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, huazeg@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., [27], [30], [9], [3]) based on blockchain have sprung up. Particularly, the appearance of many blockchain-based cryptocurrencies (e.g., Bitcoin[24], 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., [18], [21], [10], [25]) 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., [13], [12], [14], [26]) 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. [25] 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 such as payment tree [16], punishment mechanism [2], and smart contracts [22]. Nevertheless, the aforementioned protocols expose the payment path directly without considering the privacy-preserving problem. To address this issue, Molavolta et al. [19] 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. [29] put forward an efficient privacy-preserving PCN relying on elliptic curve cryptography, and Mohanty et al. [23] proposed an efficient and privacy-preserving PCN with an enhanced HTLC protocol named $n$-HTLC. Molavolta et al. [20] 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 com-
GMHL: Generalized Multi-Hop Locks

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

<table>
<thead>
<tr>
<th>Construction</th>
<th>Atomicity</th>
<th>Unlinkability</th>
<th>Generality</th>
<th>Lightweight setup</th>
<th>Required functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTLC [25]</td>
<td>•</td>
<td>o</td>
<td>o</td>
<td>•</td>
<td>HTLC</td>
</tr>
<tr>
<td>FC’19 [22]</td>
<td>•</td>
<td>o</td>
<td>o</td>
<td>•</td>
<td>smart contract</td>
</tr>
<tr>
<td>USENIX Sec’21 [2]</td>
<td>•</td>
<td>o</td>
<td>o</td>
<td>•</td>
<td>HTLC</td>
</tr>
<tr>
<td>CCS’17 [19]</td>
<td>•</td>
<td>•</td>
<td>o</td>
<td>•</td>
<td>HTLC</td>
</tr>
<tr>
<td>FC’20 [29]</td>
<td>•</td>
<td>•</td>
<td>o</td>
<td>•</td>
<td>ECC</td>
</tr>
<tr>
<td>NDSS’19 [20]</td>
<td>•</td>
<td>•</td>
<td>o</td>
<td>•</td>
<td>ECDSA/Schnorr</td>
</tr>
<tr>
<td>SP’21 [28]</td>
<td>•</td>
<td>•</td>
<td>o</td>
<td>•</td>
<td>MPC and signature verification</td>
</tr>
<tr>
<td>Our construction</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>signature verification</td>
</tr>
</tbody>
</table>

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. 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 underlining 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 [28], 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 $G = (V, E)$, where the set $V$ of vertices represents the user accounts and the set $E$ of weighted edges represents the payment channels. The non-negative number associated with the vertex $U \in V$ denotes the fees it charges for forwarding a payment. The weight of a directed edge $(U_1, U_2) \in E$ 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 $\alpha$ coins to receiver $R$ through a path $S \rightarrow U_1 \rightarrow ... \rightarrow U_n \rightarrow R$. Each user $U_i$ on this path must have a capacity $\gamma_i \geq \alpha'_i$ where $\alpha'_i = \alpha - \sum_{k=1}^{i} \text{fee}(U_k)$ to ensure the payment can be successfully completed. Thus $S$ starts the payment with $\alpha^*$ coins where $\alpha^* = \alpha + \sum_{k=1}^{n} \text{fee}(U_k)$ to guarantee that $R$ will receive exactly $\alpha$ coins. We refer readers to [19] for further details.

2.2 Preliminaries

**Notations.** We denote by $1^\lambda \in \mathbb{N}^+$ the security parameter and denote by $x \leftarrow S$ the uniformly sampling of an element from a set $S$. 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 use $L$ to denote a set of positive instances corresponding to the relation $R$, where $L = \{x | \exists w \text{ s.t. } R(x, w) = 1\}$. A non-interactive zero-knowledge proof scheme NIZK consists of two algorithms, $P_{\text{NIZK}}$ and $V_{\text{NIZK}}$. $P_{\text{NIZK}}$ is a prover algorithm and its expression is $\pi \leftarrow P_{\text{NIZK}}(x, w)$. $V_{\text{NIZK}}$ is a verification algorithm and its expression is $\{0, 1\} := V_{\text{NIZK}}(x, \pi)$. The NIZK scheme ensures that the prover can prove to the verifier that he does know the secret without revealing additional
knowledge to the verifier. We model the security of a NIZK scheme by an ideal functionality $F_{NIZK}$ in Appendix and refer readers to [5] for the definition of security of zero-knowledge functionality in UC framework.

