# BlindHub: Bitcoin-Compatible Privacy-Preserving Payment Channel Hubs Supporting Variable Amounts

Xianrui Qin
The University of Hong Kong
xrqin@cs.hku.hk

Shimin Pan
The University of Hong Kong
smpan@cs.hku.hk

Arash Mirzaei
Monash University
arash.mirzaei@monash.edu

Zhimei Sui
Monash University
zhimei.sui1@monash.edu

Oğuzhan Ersoy
Radboud University
Delft University of Technology
oguzhan.ersoy@ru.nl

Amin Sakzad
Monash University
amin.sakzad@monash.edu

Muhammed F. Esgin
Monash University and CSIRO's Data61
Muhammed.Esgin@monash.edu

Joseph K. Liu
Monash University
joseph.liu@monash.edu

Jiangshan Yu
Monash University
jiangshan.yu@monash.edu

Tsz Hon Yuen
The University of Hong Kong
thyuen@cs.hku.hk

2023 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-9336-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SP46215.2023.10179427

***Abstract*—Payment Channel Hub (PCH) is a promising solution to the scalability issue of first-generation blockchains or cryptocurrencies such as Bitcoin. It supports off-chain payments between a sender and a receiver through an intermediary (called the tumbler). Relationship anonymity and value privacy are desirable features of privacy-preserving PCHs, which prevent the tumbler from identifying the sender and receiver pairs as well as the payment amounts. To our knowledge, all existing Bitcoin-compatible PCH constructions that guarantee relationship anonymity allow only a (predefined) fixed payment amount. Thus, to achieve payments with different amounts, they would require either multiple PCH systems or running one PCH system multiple times. Neither of these solutions would be deemed practical.**

**In this paper, we propose the first Bitcoin-compatible PCH that achieves relationship anonymity and supports variable amounts for payment. To achieve this, we have several layers of technical constructions, each of which could be of independent interest to the community. First, we propose *BlindChannel*, a novel bi-directional payment channel protocol for privacy-preserving payments, where one of the channel parties is unable to see the channel balances. Then, we further propose *BlindHub*, a three-party (sender, tumbler, receiver) protocol for private conditional payments, where the tumbler pays to the receiver only if the sender pays to the tumbler. The appealing additional feature of BlindHub is that the tumbler cannot link the sender and the receiver while supporting a variable payment amount. To construct BlindHub, we also introduce two new cryptographic primitives as building blocks, namely *Blind Adaptor Signature* (BAS), and *Flexible Blind Conditional Signature* (FBCS). BAS is an adaptor signature protocol built on top of a blind signature scheme. FBCS is a new cryptographic notion enabling us to provide an atomic and privacy-preserving PCH. Lastly, we instantiate both BlindChannel and BlindHub protocols and present implementation results to show their practicality.**

## I. Introduction

Payment Channels (e.g. [2], [14], [42]) are regarded as one of the most widely deployed solutions to the scalability of Bitcoin. A payment channel allows users to deposit a certain amount of coins in a shared address (the *channel*) controlled by both. The corresponding transaction will be stored on-chain. Both parties can then exchange authenticated off-chain transactions to re-distribute the channel funds. Users finally close the channel by publishing the last authenticated transaction on-chain. This splits the channel coins among parties according to the last agreed distribution.

The payment channel model allows payments between only two users. If there are more than two users, each pair of users needs to establish their own payment channel to facilitate the payment, which is a non-scalable approach. To solve this issue, Payment Channel Networks (PCN) (e.g. [19], [42]) enable two users with no direct payment channel to pay each other through the channels of some intermediaries. Nevertheless, PCN payments may require multi-channel paths and intermediaries to actively participate in relaying the payments, which can lead to their failure.

As an alternative, a Payment Channel Hub (PCH) [18], [23], [25], [27], [48] deploys a star topology where users can pay each other via a single intermediary (called the *tumbler*). However, having a single intermediary raises two issues: (i) The tumbler might steal coins from the sender by not forwarding the payment to the receiver, and (ii) The tumbler might link the sender to the receiver. These security and privacy issues can be linked to *atomicity*, *value privacy* and *relationship anonymity* properties [1], [20], [23], [27], [33],

DOI 10.1109/SP46215.2023.00116

[48][1]. Atomicity ensures balance security of honest parties (i.e. sender, receiver, and tumbler), and value privacy and relationship anonymity guarantee that the tumbler cannot know the payment amount and cannot associate the sender and receiver of a payment, respectively.

Another important property for a PCH is *interoperability* which determines the variety of cryptocurrencies supported by the PCH construction. A highly interoperable PCH allows the tumbler to relay payments between users who possess wallets in a wide variety of cryptocurrencies. Among the existing PCHs, [23], [48] provide the highest interoperability by requiring the most basic functionalities from the underlying cryptocurrencies, i.e., digital signature and timelocks, and hence supporting the most varied cryptocurrencies.

## A. Problem Statement

To the best of our knowledge, all the existing Bitcoin-compatible PCH constructions that guarantee relationship anonymity would require the transaction amount to be fixed. Fixing the amount requires either multiple PCH systems or running one PCH system multiple times. For example, assume Alice wants to pay Bob $n$ coins, then it requires either (i) $\lfloor \log_2 n \rfloor$ PCH systems, whose denominations are fixed to $1, 2, 4, \ldots, 2^{\lfloor \log_2 n \rfloor}$ coins, respectively, or (ii) one PCH system to be run for $O(n)$ times. However, for the first approach, it is unknown how to preserve the relationship anonymity and atomicity across multiple PCH systems simultaneously (existing fixed amount PCH systems only guarantee the security of a single PCH system). Running only one PCH system multiple times for one payment will also be very inefficient. This state of affairs in the state-of-the-art leads us to consider the following question:

*Is it possible to construct a Bitcoin-compatible PCH system with value privacy and relationship anonymity that also supports variable payment amounts?*

## B. Our Contributions

In this paper, we construct a new PCH as an affirmative answer to the above question. Specifically,

