Xie, H.; Fei, S.; Yan, Z.; Xiao, Y.

SofitMix: A Secure Offchain-Supported Bitcoin-Compatible Mixing Protocol

Published in:
IEEE Transactions on Dependable and Secure Computing

DOI:
10.1109/TDSC.2022.3213824

Published: 01/09/2023

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

SofitMix: A Secure Offchain-Supported Bitcoin-Compatible Mixing Protocol

Haomeng Xie, Shufan Fei, Zheng Yan*, *Senior Member, IEEE,* and Yang Xiao, *Graduate Student Member, IEEE,*

Abstract—Privacy preservation is highly expected in the Bitcoin Network. However, only applying pseudonyms cannot completely ensure anonymity/unlinkability between payers and payees. Current approaches mainly depend on a mixer service, which obfuscates payer-payee relationships of transactions. While the mixer service improves transaction privacy, it still suffers from some severe security threats (e.g., DoS attack and collusion attack), and does not support effective and reliable off-chain payment in a parallel mode. In this paper, we propose a mixing protocol for the Bitcoin Network based on zero-knowledge proof, called SofitMix. It is the first mixing protocol that can effectively resist both the DoS attack and the collusion attack. It can also support a set of parallel off-chain payments in a reliable way no matter whether some payers abort a transaction. We analyze and prove SofitMix security following the Universal Composability model with regard to fair exchange, unlinkability, collusion-resistance, DoS-resistance and Sybil-resistance. Through a proof-of-concept implementation, we demonstrate its validity and fairness. We also show its advance on off-chain payment reliability and DoS attack resistance, compared to TumbleBit.

Index Terms—Secure Mixer, Blockchain, Bitcoin, Zero-Knowledge Proof, Anonymity.

1 Introduction

Bitcoin, a digital currency platform created by Nakamoto in 2008 [1], has profoundly shaped the financial industry and our society. Behind Bitcoin's huge popularity [2], [3], [4], anonymity is one of its most appealing features to preserve transaction privacy. Bitcoin's anonymity mainly relies on pseudonyms to theoretically cuts off the connection between transactions and the real identities of involved parties. Moreover, payers and payees can change their pseudonyms from time to time to disguise their identities.

However, an increasing amount of research [5], [6], [7] has shown that the actual anonymity of Bitcoin is much weaker than our thoughts. The blockchain of Bitcoin is a public ledger storing all transactions that indicate the movement of funds from payers to payees, which could disclose the linkage between their pseudonyms. Once a pseudonym is linked to the real identity of a party, all related pseudonyms and transactions can be linked to that party, leading to collapse of anonymity.

To improve the anonymity of cryptocurrency, some types of new anonymous cryptocurrency (e.g. Zerocash [8] and Monero [9]) were proposed and achieved completely unlinkability. However, they are not compatible with Bitcoin and have a risk of liquidity squeeze. Optionally, an anonymity-enhancing service which is compatible with Bitcoin called mixer has emerged. In a nutshell, a mixer is a party in the Bitcoin Network that is responsible for receiving bitcoins (BTCs) from a set of payers and then obfuscating the relationships between transaction parties before re-sending BTCs to final payees. In this way, Bitcoin analysts cannot infer the linkage between a payee and an actual payer from a series of transactions in the public Bitcoin blockchain.

While the introduction of the mixer into Bitcoin sheds light on anonymity and unlinkability solutions, its applications in practice still confront several security and performance challenges that have not yet been well addressed.

First, existing mixer services [10], [11], [12], [13], [14], [15], [16] suffer from DoS attacks. As we know, the process of recording a transaction on the Bitcoin blockchain needs to consume a transaction fee (F_{tran}) to reward miners. However, some existing schemes require the mixer to first escrow α BTCs on the blockchain for a specific period (Δt) before mixing a payment. Thus, a powerful well-funded attacker (i.e., a payer/payee) can repetitively join and then intentionally abort a mixing protocol at a cost of insignificant F_{tran}, which could lock the mixer's limited amount of BTCs for a long time and cause a specific DoS attack.

Second, collusion attack, a situation when a malicious party (payer/payee) colludes with the mixer to defraud its counterparty (payee/payer), has not been well resisted. Although the state-of-the-art protocol, TumbleBit [10], carefully discusses this problem and offers a partial solution, it does not consider such a problem that a malicious payer may lie to a payee that it has sent the payee a token that can be used to claim α BTCs from the mixer, so that the payee cannot acquire its BTCs eventually. Vice versa, a malicious payee could lie to the payer that it never received the token and require the payer to pay it again. Although

- *Z. Yan (corresponding author) is with the State Key Lab of ISN, School of Cyber Engineering, Xidian University, Xi'an, China, and the Department of Communications and Networking, Aalto University, Espoo, Finland. E-mail: zyan@xidian.edu.cn; zheng.yan@aalto.fi.*
- *H. Xie is with the State Key Lab of ISN, School of Cyber Engineering, Xidian University, Xi'an, China. E-mail: haomengxie@foxmail.com.*
- *S. Fei is with the State Key Lab of ISN, School of Cyber Engineering, Xidian University, Xi'an, China. E-mail: shufanfei@gmail.com.*
- *Y. Xiao is with the Department of Computer Science, Virginia Polytechnic Institute and State University, VA, USA. E-mail: xiaoy@vt.edu.*

the mixer can be sued when it colludes with other parties, victims cannot provide any evidence to verify the above described malicious behaviors. Thus, it is difficult to ensure fair exchange when honest parties suffer from a collusion attack.

Third, current state-of-the-art solutions of Bitcoin transaction privacy typically yield cumbersome on-chain transaction overhead. We note that an established practice to solve this scalability issue is off-chain payment [17], [18] that utilizes limited transactions recorded on the blockchain to replace a vast amount of payments between two parties. However, previous mixing protocols [11], [12], [13], [14], [15], [16], [19] only support a classic model in which all transactions are recorded on-chain in an epoch. Although TumbleBit [10] states that it can support an off-chain model in which payers can process multiple payments in an epoch in an off-chain fashion, we found that off-chain payments of payees have a strong coupling relation. It can only be executed sequentially, and the execution of subsequent offchain payments depends on the success of previous offchain payments. Thus, payment efficiency cannot be effectively guaranteed in practice. In addition, the payee may fail to acquire all BTCs if one of the payers aborts the protocol in a parallel execution mode. Therefore, a highly reliable mixing protocol that can support parallel off-chain payments is needed.

Based on the above discussion, we can see that the current Bitcoin Network highly expects a secure and reliable mixing protocol that can resist both DoS and collusion attacks, stably support parallel off-chain payments and offer fair exchange to all involved parties including the mixer.

In this paper, we present a novel Secure off-chain-enabled bitcoin-compatible Mixing protocol, named SofitMix. SofitMix re-constructs the order of transactions to mitigate the DoS attack by tactfully utilizing the hash-time-lock transaction. It applies signatures and transactions recorded on the blockchain to effectively prevent the collusion attack. At the same time, it employs the multiple-input and multiple-output (MIMO) transaction of Bitcoin to decouple off-chain payments, which then can be operated in parallel and efficiently. Different from the puzzle-promise protocol used in TumbleBit, SofitMix employs zero-knowledge proofs to guarantee unlinkability. To the best of our knowledge, SofitMix is the first Bitcoin mixing protocol that addresses all the aforementioned challenges. SofitMix operates in epochs (i.e., mixing periods) and realizes transaction anonymity with zero-knowledge proofs. After starting a new epoch, a payer can join in the SofitMix by firstly escrowing enough BTCs in an escrow transaction recorded on-chain. In the above process, the payer needs to pay a F_{tran} to miners following the transaction rules in the Bitcoin network. When the payer makes a payment to the mixer, it also generates a token and provides the corresponding token evidence to a target payee. Correspondingly, the mixer will escrow an equal value of BTCs on the blockchain. These BTCs should be paid to the payee, once the payee can prove that it knows the token generated by the payer without disclosing its real identity through a zero-knowledge proof. In case that the mixer does not transfer the payment to the payee as the payer's expectation, the payer can cancel its payment by revealing the token to reclaim its BTCs. If the payer reclaims its BTCs after the mixer transferred BTCs to the payee, the mixer can also utilize the revealed token to reclaim equal BTCs at once, which avoids its BTCs to be locked for a long time. In one epoch, SofitMix allows the mixer to process a large amount of payments between a payer and a payee with only four transactions recorded on-chain, thus significantly improves its throughput and releases congestion of the Bitcoin network.

Specifically, SofitMix is designed to resist both the DoS attack and the collusion attack. Since the payer is required to pay a F_{tran} to Bitcoin miners to join in the protocol, malicious payers who attempt to mount a DoS attack need to lock their BTCs for time Δt and pay an extra F_{tran}, which degrades the incentive to raise the DoS attack. In addition, if a party colludes with the mixer to defraud its counterparty, evidence (i.e., the token, transactions on-chain and signed messages during SofitMix protocol execution) can be used to disclose malicious parties, which discourages malicious collusion behaviors in SofitMix. We declare that SofitMix is not contradictory to using mixing to protect privacy. Because the signed messages are only kept by the payer and the payee. Anyone except for the payer and the payee cannot disclose the linkage and the real identities of theirs. Thus, SofitMix can be used in such a situation that two parties who know but do not fully trust with each other intend to make privacy-preserving payments with BTCs. Besides, SofitMix not only supports parallel off-chain payments, but also uses a multiple input and multiple output (MIMO) mechanism to separate payments from each other and allows involved parties to make independent decisions on payments. Even some payers abort the protocol, the failure of their payments has no effect on payees' redemption of other related payments. Therefore, SofitMix offers high reliability in off-chain payments.

In summary, this paper has the following contributions:

- We propose SofitMix, the first Bitcoin mixing protocol that can resist DoS attack and support fair exchange when suffering from collusion attacks. It can also reliably support parallel off-chain payments even if some payers abort the SofitMix protocol.
- We perform both security analysis and formal proof under the Universal Composability (UC) model [22] to show its security and privacy in terms of fair exchange, unlinkability, and resistance on collusion, DoS attacks, and Sybil attacks.
- We implement SofitMix in the Bitcoin network. Performance evaluation shows that its transaction size is smaller than that of TumbleBit [10]. In addition, we demonstrate its validity and fairness and show its advance on parallel off-chain payment reliability and DoS resistance, compared also with TumbleBit.

The rest of this paper is organized as follows. Section 2 provides background knowledge and reviews related work. Section 3 introduces the system model and security model of SofitMix, as well as specifies its design goals. In Section 4, we describe SofitMix in detail. Section 5 analyzes the security and privacy of SofitMix proves its security based on the Universal Composability model, followed by SofitMix implementation and performance evaluation results in Section 6. Finally, we conclude our paper in the last section.

