GMHL: Generalized Multi-Hop Locks for Privacy-Preserving Payment Channel Networks

Zilin Liu, Anjia Yang, Jian Weng, Tao Li, Huang Zeng, and Xiaojian Liang

College of Cyber Security, Jinan University, Guangzhou, China
zilinliu.ariel@gmail.com, anjiayang@gmail.com, cryptjweng@gmail.com, Xngzelt@gmail.com, huagzeg@gmail.com, im.liangxj@gmail.com

Abstract. Payment channel network (PCN), not only improving the transaction throughput of blockchain but also realizing cross-chain payment, is a very promising solution to blockchain scalability problem. Most existing PCN constructions focus on either atomicity or privacy properties. Moreover, they are built on specific scripting features of the underlying blockchain such as HTLC or are tailored to several signature algorithms like ECDSA and Schnorr. In this work, we devise a Generalized Multi-Hop Locks (GMHL) based on adaptor signature and randomizable puzzle, which supports both atomicity and privacy preserving (unlinkability). We instantiate GMHL with a concrete design that relies on a Guillou-Quisquater-based adaptor signature and a novel designed RSA-based randomizable puzzle. Furthermore, we present a generic PCN construction based on GMHL, and formally prove its security in the universal composability framework. This construction only requires the underlying blockchain to perform signature verification, and thus can be applied to various (non-/Turing-complete) blockchains. Finally, we simulate the proposed GMHL instance and compare with other protocols. The results show that our construction is efficient comparable to other constructions while remaining the good functionalities.

Keywords: Generalized Multi-hop Locks · Payment Channel Network · Privacy Preserving · Blockchain.

1 Introduction

In recent years, the craze of blockchain has swept the world and people are increasingly paying attention to it. A great number of applications (e.g., [28], [31], [10], [3]) based on blockchain have sprung up. Particularly, the appearance of many blockchain-based cryptocurrencies (e.g., Bitcoin[25], Ethereum[4]) has made decentralized payments come to reality. Unlike traditional centralized payment, the confirmation of transactions does not rely on a centralized payment system (e.g., a bank), but instead a public distributed ledger owned by multiple parties on the blockchain. This transaction mode has several advantages, such as high transparency, easy transmission and freedom from inflation.

However, cryptocurrencies are still suffering the scalability problem, which prevents them from playing a bigger role. The scalability problem includes two aspects: on the one hand, the confirmation of a transaction has a high latency, which limits the transaction throughput of blockchain; on the other hand, it is difficult for assets or data to interact between different blockchains, which limits cross-currency payments.

Plenty of research (e.g., [19], [22], [11], [26]) has been conducted to overcome the scalability issues. Among them, Payment Channel (PC) is a very promising solution. In simple terms, a payment channel works as following procedures: Alice and Bob first jointly generate an on-chain transaction and pledge a certain amount of coins as deposits on it to open a channel. Then they can make multiple local transactions meanwhile updating the balance of the channel with no need of reporting these records to blockchain. Before the channel expires, they close the channel by issuing an on-chain transaction to allocate their deposits. PC significantly alleviates the problem of low throughput and high transaction fees on blockchain. It has grown into two branches: payment channel hub (PCH) and payment channel network (PCN). PCH is a connection of two payment channels that allows for payments between sender and receiver through an intermediary. And several constructions (e.g., [14], [13], [15], [27]) for PCH are proposed over the years. Different from PCH, PCN is a connection of multiple payment channels that allows for payments between sender and receiver through multiple intermediaries, which is the focus of this paper. A fundamental requirement upon building a PCN is to support atomicity that means all the payments are either successful or returned back to the participators, which can effectively prevent wormhole attacks. In addition, participators may not want to disclose their identities or even be linked from different transactions. Finally, each user may own multiple cryptocurrency accounts and it is desirable to propose a generic PCN construction that can be applied to various blockchains for cross-chain payments.

State of the art in PCNs. Poon et al. [26] firstly introduced a PCN proposal, namely Hash Time-Lock Contracts (HTLC) which is based on a special scripting feature of blockchain. It is the cornerstone of future research, but it does not consider atomicity (e.g., vulnerable to wormhole attacks). In order to ensure atomicity, a number of researchers improved the HTLC with new techniques. Miller et al. [23] introduced a PCN called sprites. It ensures atomicity, reduces the collateral and improves the liveness of PCN with the help of smart contract. Jourenko et al. [17] proposed a PCN construction, relying on a technique called payment tree, that provides atomicity and low collateral. Aumayr et al. [2] obtains atomicity and low collateral by improving the PCN construction with a punishment mechanism. Nevertheless, the aforementioned protocols do not consider the privacy-preserving problem, such as the payment path is exposed directly or the transactions can be linked since they use the same hash value. To address this issue, Malavolta et al. [20] proposed the first provable privacy-preserving protocol for PCN, but the high-level security requires relatively expensive cost in terms of computation and communication overhead. After that, Tripathy et al. [30] put forward an efficient privacy-preserving PCN relying on Elliptic curve based Time-Lock Contract (ETLC), and Mohanty et al.

Construction	Atomicity	Unlinkability	Generality	Lightweight setup	Required functionality
HTLC [26]	○	○	○	•	HTLC
FC'19 [23]	•	○	○	•	smart contract
USENIX Sec'21 [2]	•	○	○	•	UTXO-based
CCS'17 [20]	•	•	○	○	HTLC
FC'20 [30]	•	•	○	•	ETLC
NDSS'19 [21]	•	•	○	○	ECDSA/Schnorr
SP'21 [29]	•	•	•	○	MPC and signature verification
Our construction	•	•	•	•	signature verification

Table 1. A comparison of state of the art in PCNs

[24] proposed an efficient and privacy-preserving PCN with an enhanced HTLC protocol named n-HTLC. Malavolta et al. [21] proposed a privacy-preserving PCN construction, based on a novel cryptographic primitive named anonymous multi-hop locks (AMHL) that employs homomorphic one-way functions, zero-knowledge protocols, and commitment schemes. In this construction, the sender needs to involve a setup phase with heavy computation, since it needs to compute locks (i.e., some calculations with homomorphic properties) for all the intermediate nodes on the path. Since the previously mentioned protocols are built on specific scripting features of the underlying blockchain such as HTLC or tailored to several signature algorithms like ECDSA and Schnorr, they can only be applied to some specific blockchains. Latter researchers pondered how to propose a PCN construction that considers atomicity, unlinkability, and generality of PCN simultaneously. Thyagarajan et al. [29] proposed a generic PCN construction using lockable signatures and gave an efficient instantiation based on BLS signatures. However, the sender is required to create a 3-party local channel with all the other participators during setup phase, which may induce heavy computation. Moreover, they need to post 2 transactions on the blockchain to close the channel.

In this paper, we propose a Generalized Multi-hop Locks (GMHL) based on adaptor signature and randomizable puzzle. Different from AMHL, the computation load in GMHL is amortized by all the participators in the path. Furthermore, we present a generic PCN construction based on GMHL that enjoys all the benefits of atomicity, privacy preserving, and generality, meanwhile with a lightweight setup phase. Our construction does not rely on advanced scripts, and the main additional operation for underlying blockchain is signature verification. As a consequence, it can be applied to various (including non-/Turing-complete) blockchains.

Our contributions. The contributions of our work can be summarized as follows:

– We devise a Generalized Multi-hop Locks, denoted by GMHL, based on adaptor signature and randomizable puzzle. It supports both atomicity and privacy preserving (unlinkability). Besides, we show how to instantiate GMHL by giving a concrete protocol built on a Guillou-Quisquater-based adaptor signature and a proposed novel RSA-based randomizable puzzle.

– We present a generic PCN construction based on GMHL. It only requires the underlying blockchain to perform signature verification, and thus can be applied to various blockchains. As shown in Table 1, compared with the general construction in SP'21 [29], our PCN construction has a lightweight setup phase in sense that the puzzles are generated by each of the participators instead of the sender. We formally prove the security of this construction in the Universal Composability (UC) framework and show that our construction satisfies the basic security properties atomicity and unlinkability. Lastly, since the cost of our PCN is dominated by the calls to GMHL, we simulate the instance and compare with other protocols about the computation cost. The results show that our protocol is efficient comparable to other protocols while remaining the good functionalities.

1.1 Organization

We introduce the background and preliminaries in Section 2 and give out the security definitions in Section 3. Then, we introduce our solution overview in Section 4, and describe the PCN construction in Section 5. Next, we analyze the security of the PCN construction in Section 6 and simulate the proposed protocol in Section 7. Finally, we conclude the paper in Section 8.

2 Background and Preliminaries

In this section, we describe the background and the preliminaries that will be used in this paper.

2.1 Payment Channel Network (PCN)

A PCN can be described as a directed graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where the set $\mathbb{V}$ of vertices represents the user accounts and the set $\mathbb{E}$ of weighted edges represents the payment channels. The non-negative number associated with the vertex $U \in \mathbb{V}$ denotes the fees it charges for forwarding a payment. The weight of a directed edge $(U_1, U_2) \in \mathbb{E}$ denotes the amount of remaining coins that U_1 can pay to U_2. A payment channel network (PCN) is used to perform transactions between two users without a directly payment channel. Assume that sender S wants to pay α coins to receiver R through a path $S \to U_1 \to ... \to U_n \to R$. Each user U_i on this path must have a capacity $\gamma_i \geq \alpha'_i$ where $\alpha'_i = \alpha - \sum_{k=1}^{i-1} fee(U_k)$ to ensure the payment can be successfully completed. Thus S starts the payment with α^* coins where $\alpha^* = \alpha + \sum_{k=1}^{n} fee(U_k)$ to guarantee that R will receive exactly α coins. We refer readers to [20] for further details.

2.2 Preliminaries

Notations. We denote by $1^\lambda \in \mathbb{N}^+$ the security parameter and $x \xleftarrow{\$} S$ the uniformly sampling of an element from a set S, respectively. We use the notation

$y \leftarrow A(x)$ to denote that inputs x to a probabilistic polynomial time (PPT) algorithm A and outputs y, and use the notation $y := A(x)$ when the algorithm A is a deterministic polynomial time (DPT) algorithm.