- We introduce BlindChannel, a new bi-directional payment channel for privacy-preserving payments. In BlindChannel, although both users reach an agreement on each channel update, only one of the users sees the way channel funds are redistributed. The other user only learns the initial and final balances published on-chain. We formalize BlindChannel in the Universal Composability Framework [9] and formally prove its security. We believe BlindChannel does not only serve BlindHub but also can be of independent interests.
- We introduce BlindHub, a three-party (sender ($\mathbb{S}$), tumbler ($\mathbb{T}$), receiver ($\mathbb{R}$)) protocol for private conditional variable-amount payments, where $\mathbb{T}$ only pays to $\mathbb{R}$ if $\mathbb{S}$ pays to $\mathbb{T}$, but $\mathbb{T}$ could not link $\mathbb{S}$ and $\mathbb{R}$. BlindHub protocol only requires digital signatures and timelocks from the underlying blockchain. Combining BlindHub and BlindChannel, we give the first Bitcoin-compatible PCH construction that achieves atomicity, relationship anonymity, and supporting variable amounts simultaneously (see Fig. 7).
- We introduce a new primitive, as a building block of BlindHub, namely *Blind Adaptor Signatures* (BAS). BAS is an adaptor signature protocol built on top of a blind signature scheme. We give a concrete instantiation and the corresponding security proof of BAS.
- We provide an instantiation of BlindHub based on an ECDSA-based BAS and randomizable signatures on randomizable commitments, which can be instantiated by the scheme in [3]. To analyze the security of BlindHub and inspired by Blind Conditional Signatures (BCS) [23], we also introduce a new cryptographic notion that we call *Flexible Blind Conditional Signatures* (FBCS). This enables us to analyze the security of BlindHub. Finally, we provide a concrete instantiation of BlindChannel that is compatible with Bitcoin utilizing garbled circuits-based zero-knowledge proofs. The implementation results show that our protocol is relatively practical, and by leveraging state-of-art proof techniques can be further optimized. The source code for implementation of our protocol can be found in https://github.com/blind-channel/blind-hub.

## C. Related Work

Monero and ZCash, the most famous privacy-preserving cryptocurrencies, provide confidential transactions. To provide relationship anonymity, they use some on-chain cryptographic mechanisms (e.g. ring signature and zero-knowledge proof) that other currencies do not necessarily support. In addition, Monero and Zcash have been a target of attacks that reduce transaction privacy [10], [13], [29], [31], [39], [52], [54], [55]. Towards a different direction, mixing protocols provide relationship anonymity using a centralized tumbler that mixes users' coins. Some of these mixing protocols are on-chain and hence suffer the scalability issue of their underlying blockchain [7], [8], [21], [28], [35], [36], [38], [45]–[47], [49], [50], [56]. In the off-chain protocols, BOLT [25] is built upon Zcash, Perun [18], NOCUST [30] and MixCT [17] can only be deployed on Turing complete blockchains (e.g. Ethereum), TeeChain [32] relies on trusted execution environments (e.g. Intel SGX) and Tumblebit [27] and A$^2$L [48] do not support variable amount payments. In Table I, we present a comparison of the state-of-the-art off-chain mixing services.

The authors of [27] informally introduced the concept of a *synchronization puzzle* enabling relationship anonymity from a corrupt hub point of view in TumbleBit. In addition, TumbleBit relies on hashed time-lock contracts (HTLCs), which suggests it cannot be an interoperable solution. Tairi, et al. [48] then introduced A$^2$L and further gave a formal security notion of synchronization puzzle in the universal

[1]In existing privacy-preserving PCHs [23], [27], [48], since the payment amount is fixed, the privacy goal (referred as *unlinkability* property) does not deal with the value privacy. However, since we do not require fixed payment amount, we modified the existing definitions for PCNs [1], [20], [33] supporting variable values into the PCH model.

TABLE I
STATE-OF-THE-ART IN OFF-CHAIN MIXING SERVICES.

| | Atomicity | Value Privacy | Relationship Anonymity | Interoperability | Amount Flexibility |
|---|---|---|---|---|---|
| BOLT [25] | ● | ● | ● | ○ (Blind signatures, Script modifications) | ● |
| Perun [18] | ● | ○ | ○ | ○ (Ethereum) | ● |
| NOCUST [30] | ● | ○ | ○ | ○ (Ethereum) | ● |
| MixCT [17] | ● | ● | ● | ○ (Ethereum) | ● |
| Teechain [32] | ● | ● | ● | ◑ (Trusted Hardware) | ● |
| Tumblebit [27] | ● | N.A.[b] | ● | ◑ (HTLC-based currencies) | ○ |
| $\text{A}^2\text{L}$[a] [48] | ● | N.A.[b] | ● | ● (Digital signature and timelocks) | ○ |
| $\text{A}^2\text{L}^+$, $\text{A}^2\text{L}^{\text{UC}}$ [23] | ● | N.A.[b] | ● | ● (Digital signature and timelocks) | ○ |
| BlindHub | ● | ● | ● | ● (Digital signature and timelocks) | ● |

[a] The model of $\text{A}^2\text{L}$ is proven to be insecure by [23].
[b] N.A.: not applicable since the amount in these protocols is fixed.

composability (UC) framework. The synchronisation puzzle of [48] is more interoperable and efficient compared to the one in TumbleBit. Very recently, Glaeser et al. [23] pointed out that there is a gap in the security model of $\text{A}^2\text{L}$, and their UC proof is flawed. To amend this situation, they proposed a new notion called blind conditional signatures (BCS) and provided the corresponding game-based security definitions. Besides, they proposed $\text{A}^2\text{L}^+$, a modified version of $\text{A}^2\text{L}$, and proved that $\text{A}^2\text{L}^+$ satisfies the security notions of BCS. However, we observe that the notion of BCS is limited. If a coin-mixing service is built upon BCS, the messages (transactions) shared between $\mathbb{T}$ and $\mathbb{S}/\mathbb{R}$ are required to be the same. This allows $\mathbb{T}$ to access the payment amount on the transaction, which enables $\mathbb{T}$ to link the sender and receiver through the amount when the amount is allowed to be variable. We later introduce FBCS to tackle this issue.