TABLE 1: A Comparison of Bitcoin Mixing protocols

Protocol	Fair Exchange	Anonymity	DoS Attack Resistance[A]	Sybil Attack Resistance	Bitcoin Compatibility	Supported Models[C]	Parallel Off-chain Payments
CoinSwap [11]	✗	✗	F_{tran}	✓	✓	CM	✗
MixCoin [19]	✗	✗	∞	✓	✓	CM	✗
Blindcoin [20]	✗	✓	∞	✓	✓	CM	✗
BSC [12]	✗	✓	F_{tran}	✓	✗	CM	✗
TumbleBit [10]	✗	✓	F_{tran}	✓	✓	CM & OM	✗
CoinJoin [13]	✓	✓	0	✗	✓	CM	✗
CoinShuffle [14]	✓	✓	0	✗	✓	CM	✗
CoinShuffle++ [15]	✓	✓	0	✗	✓	CM	✗
Coinparty [21]	✗	✓	∞	✓	✓	CM	✗
Xim [16]	✓	✓	F_{tran}	✓	✓	CM	✗
SofitMix	✓	✓	$n \cdot (F_{tran} + \alpha \cdot \Delta t)^B$	✓	✓	CM & OM	✓

[A] A larger value implies a stronger ability to resist the DoS attack.
[B] n is the number of payments included in a transaction.
[C] CM: Classic Model, OM: Off-chain Model.

2 Background and Related Work

2.1 Bitcoin Transactions

We first overview how Bitcoin works by introducing its transaction structure.

Transactions: Transactions in Bitcoin record the movement of funds from an original transaction T_{origin} to a fulfilling transaction $T_{fulfill}$. The redeem condition in the original transaction stipulates a condition that must be met in order to spend BTCs in this transaction. The fulfilling transaction provides script data that meets the redeem condition in the original transaction, which results in BTCs transformation from the original transaction to the fulfilling transaction. Each transaction can include multiple inputs and multiple outputs (MIMO). In the Bitcoin network, the funds in an output of the original transaction can only be transferred to an input of the fulfilling transaction. Next, we review common redeem conditions used in current Bitcoin transactions.

Hashlock Condition ($h(a)$): Hashlock condition $h(a)$ in the original transaction stipulates that the fulfilling transaction that intends to spend the BTCs in the original transaction needs to reveal the preimage of the hash ($h(a)$).

Timelock Condition (T): The BTCs in the original transaction with a timelock condition will be locked for a period of (T). The funds cannot be claimed until the time T expires.

Signing Condition (X): This condition stipulates that the fulfilling transaction needs to be signed by a specific user (X) with secret key (SK_X) under the Elliptic Curve Digital Signature Algorithm (ECDSA) over a *Secp256* elliptic curve.

2-of-2 Escrow Condition ($X \wedge Y$): This condition in the original transaction stipulates that a fulfilling transaction needs to be signed by two parties (X, Y) cooperatively. Actually, in case that one party is out of work and they cannot build a fulfilling transaction cooperatively, an original transaction with a 2-of-2 escrow condition (i.e., an escrow transaction) often has a timelock condition (T), with which the other party can reclaim all BTCs after time *T* elapses. We heavily rely on the transaction with this type of condition to construct a payment channel that is necessary for off-chain payments [17], [18] between two parties (*X*,*Y*). An off-chain payment usually proceeds in three phases: (1) Escrow Phase in which one party X opens a payment channel with the other party *Y* by escrowing enough BTCs in a 2-of-2 escrow transaction posted on the blockchain; (2) Payment Phase in which one party *X* pays the other party *Y* via off-chain transactions that are not recorded on-chain. Every time *X* intends to transmit a new payment to *Y*, *X* reconstructs and signs a new off-chain transaction with a value that is the sum of all previously generated payments and the underlying payment. *Y* keeps these off-chain transactions locally and does not post them on-chain. We note that all off-chain transactions kept by *Y* are all fulfilling transactions of the escrow transaction, and only one off-chain transaction can be confirmed on blockchain eventually; (3) Decision Phase: before time *T* elapses, *X* and *Y* can cooperatively sign and post a new fulfilling transaction that re-allocates the funds in the escrow transaction, or *Y* can independently sign and post the last off-chain transaction from which *Y* can get the last specified amount of BTCs. In addition, *X* can reclaim all BTCs from the escrow transaction after time *T* elapses. In this way, performing two on-chain transactions can process a large number of payments between two parties in a payment channel, which dramatically improves payment velocity and volume in the Bitcoin Network.

Hash-time-lock Condition $(h(a) \vee T)$: A transaction with this condition includes a hashlock condition $h(a)$ and a timelock condition T, which stipulates one party can claim the payment prior to the deadline by revealing the preimage of the hash $h(a)$, or forgive the funds and return them to the other party after the time T expires.

2.2 Mixer Services in Bitcoin

In this subsection, we review recent centralized and decentralized mixer services in Bitcoin. We overlook the zero-knowledge contingent payment protocol [23] and the payment channel network [24] because they focus on different payment scenarios.

Mixer was first introduced in 1981 [25] and originally applied in anonymous communications. In recent years, it has been integrated into blockchain to mitigate de-anonymity attacks [26].

Maxwell first attempted to introduce the mixer into Bitcoin and proposed a centralized mixing protocol called CoinSwap [11]. However, a curious mixer can disclose the

linkage between payers and payees. Mixcoin [19], proposed by Bonneau, cannot completely restrict the mixer from stealing BTCs. An improved mixing protocol called Blindcoin [20] was then proposed by Valenta et al. to solve the unlinkability problem against a curious mixer. However, like Mixcoin, the mixer's misbehaviors are only auditable. Blindly Signed Contracts (BSC) [12] provides anonymity against the mixer and can completely prevent the mixer from stealing BTCs. Unfortunately, this protocol is not Bitcoin-compatible. TumbleBit [10] is a notable mixing protocol that achieves Bitcoin compatibility and unlinkability. Although it was claimed that Tumblebit can support fair exchange when suffering from collusion attack, we note that an honest party has no evidence to prove malicious behaviors of its counterparty.

Maxwell developed a prototype of a decentralized mixer in Bitcoin called CoinJoin [13]. To achieve unlinkability, CoinShuffle [14] and its variant CoinShuffle++ [15] were proposed. These decentralized mixing protocols perform mixing in a single transaction. The transaction size of Bitcoins restricts the number of users to 538 per mix. In addition, they are not secure enough to resist DoS attacks and Sybil attacks. These threats were considered in Coinparty [21]. However, it cannot support fair exchange when mixers collude. Xim [16] is a totally different protocol that can resist the DoS attack. However, it takes a long waiting time in a round of mixing. Although decentralized mixer can use one on-chain transaction to complete Bitcoin mixing for n users, it has a inherent disadvantage that all payments have strong coupling relationship. If one party aborts the protocol, then all parties cannot complete the mixing in this epoch.

In Table 1, we compare the related works with SofitMix in terms of fair exchange, anonymity, DoS attack resistance, Sybil attack resistance, Bitcoin compatibility, supported models, parallel off-chain payments. We can see that SofitMix offers more advanced functionalities than other works. It is a bitcoin-compatible mixing protocol. It not only addresses the anonymity problem, but also supports parallel off-chain payments and fair exchange even confronting collusion attack. It can also effectively mitigate the DoS attack and resist the Sybil attack.

3 Problem Statement

3.1 System Model

There are three types of parties involved in SofitMix, named mixer, payer and payee. The mixer is responsible for transferring BTCs from payers to payees and is compensated with mixing fees (F_{mix}). Each party involved in this protocol has a pair of long-term keys (PK_p,SK_p) that are related to its real identity. As a for-profit party represented by PK_M, each mixer is motivated to maintain its reputation. For payers and payees, their long-term keys are independent from the Bitcoin's key-pair system. These long-term keys are only used for off-chain payment signing processes as evidence to deal with the collusion attack. We assume that payees always choose fresh and ephemeral addresses to receive payments in each epoch for enhancing their privacy. We further assume that multiple mixers $\{M_i\}$ could exist in the system and each mixer is equipped with multiple backup servers in order to avoid single point failure.

3.2 Security Model

We first assume that all parties are rational, selfish and potentially malicious. They may attempt to steal BTCs and deviate from the mixing protocol in order to pursue their biggest profits. In addition, the mixer may de-anonymize a target transaction, or attempt to launch Sybil attack by forging identities of transaction parties to reduce the proportion of real parties in an anonymity set and lower the anonymity level during a mixing round. Payers or payees may mount DoS attacks by intentionally aborting the SofitMix protocol to break the capability of the mixer. Either payers or payees may collude with the mixer to defraud its counterparty to steal BTCs.

In our research, we assume that a payer and a payee communicate with each other via a secure and anonymous channel. The mixer can gain neither communication information between other parties nor their network information (IP addresses) with the help of anonymous communication techniques (e.g., Tor anonymity networks). In reality, this anonymous communication technique can be replaced by reliable encrypted communication software (e.g., Telegram). We further assume that as a for-profit party, the mixer would not intentionally abort transactions to decrease the anonymity level since such a behavior would undermine its reputation.

3.3 Design Goals

The design goals of SofitMix with regard to security and privacy are listed below:

Unlinkability: A mixing protocol holds unlinkability if no one rather than the pair of payer and payee can tell which payer paid to which payee in an epoch. Note that we assume the mixer does not collude with any party when considering this property.

Fair Exchange: A mixing protocol holds fair exchange if and only if 1. the protocol either ensure that all honest parties receive BTCs as expected (i.e. stipulated by the payment phase), or release no valid payment requests; 2. the protocol terminates after a fixed time.

DoS Attack Resistance: A mixing protocol is DoS attack resistance if all parties have an effective method to prevent attackers from intentionally aborting the protocol to lock their limited amount of BTCs for a long time.

Sybil Attack Resistance: A mixing protocol is Sybil attack resistance if the protocol can effectively prevent a malicious party from creating many different pseudonyms to shrink the proportion of real parties in the anonymity set and de-anonymize a target party.

4 SofitMix Design

In this section, we first introduce the preliminary and Notations used in SofitMix. Then, we overview SofitMix by summarizing its key ideas, followed by the details of SofitMix design.

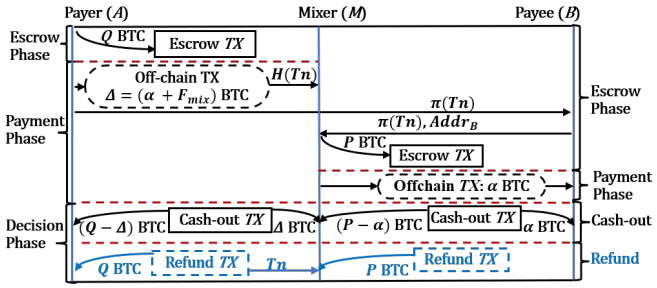
4.1 Preliminary and Notations

Non-Interactive Zero-Knowledge Proof (NIZK): Let $R \subseteq \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ be an NP relation that consists a pair of the form (x, w), where w is a witness (private

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3213824

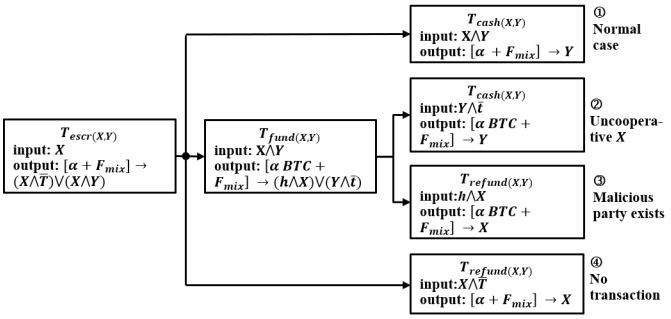
TABLE 2: The Description of Notations

Notation	Description
$T^X_{fulfill(X,Y)}$	a fulfilling transaction between X and Y signed by X
$T_{escrow(X,Y)}$	an escrow transaction between X and Y
$T_{fund(X,Y)}$	a funding transaction with a hash-time-lock redeem condition that performs the 2-of-2 escrow condition of $T_{escrow(X,Y)}$
$T_{refund\rightarrow escrow(X,Y)}$	a refund transaction that performs the timelock condition of $T_{escrow(X,Y)}$ to refund BTCs to X
$T_{refund\rightarrow fund(X,Y)}$	a refund transaction that performs the hash condition of $T_{fund(X,Y)}$ to refund BTCs to X
$T_{cash\rightarrow escrow(X,Y)}$	a cash-out transaction that performs the 2-of-2 escrow condition of $T_{escrow(X,Y)}$ to redeem BTCs to Y
$T_{cash\rightarrow fund(X,Y)}$	a cash-out transaction that performs the timelock condition of $T_{fund(X,Y)}$ to redeem BTCs to Y



* Note that the escrow phase of payee (B) partially overlaps with the payment phase of payer (A) because the mixer (M) should construct a payment channel by posting an escrow transaction after A has generated valid payments and locked BTCs in the blockchain to resist the DoS attack.
* Decision phase includes cash-out phase or refund phase.

