence of games.

Game 0 is the real experiment $\text{Confidentiality}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda)$ described in Definition 3.4, where C chooses $b = 0$.

Game 1 is the same as Game 0 except that C generates $\pi_{\text{mem}}^{(0)}$ with $\mathsf{Sim}(S, t, \mathsf{pk}_r, c_r^{(0)})$ in the challenge phase. Then we can derive that Game 0 is computationally indistinguishable from Game 1 via defining a distinguisher $\mathcal{D}_1$ to break the zero-knowledge property of Mem.

The algorithm $\mathcal{D}_1$ proceeds as follows. Given σ generated from $(\sigma, \tau) \leftarrow \mathsf{Mem.Setup}(1^\lambda)$, $\mathcal{D}_1$ simulates either Game 0 or Game 1. Instead of computing $\pi_{\text{mem}}^{(0)}$ in the challenge phase, $\mathcal{D}_1$ sends $(S, t, \mathsf{pk}_r, c_r^{(0)}; l, v_0, r_l, r_{\text{red}}, r_t)$ to his challenger. Then $\mathcal{D}_1$ receives

$$\pi_{\text{mem}}^{(0)} \leftarrow \mathsf{Mem.Prove}(S, t, \mathsf{pk}_r, c_r^{(0)}; l, v_0, r_l, r_{\text{red}}, r_t),$$

when $\beta = 0$, otherwise, $\pi_{\text{mem}}^{(0)} \leftarrow \mathsf{Sim}(S, t, \mathsf{pk}_r, c_r^{(0)})$. We can observe that if $\beta = 0$, $\mathcal{D}_1$ simulates Game 0, otherwise, $\beta = 1$, $\mathcal{D}_1$ simulates Game 1. Upon the adversary distinguishing Game 0 and Game 1 outputs β', $\mathcal{D}_1$ forwards β' to his own challenger. Thus,

$$\left|\mathsf{Adv}^{\text{game1}}_{\mathcal{A},\text{UMCT}}(\lambda) - \mathsf{Adv}^{\text{game0}}_{\mathcal{A},\text{UMCT}}(\lambda)\right| \le \mathsf{Adv}^{\text{zk}}_{\mathcal{A},\mathsf{Mem}}(\lambda).$$

Game 2 is the same as Game 1 except that, in the challenge phase, C escrows coin with value v_1 and outputs $(\mathsf{ctx}_{\text{esc}}^{(1)}, \mathsf{token}, \mathsf{pk}_r, c_r^{(0)}, \pi_{\text{mem}}^{(0)}, t)$. Then we can derive that Game 1 is computationally indistinguishable from Game 2 via defining a distinguisher $\mathcal{D}_2$ to break the confidentiality property of the underlying confidential system.

The algorithm $\mathcal{D}_2$ proceeds as follows. Given $\mathsf{pp}_{\text{ctx}} \leftarrow \mathsf{CT.Setup}(1^\lambda)$, $\mathcal{D}_2$ simulates either Game 1 or Game 2. Instead of executing CT.Create to generate $\mathsf{ctx}_{\text{esc}}^{(\beta)}$ in the challenge phase, $\mathcal{D}_2$ sends $(\mathsf{sk}_e, \mathsf{pk}_t, v_0, v_1)$ to his challenger. Then $\mathcal{D}_2$ receives $\mathsf{ctx}_{\text{esc}}^{(\beta)} \leftarrow \mathsf{CT.Create}(\mathsf{sk}_e, \mathsf{pk}_t, v_\beta)$. We can observe that if $\beta = 0$, $\mathcal{D}_2$ simulates Game 1, otherwise, $\beta = 1$, $\mathcal{D}_2$ simulates Game 2. Upon the adversary distinguishing Game 1 and Game 2 outputs β', $\mathcal{D}_2$ forwards β' to his own challenger. Thus,

$$\left|\mathsf{Adv}^{\text{game2}}_{\mathcal{A},\text{UMCT}}(\lambda) - \mathsf{Adv}^{\text{game1}}_{\mathcal{A},\text{UMCT}}(\lambda)\right| \le \mathsf{Adv}^{\text{confidential}}_{\mathcal{A},\text{CT}}(\lambda).$$

Game 3 is the same as Game 2 except that C replaces $c_r^{(0)}$ with $c_r^{(1)}$ in the challenge phase and outputs $(\mathsf{ctx}_{\text{esc}}^{(1)}, \mathsf{token}, \mathsf{pk}_r, c_r^{(1)}, \pi_{\text{mem}}^{(1)}, t)$. Then we can derive that Game 2 is computationally indistinguishable from Game 3 via defining a distinguisher $\mathcal{D}_3$ to break the CPA security of PKE.