**Comparison with BOLT [25].** BOLT is also an off-chain mixing protocol that achieves privacy-preserving variable-amount payments. On a high level, BlindHub and BOLT both achieve privacy for variable-amount payments by hiding the amount and leveraging blind signatures to validate the channel update. However, there are some differences between BlindHub and BOLT. For example, BOLT relies on the anonymous payment channel (APC) scheme that allows different transactions in the same channel to be unlinkable. This is important to achieve privacy and atomicity simultaneously in BOLT. However, the APC scheme used in BOLT utilizes a blind signature scheme that is not Bitcoin-compatible.[2] We use a different idea to achieve the privacy and atomicity, for which we leverage adaptor signatures, randomizable puzzles, and randomizable signatures. In particular, the adaptor signature enables the atomicity of the payments on both sides of the tumbler. The randomizable puzzle helps to transfer the adaptor witness in an unlinkable way, and the randomizable signature is used to link the adaptor witness and the amount.

## II. SOLUTION OVERVIEW

We first give the system model and security and privacy goals for the PCH construction, then, we provide an overview of our solution. It is noted that the privacy definitions are adopted from [20].

[2]In [37], it is conjectured that BOLT can be modified to be Bitcoin-compatible with the cost of using hash-based commitments and generic circuit-based multi-party computation for blind signing with ECDSA.

**System Model**. BlindHub, as a payment channel hub (PCH) protocol, is composed of a payment hub (referred as Tumbler $\mathbb{T}$) and multiple users, who have established payment channels with the Tumbler $\mathbb{T}$. PCH allows one user (referred as Sender $\mathbb{S}$) to be able to pay another (referred as Receiver $\mathbb{R}$) via $\mathbb{T}$. Users have authenticated communication channels with $\mathbb{T}$, and any two users intending to make payments also have a private communication channel. For the sake of simplicity, we focus on the payment of a single pair of sender and receiver. It is noted that there can be multiple senders paying multiple receivers at the same time.

**Threat Model.** As commonly done in the literature [27], [33], [34], we consider a static attacker, who corrupts parties at the beginning of each epoch. Note that the privacy of sender/receiver could be compromised if the tumbler colludes with receiver/sender. We will discuss this in more details in the Appendix A. Moreover, it is important to state that Blindhub runs over epochs, and our privacy guarantees are valid if the users do not abort during an epoch. More specifically, Blindhub guarantees relationship anonymity (which we will introduce later) when all the payments are successful. However, anonymity can be undermined if the tumbler launches the abort attack. In the case of an abort, the anonymity set of the sender and receiver pairs will be reduced from the epoch set to the uncompleted payments within the epoch. For example, if the tumbler aborts a payment from the sender, the receiver whose payment also fails is regarded as the potential one that is linked to the sender. Note that privacy in the presence of an abort is a common problem in existing schemes that rely on epoch-based anonymity sets, such as $\text{A}^2\text{L}$ [48], $\text{A}^2\text{L}^+/\text{A}^2\text{L}^{\text{UC}}$ [23] and TumbleBit [27]. However, we believe that a rational tumbler would not abort because it would also hurt its reputation. Yet, how to avoid the abort attack in the Bitcoin-compatible PCHs remains an interesting open problem.

### A. Security and Privacy Goals

We now informally define the security and privacy goals of PCH. The formal definitions are given in Appendix C.

**Griefing Resistance.** The PCH should only initiate a payment procedure if $\mathbb{R}$ can prove that the payment request are previously backed by some coins locked by a $\mathbb{S}$ during the payment procedure.

**Atomicity.** For any payment of $\mathsf{m}$ coins from $\mathbb{S}$ to $\mathbb{R}$, the PCH should ensure that either $\mathbb{R}$ receives $\mathsf{m}$ coins from $\mathbb{T}$ and $\mathbb{T}$ receives $\mathsf{m}$ coins from $\mathbb{S}$, or both parties receive none.
**Value Privacy.** $\mathbb{T}$ should not know the payment amount between $\mathbb{S}$ and $\mathbb{R}$.
**Relationship Anonymity.** $\mathbb{T}$ should not be able to find out if there is any relation between $\mathbb{S}$ and $\mathbb{R}$ of a specific payment.

## B. Our Solution

We present a payment channel hub that achieves griefing resistance, atomicity, value privacy and relationship anonymity and supports variable amounts simultaneously. Below we propose our solution in an incremental way. We first give a naive approach to solve the problem, then we discuss the challenges of this naive approach and show how to overcome them. We repeat this process until we reach the final version of our protocol.
Recall that in $A^2L$ [48] or Tumblebit [27], the amount is required to be fixed for achieving the relationship anonymity since, otherwise, $\mathbb{T}$ can easily link the corresponding sender and receiver just by observing which sender-receiver pair shares the same amount. To circumvent the fix-amount limitation, our idea is to hide the amount from $\mathbb{T}$, so that $\mathbb{T}$ can no longer learn the relationship information from the amount. Specifically, we hide the amount by committing to it. However, using this approach for our purpose is far from simple. Recall that in a payment channel, users need to reach an agreement on the channel state when they update the channel. If one user commits and hides the amount, the other user will be prevented from confirming the channel state, which will lead to a failure of channel update.
**BlindChannel.** For the challenge of updating the channel while hiding the amount, we propose a new model for the payment channel, to capture this scenario, called *BlindChannel*, as shown in Fig. 1.