Fig. 1: Protocol Overview



*$(X, Y, \bar{T}, \bar{t}, h)$ represents (A, M, T_1, t_1, h_1) in the payment channel $C(A, M)$, and represents (M, B, T_2, t_2, h_2) in the payment channel $C(M, B)$, in which $T_1 < t_1 < T_2 < t_2$, $h_1 = H(a)$, $h_2 = H(a \oplus b)$.
*There is no F_{mix} in the payment channel $C(M, B)$.
*If all parties process correctly, M and B can obtain BTCs through the path ①. If A or M is unwilling to cooperate to close the payment channel, M or B can obtain BTCs through the path ②. If there is no payment after creating the payment channel $C(A, M)$ or $C(M, B)$, A or M can reclaim all BTCs through the path ③ after time T_1 or T_2. If M or B aborts the protocol after A generates a payment, A can reclaim its BTCs through the path ④. However, if A maliciously reclaims its BTCs after M generates a payment, M can immediately reclaim its BTCs through the path ④ in the same way.

Fig. 2: Cases of Possible Transactions

input) and x is a statement, (e.g., $R = \{(x, w)|x = SHA256(w)\}$). Let L be a set of positive instances of R (i.e., $L = \{x|\exists w, s.t.\ R(x, w) = 1\}$). The NIZK lets a prover $\mathcal{P}$ take the (x, w) as input and generate a single message π. A verifier $\mathcal{V}$ can be convinced that $R(x, w) = 1$ without revealing the witness w if π is correct. We declare the NIZK has the following properties:

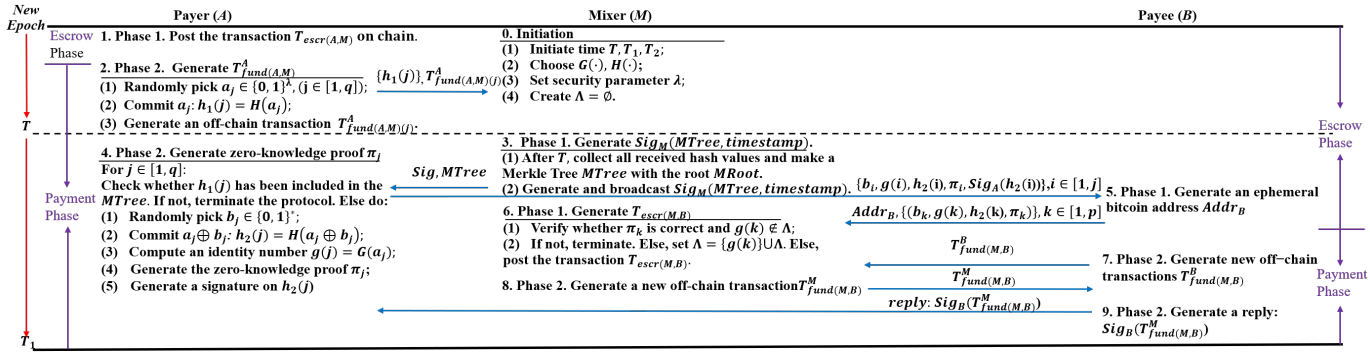
- **Completeness:** The verifier always accepts an honestly-computed proof π for a statement $x \in L$;
- **Soundness:** The verifier always rejects a proof π for a statement $x \notin L$, only except with a negligible probability;
- **Zero-Knowledge:** A correct proof π cannot reveal any information about the secret w.

Besides, for ease of understanding, we describe the notations of all types of transactions used in the SofitMix in Table 2.

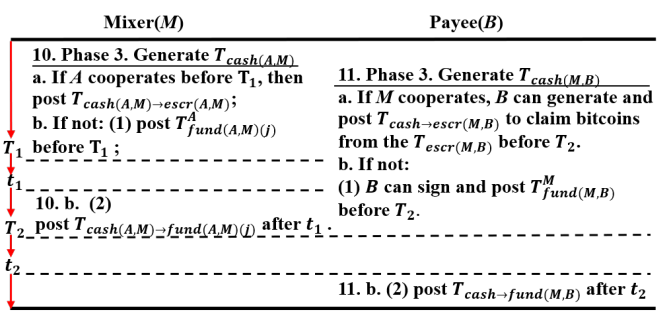
4.2 SofitMix Overview

We provide an overview of the SofitMix protocol with an assumption that every payment has the same value of α BTCs. At its core, SofitMix utilizes zero-knowledge proof π to convince the mixer that a payer intends to pay the proof holder (a payee) α BTCs and has already paid the mixer the same value of BTCs. The mixer will pay the payee α BTCs after receiving such a zero-knowledge proof π from the payee.

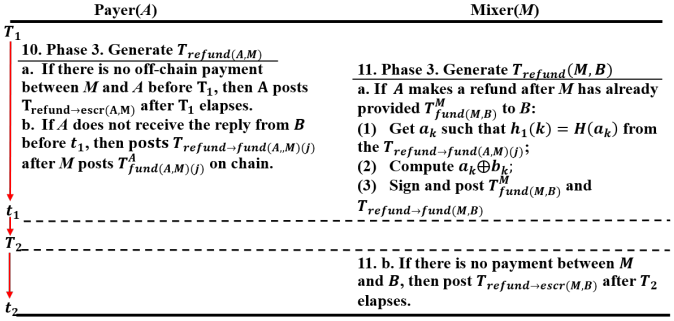
Fig. 1 shows the process of making payments between a payer A and a payee B through a mixer M. The SofitMix protocol executes in three phases in an epoch. In the **Escrow Phase**, a number of payers and the mixer escrow BTCs on the Bitcoin blockchain. A payer A first opens a payment channel with the mixer M by generating an on-chain original transaction that escrows Q BTCs. The mixer also opens a payment channel with the corresponding payee B and escrows P BTCs after the corresponding payee B provides an ephemeral address $Addr_B$ and p zero-knowledge proofs π to it, where $P = p \times \alpha$. In the **Payment Phase**, the payer/mixer can make off-chain payments to the mixer/payee. The payer can transfer up to q off-chain payments ($q = 1$ in the classic model and $q \geq 1$ in the off-chain model) to the mixer and the value for each payment is $\alpha + F_{mix}$ BTCs ($q \times (\alpha + F_{mix}) \leq Q$). Following each above payment, the payer A transfers an encrypted token $H(Tn)$ to the mixer M. Meanwhile, the payer A provides the payee B a zero-knowledge proof $\pi(Tn)$ on Tn, which can be used to convince the mixer that it knows such a token but does not indicate which one. The mixer M will provide the payee B α BTCs after receiving such a proof from the payee B. In the **Decision Phase**, all involved parties claim BTCs. This phase includes two sub-phases named cash-out phase



(a) Escrow Phase and Payment Phase



(b) Cash-out Phase



(c) Refund Phase

Fig. 3: The Detailed SofitMix Protocol

and refund phase, respectively. If every party processes normally, the cash-out phase will be activated. The mixer closes the payment channel and claims $q \times (\alpha + F_{mix})$ BTCs from A, while the payee B also closes payment channels with M and claims $p \times \alpha$ BTCs with the help of p zero-knowledge proofs. However, if the payee B does not receive the payment from M within a specific period, the payer A will proceed the refund phase and reclaim BTCs by posting a refund transaction in which the token Tn is revealed to the mixer. If the payer maliciously reclaim BTCs, the mixer can reclaim its corresponding BTCs immediately by using the revealed token Tn.

4.3 The SofitMix Protocol

In this section, we describe the SofitMix protocol in detail. Fig. 2 show the cases of possible transactions involved in this protocol. In this figure, the input in a transaction denotes the script data and the output denotes the amount of funds with its redeem condition. An arrow points from an original transaction to a fulfilling transaction. Only when the input in the fulfilling transaction satisfies the redeem condition in its corresponding original transaction, funds in the original transaction can be transferred to the fulfilling transaction. For easy understanding, we ignore the transaction fee rewarding to a miner in the figure.

As shown in Fig. 3, the mixer initiates the system by setting a security parameter λ and time points T, T_1 and T_2 according to block height, in which $T < T_1 < T_2$. T is used to restrict the off-chain payment period between payers and the mixer. T_1 and T_2 are used to stipulate the period for which funds need to be locked in escrow transactions. Then, it creates a set Λ to record zero-knowledge proofs on the used payments. The mixer also stipulates two different hash functions $H(\cdot)$ and $G(\cdot)$, in which $H(\cdot)$ is collision-resistant. These parameters (T, T_1, T_2, $H(\cdot)$, $G(\cdot)$, λ) are open to the public. This step is only executed once during the setup of the mixer. And the set Λ will be reset when starting a new epoch.

Phase 1: Escrow. Each payer or payee that wants to transfer or receive BTCs via the mixer in the upcoming epoch constructs a 2-of-2 escrow transaction with the mixer in this phase. At the beginning of an epoch, a payer A first gets parameters T, T_1, $H(\cdot)$, $G(\cdot)$ and λ from the mixer's server and opens a payment channel with the mixer by depositing Q BTCs in a 2-of-2 escrow transaction $T_{escr(A,M)}$. Then A can make up to q payments in the following payment phase, in which $Q \geq q \times (\alpha + F_{mix})$ where $q = 1$ if the mixing model is classic and $q \geq 1$ if the off-chain model is adopted. Funds in this transaction can be redeemed by a fulfilling transaction signed by both A and M, or by A after time T_1 elapses. Similarly, a payee B gets parameters T, T_2, $H(\cdot)$, $G(\cdot)$ and λ from the mixer's server and requires the mixer to open a payment channel after receiving a zero-knowledge proof generated by a payer in the following payment phase. We note that M executes the escrow phase when A proceeds in the payment phase. For ease of understanding, we describe the process of constructing a payment channel between M and B in the following phase.

