SAMCU: Secure and Anonymous Multi-Channel Updates in Payment Channel Networks

Jianhuan Wang, Shang Gao *Member, IEEE*,
Guyue Li *Member, IEEE*, Keke Gai *Senior Member, IEEE*, and Bin Xiao, *Fellow, IEEE*

***Abstract*—The Payment Channel Network (PCN) has emerged as an extensively adopted solution to address the scalability issues of Bitcoin by efficient off-chain updates. However, conflicts arise while existing update protocols are pursuing multiple goals of security, privacy, and expressiveness. In this work, we propose a new off-chain update protocol, Secure and Anonymous Multi-Channel Updates (SAMCU), which is developed on the basis of Unspent Transaction Output (UTXO). SAMCU aims at achieving goals of internal anonymity, balance security, and multi-channel updates simultaneously, which has not been done before. To achieve these goals, we exploit the technique of updating graph splitting (UGS) to make participants aware of only the identities of their neighboring sub-graphs, thereby ensuring internal anonymity in multi-channel updates. Then, to avoid security issues arising from equal sub-graphs, we further propose an Enable Payment Transaction Tree (EPTT) to guarantee balance security for each honest protocol participant. Moreover, we optimize the performance of our solution, reducing transaction fees by splitting transactions and the number of communication connections by hierarchical communication. To evaluate the performance of the SAMCU, we implement a prototype involving up to 100 updating payment channels. Experimental results demonstrate that SAMCU outperforms the state-of-the-art, resulting in approximately 70% savings in communication connections and a 66% reduction in on-chain transaction fees when the number of updating payment channels is 100.**

***Index Terms*—Anonymity, Multi-channel Updates, Payment Channel Networks (PCNs), Bitcoin, Blockchain**

I. Introduction

THE advent of Bitcoin has revolutionized the concept of ledger systems by introducing a novel decentralized approach. With its inherent advantages (e.g., irreversibility, pseudonymity, and transparency), Bitcoin has achieved significant development in the financial landscape. However, a critical drawback of this groundbreaking cryptocurrency is its limited throughput compared to traditional financial systems.

This work was supported in part by Grants from the HK RGC GRF (Project No. PolyU 15209822); in part by NSFC/RGC Joint Research Scheme (Project No. N_PolyU529/22 and No. 62261160391); and in part by the Natural Science Foundation on Frontier Leading Technology Basic Research Project of Jiangsu under Grant BK20222001.

J. WANG, S. Gao and B. Xiao (corresponding author) are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mails: 21122251r@connect.polyu.hk; shang-jason.gao@polyu.edu.hk; b.xiao@polyu.edu.hk).

G. Li is with the School of Cyber Science and Engineering, Southeast University, China (e-mail: guyuelee@seu.edu.cn).

K. Gai is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: gaikeke@bit.edu.cn).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org., provided by the author. The material includes an appendix. Contact b.xiao@polyu.edu.hk for further questions about this work.

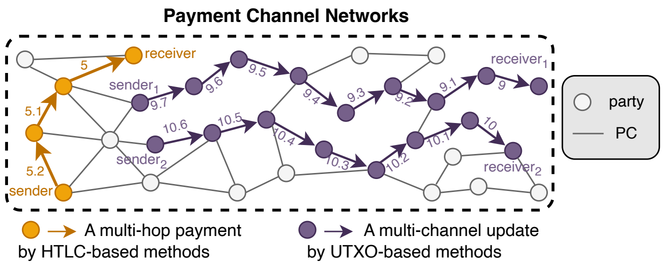


Fig. 1: Examples of the two update modes (i.e., Multi-hop payments and multi-channel updates) supported by HTLC-based and UTXO-based methods, respectively. The values on the payment channels represent the amounts transferred over the channels, with intermediaries within the payment network charging transaction fees (e.g., 0.1 coins) for supporting payments.

This limitation has spurred the exploration of various avenues by community members and researchers to enhance Bitcoin's scalability.

To address the scalability challenge, numerous efforts have been directed towards off-chain protocols. Off-chain protocols reduce the burden on the blockchain by facilitating transactions off-chain between involved parties and resorting to on-chain interactions only when necessary. For instance, payment channels (PCs), which are among the most widely adopted off-chain protocols, require the publication of only two on-chain transactions on the blockchain. One transaction is dedicated to establishing the payment channel, while the other facilitates its closure. Further research has proposed the concept of *payment channel networks (PCNs)*, which enables two users without a direct payment channel to engage in off-chain transactions. By leveraging a multi-hop pathway comprising intermediate parties with existing payment channels, users can route payments through the network.

A crucial challenge that arises in PCN involving multiple payment channels is ensuring atomic updates. Atomicity refers to the property of a transaction being executed entirely or not at all, guaranteeing the consistency of the involved channels' states. Currently, there are two main approaches to achieve atomic updates across multiple payment channels: *Hash Time-Lock Contract (HTLC)-based update protocols* and *Unspent Transaction Output (UTXO)-based update protocols*. HTLC-based update protocols [1] can achieve strong anonymity, and participants only need to generate transactions of constant size. However, it only supports multi-hop payments (MHPs, i.e., one-

TABLE I: State of the art in PCNs

	Update Mode	Collateral	Identity Privacy	Balance Security	On-chain TX size
LN [1]	MHP	$\mathcal{O}(n)$	Yes	Yes	$\mathcal{O}(1)$
AMCU [2]	MCU	$\mathcal{O}(1)$	No	No	$\mathcal{O}(n)$
PT [3]	MHP	$\mathcal{O}(\log_n)$	No	Yes	$\mathcal{O}(1)$
Blitz [4]	MHP	$\mathcal{O}(1)$	Yes	Yes	$\mathcal{O}(1)$
Thora [5]	MCU	$\mathcal{O}(1)$	Internal leak	Yes	$\mathcal{O}(n)$
Our solution	MCU	$\mathcal{O}(1)$	Yes	Yes	$\mathcal{O}(\frac{n}{m}+m)$

to-one transfers) and suffers from inefficient capital utilization caused by linear collateral requirements (i.e., the longer the MHP, the longer it takes for users' funds to be unlocked from HTLCs). In contrast, UTXO-based update protocols achieve multi-channel updates (MCUs, i.e., atomic updates across multiple channels with arbitrary topologies) and only require constant collateral, thereby enabling complex financial applications (e.g., crowdfunding and mass payment) on the Blockchain-like platforms. We illustrate examples of both MHPs and MCUs in Fig. 1.

Despite the advantages of UTXO-based update protocols [2, 4, 3, 5], they still have the following limitations: (a) **Lack of internal anonymity**. To guarantee the balance security of participants, UTXO-based methods require participants to be aware of the public addresses of all involved parties to generate the *Enable Payment* transactions collaboratively. This process results in the lack of *internal anonymity*, where intermediaries know the identities of payment senders and receivers. Existing UTXO-based methods [4] achieve internal anonymity only at the cost of sacrificing MCUs (i.e., only support MHPs), highlighting the significant challenge of enhancing internal anonymity in MCUs. (b) **Large size of transactions for on-chain enforcing payment**. In MCUs, each payment sender needs to find several parties as intermediates to transfer the funds to her corresponding receiver. In some complex applications, such as multiple payment senders transferring funds to multiple payment receivers, the number of channels involved in the protocol can easily reach several dozen. Existing UTXO-based update protocols become impractical in complex applications because they require the enable payment transaction to have the same number of outputs as the number of channels involved. For instance, Thora [5] requires users to emit an enable payment transaction of size 3.9Kb (transaction fees of approximately 7.7 USD in Dec. 2023) to the blockchain when their channels are disputed and the number of updating payment channels reaches 100. (c) **Quadratic number of communication connections**. All parties participating in the UTXO-based update protocols need to communicate with each other, necessitating the creation of a quadratic number of connections. This communication overhead becomes substantial as the number of channels increases. The large number of connections imposes limitations on the simultaneous participation of users in the MCUs, as the number of connections that personal computers can establish simultaneously is limited.

To address these limitations, we introduce a **S**ecure and **A**nonymous **M**ulti-**C**hannel **U**pdates (SAMCU) for PCN that achieves internal anonymity and balance security for MCUs. To achieve PCN internal anonymity, we firstly introduce Updating Graph Splitting (UGS), which divides an updating graph (UG) into several chain-like sub-graphs (SGs), where an *updating graph (UG)* represents the topology of multiple channels needing updates. Based on UGS, our protocol ensures that each party knows only the identities of the parties within the SG where she belongs, rather than all the parties in the UG, thereby preserving anonymity for both senders and receivers. We also propose a novel enable payment transaction tree with necessary timelocks and signature locks to ensure balance security for every honest party while maintaining internal anonymity.

Secondly, our method utilizing UGS reduces the size of enable payment transactions from $O(n)$ to $O(\frac{n}{m}+m)$, where n denotes the number of updating channels, m denotes the number of sub-graphs, as illustrated in Table I. Because participants only need to generate transactions containing the addresses of parties within their sub-graphs, excluding the need for addresses of all parties in the updating graph. Thirdly, to reduce the number of communication connections, we propose a hierarchical communication strategy based on UGS where each sub-graph designates a subset of parties as key parties. These key parties establish complete connections among themselves to facilitate inter-sub-graph information exchange. Parties in different SGs can communicate via these key parties without establishing direct connections, thereby reducing cross-sub-graph connections.

Our contributions can be summarized as follows:

- **SAMCU Protocol.** We propose SAMCU, the first UTXO-based update protocol to achieve multi-channel updates, anonymity, and constant collateral simultaneously in Section III and Section IV. Compared with previous protocols for MCUs, SAMCU achieves (i) internal anonymity (i.e., sender/receiver privacy), (ii) smaller sizes & lower fees of on-chain enforcing payment transactions, and (iii) fewer communication connections.
- **Security and Privacy Analysis:** We conduct a security and privacy analysis of SAMCU in Section V. We show that each honest party engaging in the protocol can ensure her *balance security* by following the protocol's procedures. In addition, we prove that SAMCU achieves *internal anonymity* when the protocol runs without disputes.
- **Implementation and Evaluation:** We implement a prototype of SAMCU protocol[1]. In addition, we perform a comprehensive performance evaluation of SAMCU protocol in Section VI. Experimental results show that SAMCU effectively reduces transaction size and number of communication connections compared to the state-of-the-art, especially in large-scale multi-channel updates (involving 100 updating PCs).

II. Background

A. Unspent Transaction Output (UTXO) Transactions

The UTXO model serves as the foundational framework for representing funds in many UTXO-based blockchains (e.g., Bitcoin). In contrast to account-based blockchains that employ

[1]https://github.com/SAMCU-PCN/SAMCU/

a key-value database in the form of $\{\mathsf{account} : \mathsf{balance}\}$ to store user assets, UTXO-based cryptocurrencies utilize a series of UTXOs to represent user assets. The total balance of a user is the sum of the balances of all UTXOs they own. In the UTXO model, a UTXO can be represented as a tuple: $\theta = (\mathsf{TXID}, \mathsf{index}, \mathsf{value}, \mathsf{ls})$. TXID is the ID of the transaction that created θ, index is the position of the output within that transaction, value is the amount of balance locked in θ, and ls is the conditions necessary for spending the balance in θ.

In this paper, we utilize two widely adopted signature conditions and two widely adopted timelock conditions in Bitcoin-like blockchains. The condition that requires a user's signature is represented by $\mathtt{OneSig}(u)$. Likewise, the condition that requires the signatures of multiple users is represented by $\mathtt{MultiSig}(u_1, ..., u_n)$. We use $\mathsf{AbsT}(T)$ to represent the absolute timelock condition, indicating that the output with this condition should be executed only before time T. We use $\mathsf{RelT}(\Delta)$ to represent the relative timelock condition, meaning the output with this condition can only be unlocked Δ time after it is posted to the blockchain. Due to the fact that transaction fees are determined by the size of transactions, directly storing complex locking scripts like $\mathtt{Multisig}$ in UTXOs leads to higher transaction fees on the blockchain. Moreover, storing locking scripts in UTXOs on the blockchain compromises privacy, as external observers can easily access the script's content in UTXOs. To mitigate these concerns, BIP 16 [6] introduced the standardization of Pay-to-Script-Hash (P2SH) transactions. In a P2SH transaction, a UTXO only stores $\mathcal{H}(\mathsf{ls})$, which is a hash value of the locking script ls. Users then need both the unlocking script and locking script to redeem funds from the UTXO.

B. Payment Channels (PCs)

Payment channels are a popular solution for enhancing blockchain scalability, allowing only for off-chain payments between two parties. The Lightning Network (LN) [1] is a significant implementation of this technology, with the total value locked on the network reaching 141 million USD as of June 2023. The key idea behind this technique is to enable two parties to make multiple payments using a payment channel while only recording two transactions on the ledger. The operations of a payment channel include opening, updating, and closing. To open a channel, both parties lock a certain amount of funds in a 2-of-2 multi-signature output. This output requires signatures from both parties to redeem their funds, ensuring their balance security. During the off-chain payments, they can create new transactions to update the channel's state (i.e., the balance of the two parties in the channel). These updates occur off-chain, enabling efficient multiple payments within the channel without sending transactions to the blockchain. When these two parties decide to close the channel, one of them can submit the final negotiated transaction to the blockchain containing signatures from both parties. Once this transaction is included by the blockchain, the parties can redeem their funds from the 2-of-2 multi-signature output and conclude the use of the channel.

C. Payment Channel Networks (PCNs)

PCNs (e.g., LN [1] and Eclair [7]) are an extension of PCs that establish a network of interconnected channels that aim at enhancing the efficiency of PCs. When two users who do not have an established PC wish to conduct off-chain transactions, they can leverage an existing network path that connects them to facilitate their transfer. There are primarily two update modes in PCN: *Multi-hop payments (MHPs)* [1, 3, 4, 8, 9] and *Multi-channel updates* [2, 5]. One critical issue in PCNs is the atomicity problem, which refers to the guarantee that the states of all channels in payment are either all updated together or none are updated, ensuring that users do not suffer losses due to partial execution. Currently, there are two kinds of approaches to achieve atomicity in PCNs: *HTLC-based update protocols* [1] and *UTXO-based update protocols* [2, 3, 4, 5]. We introduce these methods below and show the comparison of existing approaches in Table I.