The algorithm $\mathcal{D}_3$ proceeds as follows. Given $\mathsf{pp}_{\text{ctx}} \leftarrow \mathsf{CT.Setup}(1^\lambda)$, $\mathcal{D}_3$ simulates either Game 2 or Game 3. Instead of computing $c_r^{(\beta)}$ when executing RequestRedeem in the challenge phase, $\mathcal{D}_3$ sends $(\mathsf{pk}_r, v_0, v_1)$ to his challenger. Then $\mathcal{D}_3$ receives $c_r^{(\beta)} = \mathsf{Enc}_{\mathsf{pk}_r}(v_\beta, r_{\text{red}})$.

We can observe that if $\beta = 0$, $\mathcal{D}_3$ simulates Game 2, otherwise, $\beta = 1$, $\mathcal{D}_3$ simulates Game 3. Upon the adversary distinguishing Game 2 and Game 3 outputs β', $\mathcal{D}_3$ forwards β' to his own challenger. Thus,

$$\left|\mathsf{Adv}^{\text{game3}}_{\mathcal{A},\text{UMCT}}(\lambda) - \mathsf{Adv}^{\text{game2}}_{\mathcal{A},\text{UMCT}}(\lambda)\right| \leq \mathsf{Adv}^{\text{cpa}}_{\mathcal{A},\text{PKE}}(\lambda).$$

Game 4 is the real experiment $\text{Confidentiality}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda)$, where C chooses $b = 1$. Then we can derive that Game 3 is computationally indistinguishable from Game 4 via defining a distinguisher $\mathcal{D}_4$ to break the zero-knowledge property of Mem. The description of $\mathcal{D}_4$ is similar to $\mathcal{D}_1$, so we directly indicate that

$$\left|\mathsf{Adv}^{\text{game4}}_{\mathcal{A},\text{UMCT}}(\lambda) - \mathsf{Adv}^{\text{game3}}_{\mathcal{A},\text{UMCT}}(\lambda)\right| \leq \mathsf{Adv}^{\text{zk}}_{\mathcal{A},\mathsf{Mem}}(\lambda).$$

In a nutshell, we can derive that

$$\begin{aligned}\mathsf{Adv}^{\text{confidential}}_{\mathcal{A},\text{UMCT}}(\lambda) &= \left|\Pr[\text{Confidentiality}^{\mathcal{A}}_{\text{UMCT}}(1^\lambda) = 1] - \frac{1}{2}\right| \\ &\leq 2\mathsf{Adv}^{\text{zk}}_{\mathcal{A},\mathsf{Mem}}(\lambda) + \mathsf{Adv}^{\text{confidential}}_{\mathcal{A},\text{CT}}(\lambda) + \mathsf{Adv}^{\text{cpa}}_{\mathcal{A},\text{PKE}}(\lambda).\end{aligned}$$

The proof of Lemma 4.4 is finished.

D PROOF OF LEMMA 5.3

Below we give the formal proof of Lemma 5.3.

Proof. First, the protocol Π^{dlr} is trivially complete (or see [5] for details.).