Phase 2: Payment. After the escrow transaction $T_{escr(A,M)}$ is confirmed on the Bitcoin blockchain, A can make payments to the mixer through off-chain transactions before time T expires, where $T < T_1$ because off-chain payments need to be executed during the period when the

Algorithm 1: $GProof$

Input: $G(\cdot), H(\cdot), a, h_1, MTree$
Output: π, b, g, h_2
Generate an authentication path $MPath$ for h_1 according to $MTree$;
Randomly pick $b \in \{0,1\}^\lambda$ and compute $h_2 = H(a \oplus b)$;
Compute $g = G(a)$;
Generate the proof $\pi \leftarrow \mathcal{P}(x = \{G(\cdot), H(\cdot), b, g, h_2, MRoot\}, w = \{a, MPath\})$;
Return π, h_2, g, b

channel is open. When A intends to make a new payment to the mixer, A produces a hash value $h_1(j) = H(a_j)$ on a new random value $a_j \in \{0,1\}^\lambda$, where $j \in [1, q]$ and $j-1$ is the number of payments it has generated. Here, a_j is the token Tn that has been described in the SofitMix overview. Then A constructs and signs a new off-chain fulfilling transaction named funding transaction $T^A_{fund(A,M)(j)}$ that points to the $T_{escr(A,M)}$ and includes j outputs. These outputs are corresponding off-chain payments sent by A, each of which includes $\alpha + F_{mix}$ BTCs and is stipulated by the hash-time-lock condition. Each output can be claimed by an independent fulfilling transaction signed by A with the preimage of hash $h_1(i)$ in which $i \in [1, j]$, or by M after time t_1. Then, A will transfer $T^A_{fund(A,M)(j)}$ and $h_1(j)$ to the mixer. Here we let $T_1 < t_1$, which reserves a deadline cushion for A to decide whether to reclaim these payments after the $T^A_{fund(A,M)(j)}$ posted on-chain in case M aborts the protocol. Otherwise, M can sign and post the $T^A_{fund(A,M)(j)}$ after t_1 elapses to gain BTCs from A but does not generate payments to B, which causes a property loss to A. After receiving $T^A_{fund(A,M)(j)}$, the mixer checks whether the transaction can be claimed by itself. If not, the mixer discards $h_1(j)$.

The mixer will stop accepting off-chain payments from the payers after time T elapses. Then, the mixer discloses all valid hash values received from the payers in this epoch and construct a Merkle Tree ($MTree$) with a root ($MRoot$). The mixer signs the $MTree$ as well as a $timestamp$ indicating the epoch number with its signing key SK_M. Then the mixer publishes the $MTree$ with the signature $Sig_M(MTree, timestamp)$ on all available public resources such as the Internet and blockchain (e.g., IPFS [27]). Then, all payers can access the $MTree$ and audit its correctness according to the transactions posted on-chain.

After receiving the $MTree$, A first checks whether all target hash values $h_1(i)$ ($i \in [1, j]$) have been included in the $MTree$. If not, A will terminate the protocol and revoke its BTCs after the $T^A_{fund(A,M)(j)}$ posted on-chain. Otherwise, A runs the $GProof$ algorithm to constructs j zero-knowledge proofs, and for each proof π_i, M can be convinced that:

1. A payer has already sent it a payment through $T^A_{fund(A,M)(j)}$;
2. The corresponding output of this payment in $T^A_{fund(A,M)(j)}$ has not been used by other parties;
3. After it promises to transfer α BTC to B, it can reclaim its BTCs immediately if the payer behaves maliciously to revoke this payment from $T^A_{fund(A,M)(j)}$.

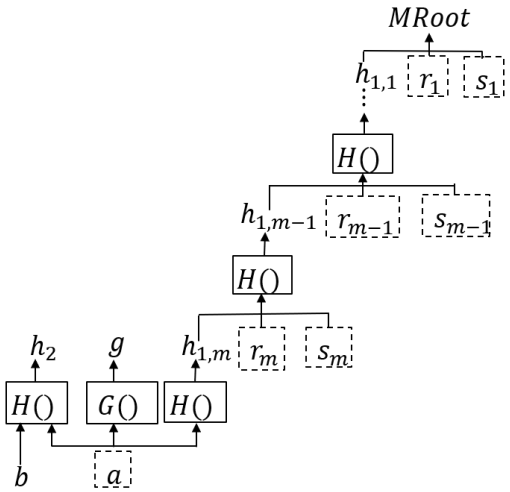


Fig. 4: The Structure of Zero Knowledge Proof

To generate such a proof, A first constructs an authentication path $MPath = \{(s_1, r_1), (s_2, r_2)...(s_m, r_m)\}$ for a hash value $h_1(i)$ according to the $MTree$ [28], in which s_l ($l \in [1, m]$) denotes a sibling of $h_1(i)$'s ancestor, $r_l \in \{'right', 'left'\}$ denotes the position of the sibling and $m = \log(n)$ denotes the depth of the $MTree$, where n is the number of payments in an epoch. Then, A picks a random value $b_i \in \{0,1\}^\lambda$ and computes the hash value $h_2(i) = H(a_i \oplus b_i)$ that will be used to construct an off-chain payment between M and B. A further computes an identification number of the payment sent to M, denoted as $g(i) = G(a_i)$. The mixer can use $g(i)$ to check whether the corresponding output of this payment in the $T^A_{fund(A,M)}$ has been used or spent by other parties. Finally, A will generate the zero-knowledge proof π_i for the following NP statements without revealing a_i and $h_1(i)$:

"Given b_i, hash value $h_2(i)$, the identification number $g(i)$ and the Merkle root $MRoot$, I know a value a_i such that:

1. $H(a_i)$ is a leaf of the $MTree$ with the root $MRoot$;
2. $g(i)$ is computed correctly: $g(i) = G(a_i)$;
3. $h_2(i)$ is well-formed: $h_2(i) = H(a_i \oplus b_i)$."

The structure of our zero knowledge proof scheme is described in Fig. 4, in which b is the public input, $(a, MPath)$ is the witness, and $(G(\cdot), H(\cdot), g, h_2, MRoot)$ is the output. Then the language L of our zero knowledge proof has the following expression, where the verification algorithm $VPath$ [28] takes a tuple $(MRoot, MPath, H(a))$ as input and outputs 1 denoting the $MPath$ is valid for the hash value h_1 according to the $MRoot$.

$$\begin{aligned} L = \{&G(\cdot), H(\cdot), b, g, h_2, MRoot) | \exists (a, MPath), \\ &s.t.\ h_2 = H(a \oplus b),\ g = G(a), \\ &VPath(MRoot, MPath, H(a)) = 1\} \end{aligned}$$

After generating the proof, A signs the hash value $h_2(i)$ using its signing key to construct an evidence $Sig_A(h_2(i))$ that denotes the corresponding payment generated by A.

Algorithm 2: $VProof$

Input: $\pi_k, g(k), \Lambda$
Output: 0 or 1
if π_k *is correct and* $g(k) \notin \Lambda$ **then**
  return 1;
else
  return 0;
end

Then, A sends the tuple $(b_i, g(i), h_2(i), \pi_i, Sig_A(h_2(i)))$ to B. We note that each payee receives only one proof in the classic model or more than one proofs in the off-chain model. After receiving p zero-knowledge proofs from A, B requires M to open a payment channel that escrows $p \times \alpha$ BTCs by sending an ephemeral address $Addr_B$ and all proofs $(b_k, g(k), h_2(k), \pi_k)$ in which $k \in [1, p]$ to M, where p is the number of payments the payee B receives in an epoch, and $p = 1$ if the classic model is applied and $p \geq 1$ if the off-chain model is executed. Note that these p zero-knowledge proofs could come from the same or different payers. After receiving the proofs from the payees, M runs the $VProof$ algorithm to check whether these proofs are valid. If for $k \in [1, p]$, it satisfies $VProof(\pi_k, g(k), \Lambda) = 1$, M opens a payment channel with B by depositing $p \times \alpha$ BTCs in a 2-of-2 escrow transaction $T_{escr(M,B)}$ timelocked by T_2. With the escrow transaction $T_{escr(M,B)}$, B can claim BTCs before T_2 with the assistance of M, which improves the efficiency of the SofitMix protocol. Then, M updates the set Λ by appending the element $g(k)$: $\Lambda = \Lambda \cup \{g(k)\}$.

To resist the DoS attack, B is required to sign and send an off-chain funding transaction $T^B_{fund(M,B_k)}$ to the mixer first. $T^B_{fund(M,B)}$ is a transaction with p outputs. Each output in the $T^B_{fund(M,B)}$ including α BTCs can be claimed by a fulfilling transaction signed by M with the preimage of hash $h_2(k)$, or by B after time t_2. If A maliciously revokes its BTCs back and reveals the preimage of hash $h_1(k) = H(a_k)$, M can compute the preimage of hash $h_2(k) = H(a_k \oplus b_k)$ and reclaim its BTCs from the $T^B_{fund(M,B)}$ immediately. After getting the $T^B_{fund(M,B)}$, M signs and transfers a similar funding transaction $T^M_{fund(M,B)}$ to B. Here we let $t_1 < T_2 < t_2$. We set $T_2 < t_2$ due to the similar reason to $T_1 < t_1$. Naturally, $t_2 > t_1$ then B cannot get payments until M received BTCs from the $T^A_{fund(A,M)}$, which protects M from losing BTCs. After receiving $T^M_{fund(M,B)}$, B makes a signature on the funding transaction denoted by $Sig_B(T^M_{fund(M,B)})$ to generate a reply message to A, which tells A that it has already received a payment from M.

Phase 3: Decision. Decision phase is initiated by M when it intends to claim BTCs from the $T_{escr(A,M)}$ during the period between T and T_1. In this phase, every party closes the payment channel and re-allocates BTCs based on payments. There would be four different results according to different choices of parties.

In normal case, if parties A, B and M process correctly, they will reach the cash-out sub-phase (as shown in Fig. 3(b)). The mixer first requires A to cooperatively generate a cash-out transaction $T_{cash \to escr(A,M)}$, which is pointed to the $T_{escr(A,M)}$ and transfers $j \times (\alpha + F_{mix})$ BTCs to M before T_1. With this single transaction, M can gain all BTCs immediately. Similarly, before time T_2 expires, B can cooperate with M to generate a cash-out transaction $T_{cash \to escr(M,B)}$ that transfers $p \times \alpha$ BTCs to B immediately. This process requires four transactions (i.e., $T_{escr(A,M)}$, $T_{cash \to escr(A,M)}$, $T_{escr(M,B)}$ and $T_{cash \to escr(M,B)}$) on-chain.

If A is unwilling to cooperate, M can sign and post the latest $T_{fund(A,M)(j)}$ first. Then, M generates a cash-out transaction $T_{cash \to fund(A,M)(j)}$ to claim $j \times (\alpha + F_{mix})$ BTCs after time t_1. Similarly, in case that M does not cooperate, B can claim BTCs by itself after time t_2 by posting $T^M_{fund(M,B)}$ and $T_{cash \to fund(M,B)}$. Considering that it is capable for M to post the $T^B_{fund(M,B)}$ anytime, which may let $T^M_{fund(M,B)}$ be a double-spending transaction, B needs to monitor the mempool of Bitcoin in real time. If it finds $T^B_{fund(M,B)}$ is posted on blockchain, it should construct a similar cash-out transaction which points to the $T^B_{fund(M,B)}$ and post it after time t_2. B can claim BTCs no matter which funding transaction is finally confirmed on the Bitcoin blockchain. This process requires six transactions (i.e., $T_{escr(A,M)}$, $T_{fund(A,M)(j)}$, $T_{cash \to fund(A,M)(j)}$, $T_{escr(M,B)}$, $T_{fund(M,B)}$ and $T_{cash \to fund(M,B)}$) on-chain.

