MixCT: Mixing Confidential Transactions from Homomorphic Commitment

Jiajun Du[1], Zhonghui Ge[1], Yu Long[1], Zhen Liu[1],
Shifeng Sun[1], Xian Xu[2], and Dawu Gu[1]

[1] Shanghai Jiao Tong University, Shanghai, China
{cqdujiajun,zhonghui.ge,longyu,liuzhen,shifeng.sun,dwgu}@sjtu.edu.cn
[2] East China University of Science and Technology, Shanghai, China
xuxian@ecust.edu.cn

Abstract. Mixing protocols serve as a promising solution to the unlinkability in blockchains. They work by hiding one transaction among a set of transactions and enjoy the advantage of high compatibility with the underlying system. However, due to the inherently public nature of the blockchains built on the account-based model, the unlinkability is highly restricted to non-confidential transactions. In the account-based model, blockchains supporting confidential payments need to trade their compatibility for unlinkability.
In this paper, we propose MixCT, a generic protocol that provides the mixing service for confidential payment systems built from homomorphic commitment in the account-based model. We formally define the security goals including safety and availability, and prove that our generic construction satisfies them. Furthermore, we provide an efficient instantiation of MixCT by the Pedersen commitment and the one-out-of-many proof. The evaluation results show that MixCT introduces a small cost for its users while being highly compatible with the underlying confidential blockchain.

Keywords: blockchain · confidential transaction · mixing service.

1 Introduction

The decentralized payment system, also known as the blockchain, provides a popular medium of exchange and has attracted a substantial number of applications. Each transaction in a blockchain is confirmed by a public ledger, with the corresponding relationship between the sender and receiver precisely kept. This, however, makes it possible to track the transaction flow and gives rise to the privacy issue. To this point, several privacy-enhanced blockchain systems have been proposed to protect ledger privacy, mainly in two aspects: the unlinkability between transactions and the confidentiality of each transaction's amount.

In the early blockchain-based cryptocurrencies [16], people use a pseudonym to break the link between a user's on-chain address and the real-world identity. This method, however, has been proved to be useless after being thoroughly studied [4,15]. Later, many efforts have been made to provide stronger unlinkability to blockchains, by mixing multiple transactions and obscuring the relations between the two sets of senders and receivers [1,18,19]. Unlike stand-alone cryptocurrencies such as [5, 10, 20, 21], this approach is highly compatible with the underlying blockchain and needs neither to start up a new blockchain nor to hard-fork an existing one. For this reason, mixing has become highly promising to enhance the unlinkability of blockchains .

The mixing procedure could be implemented with or without a hub. CoinShuffle [18] is a typical non-hub scheme to mix Bitcoin. Users are required to encrypt their output addresses layer by layer and decrypt them in a pre-designed order. CoinShuffle++ [19] proposes a P2P mixing protocol, DiceMix, to generate a decentralized CoinJoin transaction in an arbitrary order. These two schemes deal with the plain-text transaction in which the transaction amount is public. ValueShuffle [17], based on DiceMix [19], can mix the confidential transaction in the way that the amount is hidden by cryptographic commitment. All these non-hub schemes suffer from large off-chain communication overhead. For example, CoinShuffle++ [19] and ValueShuffle [17] require every user to communicate with each of the other participants to generate a CoinJoin transaction. The communication cost is even more when there are malicious participants.

In contrast to the non-hub mixing, in the *hub-based* solutions, a *trustless* tumbler acting as the hub provides the *mixing service* for users. In this way, neither an involved P2P mixing protocol nor participant coordination is required, which makes it quite feasible in practice. TumbleBit [12] and A^{2}L [22] are typical hub-based schemes that prevent the tumblers from linking users during mixing. However, most existing hub-based mixing service works for non-confidential blockchain only. Due to the public nature, anyone can link two transactions' participants together by matching the equal values on the input and output sides. So only fixed and equal amount mixing is supported [12, 14, 22]. As far as we know, the only confidential payment system that supports mixing is [17], a non-hub mixing working in the unspent-transaction-output (UTXO) model.

Unlike the UTXO model built from Bitcoin, the account-based model introduced by Ethereum supports smart contracts, which makes it well-suited for real-world applications. Mixing is even more important in the account-based model. Möbius [14] uses a smart contract as a tumbler for the first time, but it does not support confidential mixing. Most recently, people have proposed several confidential payment systems in the account-based model, such as basic Zether [7] and PGC [8]. Unfortunately, neither of them supports unlinkability. Though [7] and [9] use sophisticated ZK-proofs and many-out-of-many proofs to allow anonymous transactions, the resulting schemes are not compatible with the original Zether anymore, and thus emerge as stand-alone cryptocurrencies. To the best of our knowledge, providing the mixing service for the confidential payment system in the account-based model is still an open problem. To solve it, there are two major challenges.

– How to achieve unlinkability without undermining the *safety* of the underlying blockchain? In other words, the add-on mixing service could neither break the underlying system's balance nor steal money from any participants.

Table 1. Comparison with previous mixing solutions

Type	Scheme	# Off-Chain Messages[1]	# Transactions /User	Model	Confidential Mixing	Payment Value	DoS Resistance
Non-hub	CoinShuffle [18]	$\mathcal{O}(n)$	1	UTXO	✗	Fixed	N/A
	CoinShuffle++ [19]	$\mathcal{O}(n)$	1	UTXO	✗	Fixed	N/A
	ValueShuffle [17]	$\mathcal{O}(n)$	1	UTXO	✓	Arbitrary	N/A
Hub-based	TumbleBit [12]	12	4	U or A	✗	Fixed	✗
	A^2L [22]	9	4	U or A	✗	Fixed	✓
	Möbius [14]	2	2	Account	✗	Fixed	✓
	Ours	2	2	Account	✓	Arbitrary	✓

[1] n is the number of the mixing users.

– How to provide an *available* mixing service? To be feasible in practice and be compatible with the underlying blockchain, the introduced scheme should resist Deny-of-Service (DoS) attacks, be easy to implement, and not change the underlying transaction format.

Contributions and roadmap. This work addresses the aforementioned two challenges with a protocol we call MixCT, which provides mixing services for the confidential payment systems in the account-based model. To be applicable, MixCT only requires that the underlying blockchain utilizes homomorphic hiding to achieve confidentiality. Our contributions are as follows.

– *Generic confidential mixing service in the account-based model.* For any confidential payment system built from the homomorphic commitment in the account-based model, MixCT can provide the mixing service with a trustless tumbler, without changing the transaction format (Section 3 and 4). MixCT can also be applied to schemes that use additively homomorphic encryption to hide transaction amounts (Section 7). A comparison between MixCT and the state-of-the-art mixing approaches is given in Table 1.
– *Formal security definition and proof.* We formalize the security goals of MixCT using the game-based security model and provide the formal security proof accordingly. In particular, we analyze the safety of the system layer and user layer comprehensively (Section 5).
– *Efficient instantiation and valid evaluation.* We instantiate MixCT with Pedersen commitment and one-out-of-many proof on Ethereum. The evaluation results show that our MixCT is cheap compared with other unlinkability solutions for the confidential payment systems. (Section 6).

Technical Overview. We give an overview of MixCT, on top of a confidential payment system built in the account-based model.

In MixCT, the tumbler divides the time into epochs, and one epoch consists of two phases: escrow and redeeming. In the escrow phase, the tumbler collects new escrow transactions from the blockchain and publicly updates the tumbler's state. In the redeeming phase, the tumbler funds the receivers who can provide the valid redeeming requests in this epoch. Each participant can validate both the escrow and redeeming transactions, while no one (including the tumbler but excluding the sender/receiver acting as the creators of the escrow/redeeming request respectively) can link an escrow transaction with a redeeming transaction in each epoch. That is, the tumbler provides the mixing service without breaking the unlinkability. We now describe how the two challenges mentioned above are settled in MixCT.

The first challenge is to maintain the safety for both the mixing users and the underlying ledger. This means that the tumbler can neither violate the unlinkability nor steal coins, and a user can neither "print money" via a mixing service nor double-spend an escrow. It is hard to satisfy all these requirements, since the transaction amounts are invisible. Our main idea to address this challenge is to utilize a one-to-one permutation. In each epoch, the tumbler forms a one-to-one permutation among the redeeming and escrow transactions. This permutation must be invisible to achieve unlinkability, so we use pseudo-random permutation to mask the escrow transaction and non-interactive zero-knowledge proof to guarantee successful redeeming.

The second challenge is to enhance the availability, including being compatible with the underlying confidential blockchain, resisting DoS attacks, and reducing the costs of providing a mixing service. To address this challenge, we use the preimage of the mask as the additional random factor to generate the redeeming transaction, so that the resulting escrow and redeeming transactions' format is unmodified compared with the underlying chain. Fortunately, this design turns out to be DoS resistance, and all the required building blocks could be instantiated with lightweight cryptographic tools, as discussed in Section 6.