**Adaptor signature scheme.** An adaptor signature scheme is defined with respect to a hard relation $R$ and a digital signature scheme $\Sigma$. It consists of four algorithms $\Xi_{R,\Sigma} = (\text{PreSig}, \text{PreVf}, \text{Adapt}, \text{Ext})$. With a statement/witness pair $(Y, y) \in R$, a secret/public key pair $(sk, pk) \leftarrow \Sigma.KGen(1^\lambda)$ and a message $m \in M$, we are able to generate a pre-signature with $\hat{\sigma} \leftarrow \text{PreSig}(sk, m, Y)$, adapt a valid signature with $\sigma := \text{Adapt}(\hat{\sigma}, y)$, verify a pre-signature with $\text{PreVf}(m, Y, \hat{\sigma})$ and extract the witness with $y := \text{Ext}(\sigma, \hat{\sigma}, Y)$. Adaptor signature was formally defined in [1]. If an adaptor signature provides pre-signature correctness, pre-signature adaptability and witness extractability, it is secure. Briefly, pre-signature correctness, or existential unforgeability under chosen message attack for adaptor signature (aEUF-CMA), ensures that any honestly generated pre-signature $\hat{\sigma}$ with respect to a statement $Y$ must be valid and the adapted signature $\sigma$ from it is valid as well. Pre-signature adaptability ensures that any valid pre-signature $\hat{\sigma}$ can be adapted into a valid signature $\sigma$ 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 $RP = (PSetup, PGen, PSolve, PRand)$. With a public parameters/trapdoor pair $(pp, td) \leftarrow PSetup(1^\lambda)$, we can generate a puzzle $Z \leftarrow PGen(pp, \zeta)$, solve the puzzle with $\zeta := PSolve(td, Z)$ and randomize the puzzle to a fresh puzzle with $(Z', r) \leftarrow PRand(pp, Z)$ which $\phi(\zeta, r)$ is the solution to the puzzle $Z'$. Randomizable puzzle was formally defined in [26], 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
via secure and authenticated communication channels. We model the attacker $A$ as a PPT machine with an interface $\text{corrupt}(\cdot)$, 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 outgoing messages of $P$ are routed through $A$. In this work, we consider the static corruption model that are frequently used in papers [19], [26], [20], namely, the attacker commits ahead of time the identifiers of the parties it intends to corrupt.

**Communication model.** We specify a synchronous communication network whose communication rounds are discrete. As in [11], [17], we denote by $F_{\text{clock}}$ 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., $F_{\text{clock}}$) ticks. In this work, we treat the ideal functionality $F_{\text{clock}}$ as a global ideal functionality in the GUC model to ensure all parties are aware of the given round. Then we denote by $F_{\text{GC}}$ the formalization of communication channels as in [11]. 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 $F_{\text{st}}$ 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 $F_{\text{ano}}$ [6] the anonymous communication channels for users. It is similar to $F_{\text{st}}$ except omitting the identifier of the sender from the message sent to the receiver.

**Payment channels.** We denote by $F_{\text{PC}}$ 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 $F_{\text{PCN}}$ ideal functionality is based on $F_{\text{PC}}$ and $F_{\text{clock}}$, we define the UC-realization with respect to the aforementioned global functionalities. We denote by $\pi$ the protocol access to $F_{\text{PC}}$ and $F_{\text{clock}}$ and denote by $\text{EXEC}_{\pi, A, E}$ the ensemble of the outputs of the environment $E$ when interacting with the attacker $A$ and users running protocol $\pi$. The UC-realization with respect to the global ideal functionalities is defined as follows:

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

**Ideal Functionality.** We define an ideal functionality $F_{\text{GMHL}}$ in $(F_{\text{GC}}, F_{\text{st}}, F_{\text{ano}})$-hybrid model for GMHL. Then formalize the notion of PCN relying on
GMHL and define an ideal functionality $F_{PCN}$ in $(F_{GC}, F_{xt}, F_{ano}, F_{GMHL})$-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 [14]. 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 $U_i^0$, there is an edge labeled with $e$ in the graph, linking from sender $U_i^0$ to some receiver $U_j^n$. An interaction graph is compatible if it explains the view of the intermediate user, namely, the number of edges labeled with $e$ incident to $U_j^n$ equals to the number of coins received by $U_j^n$. 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

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, $U_n$ generates a cryptographic puzzle $P$ and sends it to $U_0$ through a secure communication channel. Note that only $U_n$ knows the solution to the puzzle $P$. 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 $U_{i+1}$ solving puzzle $P$. In the release phase, $U_n$ updates the channel between $U_{n-1}$ and $U_n$ with the solution to puzzle $P$ and releases the coins promised by $U_{n-1}$ before. Our protocol guarantees that the solution to the puzzle can be
extracted from a valid channel update. Therefore, $U_{n-1}$ can get the solution and release coins from $U_{n-2}$ after $U_n$ updating the channel. The operation of the intermediate user $U_i$ is the same as that of $U_{n-1}$. Until the sender $U_0$ receiving 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 former added randomness $r$.

Using these tools, our solution for atomicity and unlinkability is shown in Figure 1. In the setup phase, Dave generates (using $\text{PGen}$) a puzzle $P_\alpha$ (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 $P_\alpha$. Since Bob does not have the solution to puzzle $P_\alpha$, he uses $\text{PRand}$ to randomize it to a fresh puzzle $P_\beta$ (i.e., the blue lock) and initiates a payment to Coral conditioned on $P_\beta$. Similarly, Coral pays to Dave conditioned on $P_\gamma$ (i.e., the green lock). In the release phase, Dave solves puzzle $P_\gamma$ (using $\text{PSolve}$), sends the solution (i.e., the green key) to Coral and releases the coins promised by Coral. Coral gets the solution to $P_\gamma$ from update channel and removes the randomness added before (using $\text{DeRand}$) to release coins. Bob does the same as Coral. Now the release phase is finished, Alice can pay coins to Dave as expected.
The GMHL protocol \( \Pi \)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \text{Setup}_{U_0} ) : ( (pk, sk) \leftarrow \text{Gen}(1^\lambda) )</td>
</tr>
<tr>
<td>2</td>
<td>( (pp, td) := (pk, sk) )</td>
</tr>
<tr>
<td>3</td>
<td>( r \leftarrow D )</td>
</tr>
<tr>
<td>4</td>
<td>( (P_0, r_0) \leftarrow \text{PGen}(pp, r) )</td>
</tr>
<tr>
<td>5</td>
<td>( \pi_{a_0} \leftarrow \text{PNIZK}({(\alpha_0</td>
</tr>
<tr>
<td>6</td>
<td>send ( (P_0, \pi_{a_0}) ) to ( U_0 )</td>
</tr>
<tr>
<td>7</td>
<td>if ( \text{PNIZK}(\pi_{a_0}, P_0) \neq 1 ) then abort ( )</td>
</tr>
<tr>
<td>9</td>
<td>return ( T )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>( \text{Lock}<em>{U_1}(P</em>{i-1}) : ) ( )</td>
</tr>
<tr>
<td>11</td>
<td>( \sigma_0 \leftarrow \text{PreSig}(sk_{U_0}, m_0, P_0) )</td>
</tr>
<tr>
<td>12</td>
<td>send ( (\sigma_0, P_0, \pi_{a_0}) ) to ( U_1 )</td>
</tr>
<tr>
<td>13</td>
<td>return ( (\sigma_0, P_0, \pi_{a_0}) )</td>
</tr>
<tr>
<td>14</td>
<td>if ( \text{PNIZK}(\pi_{a_1-1, \sigma_{a_1-1}}) = 1 ) then abort ( )</td>
</tr>
<tr>
<td>15</td>
<td>( (P_i, r_i) \leftarrow \text{PRand}(pp, P_{i-1}) )</td>
</tr>
<tr>
<td>16</td>
<td>( \pi_{a_1} \leftarrow \text{PNIZK}({(\exists a_{i}</td>
</tr>
<tr>
<td>17</td>
<td>send ( (\pi_i, P_i, \pi_{a_1}) ) to ( U_{i+1} )</td>
</tr>
<tr>
<td>18</td>
<td>return ( (\pi_i, \pi_{a_1}, \sigma_{a_1}) )</td>
</tr>
<tr>
<td>19</td>
<td>if ( \text{Vrfy}(\sigma_{a_1}) \neq 1 ) then abort ( )</td>
</tr>
<tr>
<td>20</td>
<td>( w_i = \text{Ext}(\sigma_{i-1}, P_i) )</td>
</tr>
<tr>
<td>21</td>
<td>( w_{i-1} \leftarrow w_i \cdot r_i^{-1} )</td>
</tr>
<tr>
<td>22</td>
<td>( \sigma_{i-1} = \text{Adapt}(\sigma_{i-1}, w_{i-1}) )</td>
</tr>
<tr>
<td>23</td>
<td>send ( \sigma_{i-1} ) to ( U_{i-1} )</td>
</tr>
<tr>
<td>24</td>
<td>return ( \sigma_{i-1} )</td>
</tr>
</tbody>
</table>