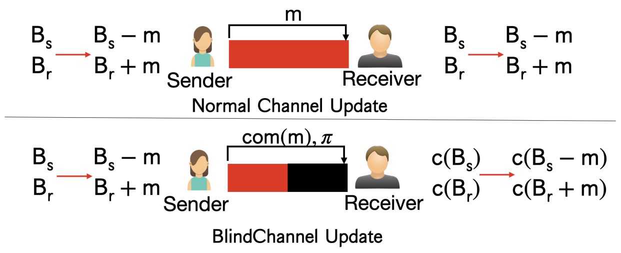

m
$\mathsf{B_s}$ → $\mathsf{B_s - m}$
$\mathsf{B_r}$ → $\mathsf{B_r + m}$
Sender
Receiver
Normal Channel Update
com(m), π
$\mathsf{c(B_s)}$ → $\mathsf{c(B_s - m)}$
$\mathsf{c(B_r)}$ → $\mathsf{c(B_r + m)}$
BlindChannel Update

Fig. 1. Comparison between a normal channel and a BlindChannel. The black colour represents the party that does not know the balance after the update. $\mathsf{c(m)}$ denotes the commitment of $\mathsf{m}$.

To present the idea of BlindChannel, we first recall some background knowledge of payment channels. Suppose there are two users sharing a channel, and their initial balances are $\mathsf{B_s}$ and $\mathsf{B_r}$ coins. Now one user (sender) wants to pay the other user (receiver) $\mathsf{m}$ coins. After the payment, the sender's and receiver's balances become $\mathsf{B_s - m}$ coins and $\mathsf{B_r + m}$ coins, respectively. In such a normal channel setting, both parties are aware of the payment amounts and their updated balances. However, in the BlindChannel setting, only one user can know the channel balances, while the other cannot. We call the former *unblind party* and the latter *blind party*. To reach an agreement between the parties on the payment amount and securely update the channel, we utilize zero knowledge proofs [12], [40]. Roughly speaking, each time they need to update the channel, the unblind party is required to send the blind party the commitments of the payment amount and their updated balances, and prove to the blind party that the payment amount in the BlindChannel equals the one committed in the given commitment. For ease of presentation, we call this proof *amount consistency proof.*
**Value Privacy: A Simple PCH from BlindChannel.** To further explain the idea, we present a simple payment channel hub based on BlindChannel, as shown in Fig. 2

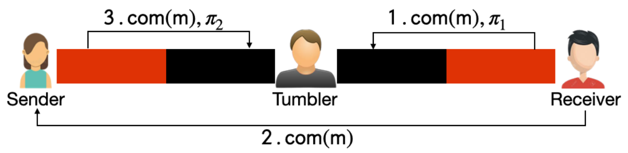


Fig. 2. A simple PCH based on BlindChannel.

In this example, $\mathbb{S}$ and $\mathbb{R}$ try to make a payment via $\mathbb{T}$ without revealing the amount. The order of payment is as follows: first, $\mathbb{T}$ pays $\mathbb{R}$, then, $\mathbb{S}$ pays $\mathbb{T}$. Assume that all parties are honest. Firstly, $\mathbb{R}$ invokes a channel update request with $\mathbb{T}$ and performs the amount consistency proof (step 1). After the success of the channel update, $\mathbb{R}$ sends a commitment of $\mathsf{m}$, $\mathsf{com(m)}$, to $\mathbb{S}$ (step 2). On receiving $\mathsf{com(m)}$, similar to step 1, $\mathbb{S}$ invokes a channel update with $\mathbb{T}$ and performs the amount consistency proof (step 3). This concludes the payment. With this example, we show a private payment between $\mathbb{S}$ and $\mathbb{R}$ without revealing the payment amount to $\mathbb{T}$, assuming the honesty of the parties. This assumption is critical to the above example since otherwise, a malicious $\mathbb{S}$ can just refuse to pay $\mathbb{T}$ after $\mathbb{R}$ is paid by $\mathbb{T}$, hence a loss of $\mathbb{T}$'s money. Namely, the *atomicity* does not hold in the malicious case.
**Atomicity: Linking Puzzle with Transaction Amount.** To ensure atomicity, we first try to use puzzles introduced in $A^2L$ [43], as shown in Fig. 3. Then, we show that this is not secure for the variable transaction amounts. Finally, we give a solution by linking the puzzle with the amount.

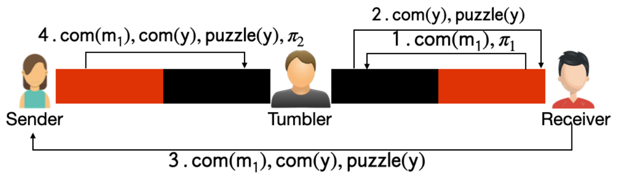

2 . com(y), puzzle(y)
4 . com($\mathsf{m_1}$), com(y), puzzle(y), $\pi_2$
1 . com($\mathsf{m_1}$), $\pi_1$
Sender
Tumbler
Receiver
3 . com($\mathsf{m_1}$), com(y), puzzle(y)

Fig. 3. Adding a puzzle to a PCH based on BlindChannel.

After adding a puzzle to our simple PCH based on BlindChannel, $\mathbb{R}$ first sends the commitment of the amount to $\mathbb{T}$ and gives the amount consistency proof. Afterwards, $\mathbb{T}$ generates a puzzle to which it already knows the solution and shares it with $\mathbb{R}$. Then, instead of directly updating the BlindChannel state with $\mathbb{R}$ to finalize the payment, $\mathbb{T}$ updates the channel with $\mathbb{R}$ to a *conditional* state. Namely, the channel update could only be completed if the condition is satisfied. Now $\mathbb{T}$ sets the condition to be the puzzle being solved. To solve the puzzle, $\mathbb{R}$ needs to send it to $\mathbb{S}$, who will buy the solution of the puzzle from $\mathbb{T}$, and sends it back to $\mathbb{R}$, and finally, $\mathbb{R}$ can claim the same amount of money from $\mathbb{T}$ with this solution.

However, this technique is not enough to guarantee the atomicity when the amount is hidden and allowed to be variable since a malicious sender can use another amount $\mathsf{m}_2$ which is smaller than $\mathsf{m}_1$ to make the payment with $\mathbb{T}$. As a result, $\mathbb{T}$ receives less money than what he sends out. Observe that the cause of this attack is that we do not correlate the amount to the puzzle solution.