HTLC-based Update Protocols. HTLC-based update protocols, as proposed by the Lightning Network (LN) [1], ensure atomicity across a multi-hop payment, which can be modeled as $\mathtt{sender} \rightarrow v_1, \ldots, v_n \rightarrow \mathtt{receiver}$. An HTLC can be constructed by two parties sharing an open payment channel (e.g., Alice and Bob) and allows Alice to lock α coins that can be released only if the contract's conditions (i.e., a hash lock and a time lock) are fulfilled. The core idea for ensuring atomicity is that each pair of adjacent parties in the multi-hop path creates HTLCs bound by the same hash lock. The preimage of this hash is known only to the receiver. When the receiver uses the preimage to redeem her funds, parties from v_n to v_1 sequentially learn the secret and use it to unlock the HTLCs they created with the previous party. This mechanism ensures that either all HTLCs are unlocked or none are, maintaining the atomicity of the MHP. Further studies employ signature algorithms to replace HTLC to enhance the security to defense against *the wormhole attacks*, and interoperability of MHPs [8, 9]. However, existing HTLC-based update protocols and signature-based update protocols still have the following limitations: (a) *Low Expressiveness.* These protocols for MHPs impose limitations on the complexity of payment scenarios, as it does not support complex structures (i.e., crowdfunding) (b) *Liner Collateral.* The duration of the locking phase for the payment sender increases linearly with the increasing number of intermediaries involved, as each intermediary needs to redeem their funds sequentially.

UTXO-based Update Protocols. Atomic Multi-Channel Update (AMCU) protocol [2] is the first attempt to solve the long-standing collateral challenge [10], aiming to achieve *constant collateral payments*. It utilizes the UTXO mechanism, whereby the amount of a transaction is atomically transferred from one or more inputs to one or more outputs to achieve atomicity across multiple channels. This protocol also supports MCUs, enabling users to perform more complex applications on UTXO-based blockchains, such as crowdfunding, mass payments, and PCN rebalancing. However, this protocol sacrifices the identity privacy of payment senders/receivers. Jourenko et al. [3] find that AMCU is not secure. It suffers from *channel closure attacks* [3], where malicious intermediaries collude to perform a double-spending attack to the enable payment transaction, rendering it unaccepted by the blockchain and compromising the atomicity of the protocol. In response to this issue, they proposed the payment trees (PT) protocol that utilizes the

UTXO mechanism to ensure the atomicity of MHPs. However, it comes with a higher collateral requirement compared to AMCU, increasing from $O(1)$ to $O(\log_n)$. Furthermore, this protocol inherits the privacy limitations of AMCU. Aumayr et al. [4] introduced Blitz, which enables MHPs in a single round of communication, contrasting with LN's two-phase commit approach. This protocol provides defense against wormhole attacks while maintaining constant collateral. Additionally, the Blitz protocol achieves identity privacy for MHPs' sender and receiver. Aumayr et al. [5] propose an alternative approach, called Thora, to mitigate the *channel closure attack* by having each channel receiver create her own enable payment transaction. This UTXO-based transaction requires the receiver's signature to become valid, thereby defending against malicious users attempting a channel closure attack. However, existing UTXO-based update protocols still fail to simultaneously achieve balance security, internal anonymity, and multi-channel updates.

III. Solution Overview

A. System Model and Threat Model

System Model. A PCN can be modeled as $\mathcal{G} := (\mathcal{V}, \mathcal{E})$. The set of the nodes $\mathcal{V}$ represents the users' addresses in the blockchain. The set of weighted edges $\mathcal{E}$ represents the opened payment channels in the PCN. Every weighted edge $(X_{v_1}, X_{v_2}) \in \mathcal{E}$ represents the balances of $v_1, v_2 \in \mathcal{V}$ stored in the PC. In this paper, we focus on MCUs in PCN. In an MCU, one or more payment senders transfer funds to one or more payment receivers. We refer to parties that only spend funds as payment senders, and those parties only receive funds as payment receivers. By utilizing specific routing algorithms [11, 12, 13, 14], these parties can discover an updating graph $\mathsf{UG} := (\{(\gamma_i, \alpha_i)\}_{i=1}^{n})$ to execute the off-chain payments, where $\{\gamma_i\}_{i=1}^{n} \subset \mathcal{G}$ represents the set of PCs linking all payment senders and receivers and $\{\alpha_i\}_{i=1}^{n}$ represents the transfer amounts for these PCs. Apart from payment senders and receivers, parties involved in the MCUs are referred to as intermediaries whose fund changes are limited to the payment fees they charge.

In our protocol, we assume the existence of a secure and authenticated communication channel among protocol participants, like TLS channel. We also assume that honest parties remain online throughout the protocol, like [4].

Threat Model. In this paper, we consider the following two types of attacks in our model: (1) Collusion Attacks (e.g., the channel closure attacks), some malicious parties may conspire to disrupt the atomicity of updates to compromise the balance security of honest parties and steal funds from honest parties; and (2) Identity Privacy Attacks, we consider a party's identity privacy compromised when their blockchain address is revealed. We assume that payment senders and payment receivers know each other's identity information, which is a reasonable assumption as payments cannot occur without knowing addresses. However, intermediate parties who assist in the payment are curious about the identities of payment senders and receivers and may try to infer this information.

B. Security and Privacy Goals

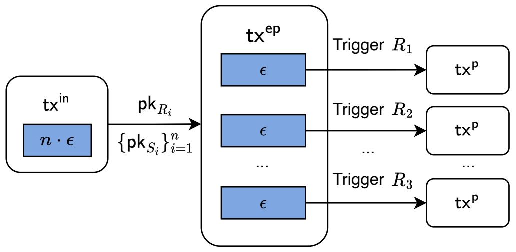
Balance Security [1, 4]: Independent of whether the protocol runs without disputes, no honest intermediate party within the update protocol loses her funds. Note that we do not consider the balance security for malicious parties, as they can compromise the atomicity of the channels they are involved in at the cost of their own funds. However, a rational attacker would not do so.

Internal Anonymity (Sender/Receiver Privacy) [4]: When the protocol runs without disputes (i.e., no party forcibly closes her payment channel on-chain), if an adversary controlling an intermediary can not determine whether any of his neighbors is one of the payment senders or one of the payment receivers, then we regard that the protocol achieves internal anonymity. This notion of anonymity is more stringent compared to that of existing MCUs protocols [5], as these protocols' preservation of parties' identity privacy is limited to external adversaries outside the protocol, while internal parties still know their identities.

C. Our Solution

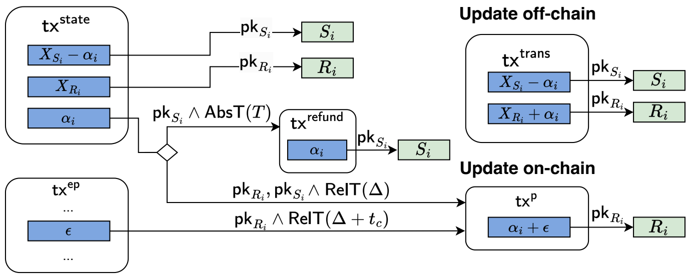
We present a novel UTXO-based update protocol that simultaneously achieves MCUs, internal anonymity, and balance security. We describe our protocol in an incremental way. We first recall the method of employing *Trigger and Response* pattern, a strategy widely utilized in previous UTXO-based update protocols, which solely achieve MCUs and balance security. Then, we give a naive approach utilizing Updating Graph Splitting (UGS) to achieve MCUs and internal anonymity while sacrificing balance security. Finally, we introduce the Enable Payment Transaction Tree (EPTT) to achieve balance security for the naive approach.

Trigger and Response Pattern. The current UTXO-based update protocols mainly employ the *Trigger and Response* pattern to address the challenge of atomic updates across multiple channels. Specifically, atomic updates require that given a $\mathsf{UG} := \{(\gamma_i, \alpha_i)\}_{i=1}^{n}$, the expected state of the channels by the timeout T has only the following clauses: (a) either all channel states are successfully updated off-chain(or enforcing payment on-chain), allowing all the channel receivers of $\{\gamma_i\}_{i=1}^{n}$ to concurrently receive the payment value $\{\alpha_i\}_{i=1}^{n}$, respectively; or (b) all the receivers in $\{\gamma_i\}_{i=1}^{n}$ does not receive $\{\alpha_i\}_{i=1}^{n}$, respectively. To solve this challenge, protocols employing the *Trigger and Response* pattern are divided into two primary operations: (1) The *Trigger* operation, where if any channel receiver forcibly claims her coins from her channel before T, she must publish an *Enable Payment* transaction to the blockchain, signaling a global event to notify other channel receivers. (2) The *Response* operation, where other channel receivers, upon observing the presence of the *Enable Payment* transaction on the blockchain, will react by forcibly claiming their coins from their channels. These operations can be executed in parallel, thereby ensuring constant collateral. Ultimately, if no channel updates occur by the time T expires, all channel states will revert to their status prior to the update. This pattern, in contrast to the HTLC-based update protocol, has the advantage of being applicable to updating graphs of any topological structure and constant collateral. We give an intuitive explanation of the *Trigger* operation and *Response* operation below.



$\mathsf{tx}^{\mathsf{in}}$
$n\cdot\epsilon$
pk_{R_i}
$\{\mathsf{pk}_{S_i}\}_{i=1}^n$
$\mathsf{tx}^{\mathsf{ep}}$
ϵ
Trigger R_1
Trigger R_2
Trigger R_3
tx^{p}

Fig. 2: Overview of the transactions for *Trigger* operation.



$\mathsf{tx}^{\mathsf{state}}$
$X_{S_i}-\alpha_i$
X_{R_i}
α_i
pk_{S_i}
pk_{R_i}
S_i
R_i
$\mathsf{pk}_{S_i}\wedge\mathsf{AbsT}(T)$
$\mathsf{tx}^{\mathsf{refund}}$
$\mathsf{pk}_{R_i},\mathsf{pk}_{S_i}\wedge\mathsf{RelT}(\Delta)$
$\mathsf{pk}_{R_i}\wedge\mathsf{RelT}(\Delta+t_c)$
$\mathsf{tx}^{\mathsf{ep}}$
ϵ
Update off-chain
$\mathsf{tx}^{\mathsf{trans}}$
$X_{R_i}+\alpha_i$
Update on-chain
tx^{p}
$\alpha_i+\epsilon$

Fig. 3: Overview of the update contract for *Response* operation.

Transactions for Trigger Operation. Some existing works (e.g., Blitz) rely on a trusted user to perform the trigger operation. Contrastingly, some existing works (e.g., Thora) eliminate the need for a trusted user by allowing any channel receiver to act as a trigger. In our approach, we adopt the latter strategy, where every channel receiver, denoted by R_i, needs to generate transactions for the trigger operation: $\mathsf{tx}^{\mathsf{in}}$ and $\mathsf{tx}^{\mathsf{ep}}$, as shown in Fig. 2. $\mathsf{tx}^{\mathsf{in}}$ is used to allocate coins for R_i, where the receiver needs to lock the required funds $n\cdot\epsilon$ in one of the outputs of $\mathsf{tx}^{\mathsf{in}}$, with ϵ representing the minimum value supported by the blockchain (e.g., 1 satoshi in Bitcoin). $\mathsf{tx}^{\mathsf{ep}}$ is a crucial transaction that triggers *Response* operations for all receivers. Its input is one of the outputs of $\mathsf{tx}^{\mathsf{in}}$, and it generates an output for each channel receiver R_i, serving as a prerequisite for R_i to execute the *Response* operation. Each output has a value of ϵ and a locking script of $\mathtt{OneSig}(\mathsf{R}_i)$. Once the transaction $\mathsf{tx}^{\mathsf{ep}}$ is generated, it is sent to each channel sender to enable her to generate the corresponding update contract for the *Response* operation based on the received $\mathsf{tx}^{\mathsf{ep}}$.

Update Contract for Response Operation. Upon receiving a set of transactions $\{\mathsf{tx}^{\mathsf{ep}}\}$ from all channel receivers, each channel sender collaborates with her respective channel receiver to generate an update contract. Fig. 3 shows an example of an update contract for the i-th channel γ_i, where we assume $\mathsf{tx}^{\mathsf{ep}}$ as the trigger transaction received from another channel receiver. The balance states of S_i and R_i in the channel are represented as X_{S_i} and X_{R_i}, respectively. S_i first uses $\mathsf{tx}^{\mathsf{state}}$ to update the state of γ_i, which can be seen as a transitional state, rendering her channel *ready* for the transfer of α_i. In $\mathsf{tx}^{\mathsf{state}}$, the sender's balance is split into two outputs: one output with a value of $X_{\mathsf{S}_i}-\alpha_i$ and another output with a value of α_i. For the second output, its condition is specified as $(\mathtt{MultiSig}(\mathsf{R}_i,\mathsf{S}_i)\wedge\mathsf{RelT}(\Delta))\vee(\mathtt{OneSig}(\mathsf{S}_i)\wedge\mathsf{AbsT}(T))$. The timelock condition $\mathsf{AbsT}(T)$ for the input of $\mathsf{tx}^{\mathsf{refund}}$ ensures that the channel sender can only refund the payment value α_i after T. Meanwhile, the timelock condition $\mathsf{RelT}(\Delta)$ for the input of tx^{p} guarantees that the execution priority of $\mathsf{tx}^{\mathsf{refund}}$ is higher than tx^{p} after T, thereby eliminating the possibility that the channel receiver redeem α_i by tx^{p} after T. Then, the state of γ_i has the following clauses: (1) S_i can wait for the expiration of the timeout T and then uses $\mathsf{tx}^{\mathsf{refund}}$ to redeem α_i. This can be viewed as rolling back γ_i to the state before the update; or (2) When R_i observes one $\mathsf{tx}^{\mathsf{ep}}$ (may be posted by her own) being confirmed on the blockchain, she can publish the corresponding tx^{p} to the blockchain to enforce her payment before T. This can be seen as forcibly updating the channel state to the state of successfully sending α_i to R_i in an on-chain way; or (3) S_i and R_i honestly update the state of γ_i using a new off-chain state transaction $\mathsf{tx}^{\mathsf{trans}}$. This transaction contains two outputs, representing the states of the S_i and R_i after transferring α_i.