Fig. 2. Algorithms and protocols for the generic construction

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

**Our approach: ** We here use adaptor signature to overcome this challenge. With 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 it 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 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 $\Pi$. This can be achieved by utilizing a randomizable puzzle and an adaptor signature $\Xi_{R,\Sigma}=(\text{PreSig}, \text{PreVf}, \text{Adapt}, \text{Ext})$ for the signature $\Sigma=(\text{Gen}, \text{Sign}, \text{Vrfy})$ used by the underlying ledger and a hard relation $R$. We assume that statement/witness pairs of $R$ are public/secret key of $\Sigma$ 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 $P_0$ related to $r$ (i.e., the secret adaptor), produces a NIZK proof $\pi_{\alpha_0}$ proving that $\alpha_0$ is the solution to puzzle $P_0$ (lines 4-6 in Figure 2) and sends a puzzle/proof pair $(P_0, \pi_{\alpha_0})$ 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 between user $U_0$ and $U_1$) and shares the (pre-signature, puzzle, proof) tuple $(\hat{\sigma}_0, P_0, \pi_{\alpha_0})$ 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 $\text{PRand}$ algorithm, produces a NIZK proof $\pi_{\alpha_i}$ over puzzle $P_i$, generates a pre-signature $\hat{\sigma}_i$ over the information that $U_i$ and $U_{i+1}$ agreed in advance 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 $\sigma_{n-1}$ (lines 19-20 in Figure 2). For intermediate user $U_i$, once he is convinced of the validity of signature $\sigma_i$, he extracts the witness $w_i$ from it using $\text{Ext}$ algorithm (lines 19-20 in Figure 2), removes the randomness added before, produces a valid signature $\sigma_{i-1}$ with $\text{Adapt}$ 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 $\Pi_{GQ}$ which is based on a Guillou-Quisquater adaptor signature $\Sigma_{GQ}$ 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 $\Phi$ to be RSA-based homomorphic encryption scheme[15], its message space $M = \mathbb{Z}_N$ and solution space $S = \mathbb{Z}_N$. Our construction is shown in Construction 1. In the construction, we wrap an integer in a puzzle with $\text{PGen}$ and $\text{PRand}$ algorithms and unwrap it with $\text{PSolve}$ algorithm.
GMHL: Generalized Multi-Hop Locks

Public parameters: \((N, r, X)\), message \(m\)