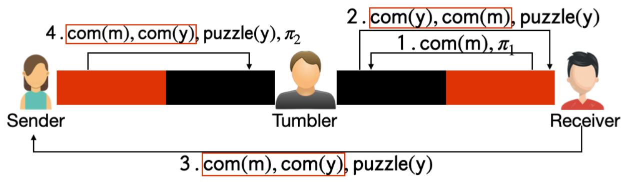

4 . com(m), com(y), puzzle(y), $\pi_2$
2 . com(y), com(m), puzzle(y)
1 . com(m), $\pi_1$
Sender
Tumbler
Receiver
3 . com(m), com(y), puzzle(y)

Fig. 4. Linking the amount with the puzzle.

To address this issue, we should *link* the amount to the puzzle solution. Besides, the *link* should be authenticated by $\mathbb{T}$. We capture the link using a red box and show this in Fig. 4. By linking the commitments, $\mathbb{T}$ is ensured that the aforementioned attack cannot be launched.

**Relationship Anonymity: Randomizable Commitment, Puzzle and Linkage.** So far, we build up a PCH satisfying atomicity, below we focus on how to guarantee the relationship anonymity. Observe that $\mathbb{T}$ can easily know the relationship between $\mathbb{S}$ and $\mathbb{R}$ by observing which pair of sender and receiver share the same commitment of the amount, the puzzle, and the link. To hide their relationship, our approach is to make all the above primitives *randomizable*. Specifically, each time $\mathbb{R}$ sends the required elements to $\mathbb{S}$, he sends randomized ones rather than the original ones. In this manner, we can hide their relationship perfectly.

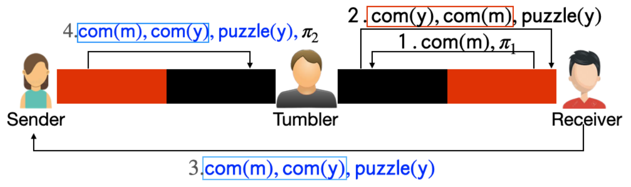

4. com(m), com(y), puzzle(y), $\pi_2$
2 . com(y), com(m), puzzle(y)
1 . com(m), $\pi_1$
Sender
Tumbler
Receiver
3. com(m), com(y), puzzle(y)

Fig. 5. Rerandomize the commitment, the puzzle and the link.

**Griefing Resistance: Linking Token with transaction amount.** The remaining security goal to achieve is griefing resistance. The idea in $\mathrm{A}^2\mathrm{L}$ [48] to achieve griefing resistance is as follows: $\mathbb{S}$ firstly asks for a one-time anonymous credential from $\mathbb{T}$ and forwards it to $\mathbb{R}$, who just shows the credential to $\mathbb{T}$ before initiating the payment. But to apply it to our scenario, we should also "link" the credential to the amount, since otherwise, the aforementioned attack can be carried out similarly.

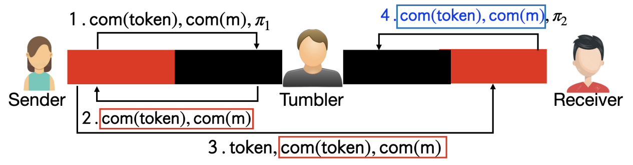

1 . com(token), com(m), $\pi_1$
4 . com(token), com(m), $\pi_2$
Sender
Tumbler
Receiver
2 . com(token), com(m)
3 . token, com(token), com(m)

Fig. 6. Protocol for the registration phase before initiating the payment. Randomized contents are put in a box with different colours.

In more detail, our approach is illustrated in Fig. 6. Firstly, $\mathbb{S}$ generates a token and commits it. Then, $\mathbb{S}$ sends the commitment of the token, the commitment of the amount, as well as an amount consistency proof to $\mathbb{T}$. Secondly, they involve in updating the channel to a conditional state. The condition is set to be a timelock state. Thirdly, if the channel update is successful, $\mathbb{T}$ returns a *linked* commitment of the token and the commitment of the amount to $\mathbb{S}$, who forwards them to $\mathbb{R}$. Then, $\mathbb{R}$ randomizes the "linked" commitment of the token and the commitment of the amount and sends them to $\mathbb{T}$. To enable $\mathbb{T}$ to check the uniqueness of the committed token, $\mathbb{R}$ attaches an additional token-uniqueness proof $\pi_{\mathsf{tup}}$ to prove that the committed token indeed has not been used before. Finally, $\mathbb{T}$ checks the validity of the "link" and the uniqueness of the token. If the check passes, $\mathbb{R}$ and $\mathbb{T}$ starts the payment.

**Our solution Overview.** After introducing our ideas in several steps, we wrap them up now and give an overview of our PCH protocol, as shown in Fig. 7. In our protocol, we instantiate the *link* as randomizable signatures on randomizable commitments (RSoRC), which will be introduced in Section III. Besides, we instantiate the randomizable puzzle as linear-only encryption, which is similar to the approach adopted in $\mathrm{A}^2\mathrm{L}^+$ [23]. Also inspired by $\mathrm{A}^2\mathrm{L}$ and $\mathrm{A}^2\mathrm{L}^+$ [23], [48], we leverage adaptor signatures to realize conditional channel update. Detailed descriptions of BlindChannel and BlindHub are provided in Section V and VI, respectively.

## III. PRELIMINARIES

We denote by $1^\lambda$, for $\lambda \in \mathbb{N}$, the security parameter. We assume that the security parameter is given as an implicit input to every function, and all our algorithms run in polynomial time in $\lambda$. We denote by $x \leftarrow \$\mathcal{X}$ the uniform sampling of the variable $x$ from the set $\mathcal{X}$. We write $x \leftarrow \mathsf{A}(y)$ to denote that a probabilistic polynomial time (PPT) algorithm $\mathsf{A}$ on input $y$, outputs $x$. We use the same notation also for the assignment of the computational results, for example, $s \leftarrow s_1 + s_2$. If $\mathsf{A}$ is a deterministic polynomial time (DPT) algorithm, we use the notation $x := \mathsf{A}(y)$. We use the same notation for expanding the entries of tuples, for example, we write $\sigma := (\sigma_1, \sigma_2)$ for a tuple $\sigma$ composed of two elements. We say a function $\mathsf{negl}$ is negligible in $\lambda$ if it vanishes faster than any polynomial with input $\lambda$.