(Non-)interactive zero-knowledge. We denote by R an NP relation and L a set of positive instances corresponding to the relation R where $L = \{x|\exists w \; s.t. \; R(x, w) = 1\}$, respectively. A non-interactive zero-knowledge proof scheme NIZK consists of two algorithms: a prover algorithm $\pi \leftarrow \mathsf{P}_{\mathsf{NIZK}}(x, w)$ and a verifier algorithm $\{0, 1\} := \mathsf{V}_{\mathsf{NIZK}}(x, \pi)$. The NIZK scheme ensures that the prover can prove to the verifier that he does know x without revealing additional knowledge to the verifier. We model the security of a NIZK scheme by an ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ in Appendix and refer readers to [5] for the definition of security of zero-knowledge functionality in the UC framework.

Adaptor signature scheme. An adaptor signature scheme is defined with respect to a hard relation R and a digital signature scheme Σ. It consists of four algorithms $\Xi_{R,\Sigma} = (\mathsf{PreSig}, \mathsf{PreVf}, \mathsf{Adapt}, \mathsf{Ext})$. With a statement/witness pair $(Y, y) \in R$, a secret/public key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda)$ and a message $m \in \mathcal{M}$, we can generate a pre-signature with $\hat{\sigma} \leftarrow \mathsf{PreSig}(\mathsf{sk}, m, Y)$, adapt a valid signature with $\sigma := \mathsf{Adapt}(\hat{\sigma}, y)$, verify a pre-signature with $\mathsf{PreVf}(m, Y, \hat{\sigma})$ and extract the witness with $y := \mathsf{Ext}(\sigma, \hat{\sigma}, Y)$. Adaptor signature was formally defined in [1]. An adaptor signature is secure if it provides pre-signature correctness, pre-signature adaptability and witness extractability. Briefly, pre-signature signature correctness ensures that any honestly generated pre-signature $\hat{\sigma}$ with respect to a statement Y must be valid and the adapted signature σ from it is valid as well. Pre-signature adaptability ensures that any valid pre-signature $\hat{\sigma}$ can be adapted into a valid signature σ with the witness y. Witness-extractability ensures that a corresponding witness y can be extracted from a valid pre-signature/signature pair $(\hat{\sigma}, \sigma)$.

Randomizable puzzle. A randomizable puzzle scheme RP consists of four algorithms $RP = (\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSlove}, \mathsf{PRand})$. With a public parameters/-trapdoor pair $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PSetup}(1^\lambda)$, we can generate a puzzle $Z \leftarrow \mathsf{PGen}(\mathsf{pp}, \zeta)$, solve the puzzle with $\zeta := \mathsf{PSolve}(\mathsf{td}, Z)$ and randomize the puzzle to a fresh puzzle with $(Z', r) \leftarrow \mathsf{PRand}(\mathsf{pp}, Z)$ which $\phi(\zeta, r)$ is the solution to the puzzle Z'. Randomizable puzzle was formally defined in [27], where the authors also claimed that it needs to satisfy correctness, security and privacy properties. Correctness ensures that the solution to the puzzle can be recovered with the trapdoor. Security guarantees that with only the puzzle and the public parameters, the adversary cannot obtain the underlying solution. Privacy ensures that given two correctly formed puzzles, randomizing one of them, it is infeasible for an adversary to figure out the randomized one even with a trapdoor oracle.

3 Security Definitions

3.1 Security and Privacy Definition

To model security and privacy we resort to universal composability (UC) framework from Canetti [7] and the synchronous version of global UC framework

(GUC) [8]. The UC framework is suitable for proofs of a concurrent composition of protocols. Under the UC framework, protocol can run concurrently, which means that even many instances are executed concurrently, protocol remains secure. We allow the composition of GMHL with other application-dependent protocols while remaining security and privacy guarantees.

Attack model. We model the parties as interactive Turing machines (ITMs). They do not communicate directly but communicate with a trusted functionality $\mathcal{F}$ via secure and authenticated communication channels. We model the attacker $\mathcal{A}$ as a PPT machine with an interface $\mathsf{corrput}(\cdot)$, which can be used to detect the internal state of the corresponding party once inputting the identifier P of a party. If a party is corrupted, all the incoming and outcoming messages of P are routed through $\mathcal{A}$. In this work, we consider the static corruption model that are frequently used in papers [20], [27], [21], namely, the attacker commits ahead of time the identifiers of the parties it intends to corrupt. Furthermore, we consider rational attackers which means an attacker must aim to obtain benefits from the attack.

Communication model. We specify a synchronous communication network whose communication rounds are discrete. As in [12], [18], we denote by $\mathcal{F}_{\mathsf{clock}}$ the notion of round. All parties promise to complete the corresponding tasks of this round and get ready to the next round before the clock (i.e., $\mathcal{F}_{\mathsf{clock}}$) ticks. In this work, we treat the ideal functionality $\mathcal{F}_{\mathsf{clock}}$ as a global ideal functionality in the GUC model to ensure all parties are aware of the given round. Then we denote by $\mathcal{F}_{\mathsf{GC}}$ the formalization of communication channels as in [12]. Consider that parties communicate via authenticated communication channels, so integrity of messages in each round of communication can be guaranteed, the attacker can only change the order of the message in the same round but cannot delay, insert or drop the message. Furthermore, we denote by $\mathcal{F}_{\mathsf{st}}$ the secure transmission functionality, which guarantees the confidentiality of the message and prevents attacker from knowing or tampering with the content of message (for a concrete functionality see [7]). Lastly, we denote by $\mathcal{F}_{\mathsf{ano}}$ [6] the anonymous communication channels for users. It is similar to $\mathcal{F}_{\mathsf{st}}$ except omitting the identifier of the sender from the message sent to the receiver.

Payment channels. We denote by $\mathcal{F}_{\mathsf{PC}}$ the generalized channels, which can be seen as a generalization of payment channels. It provides the backbone for a payment channel: Create for opening a payment payment, Update for updating the balances of the parties on the same payment channel and Close for closing a payment channel.

(Global) Universal composability. We outline the notion of secure realization in the UC framework[7] and GUC framework[8]. In short, if the environment (i.e., the distinguisher) is unable to distinguish whether interacting with a protocol or an ideal functionality, we define that a protocol realizes an ideal functionality. Since our $\mathcal{F}_{\mathsf{PCN}}$ ideal functionality is based on $\mathcal{F}_{\mathsf{PC}}$ and $\mathcal{F}_{\mathsf{clock}}$, we define the UC-realization with respect to the aforementioned global functionalities. We denote by π the protocol access to $\mathcal{F}_{\mathsf{PC}}$ and $\mathcal{F}_{\mathsf{clock}}$ and denote by $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$ the ensemble of the outputs of the environment $\mathcal{E}$ when interacting with the attacker

$\mathcal{A}$ and users running protocol π. The UC-realization with respect to the global ideal functionalities is defined as follows:

Definition 1. (Global Universal Composability) *A protocol π UC-realizes an ideal functionality $\mathcal{F}$ with respect to a global channel $\mathcal{F}_{\mathsf{PC}}$ and a global clock $\mathcal{F}_{\mathsf{clock}}$ if for any* PPT *adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$, such that for any environment $\mathcal{E}$, the ensembles* $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{E}}^{\mathcal{F}_{\mathrm{PC}},\mathcal{F}_{\mathrm{clock}}}$ *and* $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}^{\mathcal{F}_{\mathrm{PC}},\mathcal{F}_{\mathrm{clock}}}$ *are computationally indistinguishable.*

Ideal Functionality. We define an ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$ in ($\mathcal{F}_{\mathsf{GC}}$,$\mathcal{F}_{\mathsf{st}}$, $\mathcal{F}_{\mathsf{ano}}$)-hybrid model for GMHL. Then formalize the notion of PCN relying on GMHL and define an ideal functionality $\mathcal{F}_{\mathsf{PCN}}$ in ($\mathcal{F}_{\mathsf{GC}}$,$\mathcal{F}_{\mathsf{st}}$,$\mathcal{F}_{\mathsf{ano}}$,$\mathcal{F}_{\mathsf{GMHL}}$)-hybrid model. The details can be seen in Section 6.

3.2 Security and Privacy Goals

Here, we introduce the security and privacy goals for a PCN.

Atomicity. A PCN should guarantee that all the payments are either successful or returned back to the participators.

Unlinkability. Any user (including an honest but curious user U_i) should not learn information that allows him to associate sender U_0 and receiver U_n of a payment. We define unlinkability in term of an interaction multi-graph as in [15]. An interaction multi-graph is a mapping of payments from a set of senders to a set of receivers. At epoch e, for each successful completed payment queried by the sender U_0^i, there is an edge labeled with e in the graph, linking from sender U_0^i to some receiver U_n^j. An interaction graph is compatible if it explains the view of the intermediate user, namely, the number of edges labeled with e incident to U_n^j equals to the number of coins received by U_n^j. Unlinkability requires that these graphs are indistinguishable. And the anonymity set depends on the number of compatible interaction graphs. Lastly, any intermediate users cannot learn any more information about the set of users in the PCN beyond their direct neighbours.

4 Solution Overview

The approach we follow to construct our locks is reminiscent of the anonymous atomic locks adopted in A^{2}L [27], but proceeds another way around and it is an extension and improvement of the original idea. It has expanded from the scenario of three-party payment to the scenario of multi-party payment. Certain changes have been made in the construction in order to provide generality. Our Generalized Multi-hop Locks (GMHL) consists of three phases: setup phase, lock phase and release phase. Intuitively, our payment paradigm relies on the fact that for all payments in a payment channel network, the previous payment can only be successfully finished if the latter payment is successfully completed.

Atomicity. Atomicity relies on conditional payment to ensure that either all payments are completed successfully (i.e., all payment channels are updated) or none are completed in a PCN.

***Our approach* :** In this work, we use cryptographic puzzle, an encoding of an instance of a cryptographic hard problem, to realize the conditional payment. Binding a cryptographic puzzle with the channel update, we can achieve the following properties: (i) the channel can be updated only after the solution to the puzzle is found and (ii) the solution to the puzzle can be extracted from a valid channel update.