**Setup**

1. \(\text{Setup}_{U_0} : \) \((N, e, d, X, x) \leftarrow \Sigma_{GQ}.KGen(1^\lambda)\)
2. \(\text{pk} = (N, e, x)\)
3. \(sk = (N, d)\)
4. \((pp, td) = (\text{pk}, \text{sk})\)
5. \(t \leftarrow Z_n\)
6. \((P_0, t_0) \leftarrow PGen(pp, t)\)
7. \(\pi_w_0 \leftarrow \text{PNIIZK}\{\exists w_0 | \text{Dec}(td, P_0) = w_0\}, w_0\)
8. \((P_0, \pi_w_0) \leftarrow \text{PNIIZK}(P_0, \pi_w_0)\)
9. If \(\text{VNIIZK}(P_0, \pi_w_0) \neq 1\) then abort
10. return \(\top\)

**Construction 1**

The randomizable puzzle scheme is constructed as follows:

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

\[\text{PGen}(pp, \zeta) : \text{parse } pp \text{ as } pk^\Phi, \text{sample } r \leftarrow S, \text{compute } c \leftarrow \text{Enc}(pk^\Phi, \zeta \cdot r), \text{set } Z := c, \text{and return } (Z, r).\]

\[\text{PSolve}(td, Z) : \text{parse } td \text{ as } sk^\Phi, \text{compute } \zeta' \leftarrow \text{Enc}(sk^\Phi, Z), \text{and return } \zeta'.\]

\[\text{PRand}(pp, Z) : \text{parse } pp \text{ as } (N, e), \text{sample } r' \leftarrow S, \text{compute } c' = c \cdot r'^e \mod N, \text{set } Z' := c', \text{and return } (Z', r').\]

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

**Theorem 1.** Let \(\Phi\) 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 \(\Phi\). For the notion of privacy, note that for a puzzle \(Z\) and its solution \(r\zeta\) and a randomizable puzzle \(Z'\) with the solution \(r'\zeta\), the space of \(r\zeta\) and \(r'\zeta\) are the same, they are all in the field \(S\), 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 \((pk, sk)\) where \(pk = (N, e, X)\) and \(sk = (N, x)\) 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 7-8 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 agreement on a randomness \(R = (k_1k_2t_0t)^e \mod N\). It is worth mentioning that
Public parameters: \((N, e, X)\), message \(m_i\)

<table>
<thead>
<tr>
<th>Public parameters: ((N, e, X)), message (m_i)</th>
<th>(\text{Lock}<em>{U_i}(\text{sk}</em>{U_i}, P_{i-1}))</th>
<th>(\text{Lock}<em>{U</em>{i+1}}(\text{sk}<em>{U</em>{i+1}}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>((P_i, t_i) \leftarrow \text{PRand}(pp, P_{i-1}))</td>
<td>(P_i, \pi_{w_i})</td>
<td>((P_{i+1}, \pi_{w_{i+1}}))</td>
</tr>
<tr>
<td>(\pi_{w_i} \leftarrow \text{PNIzk}({3w_i</td>
<td>\text{Dec}(td, P_i) = w_i}, w_i))</td>
<td>If (\text{NIZK}(P_i, \pi_{w_i}) \neq 1) then abort</td>
</tr>
<tr>
<td>(r_2 = k_2^s \mod N)</td>
<td>(\pi_{w_{i+1}} \leftarrow \text{PNIzk}({3w_{i+1}</td>
<td>r_{i+1} = k_2^s \mod N}, k_2))</td>
</tr>
</tbody>
</table>