Secondly, for the proving of the SHVZK, we describe a simulator to "simulate" the transcripts between the prover and the verifier. Specifically, The simulator chooses $\mathbf{z} \leftarrow\$ \mathbb{Z}_p^N$ as a witnesses. In each recursive step, it samples $r_0, r_1 \leftarrow\$ \mathbb{Z}_p$ and computes

$$L = \mathbf{g}_{[N':]}^{\mathbf{z}_{[:N']}} \cdot g_{\text{red}}^{r_0}, \quad R = \mathbf{g}_{[:N']}^{\mathbf{z}_{[N':]}} \cdot g_{\text{red}}^{r_1}.$$

Suppose that at the last step the input commitment is c' and the challenge is x, simulator randomly chooses $\theta_0, \theta_1 \leftarrow\$ \mathbb{Z}_p$ and computes $d = c'^x g_0^{\theta_0} g_{\text{red}}^{\theta_1}$. In this way, the transcript $(c, L_0, R_0, x_0, L_1, \cdots, d, x, \theta_0, \theta_1)$ generated by the simulator is valid and has the identical probability distributions with the real proof.

Finally, for the proof of the soundness, we use an inductive argument to construct an extractor $\mathcal{X}$. In each step, the extractor can either extract a witness or a discrete log relation.

- If $N = 1$, rewinding prover $\mathcal{P}$ to get 2 transcripts $(d, x, \theta_0, \theta_1)$, $(d, x', \theta_0', \theta_1')$ such that,

$$d = g_0^{z_0 x + \theta_0} g_{\text{red}}^{z_{\text{red}} x + \theta_1} = g_0^{z_0 x' + \theta_0'} g_{\text{red}}^{z_{\text{red}} x' + \theta_1'}. \tag{16}$$

  Therefore, the extractor can either compute the discrete logarithm relation between pk_0 and pk_{red} at least or extract $z_0 = \frac{\theta_0 - \theta_0'}{x' - x}, z_{\text{red}} = \frac{\theta_1 - \theta_1'}{x' - x}$.
- If $N > 1$, suppose that in the $(k+1)_{\text{th}}$ step, we can extract $\mathbf{z}'$ and z'_{red} such that $\mathbf{g}'^{\mathbf{z}'} g_{\text{red}}^{z'_{\text{red}}} = c'$. Now we can prove the extractor can either extract the witnesses of the k_{th} recursive step or compute a non-trivial discrete logarithm relation between the generators. Concretely, in the k_{th} recursive step, rewind the prover $\mathcal{P}$ four times and obtain four challenges x_1, x_2, x_3, x_4, $x_i \neq \pm x_j$, $1 \leq i < j \leq 4$, and four pairs of witnesses $\mathbf{z}'^{(i)}, z'^{(i)}_{\text{red}}$ such that

$$c \cdot L^{x_i^2} \cdot R^{-x_i^2} = (\mathbf{g}_{[:N']}^{x_i^{-1}} \circ \mathbf{g}_{[N':]}^{x_i})^{\mathbf{z}'^{(i)}} g_{\text{red}}^{z'^{(i)}_{\text{red}}}, i = 1, 2, 3, 4. \tag{17}$$

We choose v_1, v_2, v_3 such that $\sum_{i=1}^3 x_i^2 v_i = 1, \sum_{i=1}^3 v_i = 0, \sum_{i=1}^3 x_i^{-2} v_i = 0$. Then we can derive that

$$\prod_{i=1}^3 (c \cdot L^{x_i^2} \cdot R^{x_i^{-2}})^{v_i} = \prod_{i=1}^3 (\mathbf{g}_{[:N']}^{x_i^{-1}} \circ \mathbf{g}_{[N':]}^{x_i})^{\mathbf{z}'^{(i)} v_i} g_{\text{red}}^{z'^{(i)}_{\text{red}} v_i}.$$

It indicates that $L = \mathbf{g}^{\mathbf{z}_L} g_{\text{red}}^{z_{\text{red}_L}}$ where $\mathbf{z}_{L[:N']} = \sum_{i=1}^3 x_i^{-1} \mathbf{z}'^{(i)} v_i$, $\mathbf{z}_{L[N':]} = \sum_{i=1}^3 x_i \mathbf{z}'^{(i)} v_i$ and $z_{\text{red}_L} = \sum_{i=1}^3 z'^{(i)}_{\text{red}} v_i$.
Similarly, repeating this process with different combinations, we can extract that

$$R = \mathbf{g}^{\mathbf{z}_R} g_{\text{red}}^{z_{\text{red}_R}}, \; c = \mathbf{g}^{\mathbf{z}_c} g_{\text{red}}^{z_{\text{red}_c}}.$$