In abnormal case, if A, B or M aborts the protocol after creating a payment channel, they will reach the refund sub-phase (as shown in Fig. 3(c)). If no payment between A and M before T_1, A can generate and post a refund transaction $T_{refund \to escr(A,M)}$ to reclaim all BTCs escrowed in $T_{escr(A,M)}$. Similarly, if no funding transactions and cash-out transactions are published on-chain before T_2, M can post a refund transaction $T_{refund \to escr(M,B)}$ to reclaim α BTCs from the $T_{escr(M,B)}$. This process requires four transactions (i.e., $T_{escr(A,M)}$, $T_{refund \to escr(A,M)}$, $T_{escr(M,B)}$ and $T_{refund \to escr(M,B)}$) on-chain.

However, if A does not receive a reply message $Sig_B(T^M_{fund(M,B)})$ from B before time t_1 or any $h_1(i)$ is not included in the $MTree$, A can reclaim its BTCs by posting a refund transaction denoted by $T_{refund \to fund(A,M)(j)}$, which reveals the preimage of hash $h_1(i)$. We note that a rational mixer should always post the latest $T^A_{fund(A,M)(j)}$ on the Bitcoin blockchain if A did not cooperate to close the payment channel. Otherwise, M has no chance to get any payments from A. On the other hand, if A behaves maliciously to reclaim its BTCs from the $T^A_{fund(A,M)(j)}$ after it got the reply from B, the mixer can get the preimage of $h_1(k)$ and thus can compute the preimage of $h_2(k)$. Therefore, the mixer can immediately reclaim its BTCs by signing and posting the $T^B_{fund(M,B)}$ and a corresponding refund transaction $T_{refund \to fund(M,B)}$. Thus, anyone who intends to lock the mixer's funds for the value of α BTC for the period Δt needs to pay $F_{tran} + \alpha \cdot \Delta t$, which is more harm than good. Similarly, the mixer is required to monitor the mempool in real time, in case a double-spending transaction $T^M_{fund(M,B)}$ posted by B invalidates its redemption.This process requires six transactions (i.e., $T_{escr(A,M)}$, $T_{fund(A,M)(j)}$, $T_{refund \to fund(A,M)(j)}$, $T_{escr(M,B)}$, $T_{fund(M,B)}$ and $T_{refund \to fund(M,B)}$) on-chain. In addition, we can find that every party is not required to must comply the protocol in order to guarantee the fair exchange, because outputs in funding transactions are mutually independent. Thus, if A aborts the protocol and

reclaims a part of BTCs from the $T_{fund(A,M)(j)}$, M can reclaim equal number of BTCs from the funding transaction $T^B_{fund(M,B)}/T^M_{fund(M,B)}$ immediately. BTCs in the remaining outputs of $T_{fund(A,M)(j)}$ and $T^B_{fund(M,B)}/T^M_{fund(M,B)}$ can be finally claimed by M and B, respectively.

We note that the escrow transaction $T_{escr(M,B)}$ is unnecessary in this protocol because only one payment included in this transaction. Optionally, M can directly post the funding transaction $T^B_{fund(M,B)}$ on-chain after receiving p valid proofs and an ephemeral address $Addr_B$ from B. With this method, B can always use two transactions (i.e., $T^B_{fund(M,B)}$ and $T_{cash\to fund(M,B)}$) to claim BTCs after time t_2, while M can always use two transactions (i.e., $T^B_{fund(M,B)}$ and $T_{refund\to fund(M,B)}$) to reclaim BTCs. However, without the transaction $T_{escr(M,B)}$, B cannot cooperate with M to claim BTCs before T_2, which reduces the efficiency of the protocol.

5 Analysis and Proof of Security and Privacy

In this section, we first analyze the security and privacy of our protocol according to the security and privacy goals defined in Section 3.3. We then prove its security and privacy in a formal way by following the UC-security model.

5.1 Security and Privacy Analysis

5.1.1 Fair Exchange

First, SofitMix executes in epoch, thus it always terminates after a fixed time. Then we show SofitMix ensures that all honest parties receive BTCs as expected (i.e. stipulated by the payment phase) or release no valid payment requests even if a party is malicious or multiple parties are collusive.

Case 1: suppose all parties are honest. Because $t_2 > t_1$, B can claim a payment if and only if M gains α BTCs from A. Thus, all honest parties can receive BTCs as expect in this case.

Case 2: suppose A is malicious. A malicious A can revoke a payment after the mixer provides a funding transaction $T^M_{fund(M,B)}$ to B. However, M can get the preimage of h_2, by which it can reclaim α BTCs from a funding transaction $T^M_{fund(M,B)}$. Thus, the honest mixer and payee release no valid payment requests in this case.

Case 3: suppose B is malicious. A malicious B can abort at the beginning of the protocol. Then A will reclaim its BTCs after time T_1. In this case, an honest payer and the mixer release no valid payment requests.

Case 4: suppose M is malicious. On one hand, If M does not provide a funding transaction $T^M_{fund(M,B)}$ to B after receiving off-chain payments from A, A reclaims its BTCs in the decision phase and B can require A to resend a payment in the next epoch. In this case, M cannot obtain any BTCs in the decision phase because there is no transaction can be utilized to reclaim BTCs between M and B. Meanwhile, as a for-profit entity, M has no motivation to abort the protocol intentionally since such a behavior could undermine its reputation. On the other hand, after providing $T^M_{fund(M,B)}$ to B and claiming BTCs from A, M cannot prevent BTCs in the $T^M_{fund(M,B)}$ from being claimed by B after time t_2 because it does not have the token a_k. Thus, the honest payer releases no valid payment requests or the honest payee receives BTCs as expected.

Case 5: suppose *A* and *M* collude. If A and M collude to lie that they have already successfully offer a payment to B, B can expose the lie by requiring A to show the reply message $Sig_B(T^M_{fund(M,B)})$. In addition, if A helps M reclaim BTCs, B can show the refund transaction $T_{refund\to fund(M,B)}$, the hash value h_2 constructing the funding transaction and the signature $Sig_A(h_2)$ provided by A to prove that M gets the primage of hash h_2 with the help of A and reclaims BTCs paid by A. Considering the hash function $H(\cdot)$ is collision resistant, it is impossible for M to compute the preimage of hash h_2 by itself. Thus, B can require A to resend a payment and an honest B receives BTCs as expected.

Case 6: suppose *B* and *M* collude. If B pretends not to receive a payment from M and requires A to resend BTCs before replying $Sig_B(T^M_{fund(M,B)})$ to A, A can reclaim the BTCs and resend a new payment to B. If B requires A to resend BTCs after replying the signature message to A, A will show the reply message $Sig_B(T^M_{fund(M,B)})$ and refuse its request. In either case, an honest A will receives BTCs as expected.

Case 7: suppose *A* and *B* collude. A and B may intend to forge a proof or utilized a used proof π to defraud the mixer. Because M will provide B a payment only when it receives a new correct proof π from B. With the soundness of the zero-knowledge proof, the probability that M accepts an incorrect π is negligible. M also verifies whether π correlates to a used payment by checking whether $g \notin \Lambda$. Thus, any parties can neither fake a new proof, nor reuse an old proof. Thus an honest M releases no valid payment request in this case.

Case 8: suppose *A*, *B* and *M* collude to impair other honest parties. This case can be reduced to one of the case from case 4 to case 6, because an honest party only interacts with M as well as its counterparties, and the outputs in funding transactions are mutually independent.

5.1.2 Unlinkability

First, the mixer cannot correlate to the specific payer from the zero-knowledge proof π which does not reveal the token a. Since every payment in the protocol has the same value and payees could receive multiple payments from the same or different payers, the mixer cannot infer the relationship between payers and payees by analyzing the values in the transactions. On the other hand, the payee can also use different ephemeral addresses for each payment to improve its unlinkability when receiving all payments from a payer. In addition, if the anonymity set size is smaller than the expectation of involved parties (payers/payees), they can join in the mixing protocol again in the next epoch. Furthermore, since the payee uses different ephemeral addresses to receive payments in different epochs, any one cannot derive the linkage between the payer and the payee from their addresses, even if this pair of payer and payee join in the protocol in multiple epochs. Moreover, our protocol proceeds in epochs and phases. All payers open their payment channels and finish off-chain payments before time T elapses, while all payees open payment channels and process off-channel payments after time T elapses. In the

cash-out phase, B claims BTCs after M claims all BTCs from A. It prevents M intentionally accelerating or delaying the interaction with A to achieve behaviors of corresponding B.

5.1.3 DoS Attack Resistance

A malicious payer may intend to lock the mixer's funds, which makes the mixer out of work. Compared with previous protocols [10], [11], [12], [13], [14], [15], [16] our protocol not only requires a payer A to pay a transaction fee, but also locks its BTCs when it initiates the protocol. In addition, if A aborts the protocol, it will reveal the token a to M, with which M can reclaim its BTCs immediately. Our protocol heavily increases the cost of DoS attack and thus effectively secures the mixer. Notably, our protocol only increases the cost of attackers, but does not increases the deposit of parties. In other hand, as a for-profit party, the mixer has no motivation to launch DoS attacks to payers, which may cause it losing reputation.

5.1.4 Sybil Attack Resistance

The mixer can forge identities of transaction parties and constructing transactions by itself to reduce the proportion of real parties in the anonymity set and de-anonymize a target party. We resist this attack by applying transaction fee. In addition, The mixer could also forge hash values in $MTree$. Our protocol can resist this attack because all parties can check the correctness of the $Sig_M(MTree, timestamp)$ and $MTree$ by verifying the difference between the number of payments included in all $T_{escr(A,M)}$ and $T_{refund(A,M)}$. As a rational and for-profit party, the mixer has no motivation to forge $MTree$, which may cause it losing reputation.

5.2 Security Proof Based on UC-Security Model

In this part, we further prove that our protocol captures all security and privacy goals proposed in Section 3.3 by applying the UC-Security model.

Definition (UC-Security): $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ denotes the ensemble of outputs of the environment $\mathcal{E}$ when interacting with an adversary $\mathcal{A}$ and parties running a protocol Π. The protocol Π UC-realizes a ideal functionality $\mathcal{F}$ if for any polytime adversary $\mathcal{A}$, there exists a polytime simulator $\mathcal{S}$ such that for any polytime environment $\mathcal{E}$, the ensembles $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable ($EXEC_{\Pi,\mathcal{A},\mathcal{E}} \approx EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$).

Thus, to prove that the SofitMix protocol is secure according to the UC-Security model, we first need to construct an ideal functionality $\mathcal{F}$ based on its design and show that the ideal functionality $\mathcal{F}$ captures all security and privacy goals. We then model a simulator $\mathcal{S}$ that runs the adversary $\mathcal{A}$ and simulates the real world SofitMix protocol execution while interacting with the ideal functionality $\mathcal{F}$.