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

\(t_{0t}\) 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 5-15 in Figure 4). Then both \(U_0\) and \(U_1\) calculate a hash \(\alpha_0\) with related to the previously agreed message \(m_0\) and randomness \(R\) (line 16 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 \((\alpha_0, t_{0t}\hat{\beta}_0)\) (lines 17-21 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 \( \text{Dec} \)) to get the secret of puzzle \( P_{n-1} \) (line 2 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 4-7 in Figure 5). At this point, \( U_i \) can adapt the pre-signature \( \hat{\sigma}_{i-1} \) to a valid signature \( \sigma_{i-1} \) because of the pre-signature adaptability feature of adaptor signature (line 8 in Figure 5). Once user \( U_{i-1} \) is certain about the validity of signature \( \sigma_{i-1} \) (line 10 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 \( \Delta \) 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 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 \( \sigma_i \) that \( \text{Verify}(\sigma_i) = 1 \) before time \( T_i \) expires, then channel \( c_i \) is updated with \( \text{c}_i\text{.cash}(U_i) -= p \) and \( \text{c}_i\text{.cash}(U_{i+1}) += p \) (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 in universal composability (UC) framework from Canetti[7] and the synchronous version of global UC framework (GUC)[8]. We first describe an ideal functionality \( \mathcal{F}_{\text{GMHL}} \) 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 \( \mathcal{F}_{\text{GMHL}} \), namely, any possible attacks in our construction can be simulated in \( \mathcal{F}_{\text{GMHL}} \). Furthermore, we specify an ideal functionality \( \mathcal{F}_{\text{PCN}} \) relying on \( \mathcal{F}_{\text{GMHL}} \) to cover the security notions of our PCN construction in Section 5.2. The proof of such construction is similar to \( \mathcal{F}_{\text{GMHL}} \).
provides three interfaces, the payment operation from sender to receiver is constant (i.e., the 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 \((<\text{pid}_0, b>, <\text{pid}_1, b>, ..., <\text{pid}_n, b>)\) where \(b\) is used to indicate whether the puzzle has been solved. At the same time, \(\mathcal{F}_{\text{GMHL}}\) 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 \(\mathcal{F}_{\text{GMHL}}\) is in Figure 7.

**Ideal functionality \(\mathcal{F}_{\text{GMHL}}\).** We define an ideal functionality \(\mathcal{F}_{\text{GMHL}}\) in \((\mathcal{F}_{\text{GC}}, \mathcal{F}_s, \mathcal{F}_{\text{ano}})\)-hybrid model for GMHL. \(\mathcal{F}_{\text{GMHL}}\) 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 \((<\text{pid}_0, b>, <\text{pid}_1, b>, ..., <\text{pid}_n, b>)\) where \(b\) is used to indicate whether the puzzle has been solved. At the same time, \(\mathcal{F}_{\text{GMHL}}\) 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 \(\mathcal{F}_{\text{GMHL}}\) is in Figure 7.

**Ideal functionality \(\mathcal{F}_{\text{PCN}}\).** We here formalize the notion of PCN relying on GMHL and define an ideal functionality \(\mathcal{F}_{\text{PCN}}\) in \((\mathcal{F}_{\text{GC}}, \mathcal{F}_s, \mathcal{F}_{\text{ano}}, \mathcal{F}_{\text{GMHL}})\)-hybrid model. \(\mathcal{F}_{\text{PCN}}\) 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., \(\Delta\)) and the current time is \(t\). In this model, we do not take transaction fees into account to ensure the security of the model. \(\mathcal{F}_{\text{PCN}}\) 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 \(\mathcal{F}_{\text{GMHL}}\). The description of the ideal functionality \(\mathcal{F}_{\text{PCN}}\) is depicted in Figure 8.
6.1 Security Analysis of GMHL

In this section we will prove the security of our GMHL by the following theorem.

**Theorem 2.** Let $\Sigma$ be EUF-CMA secure signature schemes, $R$ be a hard relation, $\Xi_{R,\Sigma}$ be a secure adaptor signature scheme and $\mathcal{R}$ be a secure and private randomizable puzzle scheme, then the construction in Figure 2 UC-realizes the ideal functionality $\mathcal{F}_{\text{GMHL}}$ in the $(\mathcal{F}_{\text{GC}}, \mathcal{F}_{\text{ano}}, \mathcal{F}_{\text{st}})$-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 $\mathcal{F}_{\text{NIZK}}$ 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 $(\hat{\sigma}_i, P_i, \pi_{\sigma_i})$, 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 $\sigma_{i-1}$ such that $\text{Verify}(\sigma_{i-1}) = 1$, before a valid signature $\sigma_i$ is output from an execution of the release phase which can
Lemma 1. For all PPT distinguishers $\mathcal{E}$ it holds that
EXEC$_{\mathcal{H}_0, \mathcal{A}, \mathcal{E}} \simeq$ EXEC$_{\mathcal{H}_1, \mathcal{A}, \mathcal{E}}$.