2 Preliminaries

Basic notations. We denote the length of a binary string x by $|x|$, and the size of a finite set S by $|S|$. We denote the security parameter by $\lambda \in \mathbb{N}$, and the uniformly random selection of one element r from a space $\mathcal{R}$ by $r \leftarrow\$\, \mathcal{R}$. $\mathsf{negl}(\lambda)$ is a negligible function with respect to λ. For algorithms, we denote an output y from a randomized algorithm A with the input x and a random factor r by $y \leftarrow \mathsf{A}(x, r)$. For $n \in \mathbb{N}$, we denote $\{1, 2, ..., n\}$ as $[n]$.

2.1 Cryptography Assumptions

Let $\mathsf{GroupGen}$ be a PPT algorithm that takes a security parameter λ as its input. $\mathsf{GroupGen}$ outputs the description of a cyclic group $\mathbb{G}$ of prime order $p \geq 2^\lambda$, and one (or more) random generator(s) $g \leftarrow\$\, \mathbb{G}$.

Definition 1 (One Way Permutation) *Let* OWP *be a one-way permutation defined over* $\mathcal{X}$ *where* $|\mathcal{X}| \geq 2^\lambda$*. Then the* OWP *is hard-to-invert, if for any* PPT *adversary* $\mathcal{A}$ *and* $y \leftarrow\$\, \mathcal{X}$*,* $\Pr[y = \mathsf{OWP}(x) \mid \mathcal{A}(\mathsf{OWP}, y) = x] \leq \mathsf{negl}(\lambda)$.

Definition 2 (Discrete Logarithm Assumption) *Run* $\mathsf{GroupGen}$ *to output* $\mathbb{G}, p, g$*. The discrete logarithm* (DLog) *assumption holds if for any* PPT *adversary* $\mathcal{A}$ *and* $h \leftarrow\$\, \mathbb{G}$*,* $\Pr[g^x = h \mid \mathcal{A}(\mathbb{G}, p, g, h) = x] \leq \mathsf{negl}(\lambda)$*. The* DLog *is a hard-to-invert instantiation of a* OWP *defined over* $\mathbb{G}$.

2.2 Additive Homomorphism Message Hiding Schemes

To achieve the confidential payment in blockchain, the transaction amount is hidden by some cryptographic tools, including commitment and public encryption. In this section, we give the definition of homomorphic commitment and its typical instantiation. Besides, we discuss the relation between homomorphic public key encryption and commitment.

An additively homomorphic commitment protocol HCom includes the following algorithms. Here we use $\mathcal{M}, \mathcal{R}, \mathcal{C}$ to denote the message space, randomness space, and the resulting space respectively.

- $\mathsf{pp} \leftarrow \mathsf{HCom.Gen}(\lambda)$. Output the public parameters pp, including the commitment key, used to hide a message $m \in \mathcal{M}$.
- $c \leftarrow \mathsf{HCom.Com}(m; r)$. On input $m \in \mathcal{M}$ and a hiding factor $r \leftarrow\!\$\, \mathcal{R}$, output the hidden result c.

We review the well-known hiding and binding properties of the commitment schemes in Appendix A.

Definition 3 (Additive Homomorphism) *We call the protocol* HCom *defined above is additive homomorphism, if for all* $\mathsf{pp} \leftarrow \mathsf{HCom.Gen}(\lambda)$*, all* $m_1, m_2 \in \mathcal{M}$ *and all* $r_1, r_2 \in \mathcal{R}$*, we have*

$$\mathsf{HCom.Com}(m_1; r_1) \oplus_C \mathsf{HCom.Com}(m_2; r_2) = \mathsf{HCom.Com}(m_1 \oplus_M m_2; r_1 \oplus_R r_2)$$

where $\oplus_M$*,* $\oplus_R$*,* $\oplus_C$ *operate on* $\mathcal{M}, \mathcal{R}$*, and* $\mathcal{C}$ *respectively.*

Pedersen Commitment. It is the most widely used additively homomorphic commitment scheme, which consists of the following algorithms. Pedersen commitment is additively homomorphic and perfectly hiding.

- $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda)$. Run $\mathsf{GroupGen}$ to generate $\mathsf{pp} = (\mathbb{G}, p, g, h)$, where $\mathbb{G}$ is a cyclic group of prime order p and g, h are two generators (commitment key) of $\mathbb{G}$. The message and randomness spaces are $\mathbb{Z}_p$.
- $c \leftarrow \mathsf{Com}(m; r)$. On input message m and randomness r, output the commitment $c = g^m h^r$.

Additive Homomorphism Public Key Encryption. The additively homomorphic message hiding could be instantiated by public key encryption that is additively homomorphic. Typically, to construct confidential transactions, Zether [7] utilizes ElGamal, and PGC [8] proposes and applies the Twisted ElGamal. These additively homomorphic encryption schemes obviously satisfy the homomorphism and hiding properties and are also binding for a fixed public key.

It is worth noting that a commitment scheme can be constructed in a standard way from a CPA secure public key encryption with perfect decryption. In our work, we use HCom as the building block for MixCT. We discuss MixCT's instantiation purely with homomorphic public key encryption in Section 7.

2.3 Non-Interactive Zero-Knowledge Proof

A non-interactive zero-knowledge (NIZK) proof is a protocol using which the prover could convince the verifier that a statement is true without leaking any further information. Let $\mathcal{R}_{\mathcal{L}} \subseteq \mathcal{X} \times \mathcal{W}$ be an $\mathcal{NP}$ relation and we say $w \in \mathcal{W}$ is a witness for the statement $x \in \mathcal{X}$ if $(x, w) \in \mathcal{R}_{\mathcal{L}}$. The corresponding language is defined as $\mathcal{L} = \{x \,|\, \exists w \in \mathcal{W} \text{ s.t. } (x, w) \in \mathcal{R}_{\mathcal{L}}\}$. A NIZK scheme consists of three algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Vf})$,

- $\mathsf{pp} \leftarrow \mathsf{NIZK.Gen}(\lambda)$. Output the public parameters pp, which will be an implicit input of the following two algorithms.
- $\pi \leftarrow \mathsf{NIZK.Prove}(x, w)$. On input the statement x and the witness w, output a proof π.
- $0/1 \leftarrow \mathsf{NIZK.Vf}(x, \pi)$. On input a statement x and a proof π, output 0 if reject and 1 otherwise.

We review the completeness, soundness, and zero-knowledge properties of the NIZK in Appendix A.

One-out-of-many proof. One-out-of-many proof (OOOM) is a proof protocol to prove that 1 out of n commitments $(c_1, c_2, \cdots, c_n)$ is committed to 0 under the commitment key ck. Formally, it is defined by the following relation.

$$\left\{ ((ck, (c_1, c_2, \cdots, c_n)), (l, r)) \;\middle|\; \begin{array}{c} \forall i \in [n] : c_i \in C_{ck} \;\wedge\; l \in [n] \;\wedge\; r \in \mathcal{R} \\ \wedge\; c_l = \mathsf{Com}_{ck}(0; r) \end{array} \right\}$$

where $\mathcal{R}$ is the randomness space and $[n]$ stands for the interval $\{1, ..., n\}$. The protocol is first introduced by Groth and Kohlweiss in [11] and an optimized construction is given in [6]. In the standard way, Fiat-Shamir transform [3] can be used to turn the protocol into a NIZK proof scheme, which requires no trusted setup and provides higher efficiency in contrast to the general-purpose NIZK schemes. Precisely, OOOM consists of two algorithms $(\mathsf{OOOM.Prove}, \mathsf{OOOM.Vf})$,

- $\pi \leftarrow \mathsf{OOOM.Prove}(ck, (c_1, c_2, \cdots, c_n), (l, r))$. On input the commitment key ck, the statement $(c_1, c_2, \cdots, c_n)$, and the witness (l, r), output the proof π.
- $0/1 \leftarrow \mathsf{OOOM.Vf}(ck, (c_1, c_2, \cdots, c_n), \pi)$. On input the commitment key ck, the statement $(c_1, c_2, \cdots, c_n)$, and the proof π, output 0 if reject and 1 otherwise.

3 Definition of Mixing Confidential Transaction System

In this section, we provide the formal definition of the mixing confidential transaction system (MixCT) in the account-based model.

3.1 Data Structure

The mixing confidential transaction system overlays an underlying blockchain supporting decentralized confidential transactions. We begin by presenting the base components required to construct the MixCT system.

Blockchain. Blockchain is a public decentralized append-only ledger that is stateful by all previously confirmed transactions. Let $\mathcal{L}_T$ be the ledger's state at time T. For any $t > 0$, $\mathcal{L}_T$ is a prefix of $\mathcal{L}_{T+t}$. By a sequence of transactions between T and $T + t$, $\mathcal{L}_T$ can be updated to $\mathcal{L}_{T+t}$. In a blockchain, users are identified by their public keys, and each user can generate one or more key pairs $(\mathsf{pk}, \mathsf{sk})$ to form transactions.