Our approach ensures the atomicity of a payment between sender U_0 and receiver U_n as follows. During the setup phase, the sender U_0 generates a cryptographic puzzle P and sends it to the rest of the users in the path. The receiver U_n additionally obtains the solution to the puzzle. In the lock phase, each intermediate user U_i $(0 \leq i \leq n-1)$ updates the channel between U_i and U_{i+1} conditioned on U_{i+1} solving the puzzle P. In the release phase, U_n updates the channel between U_{n-1} and U_n with the corresponding solution and releases the coins promised by U_{n-1} before. Then U_{n-1} can get the solution and release coins from U_{n-2} after U_n updated the channel. The operation between the successive pair of users U_i and U_{i+1} is the same as that of U_{n-1} and U_n. Until the sender U_0 receives the update channel, the whole payment is finished.

Unlinkability. The aforementioned approach provides atomicity but does not guarantees unlinkability. All payments in a PCN use the same puzzle P, which means that any user (including an honest but curious intermediate user U_i) can easily link all participators in a PCN.

***Our approach* :** We use cryptographic randomizable puzzle to overcome this issue. Compared with cryptographic puzzle, cryptographic randomizable puzzle has two more features: (i) a certain puzzle P can be randomized to a fresh puzzle P' with a randomness r, and (ii) the solution to puzzle P' can be obtained with the solution to puzzle P and the former added randomness r.

Using this tool, our solution for atomicity and unlinkability is as follows. In the setup phase, the sender U_0 generates a puzzle P and randomizes it with a series of randomnesses r in parallel to a series of fresh puzzles. The solutions (i.e., k_i and k_{i+1}) to a successive pair of puzzles P_i and P_{i+1} have a relation $k_{i+1} = \phi(r_{i+1}, k_i)$. Then U_0 sends the puzzle tuple (P_i, P_{i+1}) and the randomness r_{i+1} to user U_i through a secure communication channel. Notice that the receiver U_n receives a puzzle P_n and its solution k_n. In the lock phase, the intermediate user U_i $(0 \leq i \leq n-1)$ updates the channel between U_i and U_{i+1} conditioned on the puzzle P_{i+1}. In the release phase, U_{n-1} can release coins after U_n released coins from U_{n-1} with solution k_n. The solution k_{n-1} is calculated by removing the randomness r_n from the solution k_n obtained from the updated channel between U_{n-1} and U_n. The rest of the users do the same. The whole payment is completed after U_0 releases his coins.

Lightweight setup. Currently, similar to other protocols, the sender is responsible for a great amount of computation that is proportional to the number of intermediate nodes and thus the setup phase is not lightweight.

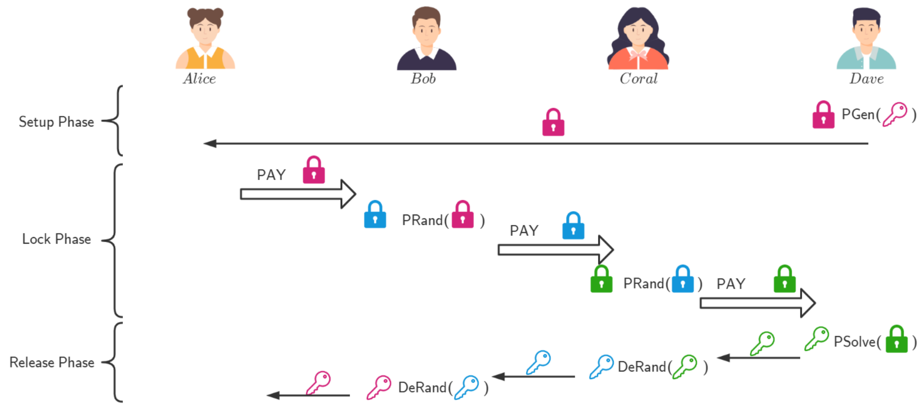


Fig. 1. The solution overview

Our approach **:** In the following, we consider to amortize the computation load to all the users on the path to obtain a lightweight setup. Our solution for atomicity, unlinkability and lightweight setup is shown in Figure 1. In the setup phase, Dave generates (using PGen) a puzzle P_α (i.e., the pink lock) and sends it to Alice through a secure communication channel. During the lock phase, Alice pays Bob conditioned on Bob solving the puzzle P_α. Since Bob does not have the solution to puzzle P_α, he uses PRand to randomize it to a fresh puzzle P_β (i.e., the blue lock) and initiates a payment to Coral conditioned on P_β. Similarly, Coral pays Dave conditioned on P_γ (i.e., the green lock). In the release phase, Dave solves puzzle P_γ (using PSolve), sends the solution (i.e., the green key) to Coral and releases the coins promised by Coral. Coral gets the solution to P_γ from update channel and removes the randomness added before (using DeRand) to release coins. Bob does the same as Coral. Now the release phase is finished, Alice can pay coins to Dave as expected.

Generality. The aforementioned approach guarantees atomicity, unlinkability and lightweight setup, but does not consider more about generality.

Our approach **:** We here use adaptor signature to overcome this challenge. Using this tool, we make our solution able to be applied to various (non-/Turing-complete) blockchains. For short, in the lock phase, U_i generates a pre-signature with respect to a puzzle and sends to U_{i+1}. During the release phase, U_{i+1} can convert the pre-signature into a valid signature with the solution to puzzle, send it to U_i and release coins. Therefore, the online operation of a payment is only to verify the validity of a signature, which can be applied to various blockchains.

In the following section, combining with adaptor signature and randomizable puzzle, we give a formalization of the aforementioned solution to guarantee the update of a channel can only be completed after the solution to a puzzle is found. In a nutshell, we first generate a randomizable puzzle and a pre-signature with respect to it. Only after solving the randomizable puzzle can the pre-signature

The GMHL protocol Π		
1	$\mathsf{Setup}_{\mathrm{U}_0}$:	$\mathsf{Setup}_{\mathrm{U}_n}(1^\lambda)$:
2		$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$
3		$(\mathsf{pp}, \mathsf{td}) := (\mathsf{pk}, \mathsf{sk})$
4		$r \xleftarrow{\$} D$
5		$(P_0, r_0) \leftarrow \mathsf{PGen}(\mathsf{pp}, r)$
6		$\pi_{\alpha_0} \leftarrow \mathsf{P}_{\mathrm{NIZK}}(\{\exists \alpha_0 \mid \mathsf{PSolve}(\mathsf{td}, P_0) = \alpha_0\}, \alpha_0)$
7		send (P_0, π_{α_0}) to U_0
8	If $\mathsf{V}_{\mathrm{NIZK}}(\pi_{\alpha_0}, P_0) \neq 1$ then abort	
9	return $\top$	return (P_0, π_{α_0})
10	$\mathsf{Lock}_{\mathrm{U}_0}(P_0)$:	$\mathsf{Lock}_{\mathrm{U}_\mathrm{i}}(P_{i-1})$:
11	$\hat{\sigma}_0 \leftarrow \mathsf{PreSig}(\mathsf{sk}_{\mathrm{U}_0}, m_0, P_0)$	If $\mathsf{V}_{\mathrm{NIZK}}(\pi_{\alpha_{i-1}}, P_{i-1}) \neq 1$ then abort
12	send $(\hat{\sigma}_0, P_0, \pi_{\alpha_0})$ to U_1	If $\mathsf{PreVf}(m_{i-1}, P_{i-1}, \hat{\sigma}_{i-1}) \neq 1$ then abort
13	return $(\hat{\sigma}_0, P_0, \pi_{\alpha_0})$	$(P_i, r_i) \leftarrow \mathsf{PRand}(\mathsf{pp}, P_{i-1})$
14		$\pi_{\alpha_i} \leftarrow \mathsf{P}_{\mathrm{NIZK}}(\{\exists \alpha_i \mid \mathsf{PSolve}(\mathsf{td}, P_i) = \alpha_i\}, \alpha_i)$
15		$\hat{\sigma}_i \leftarrow \mathsf{PreSig}(\mathsf{sk}_{\mathrm{U}_\mathrm{i}}, m_i, P_i)$
16		send $(\hat{\sigma}_i, P_i, \pi_{\alpha_i})$ to U_{i+1}
17		return $(\hat{\sigma}_i, P_i, \pi_{\alpha_i})$
18	$\mathsf{Release}_{\mathrm{U}_\mathrm{i}}(P_i, \hat{\sigma}_{i-1}, \hat{\sigma}_i, \sigma_i)$:	$\mathsf{Release}_{\mathrm{U}_n}(P_{n-1}, \hat{\sigma}_{n-1})$:
19	If $\mathsf{Vrfy}(\sigma_i) \neq 1$ then abort	$w_{n-1} = \mathsf{PSolve}(\mathsf{td}, P_{n-1})$
20	$w_i = \mathsf{Ext}(\hat{\sigma}_i, \sigma_i, P_i)$	$\sigma_{n-1} = \mathsf{Adapt}(\hat{\sigma}_{n-1}, w_{n-1})$
21	$w_{i-1} = w_i \cdot r_i^{-1}$	send σ_{n-1} to U_{n-1}
22	$\sigma_{i-1} = \mathsf{Adapt}(\hat{\sigma}_{i-1}, w_{i-1})$	return σ_{n-1}
23	send σ_{i-1} to U_{i-1}	
24	return σ_{i-1}	

Fig. 2. Algorithms and protocols for the generic construction

be converted into a valid signature. After getting the valid signature, we can combine it with the corresponding pre-signature to get the solution to the puzzle.

5 Our Construction

We here illustrate the proposed GMHL and show how to realize it in PCN.

5.1 The Proposed GMHL

We now present our GMHL and denote by Π. This can be achieved by utilizing a randomizaable puzzle and an adaptor signature $\Xi_{R,\Sigma}$=(PreSig,PreVf, Adapt,Ext) for the signature Σ=(Gen,Sign,Vrfy) used by the underlying ledger and a hard relation R. We assume that statement/witness pairs of R are public/secret key of Σ and a constant amount of coins (i.e. p) for each payment so as to avoid others linking the users in a PCN. The protocol consists of three phases: setup phase, lock phase and release phase. The algorithms of our protocol are given in Figure 2. We discuss each phase separately at a high level here.

Setup phase. In the setup phase, user U_n first obtains a public/secret key pair (pk, sk) through the generation algorithm of the signature scheme (line 2 in Figure 2). Set the public/secret key pair (pk,sk) as the public parameter/trapdoor pair (pp,td) of the randomizable puzzle. Then U_n uniformly samples an element r from a set D, generates a randomizable puzzle P_0 with related to r (i.e., the secret adaptor), produces a NIZK proof π_{α_0} proving that α_0 is the solution to puzzle P_0 (lines 4-6 in Figure 2) and sends a puzzle/proof pair (P_0, π_{α_0}) to U_0. Once U_0 is convinced of the validity of such pair, the GMHL is initialized.