When the update contact for a payment channel is accurately created, the channel receiver will send a notification message to all channel senders. Upon receiving all notification messages from all channel receivers, a channel sender sends her signature for the corresponding $\mathsf{tx}^{\mathsf{ep}}$ to each channel receiver. This process ensures that once a $\mathsf{tx}^{\mathsf{ep}}$ becomes valid (having obtained all necessary signatures), all transactions and update contracts are generated, and each channel γ_i becomes *ready* for the transfer of α_i. Then, if any channel receiver needs to enforce her payment on-chain when there is a dispute on her channel, she will post $\mathsf{tx}^{\mathsf{in}}$, $\mathsf{tx}^{\mathsf{ep}}$, and tx^{p} (response to her own trigger) to the blockchain. Upon observing $\mathsf{tx}^{\mathsf{ep}}$ being confirmed in the blockchain, other channel receivers will close their channels and publish their corresponding tx^{p} to the blockchain to conduct the *Response* operations, ensuring that all channel receivers can enforce their corresponding payment on-chain simultaneously. The timelock condition $\mathsf{RelT}(t_c+\Delta)$ for $\theta_{\mathsf{tx}^{\mathsf{ep}}/\mathsf{tx}^{\mathsf{p}}}$ ensures sufficient time for other channel receivers to close their channels and subsequently publish their tx^{p} transactions to the blockchain, where $\theta_{\mathsf{tx}_1/\mathsf{tx}_2}$ represents the output connecting tx_1 and tx_2, and t_c represents the maximum time for closing a payment channel.

However, when engaging in protocols that follow the *Trigger and Response* pattern, intermediaries acting as channel receivers need to know the list of addresses of all channel receivers (denoted by $\mathbb{R}$) to construct $\mathsf{tx}^{\mathsf{ep}}$. In addition, they also need to know the list of addresses of all channel senders (denoted by $\mathbb{S}$) to unlock the input of their corresponding $\mathsf{tx}^{\mathsf{ep}}$. Consequently, this information allows participants to easily determine that $\mathbb{S}\setminus\mathbb{R}$ represents the addresses of payment senders and $\mathbb{R}\setminus\mathbb{S}$ represents the addresses of payment receivers.

Internal Anonymity: Updating Graph Splitting. We introduce a naive approach to ensure internal anonymity for UTXO-based update protocols through our novel technique, termed updating graph splitting. We present an example (in Fig. 4) to illustrate the process of UGS and show how UGS can be utilized to achieve internal anonymity.

Given a UG $\mathsf{UG}:=(\{(\gamma_i,\alpha_i)\}_{i=1}^n)$, we split it into multiple SGs to ensure anonymity for the payment senders and receivers. We use an example to illustrate the structure of a split graph shown in Fig. 4(a), where the graph UG is divided into four

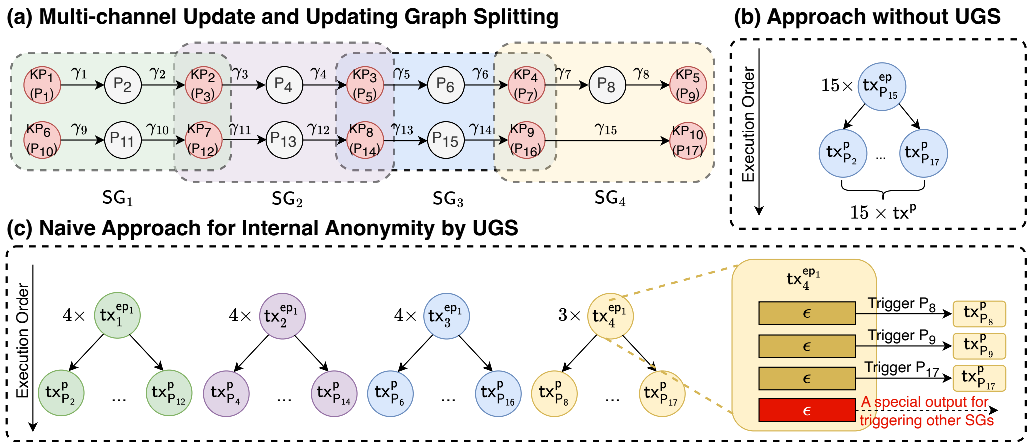

(a) Multi-channel Update and Updating Graph Splitting
(b) Approach without UGS
(c) Naive Approach for Internal Anonymity by UGS
SG1
SG2
SG3
SG4
Execution Order
15× tx^ep_P15
15 × tx^p
Trigger P8
Trigger P9
Trigger P17
A special output for triggering other SGs

Fig. 4: (a) An example of updating graph splitting. The updating graph depicts a two-to-two payment. The two payment senders transfer their funds to the two payment receivers through 7 and 6 intermediaries, respectively. This updating graph is divided into four sub-graphs by UGS, denoted as $\mathsf{SG}_1, \cdots, \mathsf{SG}_4$. Each sub-graph has 3 or 4 channels. (b) The traditional approach employs the *Trigger and Response* pattern, where a channel receiver initiates an enable payment transaction with 15 outputs to trigger 15 channel receivers, respectively. (c) The naive approach utilizes UGS to achieve internal anonymity, where each channel receiver generates a $\mathsf{tx}^{\mathsf{ep}_1}$ to trigger other channel receivers. $\mathsf{tx}^{\mathsf{ep}_1}$ consists solely of outputs that trigger each receiver within her SG, as well as a special output containing global information for triggering channel receivers in other SGs. Consequently, a channel receiver only knows the identities within her own SG, rather than the identities of all channel receivers, to create $\mathsf{tx}^{\mathsf{ep}_1}$.

SGs, denoted by $\mathsf{SG}_1, ..., \mathsf{SG}_4$. Note that the splitting involves dividing the channels into several subsets, leading to overlapping parties across the SGs. For example, P_3 and P_{12} belong to both SG_1 and SG_2. These overlapping parties, along with the payment senders/receivers, collectively form the key parties (KPs), denoted as KP_i. We define parties other than the key parties as normal parties (NPs). To better introduce our protocol, we introduce additional notions for these channels and parties. For each SG SG_i, the individual channels in SG_i are labeled as $\{\gamma_{i,j}\}_{j=1}^{c_i}$, where c_i represents the number of channels within SG_i. The j-th parties in SG_i are represented by $\mathsf{P}_{i,j}$, and $\mathbb{P}_i$ represents the set $\{\mathsf{P}_{i,j}\}_{j=1}^{n_i}$, where n_i represents the number of parties within SG_i. Similarly, $\mathsf{S}_{i,j}$ and $\mathsf{R}_{i,j}$ represents the sender and receiver of the j-th channel in SG_i, respectively. $\mathbb{S}_i$ and $\mathbb{R}_i$ represent the set $\{\mathsf{S}_{i,j}\}_{j=1}^{c_i}$ and $\{\mathsf{R}_{i,j}\}_{j=1}^{c_i}$, respectively.

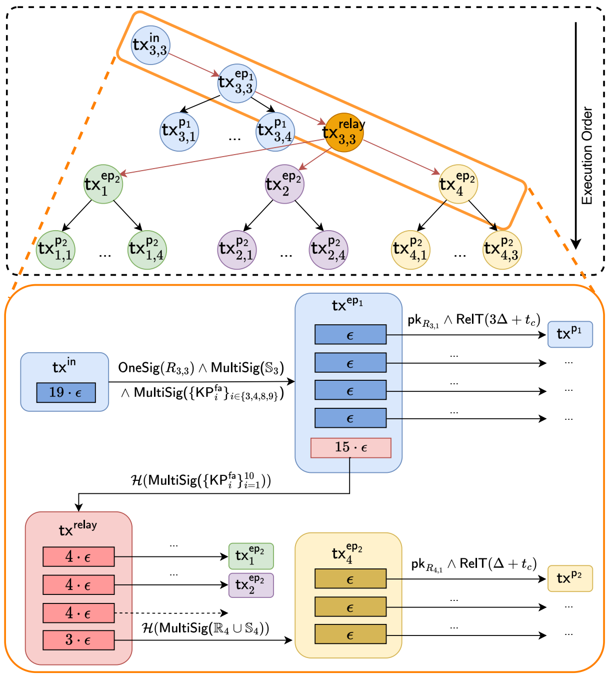
Principle for UGS. To better introduce the principle, we first define the concept of a *chain*, which refers to a series of PCs that link a payment sender to a payment receiver. A UG must consist of several chains. For example, the UG in Fig. 4(a) consists of two chains (i.e., P_1 to P_9 and P_{10} to P_{17}). The splitting must adhere to the following principles: (a) Payment senders&receivers must be key parties. They need to know more information to ensure their balance security than normal parties; (b) Each SG must contain at least one PC from each chain. This ensures that the number of segments in each chain corresponds to the number of SGs, and (c) The UG must be divided into at least three SGs. Currently, the average number of hops required for a one-to-one transfer between any two users in PCN is 9 (as of Feb. 2024 in LN) [15]. Therefore, it is not difficult to split an updating graph into three or more SGs. For small updating graphs that cannot be split into more than three SGs, we discuss how our protocol ensures their internal anonymity in Appendix A. We give an example algorithm for UGS in Appendix B.

Upon UGS, we establish a privacy requirement critical for our protocol to achieve internal anonymity within MCUs. Specifically, this privacy requirement requires that each party in UG knows only the public addresses of parties within their respective SGs, rather than all parties' public addresses in UG, thereby ensuring internal anonymity. For example, P_4, who belongs to SG_2, knows only the public addresses of the parties in SG_2 (i.e., P_3 to P_5 and P_{12} to P_{14}). KP_3, who belongs to both SG_2 and SG_3, only knows the public addresses of the parties in both SG_2 and SG_3 (i.e., P_3 to P_7 and P_{12} to P_{16}). Therefore, an intermediary can not know the public addresses of payment senders and receivers simultaneously when the number of SGs exceeds two. Even if a party and one of the payment senders/receivers are in the same SG (e.g., P_7 and P_9), the intermediary P_7 cannot determine if the known neighbors (e.g., P_9) are payment senders/receivers or just other intermediaries.

Naive Approach. We propose a naive approach to fulfill the privacy requirements for achieving internal anonymity in MCUs. After splitting the UG, we require parties within each SG to generate transactions (i.e., $\mathsf{tx}^{\mathsf{ep}}$ and tx^{p}) solely with her SG's parties for their respective SGs. For the generation of transactions $\mathsf{tx}^{\mathsf{ep}}$ and tx^{p}, parties are only required to know the identities of parties in their respective SGs. However, $\mathsf{tx}^{\mathsf{ep}}$ in Fig. 2 can solely trigger parties within the same SG and cannot trigger parties of other SGs. To overcome this issue, we introduce a novel enable payment transaction, denoted as $\mathsf{tx}^{\mathsf{ep}_1}$, as depicted in Fig. 4(c). Unlike $\mathsf{tx}^{\mathsf{ep}}$, this transaction has an extra output that serves as a notifier, alerting channel receivers in other SGs to respond. The value of this output is ϵ, and the locking script of the output is set to the notifier of this specific MCU (e.g., the sid of this update). Consequently, when any $\mathsf{tx}^{\mathsf{ep}_1}$ is published on the blockchain, all channel receivers in other SGs can respond to this trigger by posting $\mathsf{tx}^{\mathsf{ep}_1}$ for their SGs and her own tx^{p}

upon detecting a UTXO labeled with the notifier of this MCU.

However, this naive approach is vulnerable to *griefing attacks*, compromising its balance security. With the timelock setting of $\theta_{\mathsf{tx}^{\mathsf{ep}_1}/\mathsf{tx}^{\mathsf{p}}}$, a rational trigger should post her $\mathsf{tx}^{\mathsf{ep}_1}$ to the blockchain before $T-3\Delta-t_c$, which ensures that $\mathsf{tx}^{\mathsf{ep}_1}$ will be on the blockchain before $T-2\Delta-t_c$. Then, after waiting for $\Delta+t_c$, when the output of $\mathsf{tx}^{\mathsf{ep}_1}$ is unlocked, the trigger can post tx^{p} to the blockchain before $T-\Delta$, and tx^{p} will be confirmed before T. If a malicious trigger delays posting her $\mathsf{tx}^{\mathsf{ep}_1}$ to the blockchain until $T-3\Delta-t_c$, although the channel receivers in the trigger's SG have enough time to post their tx^{p} before $T-\Delta$ (accepted before T), other channel receivers do not have enough time to post $\mathsf{tx}^{\mathsf{ep}_1}$ for enabling payment for their SGs and tx^{p} to the blockchain, thereby breaking atomicity among all SGs.