Account-based model. In the account-based model, the blockchain maintains a mapping $\mathsf{bal}[\mathsf{pk}]$ from each user's public key pk to its balance v. The public key pk works as an address to receive transactions from other users. The user's secret key sk is used to generate transactions to other accounts. Once a transaction is confirmed on the blockchain, the transaction amount will be added/subtracted to/from the balance of the receiver/sender's account.

Smart Contract. A smart contract is a piece of code executed on the blockchain, receiving coins and handling them according to predefined rules when triggered by users. Typically, the smart contract works in the account-based model. It is worth noting that the smart contract has a public key pk_{sc} as its address, with no corresponding secret key.

Confidential Transaction. The confidential transaction [13] is a special form of blockchain transaction in which the transaction amount is hidden by some cryptography tools. Without loss of generality, a confidential transaction ctx can be described as

$$\mathsf{ctx} = (\mathsf{pk}_s, \mathsf{pk}_r, c, \mathsf{aux}),$$

where $\mathsf{pk}_s/\mathsf{pk}_r$ is the public key of the transaction's sender/receiver respectively, c is the confidential transaction amount[3], and aux is the application-dependent auxiliary information used to check the validity of the transaction. In particular, if the confidential transaction is output by a smart contract with address pk_{sc}, then the format of aux is different, i.e., it contains neither a signature nor any information derivable from the secret key. Once a ctx gets confirmed by the blockchain, it will be recorded on the ledger. Simultaneously, the sender and the receiver's accounts will be updated.

To generically describe the generation and verification of ctx, a confidential transaction scheme could be described by two main functions $\mathsf{CT} = (\mathsf{Create}, \mathsf{Verify})$,

- $\mathsf{ctx} \leftarrow \mathsf{CT.Create}(\mathsf{sk}_s, \mathsf{pk}_r, v)$. pk_s's owner uses the secret key sk_s to create a confidential transaction ctx, which transfers amount v to the receiver's address pk_r in the confidential form. In particular, if Create is called by a receiver and run by a smart contract to fund amount v to the receiver pk_r, we denote it as $\mathsf{CT.Create}(\mathsf{pk}_{sc}, \mathsf{pk}_r, v)$. In this case, the resulting ctx is valid only if this function has been run by the smart contract.
- $1/0 \leftarrow \mathsf{CT.Verify}(\mathsf{ctx})$. Every blockchain user validates ctx with this function.

As far as we know, all the existing confidential transaction schemes in the account-based model utilize homomorphic cryptography primitives, including homomorphic commitment and public key encryption.

[3] c could be a vector, such as in PGC or Zether.

3.2 Entities

In MixCT, in addition to the common users of the blockchain, there are two more types of entities.

Mixing Service Users. Mixing service users are a set of the underlying blockchain users willing to utilize the mixing service. Each mixing service user controls a pair of accounts, acting as the sender and the receiver. Without ambiguity, we also use "user" to stand for a mixing service user later.

Tumbler. A tumbler is a *trustless* medium that works as an unlinkable payment hub to provide the mixing service for users. More specifically, senders escrow their coins to the tumbler, and receivers redeem coins from it. For safety, the tumbler can neither violate the unlinkability between the escrow and redeeming transactions, "print money", nor steal money from users.

One main obstacle in designing a qualified tumbler lies in how to moderate its capacity. That is, the tumbler can validate the received escrow/redeeming transactions but cannot "open" them. Otherwise, the unlinkability is broken immediately. In MixCT, we use a smart contract with the public address pk_t as the tumbler, which solves this paradox perfectly. Since the tumbler, as a smart contract publicly executed by all blockchain miners, has no secret inputs, and has the same view as the blockchain users.

3.3 Definition of the Mixing Confidential Transaction System

For simplicity, we omit the transaction propagation and verification procedures run by the underlying confidential blockchain and focus on the add-on only. A mixing confidential transaction system in one epoch is a tuple of polynomial-time algorithms described as follows, and Figure 1 provides the overview of the MixCT.

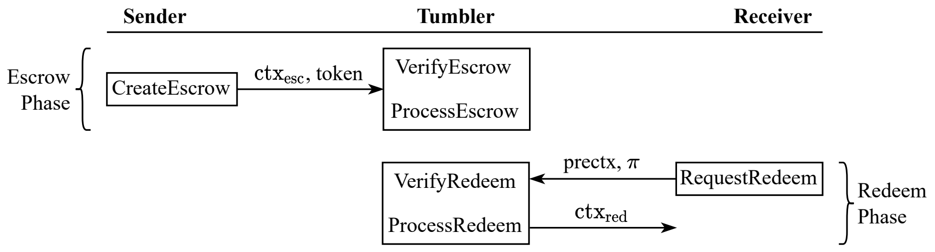

Fig. 1. Overview Sequence Diagram of MixCT.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$. Given a security parameter λ, the underlying blockchain executes this algorithm to generate all public parameters used in the system.
- $(\mathsf{ctx}_{\text{esc}}, \mathsf{token}, \mathsf{td}) \leftarrow \mathsf{CreateEscrow}(\mathsf{sk}_s, \mathsf{pk}_t, v)$. A sender with secret key sk_s executes this algorithm to generate a confidential transaction $\mathsf{ctx}_{\text{esc}}$, to escrow v coins to the tumbler with account address pk_t. Besides, this algorithm outputs a randomness td to generate a *one-time* escrowing token, and td is the trapdoor to redeem $\mathsf{ctx}_{\text{esc}}$.

- $0/1 \leftarrow \mathsf{VerifyEscrow}(\mathsf{ctx}_{\text{esc}}, \mathsf{token}, \mathsf{state})$. Given an escrow transaction $\mathsf{ctx}_{\text{esc}}$ and the corresponding token, the tumbler executes this algorithm to check the validity of $\mathsf{ctx}_{\text{esc}}$ and the uniqueness of token. That is, the token has never been recorded in the state, which is initially empty.
- $\mathsf{state}' \leftarrow \mathsf{ProcessEscrow}(\mathsf{ctx}_{\text{esc}}, \mathsf{token}, \mathsf{state})$. After verification, the tumbler uses the new received transaction $\mathsf{ctx}_{\text{esc}}$ and the corresponding token to update its storage state to state'. We note that since the tumbler works in a blockchain, its state is always publicly accessible.
- $(\mathsf{prectx}, \pi) \leftarrow \mathsf{RequestRedeem}(\mathsf{sk}_r, \mathsf{pk}_t, \mathsf{ctx}_{\text{esc}}, \mathsf{td}, \mathsf{state})$. The receiver with secret key sk_r executes this algorithm to generate a confidential *pre*-redeeming transaction prectx to be sent to the tumbler, based on an escrowing transaction $\mathsf{ctx}_{\text{esc}}$. The key idea is that the receiver who gets the trapdoor td of the $\mathsf{ctx}_{\text{esc}}$, can generate a valid redeeming proof π for prectx.
- $0/1 \leftarrow \mathsf{VerifyRedeem}(\mathsf{prectx}, \pi, \mathsf{state})$. The tumbler executes this algorithm to verify the validity and uniqueness of the pre-redeeming transaction prectx, by using the redeeming proof π and state.
- $(\mathsf{ctx}_{\text{red}}, \mathsf{state}') \leftarrow \mathsf{ProcessRedeem}(\mathsf{pk}_t, \mathsf{pk}_r, \mathsf{prectx}, \pi, \mathsf{state})$. Run by the tumbler. If prectx and the corresponding proof π are valid, the tumbler parses prectx and updates state to state', to prevent double-spend attacks. Then the tumbler uses prectx to generate a complete redeeming transaction $\mathsf{ctx}_{\text{red}}$ to the receiver with pk_r, and broadcasts this transaction. At this point, the receiver's account $\mathsf{bal}[\mathsf{pk}_r]$ will be updated by the additively homomorphic operation of the underlying confidential blockchain, which finishes the redeeming procedure.

We note that the last three algorithms are called by a receiver with pk_r and run by the tumbler with pk_t. We decouple the procedure to make it clearer.

3.4 Preconditional Assumptions

We assume that the underlying blockchain's confidentiality cannot be broken and do not consider network layer attacks, since they are orthogonal to our work. We also assume that the underlying blockchain does not support anonymous transactions, and at least 2 out of n users cannot be controlled by the adversary.

3.5 Security Goals for Confidential Mixing Service

Many previous works [14,17,18] have proposed several security requirements for the mixing service. However, most of them focus on non-confidential mixing. The security definition for the confidential mixing service in the account-based model is still missing. To this point, we propose 6 security goals for MixCT. In what follows, we give an informal description of these security goals, and the formal definitions can be found in Section 5.