Lock phase. In the lock phase, user U_0 generates an adaptor signature $\hat{\sigma}_0$ over the previously agreed message m_0 (e.g., the transaction id) and shares the (pre-signature, puzzle, proof) tuple to U_1 (lines 11-12 in Figure 2). For intermediate user U_i, if no abortion arises during the verification of such tuple (lines 11-12 in Figure 2), he randomizes the puzzle P_{i-1} to P_i using PRand algorithm, produces a NIZK proof π_{α_i} over puzzle P_i, generates a pre-signature $\hat{\sigma}_i$ over the mutually agreed information and sends tuple $(\hat{\sigma}_i, P_i, \pi_{\alpha_i})$ to U_{i+1} (lines 13-16 in Figure 2). At this point the lock phase is finalized and we can turn to the release phase.

Release phase. In the release phase, after user U_n receives the tuple $(\hat{\sigma}_{n-1}, P_{n-1}, \pi_{\alpha_{n-1}})$ and confirms its validity, he solves puzzle P_{n-1} for obtaining w_{n-1} and converts the pre-signature $\hat{\sigma}_{n-1}$ into a valid signature σ_{n-1} (lines 19-20 in Figure 2). For intermediate user U_i, once he is convinced of the validity of signature σ_i, he extracts the witness w_i from it using Ext algorithm (lines 19-20 in Figure 2), removes the randomness added before, produces a valid signature σ_{i-1} with Adapt algorithm (lines 21-22 in Figure 2) and sends it to user U_{i-1}.

Guillou-Quisquater-based Instance of GMHL. To show how to instantiate GMHL, we present a concrete instance Π_{GQ} which is based on a Guillou-Quisquater adaptor signature Σ_{GQ} and a randomizable puzzle. Before describing the construction of the instance, we first propose a specific RSA-based randomizable puzzle.

RSA-based randomizable puzzle. We set the encryption scheme Φ to be RSA-based homomorphic encryption scheme[16], its message space $\mathcal{M} = \mathbb{Z}_N$ and solution space $\mathcal{S} = \mathbb{Z}_N$. Our construction is shown in Construction 1. In the construction, we wrap an integer in a puzzle with PGen and PRand algorithms and unwrap it with PSolve algorithm.

Construction 1 *The randomizable puzzle scheme is constructed as follows:*

$\mathsf{PSetup}(1^\lambda)$*: sample a key pair* $(\mathsf{pk}^\Phi, \mathsf{sk}^\Phi) \leftarrow \mathsf{KGen}(1^\lambda)$*, set* $\mathsf{pp} := \mathsf{pk}^\Phi$ *and* $\mathsf{td} := \mathsf{sk}^\Phi$*, and return* $(\mathsf{pp}, \mathsf{td})$.

$\mathsf{PGen}(\mathsf{pp},\zeta)$*: parse* pp *as* pk^Φ*, sample* $r \stackrel{\$}{\leftarrow} \mathcal{S}$*, compute* $c \leftarrow \mathsf{Enc}(\mathsf{pk}^\Phi, \zeta \cdot r)$*, set* $Z := c$*, and return* (Z, r)*.*

$\mathsf{PSolve}(\mathsf{td},Z)$*: parse* td *as* sk^Φ*, compute* $\zeta' \leftarrow \mathsf{Enc}(\mathsf{sk}^\Phi, Z)$*, and return* ζ'*.*

$\mathsf{PRand}(\mathsf{pp},Z)$*: parse* pp *as* (N, e)*, sample* $r' \stackrel{\$}{\leftarrow} \mathcal{S}$*, compute* $c' = c \cdot r'^e \bmod N$*, set* $Z' := c'$*, and return* (Z', r')*.*

The security of our construction is shown by the following theorem.

Theorem 1. *Let* Φ *be an encryption scheme, the construction 1 is a correct, secure and private randomizable puzzle scheme.*

Proof. (*sketch*) Correctness and security follows straightforwardly from the correctness and security properties of the encryption scheme Φ. For the notion of privacy, note that for a puzzle Z and its solution $r\zeta$ and a randomizable puzzle

	Public parameters: (N, e, X), message m		
	$\mathsf{Setup}_{\mathrm{U}_0}$:		$\mathsf{Setup}_{\mathrm{U}_n}(1^\lambda)$:
1			$(N, e, d, X, x) \leftarrow \Sigma_{GQ}.\mathsf{KGen}(1^\lambda)$
2			$\mathsf{pk} = (N, e, x)$
3			$\mathsf{sk} = (N, d)$
4			$(\mathsf{pp}, \mathsf{td}) = (\mathsf{pk}, \mathsf{sk})$
5			$t \xleftarrow{\$} Z_n$
6			$(P_0, t_0) \leftarrow \mathsf{PGen}(\mathsf{pp}, t)$
7		$\xleftarrow{(P_0, \pi_{w_0})}$	$\pi_{w_0} \leftarrow \mathsf{P}_{\mathrm{NIZK}}(\{\exists w_0 \mid \mathsf{Dec}(\mathsf{td}, P_0) = w_0\}, w_0)$
8	If $\mathsf{V}_{\mathrm{NIZK}}(P_0, \pi_{w_0}) \neq 1$ then abort		
9	return $\top$		return (P_0, π_{w_0})

Fig. 3. The setup phase between user U_0 and U_n

Public parameters: (N, e, X), message m_i		
$\mathsf{Lock}_{\mathrm{U_i}}(\mathsf{sk}_{\mathrm{U_i}}, P_{i-1})$:		$\mathsf{Lock}_{\mathrm{U_{i+1}}}(\mathsf{sk}_{\mathrm{U_{i+1}}})$:
$(P_i, t_i) \leftarrow \mathsf{PRand}(\mathsf{pp}, P_{i-1})$		
$\pi_{w_i} \leftarrow \mathsf{P}_{\mathrm{NIZK}}(\{\exists w_i \mid \mathsf{Dec}(\mathsf{td}, P_i) = w_i\}, w_i)$	$\xrightarrow{(P_i, \pi_{w_i})}$	
		If $\mathsf{V}_{\mathrm{NIZK}}(P_i, \pi_{w_i}) \neq 1$ then abort
		$k_2 \xleftarrow{\$} Z_N$
		$r_2 = k_2^e \ mod \ N$
	$\xleftarrow{(r_2, \pi_2)}$	$\pi_2 \leftarrow \mathsf{P}_{\mathrm{NIZK}}(\{\exists k_2 \mid r_2 = k_2^e \ mod \ N\}, k_2)$
If $\mathsf{V}_{\mathrm{NIZK}}(r_2, \pi_2) \neq 1$ then abort		
$k_1 \xleftarrow{\$} Z_N$		
$r_1 = k_1^e \ mod \ N$		
$\pi_1 \leftarrow \mathsf{P}_{\mathrm{NIZK}}(\{\exists k_1 \mid r_1 = k_1^e \ mod \ N\}, k_1)$	$\xrightarrow{(r_1, \pi_1)}$	
		If $\mathsf{V}_{\mathrm{NIZK}}(r_1, \pi_1) \neq 1$ then abort
$R = r_1 r_2 P_i$		$R = r_1 r_2 P_i$
$\alpha_i = H(m_i, R)$		$\alpha_i = H(m_i, R)$
	$\xleftarrow{y_2}$	$y_2 = k_2 \mathsf{sk}_{\mathrm{U_{i+1}}}^{\alpha_i} \ mod \ N$
$y_1 = k_1 \mathsf{sk}_{\mathrm{U_i}}^{\alpha_i} \ mod \ N$	$\xrightarrow{y_1}$	
$\hat{\beta}_i = y_1 y_2$		$\hat{\beta}_i = y_1 y_2$
return $(\hat{\sigma}_i = (\alpha_i, \hat{\beta}_i),\ P_i,\ \pi_{w_i})$		return $(\hat{\sigma}_i = (\alpha_i, \hat{\beta}_i),\ P_i,\ \pi_{w_i})$

Fig. 4. The lock phase between user U_i and U_{i+1}

Z' with the solution $r'\zeta$, the space of $r\zeta$ and $r'\zeta$ are the same, they are all in the field $\mathcal{S}$, which implies that Z and Z' are information-theoretically unlinkable.

The description of the instance. The algorithms are given in Figure 3, Figure 4 and Figure 5. This construction consists of three phases: setup phase, lock phase and release phase. Next we will discuss each phase separately.

Setup phase. As shown in Figure 3, user U_n first runs the Guillou-Quisquater signature scheme to get a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$ where $\mathsf{pk} = (N, e, X)$ and $\mathsf{sk} = (N, x)$ and set them as public parameter pp and trapdoor td separately (line 1-4 in Figure 3). Then he picks a random integer t and generates a puzzle P_0 over it (lines 5-6 in Figure 3). He sends puzzle P_0 to U_0 along with the corresponding NIZK proof (lines 7 in Figure 3). Once U_0 is convinced of the validity of such pair, the setup phase is finished.

Lock phase. In the lock phase, for user U_0, since puzzle P_0 was generated by U_n, he just needs to execute a coin tossing protocol with U_1 to come to an

Public parameters: (N, e, X)			
	$\mathsf{Release}_{U_{i-1}}$:		$\mathsf{Release}_{U_{i+1}}(\hat{\sigma}_i, \sigma_i, P_i)$:
1			$w_{i-1} = \mathsf{PSolve}(\mathsf{td}, P_{i-1})$ //for i=n
2			parse σ_i as (α_i, β_i) //for i=1,2,...,n-1
3			parse $\hat{\sigma}_i$ as $(\alpha_i, \hat{\beta}_i)$
4			$w_i = \beta_i / \hat{\beta}_i$
5			$w_{i-1} = w_i \cdot t_i^{-1}$
6		$\xleftarrow{\sigma_{i-1}}$	$\sigma_{i-1} = (\alpha_{i-1}, \hat{\beta}_{i-1} w_{i-1})$
7	If $\mathsf{Vrfy}(\sigma_{i-1}) \neq 1$ then abort		
8	return $\top$		return σ_{i-1}

Fig. 5. The release phase between user U_i and U_{i+1}

agreement on a randomness $R = (k_1 k_2 t_0 t)^e \ mod \ N$. It is worth mentioning that $t_0 t$ is unknown to U_0 and U_1 and the use of k_1 and k_2 here is to blind the puzzle P_0. Due to the homomorphic feature of RSA encryption scheme, randomness can be continuously multiplied. The randomness R is calculated by exchanging r_1 and r_2 with each other and multiplied with puzzle P_0, the corresponding proof of consistency is attached together (lines 4-12 in Figure 4). Then both U_0 and U_1 calculate a hash α_0 with related to the previously agreed message m_0 and randomness R (line 13 in Figure 4). Next U_0 and U_1 execute a coin tossing protocol again to agree on an "almost valid" signature $(\alpha_0, \hat{\beta}_0)$ while the valid form is $(\alpha_0, t_0 t \hat{\beta}_0)$ (lines 14-16 in Figure 4). Note that the lacking part of the "almost valid" signature is the secret of puzzle P_0. For intermediate user U_i, he has one more operation than user U_0: randomizing puzzle P_{i-1} to a fresh puzzle P_i and producing the corresponding NIZK proof (lines 1-2 in Figure 4).