Execution Order
$\mathsf{tx}^{\mathsf{in}}_{3,3}$
$\mathsf{tx}^{\mathsf{ep}_1}_{3,3}$
$\mathsf{tx}^{\mathsf{p}_1}_{3,1}$
$\mathsf{tx}^{\mathsf{p}_1}_{3,4}$
$\mathsf{tx}^{\mathsf{relay}}_{3,3}$
$\mathsf{tx}^{\mathsf{ep}_2}_{1}$
$\mathsf{tx}^{\mathsf{ep}_2}_{2}$
$\mathsf{tx}^{\mathsf{ep}_2}_{4}$
$\mathsf{tx}^{\mathsf{p}_2}_{1,1}$ … $\mathsf{tx}^{\mathsf{p}_2}_{1,4}$
$\mathsf{tx}^{\mathsf{p}_2}_{2,1}$ … $\mathsf{tx}^{\mathsf{p}_2}_{2,4}$
$\mathsf{tx}^{\mathsf{p}_2}_{4,1}$ … $\mathsf{tx}^{\mathsf{p}_2}_{4,3}$
$\mathsf{tx}^{\mathsf{in}}$
$19\cdot\epsilon$
$\mathsf{OneSig}(R_{3,3}) \wedge \mathsf{MultiSig}(\mathbb{S}_3)$
$\wedge\, \mathsf{MultiSig}(\{\mathsf{KP}^{\mathsf{fa}}_i\}_{i\in\{3,4,8,9\}})$
$\mathsf{tx}^{\mathsf{ep}_1}$
ϵ
$15\cdot\epsilon$
$\mathsf{pk}_{R_{3,1}} \wedge \mathsf{RelT}(3\Delta+t_c)$
$\mathsf{tx}^{\mathsf{p}_1}$
$\mathcal{H}(\mathsf{MultiSig}(\{\mathsf{KP}^{\mathsf{fa}}_i\}^{10}_{i=1}))$
$\mathsf{tx}^{\mathsf{relay}}$
$4\cdot\epsilon$
$4\cdot\epsilon$
$4\cdot\epsilon$
$3\cdot\epsilon$
$\mathsf{tx}^{\mathsf{ep}_2}_1$
$\mathsf{tx}^{\mathsf{ep}_2}_2$
$\mathcal{H}(\mathsf{MultiSig}(\mathbb{R}_4 \cup \mathbb{S}_4))$
$\mathsf{tx}^{\mathsf{ep}_2}_4$
$\mathsf{pk}_{R_{4,1}} \wedge \mathsf{RelT}(\Delta+t_c)$
$\mathsf{tx}^{\mathsf{p}_2}$

Fig. 5: An example of an EPTT that adopts the *Trigger-Response* pattern. In this tree, $\mathsf{R}_{3,3}$ acts as the trigger, being the first channel receiver to publish $\mathsf{tx}^{\mathsf{in}}$ and $\mathsf{tx}^{\mathsf{ep}_1}$ for enforcing payment. Upon observing $\mathsf{tx}^{\mathsf{ep}_1}$ on the blockchain, other receivers can respond to this trigger, i.e., they publish their corresponding transactions following the tree to enforce their payments. Note that $\mathsf{tx}^{\mathsf{relay}}$ includes four outputs (1 output per SG). Since SG_3 does not require additional triggering by $\mathsf{tx}^{\mathsf{ep}_2}$, the output for SG_3 remains unspent. For clarity, it is hidden from the illustration.

Balance Security: EPTT. To ensure balance security, we introduce the enable payment transaction trees that follow the *Trigger and Response* pattern. Specifically, we require that each channel receiver creates a transaction tree where they act as the trigger. Fig. 5 shows an example of an EPTT, corresponding to the updating graph depicted in Fig. 4. In this tree, $\mathsf{R}_{3,3}$ is the trigger, with the other channel receivers acting as responders. This tree is structured into five layers. The first and second layers are $\mathsf{tx}^{\mathsf{in}}$ and $\mathsf{tx}^{\mathsf{ep}_1}$, respectively, facilitating the enforcing payment for the trigger's SG, SG_3. The third layer includes payment transactions from channel receivers in SG_3 and a special transaction $\mathsf{tx}^{\mathsf{relay}}$, which initiates response operations for channel receivers in other SGs. $\mathsf{tx}^{\mathsf{relay}}$ outputs for each SG, enabling $\mathsf{tx}^{\mathsf{ep}_2}$ of all other SGs, respectively. Notably, $\mathsf{tx}^{\mathsf{ep}_2}$ is similar to $\mathsf{tx}^{\mathsf{ep}}$, with the primary difference being that its input comes from $\mathsf{tx}^{\mathsf{relay}}$ instead of $\mathsf{tx}^{\mathsf{in}}$. The fifth layer consists of payment transactions from channel receivers in other SGs responding to the trigger. For simplicity in describing our protocol, we use $\mathsf{tx}^{\mathsf{p}_1}$ and $\mathsf{tx}^{\mathsf{p}_2}$ instead of tx^{p} to represent the payment transactions following $\mathsf{tx}^{\mathsf{ep}_1}$ and $\mathsf{tx}^{\mathsf{ep}_2}$, respectively.

Compared to the naive approach, the structure of EPTT prescribes a sequential execution order for enable payment transactions of SGs: initially triggering $\mathsf{tx}^{\mathsf{ep}_1}$ for the parties in the SG where the trigger is in, followed by $\mathsf{tx}^{\mathsf{ep}_2}$ for parties in other SGs. This allows us to employ timelock settings to safeguard against griefing attacks. We describe the setting of timelocks of the transaction in EPTT, as shown in Fig. 5. We retain the timelock condition for $\theta_{\mathsf{tx}^{\mathsf{ep}_2}/\mathsf{tx}^{\mathsf{p}_2}}$ as $\mathsf{RelT}(\Delta+t_c)$ and do not set any timelock conditions on $\theta_{\mathsf{tx}^{\mathsf{ep}_1}/\mathsf{tx}^{\mathsf{relay}}}$ and $\theta_{\mathsf{tx}^{\mathsf{relay}}/\mathsf{tx}^{\mathsf{ep}_2}}$. In these timelock settings, a rational trigger must post $\mathsf{tx}^{\mathsf{ep}_2}$ before $T-3\Delta-t_c$ for her $\mathsf{tx}^{\mathsf{p}_2}$ to be confirmed by the blockchain before T. Consider the blockchain delay Δ, $\mathsf{tx}^{\mathsf{relay}}$ and $\mathsf{tx}^{\mathsf{ep}_1}$ must be posted before $T-4\Delta-t_c$ and $T-5\Delta-t_c$, respectively, to allow responders to post $\mathsf{tx}^{\mathsf{ep}_2}$ to the blockchain before $T-3\Delta-t_c$. To ensure that the trigger will definitely post $\mathsf{tx}^{\mathsf{ep}_1}$ before $T-5\Delta-t_c$, we set $\mathsf{RelT}(3\Delta+t_c)$ for $\theta_{\mathsf{tx}^{\mathsf{ep}_1}/\mathsf{tx}^{\mathsf{p}_1}}$. With this setting, a rational receiver must post $\mathsf{tx}^{\mathsf{ep}_1}$ before $T-5\Delta-t_c$ for her $\mathsf{tx}^{\mathsf{p}_1}$ to be confirmed by the blockchain before T.

We require parties from different SGs to collaborate in creating their transaction trees, with key parties acting as bridges for information dissemination between the SGs. To preserve key parties' anonymity, we require key parties to generate one-time-use fresh addresses $\{\mathsf{KP}^{\mathsf{fa}}_i\}^n_{i=1}$ and use them as their identifiers for communication among themselves without revealing their real identities. We describe the creation of the $\mathsf{R}_{3,3}$'s tree as an example (cf. Fig. 5). After $\mathsf{R}_{3,3}$ creates a transaction $\mathsf{tx}^{\mathsf{ep}_1}$ containing SG party identity information through communication with her SG's parties, she sends $\mathsf{tx}^{\mathsf{ep}_1}$ to the key parties in SG_3. Then, the key parties generate the corresponding $\mathsf{tx}^{\mathsf{relay}}$ and send it to key parties in other SGs. These key parties in other SGs then collaborate with parties from their SGs to create the final two layers of the tree (each SG's $\mathsf{tx}^{\mathsf{ep}_2}$ and the corresponding $\mathsf{tx}^{\mathsf{p}_2}$), respectively. Information transfer between SGs consists solely of details regarding $\mathsf{tx}^{\mathsf{relay}}$ transactions conducted by key parties. As $\mathsf{tx}^{\mathsf{relay}}$'s input's unlocking script is $\mathtt{MultiSig}(\{\mathsf{KP}^{\mathsf{fa}}_i\}^n_{i=1})$, and its output for each SG is a hash of a locking script, no identity information is exposed. Furthermore, the generation of transaction trees for every channel receiver can be executed in parallel, thereby not increasing the time complexity.

There are two ways in which the atomicity of an EPTT can be compromised: (1) If the EPTT becomes valid (i.e., all transactions in the tree are signed) when only a portion of PCs are *ready*, then the receivers in payment channels not yet ready will be unable to enforce payment using $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ in case of disputes in their channels. (2) The outputs of transactions are double-spent maliciously by unexpected transactions.

Ensure Readiness of All PCs. To prevent the transaction tree from becoming valid before all channels update their channel states by $\mathsf{tx}^{\mathsf{state}}$, we add a signature condition

Functions for Generating Transactions.

```
GenTxIn($\mathsf{R}_{i,j}, b, \{\gamma_{i,j}\}_{j=1}^{c_i}$)
  1) $\mathsf{ls} := \texttt{MultiSig}(\mathsf{R}_{i,j} \cup \mathbb{S}_i \cup \{\mathsf{KP}_k^{\mathsf{fa}}\}_{\forall\ \mathsf{KP}_k \in \mathsf{SG}_i})$
  2) Return $\mathsf{tx}^{\mathsf{in}} := (id, \mathsf{input}, \theta_1, \theta_2)$, where input is utilized by $\mathsf{R}_{i,j}$
     to unlock a existing UTXO with $a$ balance, $\theta_1 := (b, \mathsf{ls})$, and
     $\theta_2 := (a-b, \texttt{OneSig}(\mathsf{R}_{i,j}))$, and $id := \mathcal{H}(\mathsf{input}, \theta_1, \theta_2)$
GenTxEP1($\{\gamma_{i,j}\}_{j=1}^{c_i}, \mathsf{tx}^{\mathsf{in}}, n, \mathcal{H}_{\mathsf{KP}}$)
  1) If $\mathsf{tx}^{\mathsf{in}}.\mathsf{output}[0].\mathsf{cash} < n \cdot \epsilon$, return $\perp$
  2) $\mathsf{OutputList} := (n \cdot \epsilon, \mathcal{H}_{\mathsf{KP}})$
  3) For each $\mathsf{R}_{i,j}$ in $\mathbb{R}_i$: $\mathsf{OutputList} := \mathsf{OutputList} \cup$
     $(\epsilon, \texttt{OneSig}(\mathsf{R}_{i,j}) \wedge \mathsf{RelT}(3\Delta + t_c))$
  4) $id := \mathcal{H}(\mathsf{tx}^{\mathsf{in}}.\mathsf{output}[0], \mathsf{OutputList})$
  5) Return $\mathsf{tx}^{\mathsf{ep_1}} := (id, \mathsf{tx}^{\mathsf{in}}.\mathsf{output}[0], \mathsf{OutputList})$
GenRelay($\mathsf{tx}^{\mathsf{ep_1}}, n, \{\mathcal{H}_{\mathsf{SG}_i}\}_{i=1}^{n_{\mathsf{SG}}}$)
  1) If $\mathsf{tx}^{\mathsf{ep_1}}.\mathsf{output}[0].\mathsf{cash} < n \cdot \epsilon$, return $\perp$
  2) $\mathsf{OutputList} := \emptyset$; For each $\mathsf{SG}_i$: $\mathsf{OutputList} := \mathsf{OutputList} \cup (c_i \cdot$
     $\epsilon, \mathcal{H}_{\mathsf{SG}_i})$
  3) $id := \mathcal{H}(\mathsf{tx}^{\mathsf{ep_1}}.\mathsf{output}[0], \mathsf{OutputList})$
  4) Return $\mathsf{tx}^{\mathsf{relay}} := (id, \mathsf{tx}^{\mathsf{ep_1}}.\mathsf{output}[0], \mathsf{OutputList})$
GenTxEP2($\mathsf{tx}_i^{\mathsf{relay}}, \{\gamma_{i,j}\}_{j=1}^{c_i}$)
  1) If $\mathsf{tx}^{\mathsf{relay}}.\mathsf{output}[i].\mathsf{cash} < c_i \cdot \epsilon$, return $\perp$
  2) $\mathsf{OutputList} := \emptyset$; For each $\mathsf{R}_{i,j}$ in $\mathbb{R}_i$: $\mathsf{OutputList} := \mathsf{OutputList} \cup$
     $(\epsilon, \texttt{OneSig}(R_{i,j}) \wedge \mathsf{RelT}(\Delta + t_c))$
  3) $id := \mathcal{H}(\mathsf{tx}^{\mathsf{relay}}.\mathsf{output}[i], \mathsf{OutputList})$
  4) Return $\mathsf{tx}^{\mathsf{ep_2}} := (id, \mathsf{tx}^{\mathsf{relay}}.\mathsf{output}[i], \mathsf{OutputList})$
GenTxState($\alpha, T, \gamma'$)
  1) Let $\theta'$ be the input of current state of $\gamma'$. Let $\theta_s =$
     $(b_s, \texttt{OneSig}(\mathsf{S}_{i,j})$ and $\theta_r = (b_r, \texttt{OneSig}(\mathsf{R}_{i,j}$ be the outputs of
     the current state of $\gamma'$, respectively. The values of $b_s$ and $b_r$ are
     the deposited balance of the sender and receiver, respectively.
  2) If $\alpha < b_s$, Return $\perp$. Otherwise, return $\mathsf{tx}^{\mathsf{state}} :=$
     $(id, \mathsf{input}, \theta_1, \theta_2, \theta_3)$, where $\theta_1 := (\alpha, (\texttt{OneSig}(\mathsf{S}_{i,j}) \wedge$
     $\mathsf{AbsT}(T)) \vee (\texttt{MultiSig}(\mathsf{S}_{i,j}, \mathsf{R}_{i,j}) \wedge \mathsf{RelT}(\Delta + t_c)))$, $\theta_2 :=$
     $(b_s - \alpha, \texttt{OneSig}(\mathsf{S}_{i,j}))$, $\theta_3 := (b_r, \texttt{OneSig}(\mathsf{R}_{i,j}))$.
GenTxRefund($\mathsf{tx}^{\mathsf{state}}$)
  1) $\mathsf{OutputList} = (\mathsf{tx}^{\mathsf{state}}.\mathsf{output}[0].\mathsf{cash}, \texttt{OneSig}(S_{i,j}))$
  2) $id := \mathcal{H}(\mathsf{tx}^{\mathsf{state}}.\mathsf{output}[0], \mathsf{OutputList})$
  3) Return $\mathsf{tx}^{\mathsf{refund}} := (id, \mathsf{tx}^{\mathsf{state}}.\mathsf{output}[0], \mathsf{OutputList})$
GenTxP1/P2($\mathsf{tx}^{\mathsf{state}}, \theta$)
  1) $\mathsf{OutputList} = (\mathsf{tx}^{\mathsf{state}}.\mathsf{output}[0].\mathsf{cash} + \epsilon, \texttt{OneSig}(R_{i,j}))$
  2) $id := \mathcal{H}((\mathsf{tx}^{\mathsf{state}}.\mathsf{output}[0], \theta), \mathsf{OutputList})$
  3) Return $\mathsf{tx}^{\mathsf{refund}} := (id, (\mathsf{tx}^{\mathsf{state}}.\mathsf{output}[0], \theta), \mathsf{OutputList})$
GenTxTrans($\gamma$)
  1) Let input be the input of current state of $\theta'$. Let $\theta_1, \theta_2, \theta_3$
     be the outputs of the current state of $\gamma'$, respectively. Re-
     turn $\mathsf{tx}^{\mathsf{trans}} := (id, \mathsf{input}, \theta_s, \theta_r)$, where $\theta_r := (\theta_1.\mathsf{cash} +$
     $\theta_3.\mathsf{cash}, \texttt{OneSig}(\mathsf{R}_{i,j}))$, $\theta_s := (\theta_2.\mathsf{cash}, \texttt{OneSig}(\mathsf{S}_{i,j}))$.
```

Fig. 6: Functions for Generating Transactions.