- **Theft prevention.** This is a twofold definition. (1) For an accepted escrow transaction generated by a sender, nobody can redeem it unless getting the

transaction's trapdoor from the sender. (2) Nobody can redeem "out of nothing". That is, nobody can generate a valid redeeming transaction without a corresponding escrow transaction.
- **Balance.** Every escrowed transaction can be redeemed with the same value as the corresponding sender has escrowed.
- **Double-spend prevention.** Every escrowed transaction cannot be redeemed more than once.
- **Unlinkability.** After a sequence of escrow and redeeming, no one (including the tumbler but excluding the actual sender and receiver) can link a redeeming transaction with an escrow transaction.
- **Confidentiality.** No information about the transaction amount is leaked during the mixing procedure, which means that the introduction of MixCT will not violate the underlying confidentiality.
- **DoS resistance.** The protocol can resist DoS attacks, and every honest party can always terminate without losing money.

In summary, the first five properties are used to guarantee "safety", which means that there is a determined but invisible permutation between the set of the escrow transactions and the set of successful redeeming transactions. The last property is to guarantee "availability" of our scheme.

4 A Generic Design of MixCT from an Additively Homomorphic Commitment

In this section, we present the generic design of the MixCT scheme. For "generic" we mean that any additively homomorphic commitment could be utilized in MixCT to provide the mixing service for the underlying confidential payment system.

First, we introduce two data structures to clarify the local (but public) state maintained by the tumbler, as mentioned in Section 3.3. We define $\mathsf{state} = (\mathbf{EPool}, \mathbf{RPool})$ as follows. In the state, **EPool** is used to generate proof for a valid redeeming, and **RPool** is used to prevent double-spend attacks. Initially, both of them are empty.

- **EPool**. Used to record the "input set" of the invisible permutation. Store all the (c_e, token) pairs built from successful escrow transactions. Specifically, parse c_e from each successful escrow transaction $\mathsf{ctx}_{\text{esc}} = (\mathsf{pk}_s, \mathsf{pk}_t, c_e, \mathsf{aux})$. Then use c_e and the corresponding token to form a record in **EPool**.
- **RPool**. Used to record the "output set" of the invisible permutation. Store all the c_r part of the successful redeeming transaction. Specifically, parse c_r from each valid pre-redeeming request $\mathsf{prectx} = (\mathsf{pk}_t, \mathsf{pk}_r, c_r, \mathsf{aux})_{\text{pre}}$. Then use c_r to form a record in **RPool**.

We are now ready to construct the protocol described in Section 3.3. Here we use the additively homomorphic commitment scheme $\mathsf{HCom} = (\mathsf{Gen}, \mathsf{Com})$, non-interactive zero-knowledge scheme $\mathsf{NIZK} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Vf})$, and a one-way permutation OWP whose domain is the same as the randomness space $\mathcal{R}$ in HCom. The constructions are as follows.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$. Run $\mathsf{pp}_{\text{Hcom}} \leftarrow \mathsf{HCom.Gen}(\lambda), \mathsf{pp}_{\text{nizk}} \leftarrow \mathsf{NIZK.Gen}(\lambda)$, and output $\mathsf{pp} = (\mathsf{pp}_{\mathsf{HCom}}, \mathsf{pp}_{\mathsf{NIZK}})$.
- $(\mathsf{ctx}_{\text{esc}}, \mathsf{token}, \mathsf{td}) \leftarrow \mathsf{CreateEscrow}(\mathsf{sk}_s, \mathsf{pk}_t, v)$. Run $\mathsf{ctx}_{\text{esc}} \leftarrow \mathsf{CT.Create}(\mathsf{sk}_s, \mathsf{pk}_t, v)$ to generate a confidential transaction. Specifically, $\mathsf{ctx}_{\text{esc}} = (\mathsf{pk}_s, \mathsf{pk}_t, c_e, \mathsf{aux})$ in which $c_e = \mathsf{HCom.Com}(v; r_e)$. Then, sample a uniform $t \leftarrow\!\$\, \mathcal{R}$ as the td, and compute $\mathsf{token} \leftarrow \mathsf{OWP}(t)$.
- $0/1 \leftarrow \mathsf{VerifyEscrow}(\mathsf{ctx}_{\text{esc}}, \mathsf{token}, \mathsf{state})$. Run $\mathsf{CT.Verify}(\mathsf{ctx}_{\text{esc}})$ to verify the confidential transaction $\mathsf{ctx}_{\text{esc}}$ with aux. Then check whether the token is in the range of OWP, and check if $(*, \mathsf{token}) \notin \mathbf{EPool}$. Output 1 if all checks pass, otherwise output 0[4].
- $\mathsf{state}' \leftarrow \mathsf{ProcessEscrow}(\mathsf{ctx}_{\text{esc}}, \mathsf{token}, \mathsf{state})$. When the transaction $\mathsf{ctx}_{\text{esc}}$ has been confirmed by the underlying blockchain, parse the confidential value c_e from $\mathsf{ctx}_{\text{esc}}$. Then update state by storing (c_e, token) into $\mathbf{EPool}$, i.e., $\mathbf{EPool} \leftarrow \mathbf{EPool} + (c_e, \mathsf{token})$.
- $(\mathsf{prectx}, \pi) \leftarrow \mathsf{RequestRedeem}(\mathsf{sk}_r, \mathsf{pk}_t, \mathsf{ctx}_{\text{esc}}, \mathsf{td}, \mathsf{state})$. Parse the confidential amount c_e from $\mathsf{ctx}_{\text{esc}}$ first. Use the td to generate a mask $c_{\text{mask}} = \mathsf{HCom.Com}(0; \mathsf{td})$, and compute the new masked amount c_r by homomorphicly adding c_{mask} to c_e, i.e., $c_r = c_e \oplus_C c_{\text{mask}}$. Then, create a pre-redeeming transaction prectx whose confidential amount is c_r. The format of prectx is $(\mathsf{pk}_t, \mathsf{pk}_r, c_r, \mathsf{aux})_{\text{pre}}$. The difference between a pre-redeeming request and a truly redeeming transaction is that prectx has not been run by the tumbler. Next, read the tumbler's $\mathbf{EPool}$ from its state. Run $\mathsf{NIZK.Prove}(\mathbf{EPool}, c_r)$ with witness td to generate a proof π for $(\mathbf{EPool}, c_r) \in \mathcal{L}_{\text{legal}}$, where $\mathcal{L}_{\text{legal}}$ is defined as

$$\left\{ (\mathbf{EPool}, c_r) \;\middle|\; \begin{array}{c} \mathsf{td} \in \textit{Domain of } \mathsf{OWP} \wedge \mathsf{token} = \mathsf{OWP}(\mathsf{td}) \\ \wedge\ \exists (c_e, \mathsf{token}) \in \mathsf{state}.\mathbf{EPool} \\ \wedge c_r = c_e \oplus_C \mathsf{HCom.Com}(0; \mathsf{td}) \end{array} \right\}^{[5]}$$

- $0/1 \leftarrow \mathsf{VerifyRedeem}(\mathsf{prectx}, \pi, \mathsf{state})$. First, parse the confidential amount c_r from prectx and check if c_r does not exist in $\mathbf{RPool}$. Validate the transaction format of prectx. Next, verify the proof π by $\mathsf{NIZK.Vf}(\mathbf{EPool}, c_r, \pi)$. Output 1 if all checks pass, otherwise output 0.
- $(\mathsf{ctx}_{\text{red}}, \mathsf{state}') \leftarrow \mathsf{ProcessRedeem}(\mathsf{pk}_t, \mathsf{pk}_r, \mathsf{prectx}, \mathsf{state})$. The tumbler constructs the complete redeeming transaction ctx from prectx by running it. Then it broadcasts the resulted transaction to confirm that $\mathsf{ctx}_{\text{red}}$ adds the transaction amount to address pk_r. Record $\mathsf{ctx}_{\text{red}}$ into $\mathbf{RPool}$ by $\mathbf{RPool} \leftarrow \mathbf{RPool} + c_r$ and output the updated state'.

[4] Note that the same c_e may existed. That is because, in different escrow transactions, a user may use the same random factor to hide the same transaction amount. In MixCT, though the (c_e, token) pair is recorded, we only require that each token is unique to identify each escrow transaction.

[5] c_r is a commitment under the same key as c_e.

5 Formal Security Definitions and Proofs

In this section, we present the formal definitions of the security goals mentioned in Section 4. We now state the main result of this paper.

Theorem 1 *The confidential mixing service tuple* $\mathsf{MixCT} = (\mathsf{Setup}, \mathsf{CreateEscrow}, \mathsf{VerifyEscrow}, \mathsf{ProcessEscrow}, \mathsf{RequestRedeem}, \mathsf{VerifyRedeem}, \mathsf{ProcessRedeem})$*, as defined in Section 4, is a safe and available confidential mixing service solution.*

We prove this theorem via 6 separate lemmas, which show that our construction satisfies all the requirements of the confidential mixing service. We assume that the adversary $\mathcal{A}$ cannot only let the honest parties participate in the mixing service but also corrupt some of them. Besides, $\mathcal{A}$ has no more access to the tumbler than getting its public key or making escrow/redeeming queries. This is because the tumble cannot open any transaction, as mentioned in Section 3.2.