Release phase. In the release phase, for user U_n, since he has the trapdoor to the cryptographic function of the puzzle, he can directly use the trapdoor function (i.e., the algorithm Dec) to get the secret of puzzle P_{n-1} (line 1 in Figure 5). As for intermediate user U_i, he can extract witness w_i from a valid pre-signature/signature pair $(\hat{\sigma}_i, \sigma_i)$ because of the witness-extractability feature of adaptor signature. Then U_i removes the former added t_i to get the solution to puzzle P_{i-1} (lines 2-5 in Figure 5). At this point, U_i can adapt the pre-signature $\hat{\sigma}_{i-1}$ to a valid signature σ_{i-1} because of the pre-signature adaptability feature of adaptor signature (line 6 in Figure 5). Once user U_{i-1} is certain about the validity of signature σ_{i-1} (line 7 in Figure 5), U_i can release the coins.

5.2 Description of Our PCN

GMHL can be generically combined with a blockchain B to construct a fully-fledged PCN. The construction of our PCN is shown in Figure 6. We denote by c_i the channel identifier between user U_i and U_{i+1}, and the coins in a payment is constant (i.e., p). We use Δ to represent a constant validity period of a signature and t to represent current time. In our construction, U_0 and U_n first execute the setup phase of GMHL protocol to initialize the entire construction. Then intermediate user U_i runs the lock phase of GMHL protocol to get the input

Public parameters: validity period $\varDelta$ of a lock, current time t, payment cash p		
$U_0(c_0)$	$U_i(c_i, c_{i+1}),\ i=1,2,...,n-1$	$U_n(c_{n-1})$
		$(P_0, \pi_{\alpha_0}) \leftarrow \mathsf{Setup}_{\mathrm{U_n}}(1^\lambda)$
		If $(P_0, \pi_{\alpha_0})=\perp$ then abort
$(\hat{\sigma}_0, P_0, \pi_{\alpha_0}) \leftarrow \mathsf{Lock}_{\mathrm{U_0}}(P_0)$		Send (P_0, π_{α_0}) to U_0
If $(\hat{\sigma}_0, P_0, \pi_{\alpha_0}) =\perp$ then abort		
If $c_0.\mathsf{cash}(U_0) < p$ then abort		
Set $T_1 = t+n\varDelta$		
Send $(\hat{\sigma}_0, P_0, \pi_{\alpha_0})$ to U_1		
$\xrightarrow{GMHL(U_0, U_1, V, p, T_1)}$	If $c_i.\mathsf{cash}(U_i) < p$ then abort	
	$(\hat{\sigma}_i, P_i, \pi_{\alpha_i}) \leftarrow \mathsf{Lock}_{\mathrm{U_i}}(P_{i-1})$	
	If $(\hat{\sigma}_i, P_i, \pi_{\alpha_i}) =\perp$ then abort	
	Send $(\hat{\sigma}_i, P_i, \pi_{\alpha_i})$ to U_{i+1}	
	Set $T_i = t + (n-i)\varDelta$	
	$\xrightarrow{GMHL(U_i, U_{i+1}, V, p, T_i)}$	$\sigma_{n-1} \leftarrow \mathsf{Release}_{\mathrm{U_n}}(P_{n-1}, \hat{\sigma}_{n-1})$
		If $\sigma_{n-1} =\perp$ then abort
		Send σ_{n-1} to U_{n-1}
	If $\mathsf{Vrfy}(\sigma_i)\neq 1$ then abort	
	$\sigma_{i-1} \leftarrow \mathsf{Release}_{\mathrm{U_i}}(P_i, \hat{\sigma}_{i-1}, \hat{\sigma}_i, \sigma_i)$	
	If $\sigma_{i-1} =\perp$ then abort	
	Send σ_{i-1} to U_{i-1}	
If $\mathsf{Vrfy}(\sigma_0)\neq 1$ then abort		

Fig. 6. Our PCN construction

for GMHL contract GMHL(U_i, U_{i+1}, V, p, T_i), additionally, the algorithm V in GMHL contract only needs to verify a signature:

• If U_{i+1} produces a valid signature σ_i that $\mathsf{Vrfy}(\sigma_i) = 1$ before time T_i expires, then channel c_i is updated with $c_i.\mathsf{cash}(U_i) -= p$ and $c_i.\mathsf{cash}(U_{i+1}) += p$ (i.e., U_i pays p coins to U_{i+1}).

• Else the channel c_i remains unchanged. (i.e., U_i takes back p coins that were locked in the contract).

6 Security Analysis

In this section, we formalize the security of GMHL and our PCN construction under the UC framework [7] and the GUC framework [8]. We first describe an ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$ to capture the honest behaviours and security properties of the interactions among users U_0, U_1, ..., U_n in the GMHL protocol, aiming to specify the input and output behaviors of our protocol and capture the adversary's possible influence in the execution. Next, we discuss that our GMHL construction in Section 5.1 emulates $\mathcal{F}_{\mathsf{GMHL}}$, namely, any possible attacks in our construction can be simulated in $\mathcal{F}_{\mathsf{GMHL}}$. We specify an ideal functionality $\mathcal{F}_{\mathsf{PCN}}$ relying on $\mathcal{F}_{\mathsf{GMHL}}$ to cover the security notions of our PCN construction in Section 5.2. Furthermore, we will consider the security against some concrete attacks and give a formal proof in the following.

Ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$. We define an ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$ in ($\mathcal{F}_{\mathsf{GC}}$,$\mathcal{F}_{\mathsf{st}}$,$\mathcal{F}_{\mathsf{ano}}$)-hybrid model for GMHL. $\mathcal{F}_{\mathsf{GMHL}}$ manages a list $\mathcal{P}$ (initially set $\mathcal{P} := \emptyset$), which is used to store the message about the cryptographic puzzles. The format of each piece of message in the list is ($< pid_0, b >$, $< pid_1, b >$, ...,

Ideal Functionality $\mathcal{F}_{\mathsf{GMHL}}$

Setup: On input (Setup, U_n) from U_0, $\mathcal{F}_{\text{GMHL}}$ proceeds as follows:
- Send (setup-req, U_0) to U_n and $\mathcal{S}$.
- Receive (setup-res, b) from U_n.
- If b=$\perp$ then abort.
- Sample $pid_0 \xleftarrow{\$} \{0,1\}^\lambda$.
- Store ($< pid_0, \perp >$, $\cdot$, ..., $\cdot$) into $\mathcal{P}$.
- Send (setuped, pid_0) to U_0, U_n and inform $\mathcal{S}$.

Lock: On input (Lock, U_{i+1}) from U_i, $\mathcal{F}_{\text{GMHL}}$ proceeds as follows:
- Send (lock-req, U_i) to U_{i+1} and $\mathcal{S}$.
- Receive (lock-res, b) from U_{i+1}.
- If b=$\perp$ then abort.
- Sample $pid_i \xleftarrow{\$} \{0,1\}^\lambda$.
- Update entry to ($< pid_0, \perp >$, $< pid_1, \perp >$, ..., $< pid_i, \perp >$, $\cdot$, $\cdot$, ..., $\cdot$) in $\mathcal{P}$.
- Send (locked, pid_{i-1}, pid_i) to U_i, (locked, pid_i) to U_{i+1} and inform $\mathcal{S}$.

Release: On input (Release, U_{i-1}, pid_{i-1}) from U_i, $\mathcal{F}_{\text{GMHL}}$ proceeds as follows:
- If tuple ($< pid_0, \perp >$, ..., $< pid_{i-1}, \perp >$, $\cdot$, ..., $\cdot$,)$\in \mathcal{P}$ or $b = \perp$ in $< pid_i, b >$ then abort.
- Send (release-req, U_i) to U_{i-1} and $\mathcal{S}$.
- Receive (release-res, b) from U_{i-1}.
- If b=$\perp$ then abort.
- Update entry to ($< pid_0, \perp >$, ..., $< pid_{i-2}, \perp >$, $< pid_{i-1}, \top >$, ..., $< pid_{n-1}, \top >$) in $\mathcal{P}$.
- Send (released, pid_{i-1}, $\top$) to U_{i-1}, U_i and $\mathcal{S}$.

Fig. 7. Ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$

$< pid_n, b >$) where b is used to indicate whether the puzzle has been solved. At the same time, $\mathcal{F}_{\mathsf{GMHL}}$ provides three interfaces, the setup interface allows a party to obtain a puzzle, the lock interface given as input a puzzle to get a randomized version of it and the release interface allows a party to check the validity of a puzzle solution and get the solution to another puzzle. The description of the ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$ is depicted in Figure 7.

Ideal functionality $\mathcal{F}_{\mathsf{PCN}}$. We here formalize the notion of PCN relying on GMHL and define an ideal functionality $\mathcal{F}_{\mathsf{PCN}}$ in ($\mathcal{F}_{\mathsf{GC}}$,$\mathcal{F}_{\mathsf{st}}$,$\mathcal{F}_{\mathsf{ano}}$,$\mathcal{F}_{\mathsf{GMHL}}$)-hybrid model. $\mathcal{F}_{\mathsf{PCN}}$ manages a list $\mathcal{C}$ (initially set $\mathcal{C} := \emptyset$) which is used to record the identifier of the opening channels. We assume that two adjacent parties on a PCN path have a channel, the amount of coins (i.e., p) in each payment on the channel is constant and is globally available to all parties, the validity period for a payment is constant (i.e., Δ) and the current time is t. In this model, we do not take transaction fees into account to ensure the security of the model. $\mathcal{F}_{\mathsf{PCN}}$ provides three interfaces, the Open for opening a channel, the Close for closing a channel and the Pay for the payment operation from sender U_0 to receiver U_n via the intermediate users U_i using in $\mathcal{F}_{\mathsf{GMHL}}$. The description of the ideal functionality $\mathcal{F}_{\mathsf{PCN}}$ is depicted in Figure 8.