5.2.1 Ideal Functionality $\mathcal{F}$

Parties involved in the SofitMix protocol are modeled as interactive Turing machines that communicate with a trusted centralized functionality $\mathcal{F}$ via secure and authenticated channels $\mathcal{F}_{SMT}^{l}$ [22]. We model the Bitcoin blockchain $\mathcal{BC}$ as a trusted append-only bulletin board with the ideal functionality $\mathcal{F}_{\mathcal{BC}}$, as proposed in [29]. $\mathcal{F}_{\mathcal{BC}}$ holds $\mathcal{BC}$ locally and updates $\mathcal{BC}$ according to newly constructed transactions from parties. All parties in the system can access $\mathcal{F}_{\mathcal{BC}}$, which then returns the whole records of $\mathcal{BC}$. The number of blocks on the blockchain is denoted by $|\mathcal{BC}|$. Time t in our model is absolute time that corresponds to the order number of a block on the blockchain, e.g., $t = |\mathcal{BC}| \pm i$, in which i is a natural number. We emphasize that $\mathcal{F}$ uses $\mathcal{F}_{SMT}^{l}$ and $\mathcal{F}_{\mathcal{BC}}$ as subroutine, i.e., our protocol is specified as a $(\mathcal{F}_{SMT}^{l}, \mathcal{F}_{\mathcal{BC}})$-hybrid model.

We use $(C(u_1, u_2), v, t)$ to denote a payment channel constructed between two parties (u_1, u_2), in which $C(u_1, u_2)$ is a channel identifier, v is the fund capacity of this channel, and t is the expiration time of the channel. For ease of notation, $C(u_1, u_2)$ is also used to denote the identifier of involved parties u_1 and u_2. We use $(C(u_1, u_2), (v', u'), msg, c)$ to denote a transaction to close the channel where v' and u' are the funds allocated to u_1 and u_2, respectively, msg denotes an additional message, and c denotes the type of a transaction, in which $c = 0$ denotes a funding transaction and $c = 1$ denotes a refunding transaction. The ideal functionality $\mathcal{F}$ also maintains two lists $\mathcal{C}$ and $\mathcal{L}$, as well as two sets $\mathcal{S}$ and $\mathcal{R}$ locally, where $\mathcal{C}$ records closed channels, and $\mathcal{L}$ is distinguished to different users and is used to record off-chain payments. $(C(u_1, u_2), (v_1, v_2), h)$ denotes the entries in $\mathcal{L}$, where $C(u_1, u_2)$ is the channel identifier, v_1 and v_2 are the funds allocated to u_1 and u_2 in this payment, h is a one-way-function value representing the unique identifier of this payment. $C(u_1, u_2)$ denotes the entries in $\mathcal{C}$. Elements in $\mathcal{S}$ are denoted as $A : (B_1 : n_1, B_2 : n_2 ... B_l : n_l)$ that represents the amount of payments allocated to each payee from a payer. Elements in $\mathcal{R}$ are denoted as $A : (B_1 : m_1, B_2 : m_2 ... B_l : m_l)$ that represents the amount of refunds initiated by payer A from each payee B.

We describe operations of $\mathcal{F}$ in Table 3. Our ideal functionality $\mathcal{F}$ can capture all security and privacy requirements proposed in Section 3.3. As a trusted third party, $\mathcal{F}$ can resist collusion attacks. In the payment phase, the payment information s' sent from $\mathcal{F}$ to the mixer M does not disclose any information of the payer. In addition, $\mathcal{F}$ locally updates $\mathcal{S}$ and $\mathcal{R}$, which stipulate how to allocate BTCs in the close channel phase according to the payments from payer A. Thus, the mixer and the payee cannot get more bitcoins than what is dictated in $\mathcal{S}$ and $\mathcal{R}$. As in Section 5.1 that provides informal security analysis, $\mathcal{F}$ can also resist the Sybil attack by using transaction fees, as well as the DoS attack because any one who intends to launch the DoS attack has to pay an extra cost.

5.2.2 Simulator $\mathcal{S}$

We model the simulator $\mathcal{S}$ that impersonates the behaviors of an adversary $\mathcal{A}dv$ and simulates the real world SofitMix protocol execution while interacting with the ideal functionality $\mathcal{F}$.

Open Channel: Two parties construct an escrow transaction with their identities $C(u_1, u_2)$, the initial capacity of the channel v, the expired time of the channel t in real world. Let u_1 be the initiator of the payment channel. We then model the simulator $\mathcal{S}$ when u_1 is malicious:

$\mathcal{S}$ parses an escrow transaction $T_{escr(u_1,u_2)}$ which is sent from an adversary $\mathcal{A}dv$ on behalf of u_1 as $(C(u_1, u_2), v, t)$. If u_1 is a payer, $\mathcal{S}$ transfers a request $(open, sid, C(A, M), v, t)$

TABLE 3: Ideal Functionality $\mathcal{F}$ for SofitMix

Open Channel: On input $(open, sid, C(u_1, u_2), v, t)$ from a party u_1, $\mathcal{F}$ processes as follow:

1. $\mathcal{F}$ first checks whether the addresses of parties are valid and u_1 has enough bitcoins. If it is the case, continue;
2. If u_1 is a payer and $t < T$, or u_1 is the mixer and $t > T$, $\mathcal{F}$ appends tuple $(C(u_1, u_2), v, t)$ on $\mathcal{BC}$, and appends tuple $(C(u_1, u_2), (v, 0), h)$ on $\mathcal{L}$ with a random value h. Otherwise, $\mathcal{F}$ aborts.

Payment: The ideal functionality $\mathcal{F}$ has different payment executions against $C(A, M)$ and $C(M, B)$.
Upon receiving $(pay, sid, C(u_1, u_2), n, t')$ from a party u_0, $\mathcal{F}$ processes as follow:

1. $\mathcal{F}$ checks whether there is a tuple $(C(u_1, u_2), v, t)$ on $\mathcal{BC}$. If it is not the case, $\mathcal{F}$ aborts. Otherwise, continue;
2. (1) If u_1 is a payer and $t' > t$, $\mathcal{F}$ sets $u' = n \times (\alpha + F_{mix})$ and $v' = v - u'$. $\mathcal{F}$ queries u_1 with n and gets a reply $s = A : (B_1 : n_1, B_2 : n_2 ... B_l : n_l)$ in which $\sum_{i=1}^{l} n_i = n$. $\mathcal{F}$ adds the entry s in $\mathcal{S}$. Then, $\mathcal{F}$ sends s to B and $s' = (B_1 : n_1, B_2 : n_2 ... B_l : n_l)$ to M via an anonymous channel.
(2) If u_1 is the mixer, $t' > t$ and $n = n'$ where $u_2 : n'$ is an entry in s', $\mathcal{F}$ sets $u' = n \times (\alpha)$ and $v' = v - u'$. Then, $\mathcal{F}$ samples a random value h' and updates L as $(C(u_0, u_1), (v', u'), h')$.
3. If any above conditions are not met, $\mathcal{L}$ rolls back to the last state and $\mathcal{F}$ aborts.

Close Channel: On input $(close, sid, uid, C(u_1, u_2), msg, c)$ from a party, $\mathcal{F}$ processes as follow:

1. $\mathcal{F}$ first parses $\mathcal{BC}$ as $(C(u_1, u_2), v, t)$ and parses $\mathcal{L}$ as $(C(u_1, u_2), (v', u'), h)$. If $C(u_1, u_2) \in \mathcal{C}$, then aborts. Otherwise, continue;
2. (1) If $c = 0$, $uid = u_1$ and $|\mathcal{B}| > t$, $\mathcal{F}$ sets $u' = 0$ and $v' = v$, and puts the tuple $(C(A, M), (v', u'), msg, 0)$ on $\mathcal{BC}$ and $C(A, M)$ on $\mathcal{C}$. $\mathcal{F}$ then terminates.
(2) If $c = 0$, $uid = u_2$ and $|\mathcal{B}| < t$, $\mathcal{F}$ puts the tuple $(C(A, M), (v', u'), msg, 0)$ on $\mathcal{BC}$ and $C(A, M)$ on $\mathcal{C}$. $\mathcal{F}$ then terminates.
(3) If $c = 1$, continue;
3. $\mathcal{F}$ queries A with n and gets a reply $r = A : (B_1 : m_1, B_2 : m_2 ... B_l : m_l)$, in which $m = \sum_{i=1}^{l} m_i$ and $m < n$. $\mathcal{F}$ adds r to $\mathcal{R}$, and sets $u' = u' - m \times \alpha$ and $v' = v' + m \times \alpha$. $\mathcal{F}$ then puts the tuple $(C(A, M), (v', u'), msg, 1)$ on $\mathcal{BC}$. $\mathcal{F}$ traverses the set $\mathcal{R}$ to query for values with B_i as the key and queries M with $(B_i : m_i)$. M can reply with m_i within time t' representing a refund requirement. Upon receiving m_i from M, $\mathcal{F}$ checks whether there is a tuple $(B_i : n_i)$ with A as the key in the set $\mathcal{S}$ in which $n_i > m_i$. If it is the case, $\mathcal{F}$ sets $u' := u' - m_i \times \alpha$ and $v' := v' + m_i \times \alpha$. $\mathcal{F}$ then puts the tuple $(C(M, B), (v', u'), msg, 1)$ on $\mathcal{BC}$ and $C(A, M)$ on $\mathcal{C}$.

to the ideal functionality $\mathcal{F}$. If u_1 is the mixer, $\mathcal{S}$ checks whether $VProof(\pi, g, \Lambda) = 1$. $\mathcal{S}$ transfers the request $(open, sid, C(A, M), v, t)$ to $\mathcal{F}$ if the verification result is correct.

Payment: Users make a payment with their identities $C(u_1, u_2)$, the value of the payment n and the time t'. Let u_1 be the user who initiates the payment request. We then model the simulator $\mathcal{S}$ when u_1 is malicious:

Upon receiving a payment request from the adversary $\mathcal{A}dv$ on behalf of u_1, $\mathcal{S}$ parses the request as $(C(A, M), b, g, h_2, MRoot, e, \pi, n, t')$ if u_1 is a payer while parses the request as $(C(M, B), n, t')$ if u_1 is the mixer. $\mathcal{S}$ then sends $(pay, sid, C(u_1, u_2), n, t')$ to $\mathcal{F}$.