The definition of the oracle $\mathcal{O}$ available to the adversary $\mathcal{A}$ and the detailed proofs of each lemma can be found in Appendix B.

Definition of the Theft Prevention. If $\mathcal{A}$ can successfully generate a pre-redeeming request, by either redeeming an honest user's escrow or redeeming out of nothing, we call the $\mathcal{A}$ wins the theft prevention game. Formally, we define theft prevention via the following security experiment between $\mathcal{A}$ and $\mathcal{C}$.

$$\mathsf{Adv}^{\text{theft}}_{\mathsf{MixCT}}(\lambda) = \Pr\left[\begin{array}{c|c} b = 1 \wedge & \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda); \\ ((c_e^*, \mathsf{token}^*) \in \mathsf{state}.\mathbf{EPool}_h \vee & \mathsf{state}, (\mathsf{prectx}^*, \pi^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}); \\ (c_e^*, \mathsf{token}^*) \notin \mathsf{state}.\mathbf{EPool}) & b \leftarrow \mathsf{VerifyRedeem}(\mathsf{prectx}^*, \pi^*, \mathsf{state}) \end{array}\right]$$

where $(\mathsf{ctx}^*_{\text{esc}}, \mathsf{token}^*)$ denotes the corresponding escrow transaction of prectx^* (if any), c_e^* is the confidential value in $\mathsf{ctx}^*_{\text{esc}}$, and $\mathsf{state}.\mathbf{EPool}_h$ (containing the honest users' escrows) is a subset of $\mathsf{state}.\mathbf{EPool}$.

Lemma 1. *Assuming the hard-to-invert property of the* OWP *and the soundness of the* NIZK*, the* MixCT *is theft prevention.*

Definition of the Balance. Here we focus on the adversary's attempt to redeem an escrowed transaction with a different value, since the adversary could not redeem the non-corrupted user's escrow or redeem out of nothing by Lemma 1. Here we define the balance attack game for the corrupted users only.

$$\mathsf{Adv}^{\text{balance}}_{\mathsf{MixCT}}(\lambda) =$$

$$\Pr\left[\begin{array}{c|c} & \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda); \\ b = 1 \wedge & \mathsf{state}, (\mathsf{ctx}^*_{\text{esc}}, \mathsf{token}^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{td}^*); \\ c_r^* = \mathsf{HCom}.\mathsf{Com}(v^{*\prime}; r^{*\prime}) \wedge & (\mathsf{prectx}^*, \pi^*), \mathsf{state}' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{state}, \mathsf{ctx}^*_{\text{esc}}, \mathsf{td}^*); \\ (v^{*\prime}, r^{*\prime}) \neq (v^*, r^* \oplus_R \mathsf{td}^*) & b \leftarrow \mathsf{VerifyRedeem}(\mathsf{prectx}^*, \pi^*, \mathsf{state}') \end{array}\right]$$

where $(\mathsf{ctx}^*_{\text{esc}}, \mathsf{token}^*) \in \mathsf{state}.\mathbf{EPool}$, and c_r^* is the confidential amount in prectx^*.

Lemma 2. *Assuming the binding of* HCom *and the soundness of* NIZK*, the* MixCT *is balance secure.*

Definition of the Double-spend Prevention. If $\mathcal{A}$ can redeem one escrow transaction twice, we call $\mathcal{A}$ wins the double-spend prevention game. The formal definition is as follows.

$$\mathsf{Adv}^{\text{double-spend}}_{\mathsf{MixCT}}(\lambda) =$$

$$\Pr\left[\begin{array}{l|c} \mathsf{ctx}^{*}_{\mathrm{red}} \text{ and } \mathsf{ctx}^{\wedge}_{\mathrm{red}} \text{ redeem} & \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda); \\ \text{the same } (\mathsf{ctx}^{*}_{\mathrm{esc}}, \mathsf{token}^{*}) & \mathsf{state} \ \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}); \\ & \mathsf{state}', (\mathsf{ctx}^{*}_{\mathrm{red}}, \pi^{*}) \ \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{state}); \\ & \mathsf{state}'', (\mathsf{ctx}^{\wedge}_{\mathrm{red}}, \hat{\pi}) \ \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}, \mathsf{state}') \end{array}\right]$$

Lemma 3. *Based on the one-to-one property of* OWP *and the soundness of the* NIZK*, the* MixCT *is double-spend prevention.*

Definition of the Unlinkability. We define the unlinkability via the following experiment between $\mathcal{A}$ and $\mathcal{C}$.

$$\mathsf{Adv}^{\text{unlink}}_{\mathsf{MixCT}}(\lambda) =$$

$$\Pr\left[\begin{array}{l|c} & \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda); \\ & (\mathsf{state}, \{\mathsf{ctx}_{\mathrm{esc}i}, \mathsf{token}_i, \mathsf{pk}_{s_i}\}_{i=0,1 \wedge \mathsf{pk}_{s_i} \in T_h}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}); \\ b' = b & b \leftarrow\!\!\$ \ \{0,1\}; \\ & (\mathsf{prectx}^{*}, \pi^{*}) \leftarrow \mathsf{RequestRedeem}(\mathsf{pk}_{r_b}, \mathsf{pk}_t, \mathsf{ctx}_{\mathrm{esc}b}, \mathsf{td}_b, \mathsf{state}); \\ & b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{prectx}^{*}, \pi^{*}, \mathsf{state}) \end{array}\right]$$

where $(\mathsf{pk}_{s_i}, \mathsf{pk}_{r_i})$ denotes a pair of sender/receiver, $(\mathsf{ctx}_{\mathrm{esc}i}, \mathsf{token}_i)$ is created by pk_{s_i}, T_h denotes the honest users' set, and neither of $\{\mathsf{pk}_{s_i}\}_{i=0,1}$ is corrupted by $\mathcal{A}$. The confidential amount in $(\mathsf{prectx}^{*}, \pi^{*})$ is c^{*}_{r}.

Lemma 4. *Assuming the* NIZK *is zero-knowledge and the* HCom *is hiding, no ppt $\mathcal{A}$ can win the unlinkability game in* MixCT *with non-negligible probability.*

Definition of the Confidentiality. For confidentiality, we mean to block potential leakage of the transaction amount. Here we define confidentiality via the following experiment between $\mathcal{A}$ and $\mathcal{C}$.

$$\mathsf{Adv}^{\text{confidential}}_{\mathsf{MixCT}}(\lambda) =$$

$$\Pr\left[\begin{array}{l|c} & \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda); \\ & (\mathsf{state}, \mathsf{pk}_s, \mathsf{pk}_r, v_0, v_1) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pp}); \\ & b \leftarrow\!\!\$ \ \{0,1\}; \\ b' = b & (\mathsf{ctx}^{*}_{\mathrm{esc}}, \mathsf{token}^{*}, \mathsf{td}^{*}) \leftarrow \mathsf{CreateEscrow}(\mathsf{sk}_s, \mathsf{pk}_t, v_b); \\ & \mathsf{state}' \leftarrow \mathsf{ProcessEscrow}(\mathsf{ctx}_{\mathrm{esc}}, \mathsf{token}, \mathsf{state}); \\ & (\mathsf{prectx}^{*}, \pi^{*}) \leftarrow \mathsf{RequestRedeem}(\mathsf{pk}_r, \mathsf{pk}_t, \mathsf{ctx}_{\mathrm{esc}}, \mathsf{td}^{*}, \mathsf{state}'); \\ & b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{prectx}^{*}, \pi^{*}, \mathsf{ctx}^{*}_{\mathrm{esc}}, \mathsf{token}^{*}) \end{array}\right]$$

where $(\mathsf{pk}_s, \mathsf{pk}_r)$ denotes a pair of sender/receiver. As natural restrictions for $\mathsf{pk}_s \in T_h$ and $\mathsf{pk}_r \in T_h$, we assume that $\mathcal{A}$ cannot control them.

Lemma 5. *Assuming the confidentiality of the underlying blockchain, the hiding property of* HCom*, and the zero-knowledge of* NIZK*, the* MixCT *satisfies confidentiality.*

Definition and analysis of the DoS Resistance. DoS attacks usually occur when a mass number of users join in some protocol, or the adversary tries to deviate the honest users from the normal execution of the protocol. In particular, for the mixing service, the adversary $\mathcal{A}$ may try to prevent the honest users from getting the service or redeeming their escrow. In our design, a mixing service's user pays the tumbler first and then can redeem it. As such $\mathcal{A}$ will pay great expenses incurred launching a DoS attack. Moreover, once a user escrows coins to the tumbler, based on the completeness of our NIZK, the user can always generate a valid redeem request, without the help of any other participants.