6.1 Concrete Attack Scenarios (Informal)

In the section, we consider some attacks against our construction and argue informally, why security and privacy still holds.

Generate a fake puzzle. Suppose there are three nodes on the path, honest user A, malicious user B and honest user C. After receiving a puzzle P from user A, user B generates a fake puzzle P' and sends it to user C. User C sends the

Ideal Functionality $\mathcal{F}_{\mathsf{PCN}}$
Open: On input (Open, c, $txid_P$) from a party P, $\mathcal{F}_{\text{PCN}}$ proceeds as follows:
- Send (Create, c, $txid_P$) to $\mathcal{S}$.
- Receive b from $\mathcal{S}$.
- If b=$\perp$ then abort.
- Add c into $\mathcal{C}$.
- Send (created, c.cid) to c.users.
Close: On input (Close, c) from a party P, $\mathcal{F}_{\text{PCN}}$ proceeds as follows:
- Send (Close, c.cid) to $\mathcal{S}$.
- Receive b from $\mathcal{S}$.
- If b=$\perp$ then abort.
- Remove c from $\mathcal{C}$.
- Send (closed, c.cid) to c.users.
Pay: On input (Pay, U_n) from U_0, $\mathcal{F}_{\text{PCN}}$ proceeds as follows:
- Retrieve all the c_i in $\mathcal{C}$, check whether c_i.users=$\{U_i, U_{i+1}\}$.
- If $c_i = \perp$ then abort.
- Send (Setup, U_n) to $\mathcal{S}$.
- Receive pid_0 from $\mathcal{S}$.
- If $pid_0 = \perp$ then abort.
- Set $T_i = t + (n-i)\Delta$, propose c_i.TLP ($\delta_i := (c_i.\mathsf{cash}(U_i) -= p, c_i.\mathsf{cash}(U_{i+1}) += p), T_i$) to U_i and U_{i+1}.
- Send (Lock, U_1), (Lock, U_2), ..., and (Lock, U_n) to $\mathcal{S}$.
- Receive a series of $pids$ (i.e., pid_1, pid_2, ..., and pid_{n-1}) from $\mathcal{S}$.
- If any one of $pid_i = \perp$ then abort.
- Send (Release, U_{n-1}), (Release, U_{n-2}), ..., and (Release, U_0) to $\mathcal{S}$.
- Receive a series of b from $\mathcal{S}$.
- If any one of b=$\perp$ or $T_i < t$ then send $\perp$ to U_i.
- Send a series of updates (Update, c_i.cid, $\delta_i := (c_i.\mathsf{cash}(U_i) -= p, c_i.\mathsf{cash}(U_{i+1}) += p)$) to $\mathcal{S}$.

Fig. 8. Ideal functionality $\mathcal{F}_{\mathsf{PCN}}$

solution S' to B, but this solution is invalid and C cannot redeem coins form B. However, after receiving such a solution, B can extract a valid solution S from it and redeems coins from A. In order to prevent this attack, we force each user to present a zero-knowledge proof to the next user that the puzzle he sends is a valid randomized puzzle.

Some users are skipped (wormhole). Suppose there are five intermediate users on the path, honest user A, attacker B, honest user C, attacker D and honest user E. Attackers B and D attempt to collude with each other to steal coins from other honest users. After receiving solution from E, D may want to send the solution to B so that they can skip the intermediate user C to steal his fees. But this attack does not work. Since the puzzle of each payment on the path is bound to a randomness, which is kept by each users. B cannot redeems coins back from A with the solution sent by D, such a solution is blinded with a randomness kept by C. Only after C removes his randomness added in this solution, can the attacker B receive a valid solution and redeem back his coins from A. Furthermore, once D wants to skip C and refuses to transfers solution to C, C will abort and the payment is stopped in the channel between C and D. However, in this case, D will suffer financial losses (E has redeemed coins from D). Therefore, rational D will not agree to collude with B to launch an attack, instead honestly sends the solution to C to complete the payment.

Ideal Functionality $\mathcal{F}_{\mathsf{NIZK}}$
On input (prove, sid, x, w) from one party P_1 or P_2, ignore the input does not related to the relation R (i.e., $(x, w) \notin$ R) or proposed former (i.e., the sid has been used), then send (proof, sid, x) to another party.

Fig. 9. Ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$

6.2 Security Analysis of GMHL

Here, we will prove the security of our GMHL by the following theorem. We prove security according to the UC framework and in the presence of *malicious adversaries* with *static corruptions*.

Theorem 2. *Let Σ be EUF-CMA secure signature schemes, R be a hard relation, $\Xi_{R,\Sigma}$ be a secure adaptor signature scheme and RP be a secure and private randomizable puzzle scheme, then the construction in Figure 2 UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$ in the $(\mathcal{F}_{\mathsf{GC}}, \mathcal{F}_{\mathsf{ano}}, \mathcal{F}_{\mathsf{st}})$-hybrid model.*

Proof. In the following proof, we assume all the adversary's message are well-formed and treat the malformed messages as aborts. Then we use a series of hybrids to gradually modify the initial experiment.

Hybrid $\mathcal{H}_0$: This corresponds to the original construction as described in Figure 2.

Hybrid $\mathcal{H}_1$: All calls to the non-interactive zero-knowledge scheme NIZK are replaced with calls to an ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ with respect to a relation R (described in Figure 9).

Hybrid $\mathcal{H}_2$: For a corrupted intermediate user U_i $(0 < i < n)$ and other honest users U_0, ..., U_{i-1}, U_{i+1}, ..., U_n, check whether U_i returns some tuple $(\hat{\sigma}_i, P_i, \pi_{\alpha_i})$, before U_0 and U_n execute the setup phase, and does not cause the honest user U_{i+1} to abort during the lock phase. If the aforementioned happens, abort the experiment and output $\perp$.

Hybrid $\mathcal{H}_3$: For a corrupted intermediate user U_i $(0 < i < n)$ and other honest users U_0, ..., U_{i-1}, U_{i+1}, ..., U_n and a pre-signature $\hat{\sigma}_{i-1}$ between U_{i-1} and U_i, check whether U_i returns a valid signature σ_{i-1} such that $\mathsf{Verify}(\sigma_{i-1}) = 1$, before a valid signature σ_i is output from an execution of the release phase which can extract the witness w_{i-1} such that satisfied $\mathsf{Verify}(\mathsf{Adapt}(\hat{\sigma}_{i-1}, w_{i-1})) = 1$, then the experiment aborts.

Hybrid $\mathcal{H}_4$: For the honest users U_i and U_{i+1} with a witness w_i extracted in the release phase, if the parties does not abort and $\mathsf{Verify}(\mathsf{Adapt}(\hat{\sigma}_{i-1}, w_i \cdot r_i^{-1}) \neq 1$, then the experiment aborts.

Simulator $\mathcal{S}$: The simulator $\mathcal{S}$ simulates the honest parties as in the previous hybrid. Assume that the actions of $\mathcal{S}$ are determined by the ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$ and described in Figure 10, Figure 11 and Figure 12.

In the following, we prove the indistinguishability of the neighboring experiments for the environment $\mathcal{E}$.

Lemma 1. *For all* PPT *distinguishers $\mathcal{E}$ it holds that*
$\mathrm{EXEC}_{\mathcal{H}_0,\mathcal{A},\mathcal{E}} \approx \mathrm{EXEC}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}$.

Simulator for setup phase

Case U_0 is honest and U_n is corrupted

Upon U_0 sending (Setup, U_0) to $\mathcal{F}_{\text{GMHL}}$, proceed as follows:

- Send (setup-req, U_0) to U_n.
- Upon (setuped, P_0, π_{α_0}) from $\mathcal{A}$ (on behalf of U_n), check if $\mathsf{V}_{\text{NIZK}}(P_0,\ \pi_{\alpha_0}) \neq 1$. If this is the case, then simulate U_0 aborting.

Case U_0 is corrupted and U_n is honest

Upon U_0 sending (setup-req, U_0) to U_n, proceed as follows:

- If U_n sends (setup-res, $\top$) to $\mathcal{F}_{\text{GMHL}}$, then sample a random number $r \xleftarrow{\$} D$, generate a puzzle $(P_0, r_0) \leftarrow (\mathsf{pp}, r)$, prove the knowledge of the puzzle $\pi_{\alpha_0} \leftarrow \mathsf{P}_{\text{NIZK}}(\{\exists \alpha_0 | \mathsf{PSolve}(\mathsf{td}, P_0) = \alpha_0\}, \alpha_0)$ and send (setuped, P_0, π_{α_0}) to U_0. Else stop.

Fig. 10. Simulator for setup phase

Simulator for lock phase

Case U_i is honest and U_{i+1} is corrupted

Upon U_i sending (Lock, U_{i+1}) to $\mathcal{F}_{\text{GMHL}}$, proceed as follows:

- Send (lock-req, U_i) to U_{i+1}.
- Upon (lock-res, b) from $\mathcal{A}$ (on behalf of U_{i+1}), check if b=$\perp$. If this is the case, then simulate U_i aborting. Otherwise, then randomize the puzzle $(P_i, r_i) \leftarrow \mathsf{PRand}(\mathsf{pp}, P_{i-1})$ (if i=0, ignore the first operation, U_0 does not need to randomize the puzzle), generate a pre-signature $\hat{\sigma}_i \leftarrow (\mathsf{sk}_{\mathrm{U_i}}, m_i, P_i)$ along with the corresponding NIZK proof $\pi_{\alpha_i} \leftarrow \mathsf{P}_{\text{NIZK}}(\{\exists \alpha_i | \mathsf{PSolve}(\mathsf{td}, P_i) = \alpha_i\}, \alpha_i)$ and send (locked, $\hat{\sigma}_i$,P_i,π_{α_i}) to U_{i+1}.