Rewrite equation (17), for each $x \in \{x_1, x_2, x_3, x_4\}$, as

$$\mathbf{g}^{\mathbf{z}_L x^2 + \mathbf{z}_c + \mathbf{z}_R x^{-2}} g_{\text{red}}^{z_{\text{red}_L} x^2 + z_{\text{red}_c} + z_{\text{red}_R} x^{-2}} = \left(\mathbf{g}_{[:N']}^{\mathbf{z}' x^{-1}} \circ \mathbf{g}_{[N':]}^{\mathbf{z}' x}\right) g_{\text{red}}^{z_{\text{red}'}}.$$

This implies that

$$\begin{aligned}\mathbf{z}' x^{-1} &= \mathbf{z}_{L[:N']} x^2 + \mathbf{z}_{c[:N']} + \mathbf{z}_{R[:N']} x^{-2}, \\ \mathbf{z}' x &= \mathbf{z}_{L[N':]} x^2 + \mathbf{z}_{c[N':]} + \mathbf{z}_{R[N':]} x^{-2}, \\ z'_{\text{red}} &= z_{\text{red}_L} x^2 + z_{\text{red}_c} + z_{\text{red}_R} x^{-2}.\end{aligned}$$

Otherwise, the extractor can obtain a non-trivial discrete log relation between the generators $(\mathbf{g}, g_{\text{red}})$. We can deduce that for $x \in \{x_1, x_2, x_3, x_4\}$,

$$x^3 \mathbf{z}_{L[:N']} + x(\mathbf{z}_{c[:N']} - \mathbf{z}_{L[:N']}) + x^{-1}(\mathbf{z}_{R[:N']} - \mathbf{z}_{c[N':]}) - x^{-3} \mathbf{z}_{L[:N']} = 0.$$

The only way the above equation holds for four challenges is

$$\mathbf{z}_{L[:N']} = \mathbf{z}_{R[N':]} = 0, \mathbf{z}_{c[:N']} = \mathbf{z}_{L[:N']}, \mathbf{z}_{R[:N']} = \mathbf{z}_{c[N':]}.$$

Then, $\mathbf{z}' = x\mathbf{z}_{c[:N']} + x^{-1}\mathbf{z}_{c[N':]}$ holds for all challenges. Therefore, we either extract $\mathbf{z}_c$ and z_{red_c} or break the discrete log relation assumption.

□

E PROOF OF THEOREM 5.4

Proof. First we prove the perfect completeness. By the perfect completeness of Π^{bin}, we claim that Π^{bin}.Verify always accepts. Besides, the fact that Π^{dlr}.Verify always accepts follows from the homomorphic property of encryption since

$$\begin{aligned}&\prod_{u=0}^{N-1} (c_e^{(u)}/c_r)^{\prod_{j=0}^{m-1} f_{j,u_j}} \prod_{k=0}^{m-1} G_k^{-x^k} \\ =&(c_e^{(l)}/c_r)^{x^m} \prod_{u=0}^{N-1} \mathsf{Enc}_{\mathsf{pk}_u}(0; \sum_{k=0}^{m-1} \rho_k^{(u)} x^k) \mathsf{Enc}_{\mathsf{pk}_r}(0; \sum_{k=0}^{m-1} \eta_k x^k) \\ =&\mathsf{Enc}_{\mathsf{pk}_l}(0; z_l) \prod_{u \neq l}^{N-1} \mathsf{Enc}_{\mathsf{pk}_u}(0; z_u) \mathsf{Enc}_{\mathsf{pk}_r}(0; z_{\text{red}}).\end{aligned}$$

Similarly, the correctness of the last equation can be derived.

Then, we describe a SHVZK simulator. It chooses $B, \{H_i\}_{i=1}^{m-1} \leftarrow\$ \mathbb{G}, \{G_i\}_{i=1}^{m-1} \leftarrow\$ \mathbb{G}^2$. Running Sim_1 in Π^{bin}, it derives A, C, D, z_A, z_C, $\{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}$ and computes $f_{j,0} = x - \sum_{i=1}^{n-1} f_{j,i}, j \in [m]$. Choosing $c \leftarrow \mathbb{G}^2$ and running Sim_2 in Π^{dlr}, it derives $\mathbf{L}, \mathbf{R}, d, \theta_0, \theta_1$. Thereafter, it picks $z_t \leftarrow\$ \mathbb{Z}_p$ and computes G_0 and H_0. The transcript generated by the simulator is valid and has the identical probability distributions with the real proof.