Lemma 6. *The* MixCT *is* DoS *resistance.*

6 An Efficient Instantiation of MixCT

We now present an efficient instantiation of our generic MixCT construction without the trusted setup. Specifically, assuming the underlying confidential payment system utilizes Pedersen commitment as the HCom, we use DLog as the OWP, and the one-out-of-many proof OOOM as the NIZK. We also construct a smart contract to act as the untrusted tumbler.

6.1 Detailed Instantiation

For simplicity, we omit the non-confidential and public verifiable information contained in escrows and redeem transactions, including $\mathsf{pk}_s, \mathsf{pk}_t, \mathsf{pk}_r, \mathsf{aux}$, etc.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(\kappa)$. Run $\mathsf{GroupGen}(\lambda)$, and output $\mathsf{pp} = (\mathbb{G}, p, g, h, f)$, where $\mathbb{G}$ is a cyclic group of prime order p and g, h, f are three generators of $\mathbb{G}$.
- $(\mathsf{ctx}_{\mathrm{esc}}, \mathsf{token}, \mathsf{td}) \leftarrow \mathsf{CreateEscrow}(\mathsf{sk}_s, v)$. Here c_e in $\mathsf{ctx}_{\mathrm{esc}}$ is the Pederson commitment of transaction amount v, and token uses DLog. Thus,

$$(c_e, \mathsf{token}) = (\mathsf{Com}(v; r), \mathsf{OWP}(t)) = (g^v h^r, f^t)$$

- $0/1 \leftarrow \mathsf{VerifyEscrow}(\mathsf{ctx}_{\mathrm{esc}}, \mathsf{token}, \mathsf{state})$. Check $\mathsf{ctx}_{\mathrm{esc}}$'s validity via the underlying confidential payment system. Check $\mathsf{token} \in \mathbb{G}$ and token is unique.
- $\mathsf{state}' \leftarrow \mathsf{ProcessEscrow}(\mathsf{ctx}_{\mathrm{esc}}, \mathsf{token}, \mathsf{state})$. If the escrow transaction $\mathsf{ctx}_{\mathrm{esc}}$ gets through the above verification, add (c_e, token) to **EPool**.
- $(\mathsf{prectx}, \pi) \leftarrow \mathsf{RequestRedeem}(\mathsf{ctx}_{\mathrm{esc}}, \mathsf{td}, \mathsf{state})$.
  - Parse c_e from $\mathsf{ctx}_{\mathrm{esc}}$, and use the $\mathsf{td} = t_l$ in token to generate c_r used in prectx, where l is the index of the sender's escrow record in **EPool**.

$$c_r = c_e \oplus_C \mathsf{Com}(0; \mathsf{td}) = \mathsf{Com}(v; r) \cdot \mathsf{Com}(0; t_l) = g^v h^{r+t_l}$$

  - Denote **EPool** at this state as $(c_e^i, \mathsf{token}_i)_{i \in [n]}$. Run $\mathsf{OOOM.Prove}$ with witness t_l to generate a proof π for $(\mathbf{EPool}, c_r) \in \mathcal{L}_{\mathrm{legal}}$, where $\mathcal{L}_{\mathrm{legal}}$ is

$$\mathcal{L}_{\mathrm{legal}} = \left\{ (\mathbf{EPool}, c_r) \;\middle|\; \begin{array}{c} \forall i \in [n] : c_e^i, \mathsf{token}_i \in \mathbb{G} \\ \wedge\ l \in [n] \wedge t_l \in \mathbb{Z}_p \wedge\ \mathsf{token}_l = \mathsf{OWP}(t_l) = f^{t_l} \in \mathbb{G} \\ \wedge\ c_r = c_e^l \oplus_C \mathsf{Com}_{g,h}(0; t_l) = g^v h^{r+t_l} \end{array} \right\}.$$

$\mathcal{L}_{\text{legal}}$ can be converted into two OOOM Language $\mathcal{L}_{\text{legal}}^{\text{PoFmt}}$ and $\mathcal{L}_{\text{legal}}^{\text{PoRed}}$[6],

$$\mathcal{L}_{\text{legal}}^{\text{PoFmt}} = \left\{ (g,h), \left(\frac{c_r}{c_e^i}\right)_{i\in[n]} \middle| \begin{array}{c} \forall i \in [n] : \frac{c_r}{c_e^i} \in \mathbb{G} \wedge \ l \in [n] \wedge \ t_l \in \mathbb{Z}_p \\ \wedge \ \left(\frac{c_r}{c_e^l}\right) = \mathsf{Com}_{g,h}(0; t_l) = h^{t_l} \end{array} \right\}^{[7]},$$

$$\mathcal{L}_{\text{legal}}^{\text{PoRed}} = \left\{ \begin{array}{c} \left(g, \frac{h}{f}\right), \\ \left(\frac{c_r}{c_e^i \cdot \mathsf{token}_i}\right)_{i\in[n]} \end{array} \middle| \begin{array}{c} \forall i \in [n] : \frac{c_r}{c_e^i \cdot \mathsf{token}_i} \in \mathbb{G} \wedge \ l \in [n] \ \wedge t_l \in \mathbb{Z}_p \\ \wedge \ \mathsf{token}_l = \mathsf{OWP}(t_l) = f^{t_l} \in \mathbb{G} \\ \wedge \left(\frac{c_r}{c_e^l \cdot \mathsf{token}_l}\right) = \mathsf{Com}_{g,\frac{h}{f}}(0; t_l) = \left(\frac{h}{f}\right)^{t_l} \end{array} \right\}.$$

Intuitively, we use $\mathcal{L}_{\text{legal}}^{\text{PoFmt}}$ and $\mathcal{L}_{\text{legal}}^{\text{PoRed}}$ to prove that the receiver uses the (c_e^l, td_l) couple to generate the unique c_r accordingly, with the correct Pedersen commitment format under the key (g,h). Obviously, with l and the corresponding witness $\mathsf{td} = t_l$, the receiver can generate $\pi = (\pi^{\text{PoFmt}}, \pi^{\text{PoRed}})$ via $\mathsf{OOOM.Prove}$.

- $0/1 \leftarrow \mathsf{VerifyRedeem}(\mathsf{prectx}, \pi, \mathsf{state})$. Use state and run $\mathsf{OOOM.Vf}$ to check (c_r, π). Output 1 if $c_r \notin \mathbf{RPool} \wedge \mathsf{OOOM.Vf}\left((g,h), \left(\frac{c_r}{c_e^i}\right)_{i\in[n]}, \pi^{\text{PoFmt}}\right) = 1$ $\wedge\ \mathsf{OOOM.Vf}\left(\left(g, \frac{h}{f}\right), \left(\frac{c_r}{c_e^i \cdot \mathsf{token}_i}\right)_{i\in[n]}, \pi^{\text{PoRed}}\right) = 1$. Output 0 otherwise.
- $(\mathsf{ctx}_{\text{red}}, \mathsf{state}') \leftarrow \mathsf{ProcessRedeem}(\mathsf{pk}_t, \mathsf{prectx}, \pi, \mathsf{state})$. The tumbler runs this to construct the complete redeeming transaction ctx from prectx, add c_r to the receiver's address, and store $\mathsf{ctx}_{\text{red}}$ into $\mathbf{RPool}$ by $\mathbf{RPool} \leftarrow \mathbf{RPool} + c_r$ and output updated state'.

6.2 Implementation and Evaluation

To show the feasibility of MixCT, we implement it with Ethereum smart contract. Here we focus on the costs introduced by the MixCT. We change the number of the mixing service users and record the costs for escrow and redeeming accordingly. The costs of escrow lie in the generation of the token. In the redeeming, the costs contain the user's off-line generation of the one-out-of-many proof, and the tumbler's online verification and processing.

As shown in Table 2, the escrow costs are fixed for all. The redeeming costs increase from 496k Gas to 3085k Gas as the escrow users' number changes from 4 to 64, which is much lower than other sophisticated approaches [7,9]. Besides, to generate/validate the proof of a redeeming transaction with no more than 64 mixing users, the time cost is less than 1.5/0.5 seconds.

Our evaluation benchmarks are collected on a laptop with a 2.9GHz AMD R7-4800H processor. We implement our design on Truffle Suite [2], a development environment for decentralized applications. We use Solidity to accomplish the smart contract of the tumbler running on top of the Ethereum virtual machine

[6] To break the safety by generating a statement that belongs to $\mathcal{L}_{\text{legal}}^{\text{PoFmt}}$ and $\mathcal{L}_{\text{legal}}^{\text{PoRed}}$, but not to $\mathcal{L}_{\text{legal}}$, one needs to solve the DLog problem in group $\mathbb{G}$.

[7] Here for $a, b \in \mathbb{G}$, we use $\frac{a}{b}$ to denote the operation $a \cdot b^{-1}$.