Case U_i is corrupted and U_{i+1} is honest

Upon U_i sending (lock-req, U_i) to U_{i+1}, proceed as follows:

- Upon (locked, $\hat{\sigma}_i$, P_i, π_{α_i}) from $\mathcal{A}$ (on behalf of U_i), check if $\mathsf{PreVf}(m_i, P_i, \hat{\sigma}_i) \neq 1$ or $\mathsf{V}_{\text{NIZK}}(P_i,\ \pi_{\alpha_i}) \neq 1$. If this is the case, then simulate U_{i+1} aborting.

Fig. 11. Simulator for lock phase

Simulator for release phase

Case U_i is honest and U_{i+1} is corrupted

Upon U_{i+1} sending (release-req, U_{i+1}) to U_i, proceed as follows:

- Upon (released, σ_i) from $\mathcal{A}$ (on behalf of U_{i+1}), check if $\mathsf{Verify}(\sigma_i) \neq 1$. If this is the case, then simulate U_i aborting.

Case U_i is corrupted and U_{i+1} is honest

Upon U_{i+1} sending (Release, U_{i+1}) to $\mathcal{F}_{\text{GMHL}}$, proceed as follows:

- Send (release-req, U_{i+1}) to U_i.
- Upon (release-res, b) from $\mathcal{A}$ (on behalf of U_i), check if b=$\perp$. If this is the case, simulate U_{i+1} aborting. Otherwise, calculate the secret adaptor w_{i+1} (for U_n, $w_{n-1} = \mathsf{PSolve}(\mathsf{td}, P_{n-1})$; for U_{i+1}, $w_{i+1} = \mathsf{Ext}(\hat{\sigma}_{i+1}, \sigma_{i+1}, P_{i+1})$, $w_i = w_{i+1} \cdot r_{i+1}^{-1}$), generate the valid signature $\sigma_i = \mathsf{Adapt}(\hat{\sigma}_i, w_i)$ and send (released, σ_i) to U_i.

Fig. 12. Simulator for release phase

Proof. The proof follows directly from the security of the non-interactive zero-knowledge scheme NIZK.

Lemma 2. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that* $\text{EXEC}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}} \approx \text{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}}$.

Proof. It is worth mentioning that the difference of the two hybrids is whether the experiment outputs $\perp$, hence we bound the probability of such an event occurs in the following. Consider that the event $\perp$ happens in the case that an honest user U_i does not abort during the lock phase with a puzzle not obtained from the setup phase. Here we bound such probability by a reduction against the existential unforgeability of the signature scheme Σ. Assume a contradiction $\Pr[\perp|\mathcal{H}_1] \geq \frac{1}{poly(1^\lambda)}$ and we construct the following reduction. The reduction receives a public key pk as input and samples an index $k \in [1, q]$, where $q \in poly(1^\lambda)$ is the bound of the total number of interactions. We redirect the calls to the signing algorithm to the signing oracle and also specify that the reduction aborts once the setup phase is called. At the same time, the reduction returns the corresponding $(\hat{\sigma}'_i, P'_i, \pi'_{\alpha_i})$ if the event $\perp$ happens, or aborts.

The reduction is clearly efficient whenever k is guessed correctly and the reduction does not abort. If the lock phase is executed without calling the setup phase, the event $\perp$ happens. The lock phase takes P'_{i-1} as input and furthermore we have that $\mathsf{V}_{\text{NIZK}}(\pi'_{\alpha_i}, P'_i) = 1$ and $\mathsf{PreVf}(m_i, P'_i, \hat{\sigma}'_i) = 1$, which implies that U_{i+1} does not abort the execution of the lock phase and $(\hat{\sigma}'_i, P'_i, \pi'_{\alpha_i})$ is a valid forgery. By assumption this happens with probability at least $\frac{1}{q \cdot poly(1^\lambda)}$, which is a contradiction and proves that $\Pr[\perp|\mathcal{H}_1] \leq negl(1^\lambda)$.

Lemma 3. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that* $\text{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}} \approx \text{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}}$.

Proof. In this part we let the event $\perp$ that triggered in $\mathcal{H}_3$ but not in $\mathcal{H}_2$. We continue to show that the probability of such event occurs can be bounded by a negligible function in the security parameter, and a bound on the probability can be reduced to the security of the RP scheme, hardness of the relation R and unforgeability of the adaptor signature scheme $\Xi_{R,\Sigma}$. Assume a contradiction $\Pr[\perp|\mathcal{H}_2] \geq \frac{1}{poly(1^\lambda)}$ and we construct the following reduction. We are convinced that the probability of an adversary breaking the RP scheme is negligible and furthermore, the security of the RP scheme and the unforgeability of the adaptor signature also imply the hardness of the relation R. Hence, the remaining is to show that the bound of the probability $\perp$ occurs in $\mathcal{H}_2$ can be reduced to the unforgeability of the adaptor signature scheme $\Xi_{R,\Sigma}$. The reduction receives a public key pk, a pre-signature $\hat{\sigma}$ and a statement P as input and samples an index $k \in [1, q]$, where $q \in poly(1^\lambda)$ is a bound of the total number of interactions. The reduction replaces $\hat{\sigma}_{i-1}$ with $\hat{\sigma}$ and P_{i-1} with P in the lock phase, and set the public key $\mathsf{pk}_{\text{U}_{\text{i-1}}}$ generated in the k-th interaction to pk. We redirect the calls to the pre-signing and signing algorithms to the pre-signing and signing oracles respectively. At the same time, the reduction returns the corresponding σ'_{i-1} if the event $\perp$ happens, or aborts.

The reduction is clearly efficient whenever k is guessed correctly and the reduction does not abort. Please note that once the event $\perp$ happens, we have that

$\mathsf{Verify}(\sigma'_{i-1}) = 1$ and the release phase is not executed. This implies that U_{i-1} does not abort in the execution of release phase and σ'_{i-1} is a valid forgery. By assumption this happens with probability at least $\frac{1}{q \cdot poly(1^\lambda)}$, which is a contradiction and proves that $\Pr[\perp |\mathcal{H}_2] \leq negl(1^\lambda)$.

Lemma 4. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that* $\mathrm{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}} \approx \mathrm{EXEC}_{\mathcal{H}_4,\mathcal{A},\mathcal{E}}$.

Proof. Here we let $\perp$ be the event triggered in $\mathcal{H}_4$ but not in $\mathcal{H}_3$. It is worth mentioned that such event can be occurred in two scenarios. First, a corrupted user U_{i-1} presents a pre-signature $\hat{\sigma}'_{i-1}$ which successfully completes the pre-verification under the key $\mathsf{pk}_{\mathrm{U_{i\text{-}1}}}$ during the lock phase while cannot adapt to a valid signature in the release phase. Second, a corrupted user U_{i+2} produces a valid signature σ_{i+1} during the release phase while cannot extract a valid witness from it latter. Note that if the former happens, then the adversary has the ability to against the pre-signature adaptability, and if the latter happens, the adversary has the ability to against the witness extractability of the adaptor signature $\Xi_{R,\Sigma}$. Assume a contradiction $\Pr[\perp |\mathcal{H}_4] \geq \frac{1}{poly(1^\lambda)}$ and reflect on the following hybrid:

• Hybrid $\mathcal{H}'_3$: The pre-signature in the lock phase is set to $\hat{\sigma}'_{i-1} \xleftarrow{\$} \{0,1\}$ which successfully completes the pre-verification under the public key $\mathsf{pk}_{\mathrm{U_{i\text{-}1}}}$.

Because of the pre-signature adaptability property of the adaptor signature scheme $\Xi_{R,\Sigma}$, we have that $\Pr[\perp |\mathcal{H}'_3]=\Pr[\perp |\mathcal{H}_3]$.

The remaining is to show that the bound of the probability the event $\perp$ occurs in $\mathcal{H}'_3$ can be reduced to the against of witness extractability of the adaptor signature scheme $\Xi_{R,\Sigma}$. Assume a contradiction $\Pr[\perp |\mathcal{H}'_3] \geq \frac{1}{poly(1^\lambda)}$ and we construct the following reduction. The reduction takes a public key $\mathsf{pk}'_{\mathrm{U_{i\text{-}1}}}$ and a pre-signature $\hat{\sigma}'_{i-1}$ as input and samples an index $k \in [1, q]$, where $q \in poly(1^\lambda)$ is the bound of the total number of interactions. Here it replaces the pre-signature $\hat{\sigma}_{i-1}$ with $\hat{\sigma}'_{i-1}$ in the release phase and sets the public key $\mathsf{pk}_{\mathrm{U_{i\text{-}1}}}$ generated in the k-th interaction to $\mathsf{pk}'_{\mathrm{U_{i\text{-}1}}}$. We redirect the calls to signing an pre-signing algorithms to signing and pre-signing oracles respectively. The reduction returns the signature σ_{i-1} of U_{i-1} once the event $\perp$ happens, or aborts.

The reduction is clearly efficient whenever k is guessed correctly and the reduction does not abort. Please note that the event $\perp$ happens when we have $\mathsf{Verify}(\sigma_{i-1}) \neq 1$ without any parties aborting. Please note that U_i is honest so we are convinced that the signature σ_i is valid, therefore, it remains to show that the computed σ'_{i-1} is invalid. Because of the pre-signature adaptability property of the adaptor signature scheme $\Xi_{R,\Sigma}$, the Adapt algorithm works as expected. The only way for $\mathcal{H}'_3$ generating an invalid signature σ'_{i-1} is to compute a wrong witness w_{i-1} from puzzle P_{i-1}. Consider that $w_{i-1} = w_{i+1} \cdot r_{i+1}^{-1} \cdot r_i^{-1}$ and both users U_i and U_{i+1} are honest, which implies that the extracted witness w_{i+1} is invalid. It means that with a valid signature σ_{i+1}, we cannot extract a valid witness w_{i+1} from it and the adversary has the ability to against the witness extractability of the adaptor signature. By assumption this happens

Phase	**NDSS'19**[21]	**SP'21**[29]	**Our Protocol**
Setup	$4.04n$	$15.80n$	12.08
Lock	$13.42n$	$7.02n$	$1.01n$
Release	$4.57n$	$1.21n$	$0.27n$
Total	$22.03n$	$24.03n$	$12.08+1.28n$

Table 2. A comparison of the computational time across PCN protocols, with a path length of n. Times are reported in milliseconds(ms).

with probability at least $\frac{1}{q \cdot poly(1^\lambda)}$, which is a contradiction and proves that $\Pr[\bot\ |\mathcal{H}_3'] \leq negl(1^\lambda)$.