Close Channel: The parties close the channel with their identities $C(u_1, u_2)$, the final bitcoin balances (v', u') and an additional message $m = (a, a \oplus b)$ if needed in the real world. Let u_1 be the party that initiates the execution. We then model the simulator $\mathcal{S}$ in two cases:

1. When u_1 is malicious:
(a) On receiving a refund transaction $T_{refund \to escr}$ from the adversary $\mathcal{A}dv$ on behalf of u_1, $\mathcal{S}$ sends $(close, sid, u_1, C(u_1, u_2), -, 0)$ to $\mathcal{F}$;
(b) On receiving a funding transaction T_{fund} from the adversary $\mathcal{A}dv$ on behalf of u_1, $\mathcal{S}$ sends $(close, sid, u_1, C(u_1, u_2), -, 0)$ to $\mathcal{F}$;
(c) On receiving a refund transaction $T_{refund \to fund}$ from the adversary $\mathcal{A}dv$ on behalf of A, $\mathcal{S}$ first checks whether it holds $h_1 = H(a)$ and $h_2 = H(a \oplus b)$. If it is the case, $\mathcal{S}$ randomly selects $x_1 \in \{0,1\}^\lambda$ and computes $y_1 = H(x_1)$, and sends $(close, sid, A, C(A, M), (x_1, y_1), 1)$ to $\mathcal{F}$. If $\mathcal{S}$ receives an element a' such that $h_1 = H(a')$ but $h_2 \neq H(a' \oplus b)$, $\mathcal{S}$ aborts.
(d) On receiving a notification that M is malicious with a message Mal_M from $\mathcal{F}$, $\mathcal{S}$ queries $\mathcal{A}dv$ with $(C(A, M), \pi, H(a), H(a \oplus b)$. If $\mathcal{A}dv$ can output a value $c = a' \oplus b$ such that $H(a \oplus b) = H(a' \oplus b)$, $\mathcal{S}$ aborts. Otherwise, $\mathcal{S}$ randomly selects $x_2 \in \{0,1\}^\lambda$ and computes $y_2 = H(x_2)$, and sends $(close, sid, A, C(A, M), (x_2, y_2), 1)$ to $\mathcal{F}$.
2. When u_2 is malicious:
(a) On receiving a funding transaction T_{fund} from the adversary $\mathcal{A}dv$ on behalf of u_2, $\mathcal{S}$ sends $(close, sid, u_2, C(u_1, u_2), -, 0)$ to $\mathcal{F}$;
(b) On receiving a cash-out transaction $T_{cash \to fund}$ from the adversary $\mathcal{A}dv$ on behalf of u_2, $\mathcal{S}$ sends $(close, sid, u_2, C(u_1, u_2), -, 1)$ to $\mathcal{F}$.

5.2.3 Security Proof

We now argue that the view of the environment in the simulation is indistinguishable with the execution in the real world. We note that the indistinguishable argument in the open channel phase and the payment phase is trivial, thus we only analyze the indistinguishability in the close channel phase in detail.

First, we assume that the simulator never aborts. If the parties execute refund operations when closing the channel, there are entries (a, h_1) and $(a \oplus b, h_2)$ on the real blockchain. However, the parties' view of $\mathcal{BC}$ in the ideal world contains entries (x_1, y_1) and (x_2, y_2). In this case, we note that the following distribution is statistically close: $((a, h_1), (a \oplus b, h_2)) \approx ((x_1, y_1)(x_2, y_2))$.

Then, we show that the probability of the simulator to abort is negligible. Let $Abort_A$ denote that $\mathcal{S}$ aborts in the case that A executes a refund operation. Let $Abort_M$ denotes that $\mathcal{S}$ aborts in the case that M executes a refund operation. By applying basic probability rules, we have $P(Abort) \leq P(Abort_A) + P(Abort_M)$. We now prove that $P(Abort) \leq \varepsilon(\lambda)$, where $\varepsilon(\lambda)$ is a negligible function with the security parameter λ.

We first consider $Abort_A$. In this case, the adversary can generate a valid zero-knowledge proof over $(G(\cdot), H(\cdot), b, g, h_2, MRoot)$ and an element a' such that $h_1 = H(a')$ but $h_2 \neq H(a' \oplus b)$. By the completeness of the zero-knowledge proof, we can have an element $w \in \{0,1\}^\lambda$ such that $h_1 = H(w)$ and $h_2 = H(w \oplus b)$. Thus, $H(w) = H(a')$, which implies $w = a'$ because $\mathcal{A}dv$ can only query the random oracle for polynomial-degree times. However, $H(w \oplus b) \neq H(a' \oplus b)$ implies that $w \neq a'$ since H is a deterministic function. The contradiction here indicates that for all PPT $\mathcal{A}dv$, we have $P(Abort_A) = 0$.

We then prove that the probability of $Abort_M$ is negligible. $Abort_M$ happens when the $\mathcal{A}dv$ can output a valid preimage of hash $H(a \oplus b)$ without knowing a. Considering a is a randomly selected value over the range of $\{0,1\}^\lambda$ and the $\mathcal{A}dv$ is polynomial time bounded, thus $\mathcal{A}dv$ can output a valid preimage of hash $H(a \oplus b)$ with a negligible probability $\varepsilon(\lambda)$. Thus, we conclude our proof that $EXEC_{\Pi,\mathcal{A},\mathcal{E}} \approx EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$.

6 Implementation and Evaluation

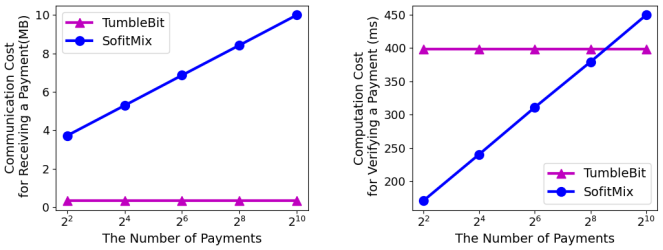
6.1 Implementation and Experimental Settings

We implemented SofitMix with Python as a client-server system based on the Bitcoin test network over a desktop running Linux 16.04 operating system and equipped with 2.3 GHZ Intel Core i5 Quad-CUP and 8.0G RAM, where we ran simulated 200 payers and 200 payees, as well as a mixer with multiple threads. Following the design of the SofitMix protocol, we built the client and the server by utilizing a Bitcoin network API called Bitcoinlib [29]. The mixer acted as a server that sets up the protocol and public parameters. The payers or payees acted as clients that join the protocol and automatically process transactions by interacting with the mixer. In total, there were 2^9 payments initiated by the payers and transferred to the payees through the mixer by following our protocol. To test the fairness of SofitMix, we arranged 4 malicious payers and 3 malicious payees who generated or transmitted abnormal transactions.

We constructed transactions with the P2SH [30] format in our implementation. In the zero-knowledge proof, we instantiated hash functions $H(\cdot)$ and $G(\cdot)$ with SHA-256 and SHA-1, respectively. Because of ZKBoo scheme [31], [32] is the most efficient scheme regarding computation, we implement the zero-knowledge proof by using the ZKBoo in C language at payers and the mixer. Herein, because of the soundness error of ZKBoo is $2/3$ per execution, we set 136 repetitions to achieve the soundness error as below 2^{-80}.

6.2 Performance Evaluation

We first evaluated the communication and computation overhead of SofitMix. Then, we executed a number of payments to test transaction size, validity, fairness, off-chain payment reliability and DoS resistance of SofitMix. All transactions happened in our tests can be found on the blockchain of the Bitcoin test network. All transaction IDs were recorded in the website: http://www.mixertransactiondata.tk/.



(a) Communication Overhead (b) Computation Overhead

Fig. 5: Performance of SofitMix on NIZK.

6.2.1 Computation and Communication Overheads for Cryptographic Algorithm

We note that the main computation and communication overhead of SofitMix come from the zero-knowledge proof. We first simulate the mixer to analyze and test the communication/computation overhead of receiving/verifying the zero-knowledge proof. We find that the result is only related to the number of payments, and independent from the behavior of the mixer (i.e., normal case or abnormal case). Fig. 5 shows that both of them increase logarithmically with the number of payments in an epoch. The computation complexity and communication complexity of SofitMix for one payment are both $\mathcal{O}(\log(n))$, where n is the number of payments received by the mixer. This result is consistent with our design of zero knowledge proof, because the depth of the $MTree$ increases logarithmically with the number of payments in an epoch.

As TumbleBit is the only implemented anonymous mixing protocol, we compared the performance of our protocol with TumbleBit. Different from SofitMix, TumbleBit designs a puzzle-promise protocol that utilizes RSA encryption algorithm to achieve unlinkability. Its communication and computation overheads are fixed and not related to the number of payments. It requires 327KB of data transmission on a wire and spends 398ms to perform a payment. In SofitMix, the communication and the computation overheads increase logarithmically with the number of payments. It has a lower computation cost than that of TumbleBit when the number of payments is smaller than 382. In addition, the communication cost for a payment is 8.5 MB and the computation cost for a payment is 378ms when there are 2^9 payments in an epoch. Although our protocol requires much higher bandwidth to transfer data than TumbleBit, it is acceptable considering the high-performance network nowadays. Notably, we demonstrate it is more reliable and secure than TumbleBit at the end of this section.

6.2.2 Transaction Size

Table 4 shows that the size of all transactions involved in our protocol. The result shows that the size of transactions in the classic model (i.e., $j = 1$) of our protocol is much smaller than that in TumbleBit. It helps reduce the transaction fee and improve the throughput of the Bitcoin network. Because of the mutually independent outputs in the funding transaction, the size of some transactions in the off-chain model (i.e. $j > 1$) increases linearly in our protocol. However, it

TABLE 4: Comparison of Transaction Size (Bytes)

Protocol	T_{escr}	$T_{cash\to escr}$	$T_{refund\to escr}$	$T_{fund(A,M)}$	$T_{fund(M,B)}$	$T_{cash\to fund}$	$T_{refund\to fund}$
SofitMix	191	348	273	$347+32j$	348	278	348
TumbleBit	191	447	373	447	-	907	651

j denotes the number of payments a payer has generated.

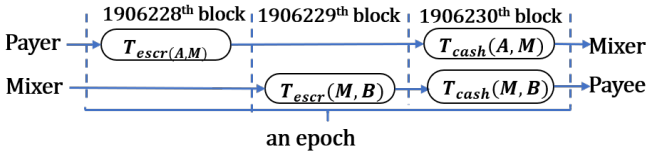


Fig. 6: Payment Test in Normal Case

can guarantee payees to gain expected BTCs even though some malicious payers abort the protocol.

6.2.3 Validity

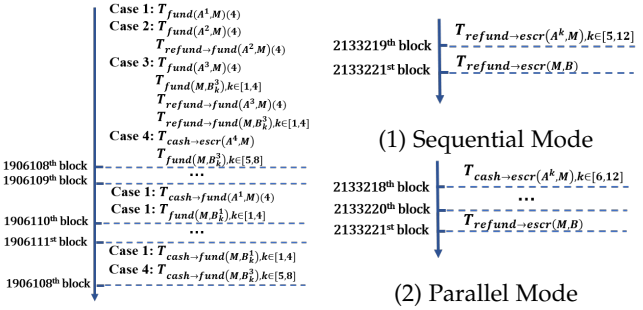
We tested the validity of our protocol by requiring all parties to follow our protocol without aborting. We successfully mixed 2×2^9 payments between 200 payers and 200 payees in the Bitcoin Test network, resulting 1424 transactions included in 3 blocks, as illustrated in Fig. 6. The protocol started at the 1906228^{th} block and ended at 1906230^{th} block. Transactions $T_{escr(A,M)}$ were posted on-chain and all payers finished the payment phase with the mixer in the 1906228^{th} block. Transactions $T_{escr(M,B)}$ were posted on-chain and all payees finished the payment phase with the mixer in the 1906229^{th} block. All parties cooperatively finished the decision phase in the 1906230^{th} block.