Table 2. The evaluation results of MixCT

#users	Escrow		Redeem			
	Size/bytes	Gas/units	Size/bytes	Prove Time/ms	Verify Time/ms	Gas/units
$n = 4$	132	128,556	2,500	173	93	496,163
$n = 8$			2,948	271	133	714,341
$n = 16$			3,396	451	185	1,080,093
$n = 32$			3,844	758	288	1,755,951
$n = 64$			4,292	1,466	462	3,085,444
n	$\mathcal{O}(1)$		$\mathcal{O}(\log n)$	$\mathcal{O}(n \log n)$		

(EVM) and use JavaScript to implement the mixing user. Especially, we use the construction of one-out-of-many proof from [6]. All code has been open-sourced[8].

7 Extension to Additively Homomorphic Encryption

As defined in Section 3.3, MixCT only requires the additive homomorphism property of the confidential transaction from the underlying blockchain. Naturally, one might wonder whether a similar construction could be deployed to blockchains using additively homomorphic encryption.

Intuitively, it is possible since there is some standard construction from a CPA secure public-key encryption with perfect decryption (e.g., both ElGamal and Twisted ElGamal satisfy these requirements), to the commitment scheme. However, there are two issues here. First, the generic confidential transaction's format would be changed to $\mathsf{ctx} = (\mathsf{pk}_s, \mathsf{pk}_r, (c_s, c_r), \mathsf{aux})$, where (c_s, c_r) are the ciphertexts of two opposite amounts sharing the same absolute value. Second, to add c_r to the receiver's account without leaking its public key, and to ensure that only the appointed receiver can successfully redeem, the sender needs to confidentially add the receiver's public address into the escrow transaction. In this way, safety could be guaranteed.

We leave the concrete mixing service based on homomorphic encryption, esp., the efficient instantiation without a trusted setup, as future work.

8 Conclusion

In this paper, we have proposed a confidential mixing service scheme in the account-based model. The proposed scheme, MixCT, allows us to mix confidential transactions constructed from additively homomorphic commitments. We formalize this protocol in the system layer and user layer comprehensively, and then prove that it satisfies both safety and availability. Experimental evaluation shows that MixCT could be instantiated by lightweight cryptographic tools.

[8] `https://github.com/dujiajun/MixCT`

References

1. CoinJoin: Bitcoin privacy for the real world, https://bitcointalk.org/?topic=279249
2. Truffle Suite, https://trufflesuite.com/
3. Abdalla, M., An, J.H., Bellare, M., Namprempre, C.: From identification to signatures via the fiat-shamir transform: Minimizing assumptions for security and forward-security. In: Advances in Cryptology — EUROCRYPT 2002 (2002)
4. Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating User Privacy in Bitcoin. In: Financial Cryptography and Data Security (2013)
5. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized Anonymous Payments from Bitcoin. In: 2014 IEEE Symposium on Security and Privacy (2014)
6. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J., Petit, C.: Short accountable ring signatures based on DDH. In: ESORICS (2015)
7. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards Privacy in a Smart Contract World. In: Financial Cryptography and Data Security (2020)
8. Chen, Y., Ma, X., Tang, C., Au, M.H.: PGC: Decentralized Confidential Payment System with Auditability. In: ESORICS (2020)
9. Diamond, B.E.: Many-out-of-Many Proofs and Applications to Anonymous Zether. In: 2021 IEEE Symposium on Security and Privacy (SP) (2021)
10. Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: A New Design for Anonymous Cryptocurrencies. In: Advances in Cryptology – ASIACRYPT 2019 (2019)
11. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Advances in Cryptology - EUROCRYPT 2015 (2015)
12. Heilman, E., AlShenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub. In: NDSS (2017)
13. Maxwell, G.: Confidential transactions (2015), https://people.xiph.org/~greg/confidential_values.txt
14. Meiklejohn, S., Mercer, R.: Möbius: Trustless Tumbling for Transaction Privacy. In: Proceedings on Privacy Enhancing Technologies (2018)
15. Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., Christin, N.: An Empirical Analysis of Traceability in the Monero Blockchain. In: Proceedings on Privacy Enhancing Technologies (2018)
16. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System https://bitcoin.org/bitcoin.pdf
17. Ruffing, T., Moreno-Sanchez, P.: ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin. In: Financial Cryptography and Data Security (2017)
18. Ruffing, T., Moreno-Sanchez, P., Kate, A.: CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In: ESORICS (2014)
19. Ruffing, T., Moreno-Sanchez, P., Kate, A.: P2P Mixing and Unlinkable Bitcoin Transactions. In: NDSS (2017)
20. Saberhagen, N.v.: CryptoNote v 2.0 (2013), https://www.semanticscholar.org/paper/CryptoNote-v-2.0-Saberhagen/5bafdd891c1459ddfd22d71412d5365de723fb23
21. Sun, S.F., Au, M.H., Liu, J.K., Yuen, T.H.: RingCT 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero. In: Computer Security – ESORICS 2017 (2017)

22. Tairi, E., Moreno-Sanchez, P., Maffei, M.: A^2L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs. In: IEEE Symposium on Security and Privacy (SP) (2021)

A Security Definitions

Definition 4 (Hiding) *An additively homomorphic commitment protocol* HCom *is hiding if a commitment reveals no information about the committed message* m*. Formally, for any* PPT *adversary* $\mathcal{A}$*,*

$$\Pr\left[\mathcal{A}(c) = b \,\middle|\, \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{HCom.Gen}(\lambda); (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pp}); \\ b \leftarrow\$ \{0,1\}; c \leftarrow \mathsf{HCom.Com}(m_b; r_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

If the probability is exactly $1/2$*, the commitment scheme is perfectly hiding.*

Definition 5 (Binding) *An additively homomorphic commitment protocol* HCom *is binding if a commitment* c *can only be opened to one message. Formally, for all* PPT $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c} m_0 \neq m_1 \wedge \\ \mathsf{Com}(m_0; r_0) = \mathsf{Com}(m_1; r_1) \end{array} \,\middle|\, \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{HCom.Gen}(\lambda); \\ (m_0, r_0), (m_1, r_1) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

If the probability is exactly 0*, the commitment scheme is perfectly binding.*

Definition 6 (Completeness) *A non-interactive zero-knowledge proof protocol* NIZK *is complete if the prover who knows a witness of a statement could always convince the verifier to accept. Formally,*

$$\Pr\left[\mathsf{NIZK.Vf}(x, \pi) = 1 \vee (x, w) \notin \mathcal{R}_{\mathcal{L}} \,\middle|\, \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{NIZK.Gen}(1^\lambda); \\ \pi \leftarrow \mathsf{NIZK.Prove}(x, w) \end{array}\right] = 1.$$

Definition 7 (Soundness) *A non-interactive zero-knowledge proof protocol* NIZK *is sound if the prover cannot convince the verifier to accept a false statement. Formally,*

$$\Pr\left[\mathsf{NIZK.Vf}(x, \pi) = 1 \wedge x \notin \mathcal{L} \,\middle|\, \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{NIZK.Gen}(1^\lambda); \\ \pi \leftarrow \mathcal{A}(\mathsf{pp}, x) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 8 (Zero-Knowledge) *A non-interactive zero-knowledge proof 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* $\mathcal{S}$ *such that for all adversaries* $\mathcal{A}$*,*

$$\left|\Pr\left[\begin{array}{c} \mathcal{A}(\mathsf{pp}, x, \pi) \\ = 1 \end{array} \,\middle|\, \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{NIZK.Gen}(1^\lambda); \\ \pi \leftarrow \mathsf{NIZK.Prove}(x, w) \end{array}\right] - \Pr\left[\begin{array}{c} \mathcal{A}(\mathsf{pp}, x, \pi) \\ = 1 \end{array} \,\middle|\, (\mathsf{pp}, x, \pi) \leftarrow \mathcal{S}(x)\right]\right| \leq \mathsf{negl}(\lambda).$$

B Security Proofs

We denote the number of sender-receiver pairs in one epoch by n, the honest user list by T_h, and the corrupted user list by T_c. We formally describe the attack behaviors as the adversary's queries to oracles implemented by a challenger $\mathcal{C}$. The oracle $\mathcal{O}$ available to the adversary may be one of the followings.

- $\mathsf{O}^T_{\text{reg}}$: $\mathcal{A}$ queries this oracle to register as the tumbler, and gets the public address.
- $\mathsf{O}^{H/C}_{\text{reg}}$: $\mathcal{A}$ queries the $\mathsf{O}^H_{\text{reg}}/\mathsf{O}^C_{\text{reg}}$ oracle to instruct an honest user/let an corrupted user pk to register in the mixing protocol. $\mathcal{C}$ keeps track of this type of queries by maintaining a T_h/T_c for honest (corrupted) users.
- $\mathsf{O}^{H/C}_{\text{esc}}$: $\mathcal{A}$ queries the $\mathsf{O}^H_{\text{esc}}/\mathsf{O}^C_{\text{esc}}$ oracle to instruct an/a honest/corrupted user $\mathsf{pk} \in T_h/\mathsf{pk} \in T_c$ to escrow. $\mathcal{C}$ keeps track of this type of queries by maintaining a $\mathbf{EPool}_h/\mathbf{EPool}_c$ for honest users.
- $\mathsf{O}^{H/C}_{\text{red}}$: $\mathcal{A}$ queries the $\mathsf{O}^H_{\text{red}}/\mathsf{O}^C_{\text{red}}$ oracle to instruct an/a honest/corrupted user $\mathsf{pk} \in T_h/\mathsf{pk} \in T_c$ to redeem. $\mathcal{C}$ keeps track of this type of queries by maintaining a $\mathbf{RPool}_h/\mathbf{RPool}_c$ for honest users.