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
EXEC$_{\mathcal{H}_1, \mathcal{A}, \mathcal{E}} \simeq$ EXEC$_{\mathcal{H}_2, \mathcal{A}, \mathcal{E}}$.
Proof. It is worth mentioning that the difference of the two hybrs is whether the experiment outputs ⊥, hence we bound the probability of such an event
occurs in the following. Consider that the event \( \bot \) 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 \( \Sigma \). Assume a contradiction \( \Pr[\bot | H_1] \geq \frac{1}{\text{poly}(1^\lambda)} \) and we construct the following reduction. The reduction receives a public key \( pk \) as input and samples an index \( k \in [1, q] \), where \( q \in \text{poly}(1^\lambda) \) 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 \((\hat{\sigma}_i', P_i', \pi_{i_0}')\) if the event \( \bot \) 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 \( \bot \) happens. The lock phase takes \( P_{i-1}' \) as input and furthermore we have that \( \text{VNIKZ}(\pi_{i_1}', P_i') = 1 \) and \( \text{PreV}(m_i, P_i', \hat{\sigma}_i') = 1 \), which implies that \( U_{i+1} \) does not abort the execution of the lock phase and \((\hat{\sigma}_i', P_i', \pi_{i_0}')\) is a valid forgery. By assumption this happens with probability at least \( \frac{1}{q \cdot \text{poly}(1^\lambda)} \), which is a contradiction and proves that \( \Pr[\bot | H_1] \leq \text{negl}(1^\lambda) \).

**Lemma 3.** For all PPT distinguishers \( \mathcal{E} \) it holds that \( \text{EXEC}_{H_2, A, \mathcal{E}} \approx \text{EXEC}_{H_3, A, \mathcal{E}} \).

**Proof.** In this part we let the event \( \bot \) that triggered in \( H_3 \) but not in \( H_2 \). 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 \( \Pr[\bot | H_2] \geq \frac{1}{\text{poly}(1^\lambda)} \) 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 \( \bot \) occurs in \( H_2 \) 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 \( q \in \text{poly}(1^\lambda) \) 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 \( pk_{i-1} \) 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 \( \hat{\sigma}_{i-1}' \) if the event \( \bot \) 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 \( \bot \) happens, we have that \( \text{Verify}(\sigma_{i-1}') = 1 \) and the release phase is not executed. This implies that \( U_{i-1} \) does not abort in the execution of release phase and \( \sigma_{i-1}' \) is a valid forgery. By assumption this happens with probability at least \( \frac{1}{q \cdot \text{poly}(1^\lambda)} \), which is a contradiction and proves that \( \Pr[\bot | H_2] \leq \text{negl}(1^\lambda) \).
Lemma 4. For all PPT distinguishers $E$ it holds that

\[ \text{EXEC}_{H_3, A, E} \approx \text{EXEC}_{H_4, A, E}. \]

Proof. Here we let $\bot$ be the event triggered in $H_4$ but not in $H_3$. 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 $pk_{U_{i-1}}$ 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 $\sigma_{i+1}$ 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 $\Pr[\bot | H_4] \geq \frac{1}{\lambda^4}$ and reflect on the following hybrid.

- Hybrid $H'_4$: The pre-signature in the lock phase is set to $\hat{\sigma}'_{i-1} \notin \{0,1\}$ which successfully completes the pre-verification under the public key $pk_{U_{i-1}}$.

Because of the pre-signature adaptability property of the adaptor signature scheme $\Xi_{R, \Sigma}$, we have that $\Pr[\bot | H'_4] = \Pr[\bot | H_3]$.