$\texttt{MultiSig}(\mathbb{S}_i \cup \{\mathsf{KP}_k^{\mathsf{fa}}\}_{\forall\ \mathsf{KP}_k \in \mathsf{SG}_i})$ to $\mathsf{tx}^{\mathsf{ep_1}}$, add a signature condition $\texttt{MultiSig}(\{\mathsf{KP}_i^{\mathsf{fa}}\})$ to $\mathsf{tx}^{\mathsf{relay}}$, and add a signature condition $\texttt{MultiSig}(\mathbb{S}_i)$ to $\mathsf{tx}^{\mathsf{ep_2}}$, as shown in Fig. 5. After a channel sender successfully updates her channel by $\mathsf{tx}^{\mathsf{state}}$, she will sign $\{\mathsf{tx}^{\mathsf{ep_1/ep_2}}\}$ for all channel receivers' trees in her SG and then sends these signatures to the parties in her SG. After receiving these signatures from all channel senders in her SGs, the key party will inform other key parties that the SGs she is responsible for are all *ready* by sending her signatures of $\{\mathsf{tx}^{\mathsf{relay}}\}$. The key party only sends her signatures of $\{\mathsf{tx}^{\mathsf{ep_1}}\}$ to the channel receivers in her SGs after confirming that all other payment channels have been ready. This process ensures that when a channel receiver receives all the signatures to satisfy the signature conditions of $\mathsf{tx}^{\mathsf{ep_1}}$, every payment channel in UG has been ready.

Prevent Double Spending. To prevent double spending, we carefully design the signature conditions of locking scripts for transactions. We add a signature condition to the input of $\mathsf{tx}^{\mathsf{ep_1}}$, requiring the signature of the owner of $\mathsf{tx}^{\mathsf{ep_1}}$ to unlock it. As a result, for the first receiver $R_{i,j}$ executing the trigger and response operations, all inputs of $\mathsf{tx}^{\mathsf{in}}, \mathsf{tx}^{\mathsf{ep_1}}$, and $\mathsf{tx}^{\mathsf{p_1}}$ require her signature to unlock the corresponding outputs. Malicious users cannot launch a double-spending attack to unlock an output requiring $R_{i,j}$'s signature without $R_{i,j}$'s participation. We add a signature condition to the input of $\mathsf{tx}^{\mathsf{ep_2}}$, requiring the signatures of all channel receivers in the SG corresponding to $\mathsf{tx}^{\mathsf{ep_2}}$ to unlock it. Consequently, if there is at least one honest key party, malicious users are unable to launch a double spending attack on $\mathsf{tx}^{\mathsf{relay}}$, $\mathsf{tx}^{\mathsf{ep_2}}$, and $\mathsf{tx}^{\mathsf{p_2}}$, which are required for other honest channel receivers to enforce payment on-chain.

The detailed definition of functions for generating transactions is shown in Fig. 6. We provide a comprehensive security analysis of the EPTT with the aforementioned strategies in Section V, demonstrating that our solution safeguards the balance security for honest intermediate parties, even in scenarios involving malicious key parties. For existing attacks (i.e., wormhole attacks and channel closure attacks), our protocol inherently resists wormhole attacks due to the atomicity in SAMCU being guaranteed by UTXO mechanism instead of secret passing. Additionally, our protocol defends against channel closure attacks as it prevents double spending, which is the core operation of such attacks. However, malicious key parties may launch a wormhole-like attack to steal transaction fees from normal parties rather than their funds. This type of attack causes relatively minor losses compared to traditional wormhole attacks. We discuss this attack in Appendix C.

Other Properties. In addition to the aforementioned security and privacy goals, our solution achieves performance improvements in the following aspects.

Small Size & Low Fee of On-chain Transaction. Compared with existing protocols, the total size of transactions for enforcing payment per person is reduced because we split the large enable payment transaction into several smaller ones. A channel receiver who incurs a dispute only needs to publish one small enable payment transaction to enforce her payment. For example, as shown in Fig. 4(b), in existing approaches, a channel receiver needs to publish $\mathsf{tx}^{\mathsf{ep}}$ with 1 input and 15 outputs. In contrast, in our approach, the channel receiver in SG_4 only needs to generate a $\mathsf{tx}^{\mathsf{ep_1}}$ with 1 input, and 4 outputs for triggering other parties in UG, as shown in Fig. 5. After observing a $\mathsf{tx}^{\mathsf{ep_1}}$ on the blockchain, parties in other SGs only need to publish $\mathsf{tx}^{\mathsf{relay}}$ and $\mathsf{tx}^{\mathsf{ep_2}}$, where the total size of these two transactions is also smaller than the $\mathsf{tx}^{\mathsf{ep}}$ in the existing approaches.

However, there remains a challenge in reducing the on-chain transaction fees for enforcing payments per person: If we employ the regular transaction fee payment mechanism (adding the fee to one of the inputs) in SAMCU, then every channel receiver needs to cover the fee for the transactions in her EPTT (except $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$) because the funds in inputs of these transactions all originate from $\mathsf{tx}^{\mathsf{in}}$). Therefore, a receiver must pay not only for transactions for enforcing her payment (i.e., $\mathsf{tx}^{\mathsf{in}}$, $\mathsf{tx}^{\mathsf{ep}_1}$) but also for transactions (i.e., $\mathsf{tx}^{\mathsf{relay}}$, $\mathsf{tx}^{\mathsf{ep}_2}$) for enforcing other receivers' payment. To solve this problem, we employ Child-Pays-For-Parent [16] to pay the fees instead of the regular method. We can modify the transactions in EPTT to support CPFP by increasing the value of the input by 1ϵ and adding an output with 1ϵ and no locking script in transactions $\mathsf{tx}^{\mathsf{relay}}$, $\mathsf{tx}^{\mathsf{ep}_1/\mathsf{ep}_2}$. Although these transactions are set with 0 transaction fees, anyone can use the CPFP strategy to pay the fees without changing the content of the parent transactions. Details of CPFP are introduced in Appendix D. With CPFP, parties only need to pay for the transaction enforcing their payments, thereby reducing the on-chain transaction fee per person.

Few Communication Connections. In existing approaches, each intermediary needs to establish network connections with all parties involved when participating in an MCU. In contrast, our solution requires intermediaries to establish connections only with the dealer, other parties within the same SG, and all key parties at most. This selective connection strategy reduces the network resources consumed by parties participating in MCUs.

IV. Constructions

A. Building Blocks

1) Digital Signature Scheme: A digital signature scheme Σ is composed of three algorithms $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$:

- $\mathsf{sk}, \mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda)$ is a PPT algorithm that takes the security parameter 1^λ and outputs a public key pk and the corresponding secret key sk.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ is a PPT algorithm that takes a secret key sk and a message m as input and outputs a signature σ.
- $\{0,1\} \leftarrow \mathsf{Verify}(\sigma, m, \mathsf{pk})$ is a DPT algorithm that takes signature σ, a message m, and a public key pk as input. It determines the validity of the signature by checking if σ was created by the secret key corresponding to pk for the given message m. If the signature is valid, the algorithm outputs 1; otherwise, it outputs 0.

2) Payment Channel Networks.: A PCN is equipped with three following operations:

- $\mathsf{createChannel}(\overline{\mathsf{tx}}^{\mathsf{deposit}}) \to \gamma$ Any two users can submit a transaction $\mathsf{tx}^{\mathsf{deposit}}$ to the blockchain to open a channel. This transaction has two inputs, each from one of the users. The output of this transaction is a 2-of-2 multi-signature address controlled by both parties. Additionally, the two parties will negotiate $\overline{\mathsf{tx}}^{\mathsf{state}}$, which can be used to redeem their respective funds from the 2-of-2 multi-signature address. It is important to note that $\overline{\mathsf{tx}}^{\mathsf{state}}$ is not immediately recorded on the blockchain at this stage. Finally, this operation generates a payment channel γ with an identifier as the 2-of-2 multi-signature address.
- $\mathsf{updateChannel}(\gamma, \mathsf{tx}^{\mathsf{state}}, t_u) \to \{0,1\}$ Either user can update the state of channel γ to the state defined by $\mathsf{tx}^{\mathsf{state}}$, where $\mathsf{tx}^{\mathsf{state}}$ is the unsigned version of $\overline{\mathsf{tx}}^{\mathsf{state}}$. When both users complete the negotiation and sign $\mathsf{tx}^{\mathsf{state}}$, this operation returns 1. Otherwise, it returns 0. The maximum execution time for this operation is t_u.
- $\mathsf{closeChannel}(\gamma, t_c)$, Either user can close the channel by publishing the latest $\overline{\mathsf{tx}}^{\mathsf{state}}$ stored in γ, which represents the latest state of the channel, to the blockchain. t_c represents the maximum execution time for this operation.

B. Protocol Description

The formal construction of our protocol is described below and shown in Fig. 7.

1) Initialization: One of the payment senders is chosen as the dealer. After dealer splitting the updating graph $\mathsf{UG} := (\{(\gamma_i, \alpha_i)\}_{i=1}^n)$, she requires all the key parties to create the fresh addresses $\{\mathsf{KP}_i^{\mathsf{fa}}\}_{i=1}^{n_{\mathsf{KP}}}$ for $\{\mathsf{KP}_i\}_{i=1}^{n_{\mathsf{KP}}}$ exclusively for this specific update. Based on $\{\mathsf{KP}_i^{\mathsf{fa}}\}_{i=1}^{n_{\mathsf{KP}}}$, dealer generates the input and output parameters required for $\mathsf{tx}^{\mathsf{relay}}$. Subsequently, the dealer distributes the relevant parameters to all parties. If all parties agree to participate in this update by sending confirmation messages to dealer, the process will proceed to the next phase.

2) Pre-Setup-SG: For each SG, for each receiver, $\mathsf{R}_{i,j}$ generate $\mathsf{tx}^{\mathsf{in}}$. The value of the output is $(c_i + n) \cdot \epsilon$. Subsequently, $\mathsf{R}_{i,j}$ generate $\mathsf{tx}^{\mathsf{ep}_1}$, with the last output being $\mathtt{MultiSig}(\{\mathsf{KP}_i^{\mathsf{fa}}\}_{i=1}^{n_{\mathsf{KP}}})$ and a value of $n \cdot \epsilon$. The remaining outputs of $\mathsf{tx}^{\mathsf{ep}_1}$ are allocated to send ϵ to other receivers within the same sub-graph. Finally, a two-step notification process ensures that each party is aware of the completion of generation by other parties before proceeding to the next phase.

3) Pre-Setup-UG: For each key party $\mathsf{KP}_k \in \{\mathsf{KP}_i\}_{i=1}^{n_{\mathsf{KP}}}$, she first generates the corresponding $\mathsf{tx}^{\mathsf{relay}}$ for all the received $\mathsf{tx}^{\mathsf{ep}_1}$ from the parties in her sub-graphs and sends $\mathsf{tx}^{\mathsf{relay}}$ to all the other KPs. Then, once a key party receives $\mathsf{tx}^{\mathsf{relay}}$ from all KPs, she generates the corresponding $\mathsf{tx}^{\mathsf{ep}_2}$ for each received $\mathsf{tx}^{\mathsf{relay}}$. Finally, KP_k sends all the generated $\mathsf{tx}^{\mathsf{relay}}$ and $\mathsf{tx}^{\mathsf{ep}_2}$ to all parties in her sub-graphs. Upon receiving them, parties will check the correctness of the received transactions.

4) Setup: The channel senders create $\mathsf{tx}^{\mathsf{state}}$ and $\mathsf{tx}^{\mathsf{refund}}$, along with $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ for each received $\mathsf{tx}^{\mathsf{ep}_1/\mathsf{ep}_2}$. Then, they send $\mathsf{tx}^{\mathsf{state}}$, all $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ and the signatures for all $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ to their channel receivers, respectively. This step ensures that the receivers can post $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ on the blockchain to redeem their balances, regardless of which $\mathsf{tx}^{\mathsf{ep}_1/\mathsf{ep}_2}$ that is eventually published. Subsequently, a two-step notification process ensures that each party knows the completion of checking received transactions by all other channel receivers before proceeding to the next phase.