**Commitment scheme.** Denote message space, randomness

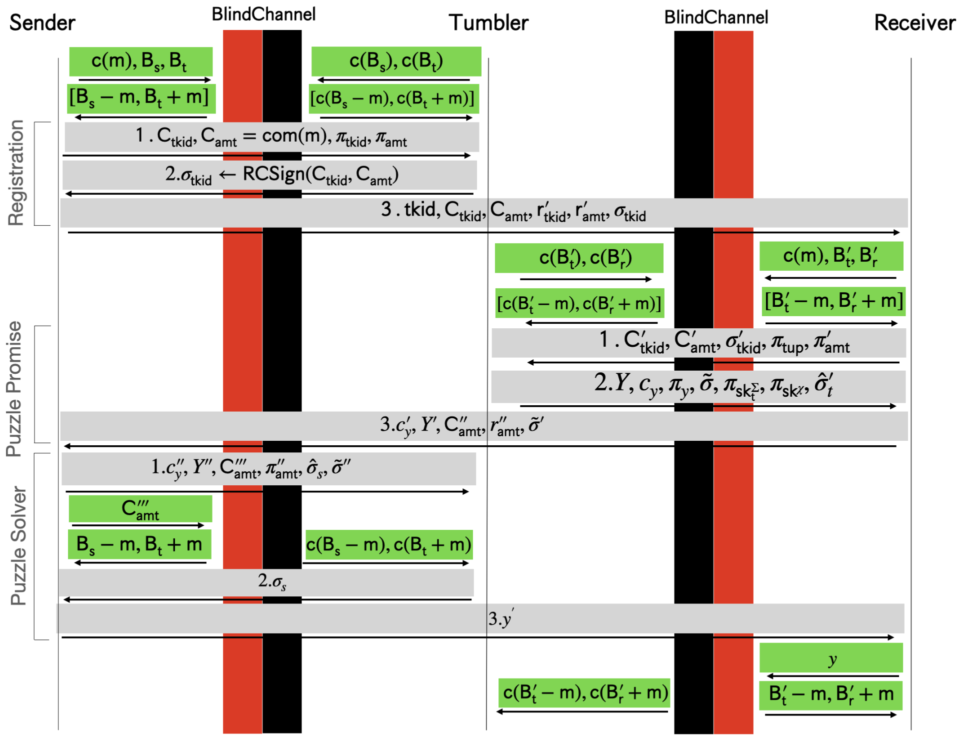


Fig. 7. Overview of our solution. Sender pays receiver via tumbler. The overview can be divided into two layers: one layer is the BlindHub protocol, which corresponds to the interactions in the gray box. BlindHub protocol can be divided into three phases: 1) registration, 2) puzzle promise, and 3) puzzle solver. For the ease of presentation, we present the transcripts transferred among the paries and ignore the process of generating them. The definitions of all the notations written in the gray box can be found in BlindHub protocol given in Section VI. Another layer is the BlindChannel protocol, which corresponds to the interactions with the BlindChannel (the long red/black rectangle) in the green box. For the sake of simplicity, we just present the simplified input and output of BlindChannel. $\mathsf{c}(\cdot)$ denotes commitment. $\mathsf{B_s}, \mathsf{B_t}$ denotes the balances of sender and tumbler in the channel shared with sender. $\mathsf{B}'_\mathsf{t}, \mathsf{B}'_\mathsf{r}$ denotes the balances of tumbler and receiver in the channel shared with receiver. The balance in the square bracket $[\cdot]$ represents that the balance is still in conditional state while balance that is not in $[\cdot]$ represents that the balance is already in the normal state.

space and commitment space as $\mathcal{M}, \mathcal{R}$, and $\mathcal{CM}$, respectively. A commitment scheme $\Pi_{\mathsf{COM}}$ consists of the following algorithms: on input $m \in \mathcal{M}, r \in \mathcal{R}$, $(r, \mathsf{C}) \leftarrow \Pi_{\mathsf{COM}}.\mathsf{com}(m, r)$, and should satisfy hiding and biding properties. Hiding states that given the commitment, one cannot determine the values. Biding requires that one cannot change the value after they have committed to it.

**Non-interactive zero-knowledge.** Let $R$ be an NP relation and $L$ be defined as the set $L := \{x \mid \exists w,\ s.t.\ R(x, w) = 1\}$. We say $R$ is a *hard relation* if: 1) there is a PPT sampling algorithm $\mathsf{GenR}$ that on input $1^\lambda$ and output a statement/witness pair $(Y, y) \in R$. 2) $R$ is poly-time decidable. 3) for all PPT $\mathcal{A}$, $\mathcal{A}$ on input $Y$ outputs a valid witness $y$ with negligible probability. A non-interactive zero-knowledge proof scheme $\Pi_{\mathsf{NIZK}}$ consists of two PPT algorithms: $\mathsf{P_{NIZK}}(w, x)$: The prover algorithm that on input a witness $w$ and its statement $x$, outputs a proof $\pi$. $\mathsf{V_{NIZK}}(x, \pi)$: The verification algorithm that on input the statement $x$ and the proof $\pi$, outputs a bit $b \in \{0, 1\}$. The prover can provide a verifier with $\pi$ to convince her of the prover's knowledge of the witness $w$ for the statement $x$ without disclosing any further information.