The remaining is to show that the bound of the probability the event $\bot$ occurs in $H'_3$ can be reduced to the against of witness extractability of the adaptor signature scheme $\Xi_{R, \Sigma}$. Assume a contradiction $\Pr[\bot | H'_3] \geq \frac{1}{\lambda^4}$ and we construct the following reduction. The reduction takes a public key $pk_{U_{i-1}}$ and a pre-signature $\hat{\sigma}'_{i-1}$ as input and samples an index $k \in [1, q]$, where $q \in \text{poly}(1^\lambda)$ 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 $pk_{U_{i-1}}$ generated in the $k$-th interaction to $pk'_{U_{i-1}}$. We redirect the calls to signing an pre-signing oracles respectively. The reduction returns the signature $\sigma_{i-1}$ of $U_{i-1}$ once the event $\bot$ happens, or aborts.

The reduction is clearly efficient whenever $k$ is guessed correctly and the reduction does not abort. Please note that the event $\bot$ happens when we have $\text{Verify}(\sigma_{i-1}) \neq 1$ without any parties aborting. Please note that $U_i$ is honest so we are convinced that the signature $\sigma_i$ is valid, therefore, it remains to show that the computed $\hat{\sigma}'_{i-1}$ 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 $H'_3$ generating an invalid signature $\hat{\sigma}'_{i-1}$ is to compute a wrong witness $w_{i-1}$ from puzzle $P_{i-1}$. Consider that $w_{i-1} = w_{i+1} \cdot r_{i+1}^{-1} \cdot r_{i-1}^{-1}$ 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 $\sigma_{i+1}$, 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 with probability at least $\frac{1}{\lambda^4}$, which is a contradiction and proves that $\Pr[\bot | H'_3] \leq \text{negl}(1^\lambda)$.

Lemma 5. For all PPT distinguishers $E$ it holds that

\[ \text{EXEC}_{H_4, A, E} \approx \text{EXEC}_{\text{GMHL}, A, E}. \]
### Table 2. A comparison of the computational time across PCN protocols, for a path length of n. Times are reported in milliseconds (ms).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>4.04n</td>
<td>15.80n</td>
<td>12.08n</td>
</tr>
<tr>
<td>Lock</td>
<td>13.42n</td>
<td>7.02n</td>
<td>1.01n</td>
</tr>
<tr>
<td>Release</td>
<td>4.57n</td>
<td>1.21n</td>
<td>0.27n</td>
</tr>
</tbody>
</table>

Proof. The differences of the two experiments are only conceptual. Hence, indistinguishability follows. This concludes the proof of Theorem 2.

#### 6.2 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 \( F_{PCN} \) in the \((F_{GC}, F_{PC}, F_{clock}, F_{GMHL})\)-hybrid model.

**Proof.** Observe that the ideal functionality \( F_{GMHL} \) enforces atomicity and un-linkability properties of a PCN as proving above. The remaining information outside \( F_{GMHL} \) are the changeable accounts and timeouts, and we set constant accounts and synchronized phases to preserve un-linkability. Furthermore, it is also trivial for simulator \( S \) to interactive with \( F_{GMHL} \) and \( F_{PC} \) on behalf of \( F_{PCN} \).

#### 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 [20] and BLS-based PCN construction from [28].

We consider an \( n \)-party payment path for a PCN, and the PCN protocols of [20] and [28] are the similar structure. Because the cost of the PCN protocols is dominated by the calls to locks, we simulated the instance of GMHL presented before and compared it with other locks about the timing cost of the operations performed in each phase: setup phase, lock phase and release phase. We measured the cost on a personal computer with 2.30GHz Intel(R) Core(TM) i7-10875H processor and 32GB memory. We used python library pypbc-0.2 for the arithmetic operations in class groups, the library rsa-4.8 for operations in RSA algorithm and the library numpy-1.22.1 for the cryptographic operations, and the zero-knowledge proof here was \( \Sigma \) scheme. We evaluated the Schnorr-based PCN construction [20], BLS-signature based PCN construction [28] and Guillou-Quisquater-based PCN construction with the aforementioned tools. We operated 1k times and calculated their average time, which are reported in Table 2.

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 [20], 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 [28], our construction not only remains good functionalities but also performs better in all phases. In summary, our construction offers improvements in computation overhead and is a promising tool to realizing efficient PCNs.

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 realizing efficient PCNs.

References