5) Confirmation: After completing the two-step notification process in the last phase, each channel sender proceeds to update the channel with the channel receiver from its un-updated state to the transmit state (represented by $\mathsf{tx}^{\mathsf{state}}$) because she can ensure that all receivers possess $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ with signatures that allow them to claim the funds from $\mathsf{tx}^{\mathsf{state}}$. If the channel is successfully updated, the channel sender signs each $\mathsf{tx}^{\mathsf{ep}_1/\mathsf{ep}_2}$ and sends it to the corresponding receiver, while the channel receiver signs each $\mathsf{tx}^{\mathsf{ep}_2}$ and sends it to the corresponding receiver. When a channel receiver gets the signatures from

SAMCU Protocol Details

- Let dealer be one of the payment senders or receivers;
- Let T be the expected end time for this update;
- Let Δ be the maximum delay time of the blockchain;
- Let t_u and t_c be the maximum execution time of updating and closing PC, respectively;
- Let $\mathsf{UG} := (\{\{\gamma_{i,j}, \alpha_{i,j}\}_{j=1}^{c_i}\}_{i=1}^{m})$ be the updating graph, where $\alpha_{i,j}$ is the transferred amount of $\gamma_{i,j}$ and $\Sigma_{i=1}^{m} c_i = n$;
- Let $n_{\mathsf{KP}}, n_{\mathsf{SG}}$ be the number of key parties, sub-graphs in UG, respectively;

Initialization (phase 1):

dealer:

1) Send $\mathtt{init}$ to all key parties $\{\mathsf{KP}_i\}_{i=1}^{n_{\mathsf{KP}}}$.

KP_i upon receiving $\mathtt{init}$ from dealer:

1) Create a fresh address $\mathsf{KP}_i^{\mathsf{fa}}$ and sends $\mathsf{KP}_i^{\mathsf{fa}}$ to dealer.

dealer upon receiving $\{\mathsf{KP}_i^{\mathsf{fa}}\}_{i=1}^{n_{\mathsf{KP}}}$:

1) Create $\mathcal{H}_{\mathsf{KP}} := \mathcal{H}(\mathtt{MultiSig}(\{\mathsf{KP}_i^{\mathsf{fa}}\}_{i=1}^{n_{\mathsf{KP}}}))$
2) Create $\mathcal{H}_{\mathsf{SG}_i} := \mathcal{H}(\mathtt{MultiSig}(\mathbb{R}_i \cup \mathbb{S}_i))$ for each SG_i.
3) Create $\mathcal{M}_{\mathsf{SG}_i} := \{\mathsf{KP}_k : \mathsf{KP}_k^{\mathsf{fa}}\}_{\mathsf{KP}_k \in \mathsf{SG}_i}$ for each SG_i (the id mapping of KPs' real identities and fresh identities in SG_i).
4) Send $\mathtt{initMsg} := (\mathcal{H}_{\mathsf{KP}}, n, \{\gamma_{i,j}\}_{j=1}^{c_i}, \mathcal{M}_{\mathsf{SG}_i})$ to all normal parties in $\mathsf{SG}_i, 1 \le i \le n_{\mathsf{SG}}$, and send $\mathtt{initMsg} := (\mathcal{H}_{\mathsf{KP}}, n, \{\mathcal{H}_{\mathsf{SG}_j}\}_{j=1}^{n_{\mathsf{SG}}}, \{\gamma_{i,j}\}_{j=1}^{c_i}, \{\gamma_{i+1,j}\}_{j=1}^{c_{i+1}}, \mathcal{M}_{\mathsf{SG}_i}, \mathcal{M}_{\mathsf{SG}_{i+1}}, \{\mathsf{KP}_k\}_{k=1}^{n_{\mathsf{KP}}})$ to KP_k in SG_i and SG_{i+1}, $1 \le k \le n_{\mathsf{KP}}$.

$\mathsf{P}_{i,j}$ upon receiving $\mathtt{initMsg}$ from dealer:

1) Verify the information of the MCU, if the party agrees to participate in this payment, send $\mathtt{init\text{-}OK}$ to dealer and go to **Pre-Setup-SG** phase, else send $\mathtt{abort}$.

Pre-Setup-SG (phase 2):

$R_{i,j}$:

1) Create $\mathsf{tx}_{i,j}^{\mathsf{in}} := \mathtt{GetTxIn}(\mathsf{R}_{i,j}, c_i + n, \{\gamma_{i,j}\}_{j=1}^{c_i})$, create $\mathsf{tx}_{i,j}^{\mathsf{ep_1}} := \mathtt{GetTxEP1}(\{\gamma_i, j\}_{j=1}^{c_i}, \mathsf{tx}_{i,j}^{\mathsf{in}}, n, \mathcal{H}_{\mathsf{KP}})$ and send $\mathsf{tx}_{i,j}^{\mathsf{ep_1}}$ to all $\mathsf{P}_{i,j}$ in $\mathbb{P}_i$.

KP_k in both SG_i and SG_{i+1} upon receiving $\{\mathsf{tx}_{i,j}^{\mathsf{ep_1}}\}_{j=1}^{c_i} \cup \{\mathsf{tx}_{i+1,j}^{\mathsf{ep_1}}\}_{j=1}^{c_{i+1}}$:

1) Check $\{\mathsf{tx}_{i,j}^{\mathsf{ep_1}}\}_{j=1}^{c_i} \cup \{\mathsf{tx}_{i+1,j}^{\mathsf{ep_1}}\}_{j=1}^{c_{i+1}}$, else send $\mathtt{abort}$ (If KP_k is only in one sub-graph SG_i, then she only needs to check $\{\mathsf{tx}_{i,j}^{\mathsf{ep_1}}\}_{j=1}^{c_i}$).
2) Send $\mathtt{Pre\text{-}Setup\text{-}SG\text{-}OK}$ to all key parites $\{\mathsf{KP}_k\}_{k=1}^{n_{\mathsf{KP}}}$.

KP_i upon receiving $\mathtt{Pre\text{-}Setup\text{-}SG\text{-}OK}$ from all key parties:

1) Send $\mathtt{Pre\text{-}Setup\text{-}UG\text{-}OK}$ to all parties in their respective sub-graphs and go to **Pre-Setup-UG** phase.

$\mathsf{P}_{i,j}$ upon receiving $\mathtt{Pre\text{-}Setup\text{-}UG\text{-}OK}$ from all key parties in SG_i:

1) Go to **Pre-Setup-UG** phase.

Pre-Setup-UG (phase 3):

KP_k in SG_i:

1) Create $\mathsf{tx}_{i,j}^{\mathsf{relay}} := \mathtt{GetTxRelay}(\mathsf{tx}^{\mathsf{ep_1}}, n, \{\mathcal{H}_{\mathsf{SG}_i}\}_{i=1}^{n_{\mathsf{SG}}})$, $1 \le j \le c_i$, according to the received $\{\mathsf{tx}_{i,j}^{\mathsf{ep_1}}\}_{j=1}^{c_i}$.
2) Send $\{\mathsf{tx}_{i,j}^{\mathsf{relay}}\}_{j=1}^{c_i}$ to all other key parties.

KP_k in SG_i upon receiving $\{\mathsf{tx}^{\mathsf{relay}}\}$ from all other key parties:

1) Check $\{\mathsf{tx}^{\mathsf{relay}}\}$, else send $\mathtt{abort}$.
2) Create $\mathsf{tx}^{\mathsf{ep_2}} := \mathtt{GetTxEP2}(\mathsf{tx}, \{\gamma_{i,j}\}_{j=1}^{c_i})$ for each tx in $\{\mathsf{tx}^{\mathsf{relay}}\}$.
3) Send $\{\mathsf{tx}^{\mathsf{relay}}\}$ and $\{\mathsf{tx}^{\mathsf{ep_2}}\}$ to all parties in $\mathbb{P}_i$.
4) Go to **Setup** phase.

$\mathsf{P}_{i,j}$ upon receiving $\{\mathsf{tx}^{\mathsf{relay}}\}$, $\{\mathsf{tx}^{\mathsf{ep_2}}\}$ from all key parties in SG_i:

1) Check $\{\mathsf{tx}^{\mathsf{relay}}\}$, $\{\mathsf{tx}^{\mathsf{ep_2}}\}$ and go to **Setup** phase, else send $\mathtt{abort}$.

Setup (phase 4):

$\mathsf{S}_{i,j}$:

1) Create $\mathsf{tx}_{i,j}^{\mathsf{state}} := \mathtt{GenState}(\alpha_{i,j}, T, \gamma_{i,j})$, and create $\mathsf{tx}_{i,j}^{\mathsf{refund}} := \mathtt{GenTxRefund}(\mathsf{tx}_{i,j}^{\mathsf{state}})$.
2) For all specious output θ in all $\mathsf{tx}_{i,j}^{\mathsf{ep_1}} \cup \mathsf{tx}_{i,j}^{\mathsf{ep_2}}$, create $\mathsf{tx}_{i,j,\theta}^{\mathsf{p_1/p_2}} := \mathtt{GenPay}(\mathsf{tx}^{\mathsf{state}}, \theta)$ and corresponding signature $\sigma_{\mathsf{S}_{i,j}}(\mathsf{tx}_{i,j,\theta}^{\mathsf{p_1/p_2}})$.
3) Send $(\mathsf{tx}_{i,j}^{\mathsf{state}}, \mathsf{tx}_{i,j}^{\mathsf{refund}}, \{\mathsf{tx}_{i,j,\theta}^{\mathsf{p_1/p_2}}\}, \{\sigma_{\mathsf{S}_{i,j}}(\mathsf{tx}_{i,j,\theta}^{\mathsf{p_1/p_2}})\})$ to $\mathsf{R}_{i,j}$.

$\mathsf{R}_{i,j}$ upon receiving transactions and signatures from $\mathsf{S}_{i,j}$:

1) Check received transactions and signatures, else send $\mathtt{abort}$.
2) Send message ($\mathtt{Setup\text{-}SG\text{-}OK}$) to all parties in $\mathbb{P}_i$.

KP_k upon receiving $\mathtt{Setup\text{-}SG\text{-}OK}$ from all $\mathsf{R}_{i,j}$ in $\mathbb{R}_i$:

1) Send message ($\mathtt{Setup\text{-}UG\text{-}OK}$) to all other key parties.

KP_i upon receiving $\mathtt{Setup\text{-}UG\text{-}OK}$ from all other key parties:

1) Send message ($\mathtt{Setup\text{-}UG\text{-}OK}$) to all parties in her one or two sub-graphs, and go to **Confirmation** phase.

$\mathsf{P}_{i,j}$ upon receiving $\mathtt{Setup\text{-}SG\text{-}OK/Setup\text{-}UG\text{-}OK}$ from all parties in SG_i:

1) Go to **Confirmation** phase.

Confirmation (phase 5):

$\mathsf{S}_{i,j}$ and $\mathsf{R}_{i,j}$ in $\gamma_{i,j}$:

1) $\mathtt{updateChannel}(\gamma_{i,j}, \mathsf{tx}_{i,j}^{\mathsf{state}}, t_u)$.
2) if the channel cannot update until the expired time t_u, send $\mathtt{abort}$.
3) $\mathsf{S}_{i,j}$ sends $\{\sigma_{\mathsf{S}_{i,j}}(\mathsf{tx}^{\mathsf{ep_1/ep_2}})\}$ to all receivers, and sends $\mathtt{Confirmation\text{-}OK1}$ to all key parties in SG_i.
4) $\mathsf{R}_{i,j}$ sends $\{\sigma_{\mathsf{R}_{i,j}}(\mathsf{tx}^{\mathsf{ep_2}})\}$ to all receivers and key parties in SG_i.

$\mathsf{R}_{i,j}$ upon receiving $\{\sigma_{\mathsf{S}_{i,j}}(\mathsf{tx}^{\mathsf{ep_1/ep_2}})\}$ from all parties in $\mathbb{S}_i$ and $\{\sigma_{\mathsf{R}_{i,j}}(\mathsf{tx}^{\mathsf{ep_2}})\}$ from all parties in $\mathbb{R}_i$:

1) Check all the signatures, else send $\mathtt{abort}$.
2) Send $\mathtt{Confirmation\text{-}OK1}$ to all parties in SG_i

KP_k in SG_i upon receiving $\mathtt{Confirmation\text{-}OK1}$ from $\mathbb{R}_i \cup \mathbb{S}_i$:

1) Send $\mathtt{Confirmation\text{-}OK1}$ and $\{\sigma_{\mathsf{KP}_k}(\mathsf{tx}^{\mathsf{relay}})\}$ to all key parties.

KP_k in SG_i upon receiving $\mathtt{Confirmation\text{-}OK1}$ and $\{\sigma_{\mathsf{KP}_k}(\mathsf{tx}^{\mathsf{relay}})\}$ from all key parties:

1) Check $\{\sigma_{\mathsf{KP}_j}(\mathsf{tx}^{\mathsf{relay}})\}_{j \ne k}$, else send $\mathtt{abort}$.
2) Send $\sigma_{\mathsf{KP}_k^{\mathsf{fa}}}(\mathsf{tx}^{\mathsf{ep_1}})$ and valid $\{\mathsf{tx}^{\mathsf{relay}}\}$ to all parties in $\mathbb{R}_i$.

$\mathsf{R}_{i,j}$ upon receiving $\sigma_{\mathsf{KP}_k^{\mathsf{fa}}}(\mathsf{tx}^{\mathsf{ep_1}})$ and valid $\{\mathsf{tx}^{\mathsf{relay}}\}$ from all key parties in SG_i:

1) Check all the signatures, else send $\mathtt{abort}$.
2) Send $\mathtt{Confirmation\text{-}OK2}$ to all parties in SG_i

KP_k in SG_i upon receiving $\mathtt{Confirmation\text{-}OK2}$ from $\mathbb{R}_i$:

1) Send $\mathtt{Confirmation\text{-}OK2}$ to all key parties.

KP_k in SG_i upon receiving $\mathtt{Confirmation\text{-}OK2}$ from all key parties:

1) Send $\mathtt{Confirmation\text{-}OK2}$ to all parties in $\mathbb{R}_i$.