In terms of the knowledge soundness, we construct an extractor $\mathcal{X}$ for Mem with logarithmic proof size exploiting extractors in subprotocol Π^{bin} and Π^{dlr}.

The extractor $\mathcal{X}$ runs the prover Mem.Prove to get A, B, C, D, $\{G_k, H_k\}_{k\in[m]}$. Then, by rewinding the prover $m+1$ times and giving challenge $x \in \{x^{(1)}, \cdots, x^{(m+1)}\}$, the extractor can obtain $m+1$ pairs $\{f_{j,i}^{(e)}\}_{j,i=0}^{m-1,n-1}, \mathbf{z}^{(e)}, z_{\text{red}}^{(e)}, z_t^{(e)}$ for $e = 1, \cdots m+1$. First, we use knowledge soundness property of the protocol Π^{bin} to extract openings $\{\delta_{l_j,i}\}_{j,i=0}^{m-1,n-1}, \{a_{j,i}\}_{j,i=0}^{m-1,n-1}$ for B and A with $\delta_{l_j,i} \in \{0,1\}$ and $\sum_{i=0}^{n-1} \delta_{l_j,i} = 1$. The openings define $l = \sum_{j=0}^{m-1} l_j n^j$ where l_j is the index of the only 1 in the sequence $(\delta_{l_j,0}, \cdots, \delta_{l_j,n-1})$. And with overwhelming probability, the extracted witnesses satisfy $f_{j,i}^{(e)} = \delta_{l_j,i} x^{(e)} + a_{j,i}$ for $e = 1, \cdots, m+1$. Then, we use the knowledge soundness property of the protocol Π^{drl} to extract openings $\mathbf{z}^{(e)}, z_{\text{red}}^{(e)}$ such that

$$\begin{aligned}&\prod_{u=0}^{N-1} (c_e^{(u)}/c_B)^{\prod_{j=0}^{m-1} f_{j,u_j}^{(e)}} \prod_{k=0}^{m-1} G_k^{-x^k} \\ &= \prod_{u=0}^{N-1} \mathsf{Enc}_{\mathsf{pk}_u}(0; z_u^{(e)}) \mathsf{Enc}_{\mathsf{pk}_r}(0; z_{\text{red}}^{(e)})),\end{aligned} \tag{18}$$

$$\prod_{u=0}^{N-1} (\mathsf{token}_u/\mathsf{token}_{\text{tmp}})^{\prod_{j=0}^{m-1} f_{j,u_j}^{(e)}} \prod_{k=0}^{m-1} H_k^{-x^k} = \mathsf{C}^{\circ}.\mathsf{Com}(0,0; z_t^{(e)}). \tag{19}$$

Based on the observation that $p_u(x) = \prod_{j=0}^{m-1} f_{j,u_j}$ and only $p_l(x)$ is with degree m, we can rewrite equation (18,19) as

$$(c_e^{(l)}/c_r)^{x^m} \prod_{k=0}^{m-1} \hat{G}_k^{-x^k} = \prod_{u=0}^{N-1} \mathsf{Enc}_{\mathsf{pk}_u}(0; z_u) \mathsf{Enc}_{\mathsf{pk}_r}(0; z_{\text{red}})), \tag{20}$$

$$(\mathsf{token}_l/\mathsf{token}_{\text{tmp}})^{x^m} \prod_{k=0}^{m-1} \hat{H}_k^{-x^k} = \mathsf{C}^{\circ}.\mathsf{Com}(0,0; z_t), \tag{21}$$

where

$$\hat{G}_k = G_k/(\prod_{u=0}^{N-1} (c_e^u/c_r)^{p_{u,k}}),\ \hat{H}_k = H_k/(\prod_{u=0}^{N-1} (\mathsf{token}_u/\mathsf{token}_{\text{tmp}})^{p_{u,k}}).$$