**Linear-Only Homomorphic Encryption.** A public key encryption scheme $\Pi_{\mathsf{Enc}}$ with a message space $\mathbb{M}$ and ciphertext space $\mathbb{C}$ consists of the following algorithms: $\mathsf{KGen}(\lambda)$: On input the security parameter $\lambda$, outputs a key pair $(\mathsf{ek}, \mathsf{dk})$. $\mathsf{Enc}(\mathsf{ek}, m)$: on input the public key $\mathsf{ek}$ and a message $m \in \mathbb{M}$, outputs a ciphertext $c \in \mathbb{C}$. $\mathsf{Dec}(\mathsf{dk}, c)$: A deterministic algorithm that on input the private key $\mathsf{dk}$ and the ciphertext $c \in \mathbb{C}$, outputs a message $m \in \mathbb{M}$. Correctness of a public key encryption scheme $\Pi_{\mathsf{Enc}}$ guarantees that for every message $m \in \mathbb{M}$ and every key pair $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{KGen}(\lambda)$, $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{ek}, m)) = m$ holds. The encryption scheme that is used in this work is required to provide CPA-security [24]. Also, $\Pi_{\mathsf{Enc}}$ is called additively homomorphic if for every $m_1, m_2 \in \mathbb{M}$ and every public key $\mathsf{ek}$ generated by $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KGen}(\lambda)$, it holds that $\mathsf{Enc}(\mathsf{ek}, m_1) \cdot \mathsf{Enc}(\mathsf{ek}, m_2) = \mathsf{Enc}(\mathsf{ek}, m_1 + m_2)$. Linear-Only encryption (LOE) [26] models homomorphic encryption by oracle queries rather than concrete algorithms. The formal description of the oracles modelled by homomorphic encryption can be found in the full verion [44] (Fig. 16 in Appendix E.4) . By homomorphically adding 0 to the desired ciphertext, this paradigm enables (perfect) re-randomization of the ciphertext.

**Digital signature scheme.** A digital signature scheme $\Sigma$ usually contains three algorithms: $\mathsf{KeyGen}(1^\lambda)$: inputs a security parameter $1^\lambda$ and outputs a secret key/public key pair $(\mathsf{sk}, \mathsf{pk})$. $\mathsf{Sign_{sk}}(m)$: input a secret key $\mathsf{sk}$ and message $m \in \{0, 1\}^*$ and outputs a signature $\sigma$. $\mathsf{Vf_{pk}}(m, \sigma)$: on the input of a public key $\mathsf{pk}$, a signature $\sigma$ and a message $m$, outputs a bit $b$ indicating whether a signature is valid $(b = 1)$ or not $(b = 0)$. The correctness property holds as long as for any key pair $(\mathsf{sk}, \mathsf{pk})$ created by the key generation function and any message $m$, if a signature $\sigma$ is produced using the signing algorithm with input $(\mathsf{sk}, m)$, the verification algorithm output 1 on input $(\mathsf{pk}, \sigma, m)$.

**Adaptor signature scheme.** An adaptor signature scheme is defined upon a hard relation $R$ and a signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vf})$ and it consists of four algorithms $\Pi_{R,\Sigma} = (\mathsf{PreSign}, \mathsf{Adapt}, \mathsf{PreVf}, \mathsf{Ext})$ with the following syntax: for each statement/witness pair $(Y, y) \in R$, $\mathsf{PreSign}(\mathsf{sk}, m, Y)$ is a PPT algorithm that outputs a pre-signature $\hat{\sigma}$ and $\sigma := \mathsf{Adapt}(\hat{\sigma}, y)$ is a valid signature. $\mathsf{PreVf}(\mathsf{pk}, m, Y, \hat{\sigma})$ is a DPT algorithm that outputs a bit $b$. Finally, $\mathsf{Ext}(\sigma, \hat{\sigma}, Y)$ is a DPT algorithm that outputs witness $y$, s.t., $(Y, y) \in R$. An adaptor signature achieves *pre-signature correctness* if the pre-signature w.r.t. a statement $Y$ is valid and it can be adapted into a full signature, from which we can extract the witness $y$. Adaptor signature achieves *unforgeability* if producing a forgery for some message $m$ is hard even given a pre-signature on $m$ w.r.t. a random statement $Y \in L_R$. Adaptor signature achieves *pre-signature adaptability* if for any valid pre-signature w.r.t. $Y$ can be completed into a valid signature using a witness $y$ with $(Y, y) \in R$. A more detailed description can be found in the full version [44] (Appendix E.1).

**Randomizable signatures on randomizable commitments scheme.** Randomizable commitment allows anyone to transform a commitment into a fresh one of the same message. In this paper, we need a signature scheme that satisfies the following requirements: 1) it enables the issuance of signatures on randomizable commitments and 2) anyone, knowing neither the signing key nor the committed message, can randomize the commitments, and compute a new signature on these randomized commitments. Specifically, we call a signature scheme randomizable on randomizable commitments (RSoRC) if it provides the following algorithms: RCSign and RCRand in addition to the ones given by a standard digital signature scheme. Given a set of randomizable commitments $\mathsf{CM}_1, \ldots, \mathsf{CM}_n$, a signer can generate a randomizable signature $\tilde{\sigma} \leftarrow \mathsf{RCSign}(\mathsf{CM}_1, \ldots, \mathsf{CM}_n)$. Given a randomizable signature $\tilde{\sigma}$ , the corresponding commitments $\mathsf{CM}_1, \ldots, \mathsf{CM}_n$ and a randomness $r$, anyone can generate a new valid randomizable signature and a set of randomized commitments $(\mathsf{CM}'_1, \ldots, \mathsf{CM}'_n, \tilde{\sigma}') \leftarrow \mathsf{RCRand}(\tilde{\sigma}, \mathsf{CM}_1, \ldots, \mathsf{CM}_n, r)$. The signature that fulfils these requirements, and which we use in our construction, can be instantiated by the signature on randomizable ciphertexts scheme [3]. Though in the scheme of [3] the message space is defined upon ciphertexts, actually, it can also be defined upon commitments which can be presented as group elements on the elliptic curves where discrete logarithm problem is hard. We give the formal definition, security properties and concrete construction of the primitive in the full version [44] (Appendix D).