Proof of Lemma 1.

Proof. (1) If a PPT $\mathcal{A}$ wants to redeem a non-corrupted user's escrow $(\mathsf{ctx}^*_{\text{esc}}, \mathsf{token}^*)$ in $\mathsf{state}.\mathbf{EPool}$, $\mathcal{A}$ must generate $(\mathsf{prectx}^*, \pi^*)$. However, for the given $(c^*_e, \mathsf{token}^*)$ parsed from $(\mathsf{ctx}^*_{\text{esc}}, \mathsf{token}^*)$, c^*_r is unique. That is because $c^*_r = c^*_e \oplus_C \mathsf{HCom.Com}(0; \mathsf{td}^*)$, and $\mathsf{token}^* = \mathsf{OWP}(\mathsf{td}^*)$. Thus, based on the one-to-one property of OWP, c^*_r in prectx^* is determined for the given $(\mathsf{ctx}^*_{\text{esc}}, \mathsf{token}^*)$. Since a PPT $\mathcal{A}$ cannot invert $\mathsf{token}^* = \mathsf{OWP}(\mathsf{td}^*)$ to get td^*, $\mathcal{A}$ can generate neither the statement c^*_r nor the corresponding proof π^*.

(2) Else if the PPT $\mathcal{A}$ can generate a valid pre-redeeming request $(\mathsf{prectx}^*, \pi^*)$ with no existing corresponding escrow transaction $(c^*_e, \mathsf{token}^*)$ in $\mathsf{state}.\mathbf{EPool}$. It obviously violates the soundness property of the used NIZK.

In conclusion, no PPT adversary has a non-negligible advantage in the theft prevention game.

Proof of Lemma 2.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ who can win this game with non-negligible advantages. Same as before, the soundness of the NIZK means $(\mathsf{ctx}^*_{\text{esc}}, \mathsf{token}^*) \in \mathsf{state}$. This means $\mathcal{A}$ generates a prectx^* with a confidential value c^*_r that can be opened to a different value of c^*_e in $\mathsf{ctx}^*_{\text{esc}}$. Recall $\mathcal{L}_{\text{legal}}$ requires $c^*_r = c^*_e \oplus_C \mathsf{HCom.Com}(0; \mathsf{td}^*) = \mathsf{HCom.Com}(v^*; r^*) \oplus_C \mathsf{HCom.Com}(0; \mathsf{td}^*) = \mathsf{HCom.Com}(v^*; r^* \oplus_R \mathsf{td}^*)$. If c^*_r can be opened with $(v^{*\prime}, r^{*\prime}) \neq (v^*, r^* \oplus_R \mathsf{td}^*)$ and π^* can pass the validity check, this means $\mathcal{A}$ can break the binding property of the HCom with non-negligible probability.

Proof of Lemma 3.

Proof. From the soundness of our NIZK, the successful redeeming of $(\mathsf{ctx}^*_{\mathrm{red}}, \pi^*)$ means that there must be a corresponding record $(\mathsf{ctx}^*_{\mathrm{esc}}, \mathsf{token}^*)$ in $\mathsf{state}.\mathbf{EPool}$. Since we use $\mathsf{state}.\mathbf{RPool}$ to ensure the uniqueness of each successful redeeming, if c^*_r and $\hat{c_r}$ (contained in the pre-redeeming transactions of $\mathsf{ctx}^*_{\mathrm{red}}$ and $\mathsf{ctx}^{\wedge}_{\mathrm{red}}$ respectively) can both pass $\mathsf{VerifyRedeem}$, they must be different.

However, c^*_r in prectx^* is determined for a given $(\mathsf{ctx}^*_{\mathrm{esc}}, \mathsf{token}^*)$. Based on the one-to-one property of OWP, no $\mathcal{A}$ can redeem $(\mathsf{ctx}^*_{\mathrm{esc}}, \mathsf{token}^*)$ more than once.

Proof of Lemma 4.

Proof. We proceed via a sequence of games. Let $\Pr[W_i]$ be the probability that $\mathcal{A}$ wins in Game_i. Let $\overset{c}{\approx}$ denote that two distributions are computationally indistinguishable.

$\mathsf{Game}_0(\lambda)$**:** This game is as in the experiment with $b = 0$. Denote the confidential amount and the proof contained in $(\mathsf{prectx}^*, \pi^*)$ as $({c_r}^*_0, \pi^*_0)$.

$\mathsf{Game}_1(\lambda)$**:** This game is the same as Game_0, except the proof π^*_0 in $\mathsf{RequestRedeem}$ is now replaced by the output π'_0 from a simulator $\mathcal{S}$ of NIZK instead of generated by $\mathsf{NIZK.Prove}$. By a direct reduction to the zero-knowledge property of the NIZK, we have $({c_r}^*_0, \pi^*_0) \overset{c}{\approx} ({c_r}^*_0, \pi'_0)$. That is, $|\Pr[W_1] - \Pr[W_0]| \le \mathsf{negl}(\lambda)$.

$\mathsf{Game}_2(\lambda)$**:** This game is the same as Game_1, except that $b = 1$ is used to generate prectx^*. Since $(\mathsf{ctx}_{\mathrm{esc}1}, \mathsf{token}_1)$ is in the same $\mathsf{state}.\mathbf{EPool}$ as $(\mathsf{ctx}_{\mathrm{esc}0}, \mathsf{token}_0)$, from the zero-knowledge property of NIZK, there is a simulator $\mathcal{S}$ that can generate the simulated proof π'_1 for ${c_r}^*_1$. Then distinguishing Game_1 and Game_2 reduces to the hiding property of the underling homomorphic commitment. So we have $({c_r}^*_0, \pi'_0) \overset{c}{\approx} ({c_r}^*_1, \pi'_1)$. Thus, $|\Pr[W_2] - \Pr[W_1]| \le \mathsf{negl}(\lambda)$.

$\mathsf{Game}_3(\lambda)$**:** This game is the same as Game_2, except that π^*_1 is used instead of the simulated proof π'_1. Because of the zero-knowledge of NIZK, we have $({c_r}^*_1, \pi'_1) \overset{c}{\approx} ({c_r}^*_1, \pi^*_1)$. That is $|\Pr[W_3] - \Pr[W_2]| \le \mathsf{negl}(\lambda)$.

From above, we conclude $({c_r}^*_0, \pi^*_0) \overset{c}{\approx} ({c_r}^*_1, \pi^*_1)$, and $\left|\Pr[b' = b] - \frac{1}{2}\right| < \mathsf{negl}(\lambda)$.

Proof of Lemma 5.

Proof. We proceed via three claims.

Claim 5.1. The $\mathsf{CreateEscrow}$ is confidential.

This conclusion is obvious, i.e., $(c^*_{e0}, \mathsf{token}^*_0) \overset{c}{\approx} (c^*_{e1}, \mathsf{token}^*_1)$. Because $c^*_{e0} \overset{c}{\approx} c^*_{e1}$ follows from the confidentiality of the underlying blockchain, and token^*_b is generated independently of c^*_{eb} by a OWP.

Claim 5.2. The $\mathsf{RequestRedeem}$ is confidential.

Let (c^*_{rb}, π^*_b) be the redeeming request of $(c^*_{eb}, \mathsf{token}^*_b)$. Then there is a random $r'' \leftarrow\$\, \mathcal{R}$, s.t. $c^*_{rb} = c^*_{eb} \oplus_c \mathsf{HCom.Com}(0; \mathsf{td}^*_b) = \mathsf{HCom.Com}(v_b; r'')$, since $\mathsf{td}^*_b \leftarrow\$\, \mathcal{R}$. Based on the confidentiality of the underlying blockchain, $c^*_{r0} \overset{c}{\approx} c^*_{r1}$. By the zero-knowledge property of NIZK, we have $(c^*_{r0}, \pi^*_0) \overset{c}{\approx} (c^*_{r0}, \pi'_0) \overset{c}{\approx} (c^*_{r1}, \pi'_1) \overset{c}{\approx} (c^*_{r1}, \pi^*_1)$, where π'_b is the simulated proof for c^*_{rb}.

Claim 5.3. No PPT adversary can link the output of a $\mathsf{CreateEscrow}$ to a $\mathsf{RequestRedeem}$.

This comes directly from the unlinkability of MixCT.