Lemma 5. *For all* PPT *distinguishers* $\mathcal{E}$ *it holds that* $\text{EXEC}_{\mathcal{H}_4,\mathcal{A},\mathcal{E}} \approx \text{EXEC}_{\mathcal{F}_{\text{GMHL}},\mathcal{A},\mathcal{E}}$.

Proof. The differences of the two experiment are only conceptual. Hence, indistinguishability follows.

This concludes the proof of Theorem 2.

6.3 Security analysis of PCN

In this section we present the security notion of our PCN construction and the proof of the following theorem.

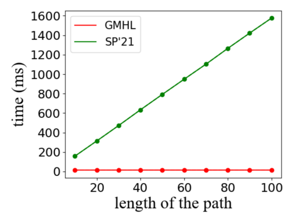
Theorem 3. *The protocol in Figure 6 UC-realizes* $\mathcal{F}_{\text{PCN}}$ *in the (*$\mathcal{F}_{\text{GC}}$*,* $\mathcal{F}_{\text{PC}}$*,* $\mathcal{F}_{\text{clock}}$*,* $\mathcal{F}_{\text{GMHL}}$*)-hybrid model.*

Proof. Observe that the ideal functionality $\mathcal{F}_{\mathsf{GMHL}}$ enforces atomicity and unlinkability properties of a PCN as proving above. The remaining information outside $\mathcal{F}_{\mathsf{GMHL}}$ are the changeable accounts and timeouts, and we set constant accounts and synchronized phases to preserve unlinkability. Furthermore, it is also trivial for simulator $\mathcal{S}$ to interactive with $\mathcal{F}_{\mathsf{GMHL}}$ and $\mathcal{F}_{\mathsf{PC}}$ on behalf of $\mathcal{F}_{\mathsf{PCN}}$.

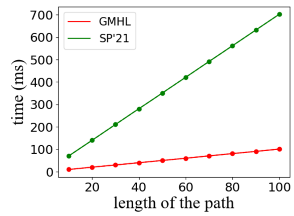
7 Performance Analysis

In this section we implemented our Guillou-Quisquater-based (GQ-based) PCN construction, then compared its performance against the Schnorr-based PCN construction from [21] and the BLS-based PCN construction from [29].

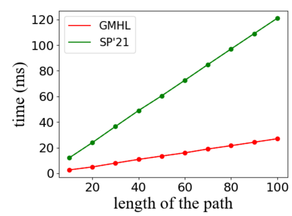
We consider an n-party payment path for a PCN, and the PCN protocols of [21] and [29] have the similar structure. Because the cost of the PCN protocols is dominated by the calls to locks, we simulate the instance of GMHL presented before and compare it with other locks about the timing cost of the operations perform in all phases. We measure the cost on a personal computer with 2.30GHz Intel(R) Core(TM) i7-10875H processor and 32GB memory. We use python library pypbc-0.2 for the arithmetic operations in class groups, the



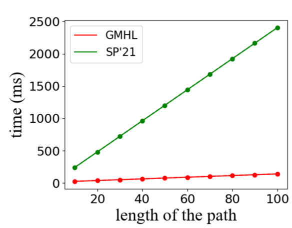
(a) computation time in the setup phase



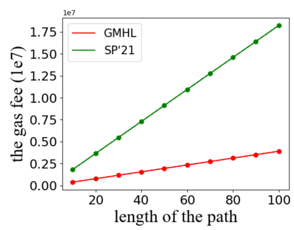
(b) computation time in the lock phase



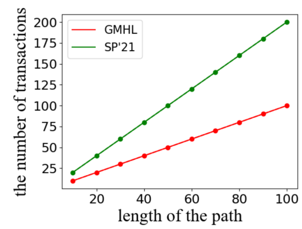
(c) computation time in the release phase



(d) time to complete payments



(e) the gas required to implement the schemes in Ropsten



(f) the number of transactions that need to be uploaded on chain

Fig. 13. The comparison between two general PCN protocols.

library rsa-4.8 for operations in RSA algorithm and the library numpy-1.22.1 for the cryptographic operations, and zero-knowledge protocols here have been implemented using Σ protocols [9]. We operate 1K times and calculate their average time, which are reported in Table 2. We do not consider the network latency in our measurements, since this does not constitute the efficiency bottleneck of our approach. From Table 2, we can see that the setup phase in our protocol outperforms other protocols in sense that the computation cost does not grow with the number of participators n while other protocols require a linear growth. Compared with the work in NDSS'19 [21], a similar work on constructing a PCN protocol, our GQ-based construction performs significantly better in the lock and release phase. Compared with another generalized construction in SP'21 [29], our construction not only remains good functionalities but also performs better in all phases.

We also compare with another general construction [29] in more details, which is shown in Figure 13. We measure the running time for each phase of the PCN protocol and the whole time to complete the entire payment. As can be seen from sub-figure (a), our protocol takes a constant time during the setup phase, while the other protocol takes time proportional to the length of the path. Besides, although the time required by our protocol in the lock and release phases also grows with the path length, it does not grow as fast as the other protocol [29]. Therefore, the time required for our protocol to complete the entire payment is much lower than that of [29]. When the path length is 100, our scheme takes only 140.08 ms, which is much lower than the 2408 ms required by [29]. Furthermore, we measure the computation cost in terms of the gas required by a smart contract implementing the protocols in Ropeten. We notice that setting up the corresponding contract of our protocol requires 39075 unit of gas per hop, while [29] needs 182390 unit of gas. Finally, we also compare the number of transactions that need to be uploaded when the payment channel network is expired. Parties involved in the PCN protocol [29] need to post 2 transactions on the chain to close their channels, instead of 1 as ours. In summary, we can find that our construction has better advantages in various comparisons.

8 Conclusion

In this work we devised a Generalized Multi-Hop Locks(GMHL) which supports both atomicity and unlinkability. We then presented a general PCN construction based on GMHL to reach all the benefits of atomocity, unlinkability and generality. Furthermore, We proposed a Guillou-Quisquater-based GMHL instance and compared with other constructions, showing that our proposal has a lightweight setup and is efficient comparable to other constructions. As GMHL dominates the performances of PCN, the proposed GMHL can be regarded as a promising tool to realize efficient PCNs.

References

1. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized channels from limited blockchain scripts and adaptor signatures. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 635–664. Springer (2021)
2. Aumayr, L., Moreno-Sanchez, P., Kate, A., Maffei, M.: Blitz: Secure multi-hop payments without two-phase commits. In: 30th USENIX Security Symposium (USENIX Security) (2021)
3. Blass, E.O., Kerschbaum, F.: Strain: A secure auction for blockchains. In: European Symposium on Research in Computer Security (ESORICS). pp. 87–110. Springer (2018)
4. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. white paper **3**(37) (2014)
5. Camenisch, J., Krenn, S., Shoup, V.: A framework for practical universally composable zero-knowledge protocols. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 449–467. Springer (2011)
6. Camenisch, J., Lysyanskaya, A.: A formal treatment of onion routing. In: Annual International Cryptology Conference (CRYPTO). pp. 169–187. Springer (2005)
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science (FOCS). pp. 136–145. IEEE (2001)
8. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Theory of Cryptography Conference (TCC). pp. 61–85. Springer (2007)

9. Damgård, I.: On σ-protocols. Lecture Notes, University of Aarhus, Department for Computer Science p. 84 (2002)
10. Dang, H., Le Tien, D., Chang, E.C.: Towards a marketplace for secure outsourced computations. In: European Symposium on Research in Computer Security (ESORICS). pp. 790–808. Springer (2019)
11. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015. vol. 9212, pp. 3–18. Springer (2015)
12. Dziembowski, S., Eckey, L., Faust, S., Hesse, J., Hostáková, K.: Multi-party virtual state channels. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT,). pp. 625–656. Springer (2019)
13. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 106–123. IEEE (2019)
14. Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 473–489 (2017)
15. Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In: Network and Distributed System Security Symposium (NDSS) (2017)
16. Johnson, R., Molnar, D., Song, D., Wagner, D.: Homomorphic signature schemes. In: Cryptographers' track at the RSA conference (CT-RSA). pp. 244–262. Springer (2002)
17. Jourenko, M., Larangeira, M., Tanaka, K.: Payment trees: Low collateral payments for payment channel networks. In: International Conference on Financial Cryptography and Data Security (FC). pp. 189–208. Springer (2021)
18. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Theory of Cryptography Conference (TCC). pp. 477–498. Springer (2013)
19. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 17–30 (2016)
20. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 455–471. ACM (2017)
21. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019. The Internet Society (2019)
22. Mavroudis, V., Wüst, K., Dhar, A., Kostiainen, K., Capkun, S.: Snappy: Fast on-chain payments with practical collaterals. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020. The Internet Society (2020)
23. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: International Conference on Financial Cryptography and Data Security (FC). pp. 508–526. Springer (2019)

24. Mohanty, S.K., Tripathy, S.: n-htlc: Neo hashed time-lock commitment to defend against wormhole attack in payment channel networks. Computers & Security **106**, 102291 (2021)
25. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review p. 21260 (2008)
26. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
27. Tairi, E., Moreno-Sanchez, P., Maffei, M.: A^2L: Anonymous atomic locks for scalability in payment channel hubs. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1834–1851 (2021)
28. Takahashi, T., Otsuka, A.: Probabilistic micropayments with transferability. In: European Symposium on Research in Computer Security (ESORICS). pp. 390–406. Springer (2021)
29. Thyagarajan, S.A.K., Malavolta, G.: Lockable signatures for blockchains: Scriptless scripts for all signatures. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco,CA, USA, 24-27 May 2021. pp. 937–954. IEEE (2021)
30. Tripathy, S., Mohanty, S.K.: Mappcn: Multi-hop anonymous and privacy-preserving payment channel network. In: International Conference on Financial Cryptography and Data Security (FC). pp. 481–495. Springer (2020)
31. Xiao, Y., Zhang, N., Li, J., Lou, W., Hou, Y.T.: Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution. In: European Symposium on Research in Computer Security (ESORICS). pp. 610–629. Springer (2020)