$\mathsf{P}_{i,j}$ upon receiving $\mathtt{Confirmation\text{-}OK2}$ from $\mathbb{R}_i$ and all key parties in SG_i:

1) Go to **Finalizing** phase.

Finalizing (phase 6):

$\mathsf{S}_{i,j}$:

1) $\mathtt{updateChannel}(\gamma_{i,j}, \mathtt{GetTxTrans}(\gamma_{i,j}), t_c)$.

$\mathsf{R}_{i,j}$:

1) If the time has expired and $\mathtt{Update\text{-}OK}$ has not been returned from $\mathsf{S}_{i,j}$ and no $\mathsf{tx}^{\mathsf{ep_1/ep_2}}$ is on the blockchain before $T - t_c - 5\Delta$, post $\mathsf{tx}^{\mathsf{in}}$, $\mathsf{tx}^{\mathsf{ep_1}}$ and $\mathsf{tx}^{\mathsf{p_1}}$ with required signature to the blockchain.

Response (runs in every block b_i) (phase 7):

$\mathsf{R}_{i,j}$:

1) If a $\mathsf{tx}^{\mathsf{ep_1}}$ for SG_i is published in the blockchain before $T - t_c - 4\Delta$, run $\mathtt{closeChannel}(\gamma_{i,j}, t_c)$ and wait $3\Delta + t_c$ blocks, then, post the $\mathsf{tx}^{\mathsf{p_1}}$ to the blockchain.
2) or if $b_i < T - t_c - 4\Delta$, a $\mathsf{tx}^{\mathsf{ep_1}}$ for $\mathsf{SG}_k (k \ne i)$ is published in the blockchain and there is no $\mathsf{tx}^{\mathsf{relay}}$ in the blockchain, post the corresponding $\mathsf{tx}^{\mathsf{relay}}$ into the blockchain and run $\mathtt{closeChannel}(\gamma_{i,j}, t_c)$. After $\mathsf{tx}^{\mathsf{relay}}$ being published in the blockchain, publish the corresponding $\mathsf{tx}^{\mathsf{ep_2}}$ for SG_i into the blockchain. After the $\mathsf{tx}^{\mathsf{ep_2}}$ and $\mathsf{tx}^{\mathsf{state}}$ being published in the blockchain, wait for the timelock conditions to be satisfied, and then post the $\mathsf{tx}^{\mathsf{p_2}}$ to the blockchain.
3) or if a $\mathsf{tx}^{\mathsf{ep_2}}$ for SG_i is published in the blockchain before $T - t_c - 2\Delta$, run $\mathtt{closeChannel}(\gamma_{i,j}, t_c)$ and wait $\Delta + t_c$ blocks, then, post the $\mathsf{tx}^{\mathsf{p_2}}$ to the blockchain.

$\mathsf{S}_{i,j}$:

1) If $b_i >= T$, the channel is closed and $\mathsf{tx}^{\mathsf{state}}$ is published in the blockchain, post the $\mathsf{tx}^{\mathsf{refund}}$ into the blockchain.

Fig. 7: SAMCU protocol.

all channel senders and receivers, she can meet the signature condition $\mathsf{MultiSig}(\mathbb{S}_i)$ of her $\mathsf{tx}^{\mathsf{ep}_1}$ and meet all the signature conditions of her $\mathsf{tx}^{\mathsf{ep}_2}$, so she sends `Confirmation-OK1` to all parties within the same sub-graph. When a key party gets message `Confirmation-OK1` from all channel receivers in her one or two sub-graphs, she will send `Confirmation-OK1`, the signatures of $\{\mathsf{tx}^{\mathsf{relay}}\}$, to all other key parties. After that, when a key party receives `Confirmation-OK1` from all key parties, it means that all channels from other sub-graphs are successfully updated. Then, the key party signs each $\mathsf{tx}^{\mathsf{ep}_1}$ and sends it and valid $\{\mathsf{tx}^{\mathsf{relay}}\}$ with all signatures to the corresponding channel receiver to all parties in her sub-graphs. When a receiver gets the signatures from all key parties with the same sub-graph, she is able to meet the second condition of her $\mathsf{tx}^{\mathsf{ep}_1}$ on the blockchain i.e., $\mathtt{MultiSig}(\{\mathsf{KP}_k^{\mathsf{fa}}\}_{\forall\, \mathsf{KP}_k \in \mathsf{SG}_i})$. Subsequently, a two-step confirmation process ensures that each party knows that all channel receivers have valid $\mathsf{tx}^{\mathsf{ep}_1}$ before proceeding to the next phase.

6) Finalization: After the last phase, every channel sender can ensure that all channels in UG are ready to be updated and all channel receivers have valid $\mathsf{tx}^{\mathsf{ep}_1}$. Then, she can start updating her channel with the channel receiver off-chain. In case a PC fails to be updated, the channel receiver can post her own $\mathsf{tx}^{\mathsf{in}}$, $\mathsf{tx}^{\mathsf{ep}_1}$, and $\mathsf{tx}^{\mathsf{p}_1}$ to enforce payment.

7) Response: All participants should watch the blockchain's status and respond accordingly when new blocks are created on the blockchain. Each channel receiver needs to watch which transactions have appeared on the blockchain. In the event that a *Trigger* transaction appears in the blockchain, she needs to post the corresponding *response* transactions to the blockchain to forcibly redeem her funds in the payment channel following the corresponding EPTT if there is a dispute in her channel. Each channel sender also needs to actively watch the blockchain. If her receiver has not redeemed her funds by posting $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ before T, the sender needs to post $\mathsf{tx}^{\mathsf{refund}}$ to redeem her funds after T.

V. Security Analysis

A. Balance Security

Theorem 1. *No honest intermediate party within SAMCU loses her funds.*

Proof. We first define four states for payment channels: (a) We define a channel as *unchanged* when its state remains unaffected by the current update, typically occurring when a participant sends a message `abort` before the confirmation phase. (b) We define a channel as *revoked* when the channel sender uses $\mathsf{tx}^{\mathsf{state}}$ to update the channel's state during the confirmation phase and subsequently redeems her funds from the channel using a $\mathsf{tx}^{\mathsf{refund}}$ transaction after time T. (c) We define a channel as *compensated* when the channel sender uses $\mathsf{tx}^{\mathsf{state}}$ to update the channel's state during the confirmation phase and the receiver redeems her funds from the channel using $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ before time T. (d) We define a channel as *successful* when the channel sender uses $\mathsf{tx}^{\mathsf{trans}}$ to update the channel's state during the finalization phase. For an honest intermediary $\mathsf{P}_{i,j}$, we consider her upstream channel (where she acts as the receiver) and downstream channel (where she acts as the sender) as γ_l and γ_r, respectively. There are three scenarios that can compromise the balance security of $\mathsf{P}_{i,j}$:

- **γ_l is unchanged, γ_r is successful or compensated.** When the upstream channel γ_l remains unchanged, the rational channel receiver of γ_l will not send `Confirmation-OK2` to other parties. As the channel sender of γ_r, she will not sign any $\mathsf{tx}^{\mathsf{ep}_1/\mathsf{ep}_2}$, thus γ_r cannot be compensated. As the channel sender of γ_r, she also will not honestly update the state of γ_r off-chain. This leads to a contradiction.
- **γ_l is revoked, γ_r is successful.** A rational $\mathsf{P}_{i,j}$ will not honestly update γ_r off-chain when she lacks the ability to enforce payment from γ_l. This leads to a contradiction.
- **γ_l is revoked, γ_r is compensated.** γ_l being revoked implies that $\mathsf{P}_{i,j}$ cannot enforce the payment independently (i.e., she does not possess a valid transaction $\mathsf{tx}^{\mathsf{ep}_1}$). γ_r being compensated indicates that the channel receiver of γ_r has already published a valid $\mathsf{tx}^{\mathsf{ep}_1/\mathsf{ep}_2}$. The legitimacy of these signatures requires $\mathsf{P}_{i,j}$'s signature, implying that $\mathsf{P}_{i,j}$ is aware of these transactions and their children transactions. If they were not aware, they would send an `abort` to others before the confirmation phase.

  Then, if $\mathsf{P}_{i,j}$ is a normal party, she can initiate her own $\mathsf{tx}^{\mathsf{p}_1/\mathsf{p}_2}$ corresponding to $\mathsf{tx}^{\mathsf{ep}_1/\mathsf{ep}_2}$ on the blockchain to redeem funds from γ_l. Otherwise, if $\mathsf{P}_{i,j}$ is a key party, two scenarios exist: (a) γ_r was compensated by $\mathsf{tx}^{\mathsf{ep}_1}$ and $\mathsf{tx}^{\mathsf{p}_1}$. In this situation, $\mathsf{P}_{i,j}$ can initiate the corresponding $\mathsf{tx}^{\mathsf{relay}}$, $\mathsf{tx}^{\mathsf{ep}_2}$, and $\mathsf{tx}^{\mathsf{p}_2}$ on the blockchain to redeem funds from γ_l. These transactions must be valid since the key party only signs $\mathsf{tx}^{\mathsf{ep}_1}$ if its child transactions are valid according to the procedure of the protocol. (b) γ_r was compensated by $\mathsf{tx}^{\mathsf{ep}_2}$ and $\mathsf{tx}^{\mathsf{p}_2}$. In this situation, the parent transaction $\mathsf{tx}^{\mathsf{relay}}$ of $\mathsf{tx}^{\mathsf{ep}_2}$ must also be on the blockchain. Then, $\mathsf{P}_{i,j}$ can initiate $\mathsf{tx}^{\mathsf{ep}_2}$ for their left SG and $\mathsf{tx}^{\mathsf{p}_2}$ following the already-on-chained $\mathsf{tx}^{\mathsf{relay}}$ on the blockchain to redeem funds from γ_l. Hence, this creates a contradiction as γ_l cannot be revoked.

All possible scenarios that compromise the balance security of an honest intermediary are contradictory. Therefore, we infer that no honest intermediary loses her funds within SAMCU. ■

B. Internal Anonymity

Theorem 2. *SAMCU achieves internal anonymity when it runs in the honest case.*

Sketch Proof. When the protocol runs without disputes, each key party only knows the public addresses of parties located in her SG(s), as well as the fresh addresses of other KPs. Similarly, every normal party only knows the public addresses of other parties within her SG and the fresh addresses of all KPs. However, as long as the protocol is executed honestly, the fresh addresses are not included on the blockchain, eliminating concerns about the possibility of linking fresh addresses to the public addresses of KPs. In summary, each party is only aware of the public addresses belonging to the parties in its neighborhood. Thus, we conclude that the SAMCU protocol achieves internal anonymity when the protocol runs without disputes. ■

TABLE II: Theoretical comparisons between SAMCU and other related protocols.

Scheme	Time Complexity		Computational Complexity		Communication Complexity						Transaction Fee&Size
					Connections			Data Size			
	Setup Time	Compensation Time	Transactions Generation	Signatures Generation	Dealer	Party	Overall	Dealer	Party	Overall	
LN	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(1)$
AMCU	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n)$
PT	$O(\log n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	$O(n^2)$	$O(\log n)$
Blitz	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n)$
Thora	$O(1)$	$O(1)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^3)$	$O(n)$
SAMCU	$O(1)$	$O(1)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(\frac{n}{m}+k)$	$O(\frac{n^2}{m}+k^2)$	$O(n^2)$	$O(n^2)$	$O(n^3)$	$O(\frac{n}{m}+m)$

VI. Evaluation

A. Theoretical Analysis

We conduct a theoretical analysis between SAMCU and other comparisons (as shown in Table II). In this table, n represents the number of participating payment channels in all protocols. m and k represent the number of sub-graphs and the number of key parties in SAMCU. We assume that the n channels can be evenly distributed among the m sub-graphs in SAMCU.

Compared to solutions for MHPs (i.e., LN, PT, Blitz), SAMCU achieves higher expressiveness (i.e., MCUs). In addition, SAMCU achieves constant setup time (i.e., generating all transactions within a finite number of rounds) and constant collateral time (i.e., parties can withdraw funds locked in transactions within constant time). On the other hand, to achieve constant setup and collateral times, like other solutions for MCUs (i.e., Thora), SAMCU incurs higher communication and computational costs compared to MHPs solutions to ensure secure synchronization within constant running time. Both communication and computation expenses in SAMCU increase complexity by one degree compared to solutions for MHPs. Furthermore, SAMCU employs transactions of size $O(\frac{n}{m}+m)$ instead of $O(1)$ for enforcing payment.

However, compared to the SOTA for MCUs (i.e., Thora), SAMCU offers the following improvements: (a) **Number of Connections**: Thora requires all parties to establish connections with every other party, resulting $O(n^2)$ connections. In our approach, during the initialization phase, the `dealer` establishes n connections with all parties involved in SAMCU. As the progresses from the pre-setup to the confirmation phase, all parties need pairwise connections within each SG, totaling $O(\frac{n^2}{m})$ connections. Communication between groups requires pairwise connections between key parties, totaling $O(k^2)$ connections. Thus, the overall number of connections is $O(\frac{n^2}{m}+k^2)$; (b) **On-Chain Transaction Size&Fee**: In protocols for MCUs, parties collaboratively create transactions to ensure their balance security, with the sizes of these transactions outlined in Table III. When disputes arise in channels, in Thora, disputing users must emit $\mathsf{tx}^{\mathsf{in}}$, $\mathsf{tx}^{\mathsf{ep}}$, and tx^{p} to the blockchain, totaling 988 + $36n$ bytes, with a complexity of $O(n)$. In SAMCU, the receiver of the first disputed channel needs to emit $\mathsf{tx}^{\mathsf{in}}$, $\mathsf{tx}^{\mathsf{ep}_1}$, and $\mathsf{tx}^{\mathsf{p}_1}$ to the blockchain, while parties in other SGs need to emit $\mathsf{tx}^{\mathsf{relay}}$, $\mathsf{tx}^{\mathsf{ep}_2}$, and $\mathsf{tx}^{\mathsf{p}_2}$ to the blockchain. The total size of transactions for enforcing payment in SAMCU are $939 + 36\frac{n}{m}$ and $842 + 36(\frac{n}{m}+m)$ bytes, respectively, with a complexity of $O(\frac{n}{m}+m)$.