Since the Vandermonde matrix with the e_{th} row given by

$$(1, x^{(e)}, \cdots, (x^{(e)})^m)$$

is invertible and the equation (20, 21) holds for distinct $x^{(1)}, \cdots, x^{(m+1)}$, we can thus obtain a linear combination $\phi_1, \cdots, \phi_{m+1}$ of the rows producing the vector $(0, \cdots, 0, 1)$ such that

$$\begin{aligned}c_e^{(l)}/c_r &= \prod_{e=1}^{m+1} \left((c_e^{(l)}/c_r)^{(x^{(e)})^m} \prod_{k=0}^{m-1} \hat{G}_k^{(x^{(e)})^k} \right)^{\phi_e} \\ &= \mathsf{Enc}_{\mathsf{pk}_l}(0; \sum_{e=1}^{m+1} \phi_e z_l^{(e)}) \cdot \mathsf{Enc}_{\mathsf{pk}_r}(0; \sum_{e=1}^{m+1} \phi_e z_{\text{red}}^{(e)}),\end{aligned}$$

$$\begin{aligned}\mathsf{token}_l/\mathsf{token}_{\text{tmp}} &= \prod_{e=1}^{m+1} \left((\mathsf{token}_l/\mathsf{token}_{\text{tmp}})^{(x^{(e)})^m} \prod_{k=0}^{m-1} \hat{H}_k^{(x^{(e)})^k} \right)^{\phi_e} \\ &= \mathsf{C}^{\circ}.\mathsf{Com}(0,0; \sum_{e=1}^{m+1} \phi_e z_t^{(e)}),\end{aligned} \tag{22}$$

which provides randomness openings

$$r_l = \sum_{e=1}^{m+1} \phi_e z_l^{(e)},\ r_{\text{red}} = \sum_{e=1}^{m+1} \phi_e z_R^{(e)},\ r_t = \sum_{e=1}^{m+1} \phi_e z_t^{(e)}.$$

□

F DESCRIPTION OF BINARY PROOF

- Prover
  - Picks $r_A, r_C, r_D, \{a_{j,i}\}_{j=0,i=1}^{m-1,n-1} \leftarrow\$ \mathbb{Z}_p$
  - Computes
    - $\forall j \in [m], a_{j,0} := -\Sigma_{i=1}^{n-1} a_{j,i}$
    - $A := \mathsf{C}^{\triangle}.\mathsf{Com}(a_{0,0}, a_{0,1}, \cdots, a_{m-1,n-1}; r_A)$
    - $C := \mathsf{C}^{\triangle}.\mathsf{Com}\left(\{a_{j,i}(1-2b_{j,i})\}_{j,i=0}^{m-1,n-1}; r_C\right)$
    - $D := \mathsf{C}^{\triangle}.\mathsf{Com}(-a_{0,0}^2, -a_{0,1}^2, \cdots, -a_{m-1,n-1}^2; r_D)$
  - Sends: A, C, D
- Verifier
  - Selects $x \leftarrow\$ \mathbb{Z}_p$ and sends x.
- Prover
  - For $\forall j \in [m], i \in [n]$, computes $f_{j,i} = b_{j,i}x + a_{j,i}$
  - Sends
    - $\{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}$
    - $z_A = r_B x + r_A$
    - $z_C = r_C x + r_D$
- Verifier checks
  - $A, B, C, D \in \mathbb{G}, \{f_{j,i}\}_{j=0,i=1}^{m-1,n-1}, z_A, z_C \in \mathbb{Z}_p$
  - $\forall j \in [m], f_{j,0} = x - \Sigma_{i=1}^{n-1} f_{j,i}$
  - $B^x A = \mathsf{C}^{\triangle}.\mathsf{Com}(\{f_{j,i}\}_{j,i=0}^{m-1,n-1}; z_A)$
  - $C^x D = \mathsf{C}^{\triangle}.\mathsf{Com}(\{f_{j,i}(x - f_{j,i})\}_{j,i=0}^{m-1,n-1}; z_C)$

Figure 5: The zero knowledge proof protocol Π^{bin} for R_{bin}.