6.2.4 Fairness

We tested our protocol by showing that fair exchange can be guaranteed in case that malicious or uncooperative parties exist. We ran 3 payers $A^i (i = 1, 2, 3)$ and 4 payees $B^j (j = 1, 2, 3)$. Every payer intended to make 4 payments. Escrow transactions $T_{escr(A,M)}$ and $T_{escr(M,B)}$ were timelocked for 2 blocks and 4 blocks respectively, while funding transactions $T_{fund(A,M)}$ and $T_{fund(M,B)}$ were timelocked for 3 and 5 blocks respectively. The mixer and the payees used an available API [33] to monitor the mempool of Bitcoin network in real time with 3KB bandwidth consumption. We show the timeline of the fairness test according to the block height when transactions confirmed on the blockchain in Fig. 7 regarding the following four cases.

Case 1: A^1 refused to cooperate with M in the decision phase even though the payment phase was successfully completed. In this case, M posted $T_{fund(A^1,M)(4)}$ (in the 1906108^{th} block in Fig. 7) first and afterwards $T_{cash\to fund(A^1,M)(4)}$ (in the 1906110^{th} block in Fig. 7) by itself to claim BTCs. In case that the mixer M refused to cooperate with B, B^1 claimed its BTCs by posting four funding transactions $T_{fund(M,B_k^1)}$ (refer to the 1906110^{th} block's Case 1) and afterwards $T_{cash\to fund(M,B_k^1)}$ (refer to the 1906112^{th} block's Case 1) by itself to claim BTCs, where $k \in [1, 4]$.

Case 2: M refused to provide payments to the payee B^2 after A^2 completed payments to M. Herein, A^2 waited for M to post $T_{fund(A^2,M)(4)}$ on the blockchain and then



(a) Transactions of SofitMix (b) Transactions of SofitMix

Fig. 7: Payment Test in Abnormal Cases

created $T_{refund\to fund(A^2,M)(4)}$ to reclaim its BTCs (shown in the 1906108^{th} block).

Case 3: A^3 maliciously reclaimed its BTCs after the mixer completed payments with the payee B^3 by using $T_{fund(A^3,M)(4)}$ and $T_{refund\to fund(A^3,M)(4)}$ (refer to the 1906108^{th} block's Case 3). The mixer got tokens from $T_{refund\to fund(A^3,M)(4)}$ posted by A^3, and reclaimed its BTCs by posting four $T_{fund(M,B_k^3)}$ and its corresponding refund transaction $T_{refund\to fund(M,B_k^3)}$, where $k \in [1, 4]$ (refer to the 1906108^{th} block).

From Fig. 7a, we can see that SofitMix can ensure fairness for all involved transaction parties even though some parties behave abnormally.

6.2.5 Reliability of Off-chain Payments

We tested reliable off-chain payments by showing that even if some payers abort, the payee can still claim remaining BTCs. Following the execution of Case 3 in the fairness test, we required A^4 to successfully make 4 off-chain payments to B^3 by $T_{cash\to escr(A^4,M)}$ (refer to the 1906108^{th} block's Case 4. So there were eight independent funding transaction $T_{fund(M,B_k^3)}$ between B and M. We show that the failure cash-out in Case 3 has no influence on the rest payment cash out. We can see from Fig. 7a that B^3 signed and posted $T_{cash\to fund(M,B_k^3)}$ to claim BTCs after the timelock expired, where $k \in [5, 8]$ (refer to the 1906112^{th} block's Case 4).

We also executed similar test on TumbleBit in a sequential mode and a parallel mode, respectively. In each mode, M produced eight cash-out transactions $T_{cash(M,B)(j)}$ and corresponding eight puzzles z_j for a payee B, where $j \in [1, 8]$. We required eight payers A^i ($i \in [5, 12]$) to pay for M from which A^i can get solutions of these puzzles. In this test, the first payer A^5 was malicious, which aborted the protocol after receiving a payment request from B. In the sequential mode, the payee B cannot execute the following payments without the help of the first payer A^5. Thus, refund transactions were posted on the blockchain when payment channels are closed (shown in the 2133219^{th} and

2133221^{st} block in Fig. 7b (1)). In the parallel mode, the remaining payers paid for M and got solutions from M by using $T_{cash \to escr(A^k,M)}$ (shown in the 2133218^{th} block in Fig. 7b (2)), where $k \in [6, 12]$. However, claiming BTCs from any cash-out transaction $T_{cash(M,B)(j)}$ requires the solution of the first puzzle. Thus, M reclaimed all BTCs by posting $T_{refund \to escr(M,B)}$ after the timelock expired (shown in the 2133221^{st} block), while B cannot claim any BTCs and fair exchange cannot be guaranteed when A^5 aborts.

Compared with TumbleBit, we can see SofitMix supports parallel off-chain payments even if some payers abort the protocol, which can enhance the reliability and improve the efficiency of off-chain payments.

6.2.6 Verification of DoS Attack Resistance

We performed a test to show that our protocol can resist DoS attack more effectively than TumbleBit by comparing the cost of an attacker. In this test, we set the capacity of both mixers as each can support ten payments in an epoch, and BTCs in the escrow transaction $T_{escr(M,B)}$ will be locked for three blocks in both protocols. Meanwhile, a payer A, who intends to launch the DoS attack, pretends to transfer ten payments to a payee B. B requires the mixer to open a payment channel that escrows $10 \cdot \alpha$ BTCs in the $T_{escr(M,B)}$. In TumbleBit, A only consumes one F_{tran} to lock all mixer's BTCs in the $T_{escr(M,B)}$ for three blocks. However, in SofitMix, A is required to consume $10 \cdot F_{tran}$ to initiate a escrow transaction $T_{escr(A,M)}$ that includes ten payments, and also needs to lock $10 \cdot \alpha$ BTCs for three blocks in a $T_{fund(A,M)}$. Afterwards, M can reclaim all BTCs immediately after A reclaims its BTCs from the $T_{fund(A,M)}$. Obviously, the attacker's cost to raise the DoS attack in TumbleBit is much lower than that in SofitMix, which implies that SofitMix is more robust than TumbleBit to resist the DoS attack.

7 Conclusion

In this paper, we proposed SofitMix. It is the first mixing protocol that can effectively resist both DoS attacks and collusion attacks. With SofitMix, it is hard for an adversary to make the mixer out of work and a malicious party cannot defraud its counterparties by colluding with the mixer. SofitMix can also reliably supports a set of parallel off-chain payments with fair exchange. It not only realizes anonymity in a Bitcoin-compatible way, but also significantly reduces transaction sizes compared with TumbleBit.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grant 62072351; in part by the Academy of Finland under Grant 308087, Grant 345072, and Grant 350464; in part by the open research project of ZheJiang Lab under grant 2021PD0AB01; and in part by the 111 Project under Grant B16037.

References

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

[2] Z. Yan and L. Peng, "Trust evaluation based on blockchain in pervasive social networking," *IEEE Blockchain Newsl.*, pp. 1–4, 2018.

[3] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[4] W. Feng, Z. Yan, L. T. Yang, and Q. Zheng, "Anonymous authentication on trust in blockchain-based mobile crowdsourcing," *IEEE Internet of Things Journal*, 2020.

[5] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and privacy in social networks*, pp. 197–223, 2013.

[6] M. Ober, S. Katzenbeisser, and K. Hamacher, "Structure and anonymity of the bitcoin transaction graph," *Future internet*, vol. 5, no. 2, pp. 237–250, 2013.

[7] L. Peng, W. Feng, Z. Yan, Y. Li, X. Zhou, and S. Shimizu, "Privacy preservation in permissionless blockchain: A survey," *Digital Communications and Networks*, 2020.

[8] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash : Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, pp. 459–474, 2014.

[9] S. Noether, "Ring signature confidential transactions for monero," *IACR Cryptol. ePrint Arch*, vol. 2015, 2015.

[10] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *Network and Distributed System Security Symposium*, 2017.

[11] G. Maxwell, "Coinswap: Transaction graph disjoint trustless trading," *Bitcoin Forum*, 2013.

[12] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," in *International conference on financial cryptography and data security*, pp. 43–60, 2016.

[13] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world," 2013.

[14] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *European Symposium on Research in Computer Security*, pp. 345–364, 2014.

[15] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "P2p mixing and unlinkable bitcoin transactions.," in *Network and Distributed System Security Sym-posium (NDSS)*, pp. 1–15, 2017.

[16] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-resistant mixing for bitcoin," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pp. 149–158, Nov. 2014.

[17] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 455–471, 2017.

[18] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf, 2016.

[19] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for bitcoin with accountable mixes," in *International Conference on Financial Cryptography and Data Security*, pp. 486–504, 2014.

[20] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for bitcoin," in *International Conference on Financial Cryptography and Data Security*, pp. 112–126, 2015.

[21] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "Coinparty: Secure multi-party mixing of bitcoins," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 75–86, 2015.

[22] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pp. 136–145, IEEE, 2001.

[23] W. Banasik, S. Dziembowski, and D. Malinowski, "Efficient zero-knowledge contingent payments in cryptocurrencies without scripts," in *European symposium on research in computer security*, pp. 261–280, Springer, 2016.

[24] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 455–471, 2017.

[25] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[26] Q. Feng, D. He, S. Zeadally, M. K. Khan, and N. Kumar, "A survey on privacy protection in blockchain system," *Journal of Network and Computer Applications*, vol. 126, pp. 45–58, 2019.
[27] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[28] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 967–984, 2018.
[29] Python-Bitcoinlib, https://github.com/petertodd/python-Bitcoinlib.
[30] https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki.
[31] ZKBoo, https://github.com/Sobuno/ZKBoo.
[32] I. Giacomelli, J. Madsen, and C. Orlandi, "Zkboo: Faster zero-knowledge for boolean circuits," in *25th USENIX Conference on Security Symposium)*, pp. 1069–1083, Aug. 2016.
[33] https://testnet.blockchain.info/api.

H. Xie received the B.Sc. degree in telecommunications engineering from Xidian University, Xi'an, China, in 2016, where he is currently continuing to pursue the Ph.D. degree in Cyber security. His research interests are in security, privacy preservation on blockchain.

S. Fei received his B.Sc. degree from China University of Mining and Technology in 2017. He is pursuing his doctorate in the School of Cyber Engineering at Xidian University. His main research interests include end-to-end communication, blockchain, and trusted execution environment.

Z. Yan (Senior Member, IEEE) received the D.Sc. degree in technology from the Helsinki University of Technology, Espoo, Finland, in 2007. She is currently a Professor in the School of Cyber Engineering, Xidian University, Xi'an, China. Her research interests are in trust, security, privacy, and security-related data analytics. Dr. Yan is an area editor or an associate Editor of IEEE INTERNET OF THINGS JOURNAL, Information Fusion, Information Sciences, IEEE Network Magazine, and Journal of Network and Computer Applications. She served as a General Chair or Program Chair for numerous international conferences, including IEEE TrustCom 2015, IFIP Networking 2021. She is a Founder Steering Committee Co-Chair of IEEE Blockchain conference. She received several awards, including 2021 N²Women: Stars in Computer Networking and Communications, Distinguished Inventor Award of Nokia, the Best Journal Paper Award issued by IEEE Communication Society Technical Committee on Big Data and the Outstanding Associate Editor of 2017 and 2018 for IEEE Access.

Y. Xiao (Graduate Student Member, IEEE) received the B.S. degree from the School of Electrical and Information Engineering, Shanghai Jiao Tong University, and the M.S. degree from the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. He is currently pursuing the Ph.D. degree with the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University (Virginia Tech), supervised by Prof. W. Lou. His research interests include network security, blockchain, and the IoT security.