Our approach does not optimize computational complexity or the data size of communication. Although the size of individual transactions is reduced, the overall size of the EPTTs remains comparable to Thora, still at $O(n^2)$. Since most of the computational time is spent generating transactions and most of the communication data consists of these transactions, both the computational time complexity and the complexity of communication data size per person are consistent with the complexity of total transaction size, remaining at $O(n^2)$, the same as Thora.

TABLE III: Comparing transaction sizes (byte).

Methods	$\mathsf{tx}^{\mathsf{in}}$	$\mathsf{tx}^{\mathsf{ep}/\mathsf{ep}_1}$	$\mathsf{tx}^{\mathsf{relay}}$	$\mathsf{tx}^{\mathsf{ep}_2}$	$\mathsf{tx}^{\mathsf{p}/\mathsf{p}_1/\mathsf{p}_2}$	$\mathsf{tx}^{\mathsf{state}}$	$\mathsf{tx}^{\mathsf{refund}}$
Thora	223	$256+36n$	null	null	500	368	272
SAMCU	223	$162+36(n/m+1)$	$162+36m$	$162+36(n/m)$	500	368	272

* $\mathsf{tx}^{\mathsf{ep}/\mathsf{ep}_1/\mathsf{ep}_2}$ and $\mathsf{tx}^{\mathsf{relay}}$ require an extra 9 bytes to support CPFP.

B. Experimental Results

We first implement a prototype of SAMCU in Python 3. We conduct a series of experiments aimed at (a) evaluating the size&fee of on-chain transactions, (b) evaluating the computation time, and (c) evaluating the communication connections, using a machine with Quad-Core Intel Core i5 (1.4 GHz).

On-chain Transaction Size and Fee. We vary the number of PCs from 20 to 100 and illustrate the total transaction size for a disputing user to enforce payment in the different numbers of SGs in Fig. 8. The result shows the total transaction size for a disputing user to enforce payment rises nearly linearly in all cases as the number of PCs increases. For the same number of PCs, the total transaction size for a user to enforce payment decreases as the number of SGs increases. In the current Bitcoin environment, recording 500 bytes of data requires a transaction fee of 1 USD. When n is 100 and m is 10, the on-chain transaction fees significantly decrease from 9.2 USD to 3.1 USD, reducing the cost of enforcing payments compared to Thora.

Computation Time. We vary the number of PCs from 20 to 100 and illustrate the computational time for a key party with different numbers of SGs in Fig. 9. The results show that the computational time for a key party exhibits quadratic growth in all cases as the number of PCs increases. For the same number of PCs, the computational time for a key party decreases as the number of SGs increases. When updating 100 channels, the total computational time for a key party is less than 1.5s.

Communication Connections. We vary the number of PCs from 20 to 100 and illustrate the number of communication connections in the different numbers of SGs in Fig. 10. In all cases, there are 3 chains in UG. The result shows that the total number of communication connections exhibits quadratic growth in all cases as the number of PCs increases. For the same number of PCs, the total number of communication connections

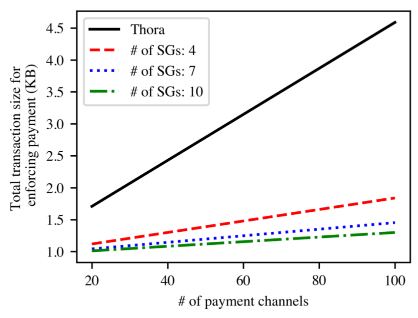


Fig. 8: Comparing total transaction sizes for enforcing payment between SAMCU and Thora.

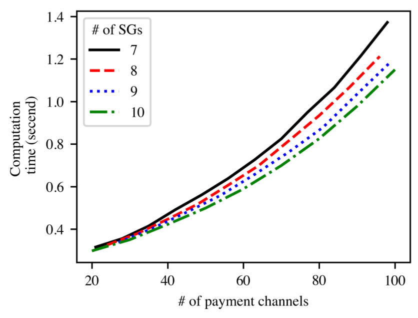


Fig. 9: Computation time for varying numbers of channels.

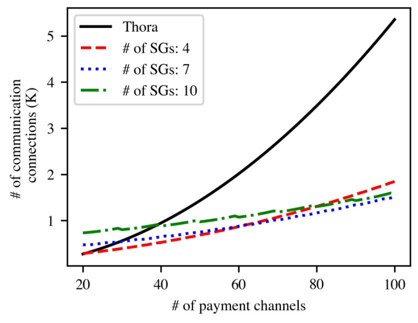


Fig. 10: Comparing the number of communication connections between SAMCU and Thora.

decreases as the number of SGs increases. When n is 100, m is 10, the connection count decreases from 5.35K (in Thora) to 1.57K (in SAMCU), signifying a reduction of 70%. This enhancement renders the protocol more practical for large-scale MCUs.

VII. Related Work

Anonymization techniques have been widely used to protect privacy in various domains (e.g., distributed learning [17], data sharing [18, 19], and person re-identification [20]). Although anonymity is a key feature of blockchain technology, external observers can still correlate different transactions made by the same user through her account addresses, ultimately tracing user behavior to de-anonymize the user entirely [21]. In recent years, significant efforts have been devoted to enhancing identity privacy on blockchains. In addition to off-chain PCNs for achieving anonymity, anonymous cryptocurrencies (e.g., Monero [22] and Zcash [23]) use advanced cryptographic techniques, such as Linkable Ring Signatures [22] and zk-SNARKs [23], to provide greater transaction anonymity than Bitcoin.

Another technique for achieving anonymity is through third-party mixing services. In this approach, Senders send their coins to a mixing pool, and the service redistributes the coins among receivers to obscure the relationship between senders and receivers. CoinJoin [24] is one of the first methods to provide a mixing service for Bitcoin, where multiple users combine their Bitcoin transactions into a single transaction. To eliminate the need for trusted users in CoinJoin, CoinShuffle [25] employs an anonymous group communication protocol to conceal the participants' identities from each other.

VIII. Conclusion

In this paper, we propose SAMCU, a secure and anonymous multi-channel update protocol for PCN. Unlike traditional solutions that sacrifice privacy for multi-channel updates, SAMCU achieves both multi-channel updates and sender/receiver privacy. Our SAMCU has lower on-chain transaction fees per person and fewer communication connections compared to previous work. For instance, when simultaneously updating 100 channels, our approach yields around 70% savings in communication connections and reduces on-chain transaction fees by about 66% relative to the state-of-the-art. Our SAMCU makes multi-channel updates more practical, particularly in large-scale multi-channels. In future work, we intend to enhance the interoperability of SAMCU, enabling secure and privacy-preserving multi-channel updates across different blockchains.

References

[1] 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*, 2017, pp. 455–471.
[2] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 801–815.
[3] M. Jourenko, M. Larangeira, and K. Tanaka, "Payment trees: Low collateral payments for payment channel networks," in *Financial Cryptography and Data Security*. Springer, 2021, pp. 189–208.
[4] L. Aumayr, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Blitz: Secure multi-hop payments without two-phase commits," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 4043–4060.
[5] L. Aumayr, K. Abbaszadeh, and M. Maffei, "Thora: Atomic and privacy-preserving multi-channel updates," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, p. 165–178.
[6] "BIP 0016," 2012, https://en.bitcoin.it/wiki/BIP_0016.
[7] "Eclair network," https://github.com/ACINQ/eclair.
[8] G. Malavolta, P. A. Moreno-Sánchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.
[9] X. Wang, C. Lin, X. Huang, and D. He, "Anonymity-enhancing multi-hop locks for monero-enabled payment channel networks," *IEEE Transactions on Information Forensics and Security*, pp. 2438–2453, 2023.
[10] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *International conference on financial cryptography and data security*. Springer, 2019, pp. 508–526.
[11] P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks," in *Proceedings 2017 Network and Distributed System Security Symposium*, 2017, pp. 1–15.
[12] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *Proceedings 2017 Network and Distributed System Security Symposium*, 2017, pp. 1–15.
[13] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: efficient dynamic routing for offchain networks," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 370–381.
[14] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 777–796.

[15] "Bitcoin visuals," https://bitcoinvisuals.com/ln-eccentricity/.
[16] "Child Pays For Parent," https://bitcoincore.org/en/releases/0.13.0/#mining-transaction-selection-child-pays-for-parent.
[17] Y. Jiang, K. Zhang, Y. Qian, and L. Zhou, "Anonymous and efficient authentication scheme for privacy-preserving distributed learning," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2227–2240, 2022.
[18] J. Sun, G. Xu, T. Zhang, X. Yang, M. Alazab, and R. H. Deng, "Privacy-aware and security-enhanced efficient matchmaking encryption," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 4345–4360, 2023.
[19] ——, "Verifiable, fair and privacy-preserving broadcast authorization for flexible data sharing in clouds," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 683–698, 2023.
[20] M. Ye, W. Shen, J. Zhang, Y. Yang, and B. Du, "SecureReID: Privacy-preserving anonymization for person re-identification," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2840–2853, 2024.
[21] H. Yousaf, G. Kappos, and S. Meiklejohn, "Tracing transactions across cryptocurrency ledgers," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 837–850.
[22] N. Van Saberhagen, "Cryptonote v 2.0," 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:2711472
[23] "Zcash," 2016, https://z.cash/.
[24] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world," Aug. 2013. [Online]. Available: https://bitcointalk.org/?topic=279249
[25] T. Ruffing and P. Moreno-Sanchez, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *European Symposium on Research in Computer Security*, vol. 8713, 2014, pp. 345–364.
[26] A. Mizrahi and A. Zohar, "Congestion attacks in payment channel networks," in *International Conference on Financial Cryptography and Data Security*. Springer, 2021, pp. 170–188.

Jianhuan Wang is currently pursuing his Ph.D. degree in the Department of Computing, the Hong Kong Polytechnic University under the supervision of Dr. Bin Xiao. He received his B.Eng. degree from the Jinan University. He received his M.S. degree from the Department of Computing, the Hong Kong Polytechnic University. His research interest lies in the blockchain security and DID security.

Shang Gao is currently a research assistant professor in the Department of Computing at the Hong Kong Polytechnic University. He obtained his B.S. degree from Hangzhou Dianzi University, China, in 2010, followed by an M.E. degree from Southeast University, China, in 2014. In 2019, he received his Ph.D. degree at the Hong Kong Polytechnic University. After graduation, he worked in Microsoft China for one year. Dr. Gao's primary research focus lies within the field of computer security and information security. He specifically specializes in applied cryptography and zero-knowledge proofs, with a keen interest in areas such as RingCT protocols, zkSNARKs, Layer2 security, and recursive SNARKs. His work has been published in many top-tier conferences and journals, including IEEE S&P, ACM CCS, IEEE INFOCOM, IEEE/ACM TON, IEEE TDSC, etc.

Guyue Li received the B.S. degree in information science and technology and the Ph.D. degree in information security from Southeast University, Nanjing, China, in 2011 and 2017, respectively.,From June 2014 to August 2014, she was a Visiting Student with the Department of Electrical Engineering, Tampere University of Technology, Tampere, Finland. She is currently an Associate Professor with the School of Cyber Science and Engineering, Southeast University, and a Visiting Scholar with the Tampere University of Technology, and Université Gustave Eiffel (ESIEE PARIS), Noisy-le-Grand, France. Her current research interests include wireless network attacks, physical-layer security solutions for 5G and 6G, secret key generation, radio frequency fingerprints, and reconfigurable intelligent surfaces.,Dr. Li was a recipient of the Young Scientist Awarded by the International Union of Radio Science (URSI), the Youth Science and Technology Prize of Jiangsu Cyber Security Association, and the A-Level Zhishan Scholar of Southeast University. She has been the Workshop Co-Chair of IEEE Conference on Vehicular Technology (VTC) from 2021 to 2022. She is currently serving as an Editor for IEEE Communication Letters and an Associate Editor for EURASIP Journal on Wireless Communications and Networking.

Keke Gai received the B.Eng. degree in automation from the Nanjing University of Science and Technology, Nanjing, China, in 2004, the M.E.T. degree in educational technology from The University of British Columbia, Vancouver, BC, Canada, in 2010, the M.B.A. degree in business administration and the M.S. degree in information technology from Lawrence Technological University, Southfield, MI, USA, in 2009 and 2014, respectively, and the Ph.D. degree in computer science from Pace University, New York, NY, USA. He is currently an Associate Professor with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He has published three books and more than 130 peer-reviewed journal articles/conference papers, including 8 ESI Highly Cited Papers (2020). His research interests include cyber security, blockchain, edge computing, cloud computing, and reinforcement learning. He received five IEEE Best Paper Awards (e.g., TrustCom 2018 and HPCC 2018) and two IEEE Best Student Paper Awards in recent five years.

Bin Xiao is a professor at the Department of Computing, the Hong Kong Polytechnic University, Hong Kong. Prof. Xiao received B.Sc and M.Sc degrees in Electronics Engineering from Fudan University, China, and a Ph.D. degree in computer science from the University of Texas at Dallas, USA. His research interests include AI and network security, data privacy, Web3, and blockchain systems. He published more than 200 technical papers in international top journals and conferences. He is currently an Associate Editor of IEEE Transactions on Cloud Computing and IEEE Transactions on Network Science and Engineering. He has been the associate editor of the IEEE Internet of Things Journal and Elsevier Journal of Parallel and Distributed Computing. He is the chair of the IEEE ComSoc CISTC committee from 2024 to 2025. He has been the track co-chair of IEEE ICDCS2022, the symposium track co-chair of IEEE Globecom 2024, ICC2020, ICC 2018, and Globecom 2017, and the general chair of IEEE SECON 2018. He is a Fellow of IEEE.