**One-more Discrete logarithm (OMDL) problem.** OMDL problem [5] says that one cannot solve $q + 1$ challenge group elements given only $q$ DL solving oracles. A more detailed definition of OMDL can be found in the full version [44] (Appendix E.2.2).

## IV. Blind Adaptor Signature

In this section, we introduce a new primitive called Blind Adaptor Signature (BAS), which is an important building block to construct BlindHub. Before introducing BAS, we first briefly recall what a blind signature is. In a blind signature scheme, a user can obtain a signature from a signer on a message $m$ such that: (1) the signer cannot recognize the signature later (blindness, which implies that the message $m$ is unknown to the signer) and (2) the user can compute only one single signature per interaction with the signer (one-more unforgeability). A more detailed definition of a blind signature can be found in the full version [44] (Appendix E.2).

### A. Blind Adaptor Signature

Combining a blind signature and an adaptor Signature, we give a new primitive called *Blind Adaptor Signature*. We first give the formal definition as follows.

**Definition 1** (Blind Adaptor Signature Scheme)**.** *A blind adaptor signature (*BAS*) scheme* $\Pi_{\mathsf{BAS}}$ *with respect to a hard relation* $R$ *with a language* $L_R := \{Y | \exists y : (Y, y) \in R\}$ *consists of the following algorithms:*

- $\mathsf{BAS.Setup}(1^\lambda)$*: It takes the security parameter* $1^\lambda$ *and returns public parameters* param.
- $\mathsf{BAS.KeyGen}(\mathsf{param})$*: It takes the public parameters* param *and returns a secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$.
- $(b, \hat{\sigma}) \leftarrow \langle \mathsf{BAS.PreSign}(\mathsf{sk}, Y), \mathsf{BAS.User}(\mathsf{pk}, m, Y) \rangle$ *an interactive protocol is run between the signer with private input a secret key* sk *and the user with signer's public key* pk *and a message* $m$ *as inputs. A statement* $Y \in L_R$ *is the public input. The signer outputs* $b = 1$ *if the interaction completes successfully and* $b = 0$ *otherwise, while the user outputs a pre-signature* $\hat{\sigma}$ *if interaction completes correctly, and* $\perp$ *otherwise.*
- $\mathsf{BAS.PreVerify}, \mathsf{BAS.Adapt}$ *and* $\mathsf{BAS.Ext}$ *are the same as* $\mathsf{PreVerify}, \mathsf{Adapt}$ *and* $\mathsf{Ext}$ *of the adaptor signature.*

For a 1-round (i.e., two messages) protocol, the interaction can be realized by the following algorithms: $(\mathsf{msg}_{\mathsf{U},0}) \leftarrow \mathsf{BAS.User}_0(\mathsf{pk}, m), (\mathsf{msg}_{\mathsf{S},1}, b) \leftarrow \mathsf{BAS.Sign}_1(\mathsf{sk}, \mathsf{msg}_{\mathsf{U},0})$, $\sigma \leftarrow \mathsf{BAS.User}_1(\mathsf{msg}_{\mathsf{S},1})$.

Below we give informal security definitions of BAS. We present the formal definitions in the Appendix B .

**One-more Unforgeability.** The unforgeability model is defined to capture the attack that the adversary returns $n$ distinct message-signature pairs when he is only given $k_2 < n$ pairs during the oracle queries. It is commonly known as the one-more unforgeability in blind signature [22].

**Blindness.** In many scenarios, blind signatures should satisfy the following blindness property: a signer cannot link a message/signature pair to a particular execution of the signing protocol. But to realize the BlindChannel (as formally defined in the full version [44] (Appendix J.2)),we only need a weak blindness: given the transcript, the signer could not figure out what the message is. Compared to the normal blindness, weak blindness allows the signer to link the message/signature pair to a particular execution, as long as the signer can obtain the message/signature pair. **Pre-signature Adaptability.** The pre-signature adaptability of $\Pi_{\mathsf{BAS}}$ is the same as that of an adaptor signature. It is because the $\mathsf{PreSign}$ algorithm is not involved in the model.

**Witness extractability.** The witness extractability guarantees that given a valid signature/pre-signature pair w.r.t. a message/statement pair $(m, Y)$ one can extract the corresponding witness $y$ of $Y$.

## V. Description of BlindChannel

This section first provides some security and privacy properties required by a payment channel in the BlindHub protocol. Then, we present an overview as well as the protocol description of BlindChannel, which was briefly mentioned in Section II. In the full version [44] (Appendix J.2) we will prove that BlindChannel is a secure realization of an ideal functionality that achieves the security and privacy properties stated in this section.

### A. Security and Privacy Properties

Below we give informal definitions of security and privacy properties required by BlindChannel. The full version of this paper [44] (Appendix J.2) provides more details.

The BlindChannel scheme $\Pi_{\mathsf{BC}}$ is secure if the followings hold: 1) the blind party could not figure out the way channel funds are redistributed. 2) A BlindChannel is successfully created/updated only if both parties in the channel agree with the creation/update. 3) An honest party $P$ in the channel has the guarantee that either the current state of the channel can be enforced on the ledger, or $P$ can enforce a state where she gets all coins in the channel.

### B. BlindChannel Overview

Similar to other payment channels, BlindChannel allows two parties to pay to each other arbitrarily many times without publishing every single transaction on the blockchain. However, in BlindChannel protocol, only one of the parties, i.e. the unblind party, determines the payment amount and the other party, i.e. the blind party, cannot see the payment amount. Also, as required by BlindHub protocol, all payments are conditioned on solving a puzzle, introduced in [34]. We start by reviewing the generalized channels [2], and then gradually introduce our solution.

**Generalized Channel.** To create a generalized channel [2], Alice (denoted by $A$) and Bob (denoted by $B$) publish a funding transaction $\mathtt{TX}_{\mathtt{FU}}$ to respectively send $a$ and $b$ coins into a shared address. Both parties also hold the same copy of two transactions, by broadcasting which they can close the channel: 1) The commit transaction $\mathtt{TX}_{\mathtt{CM}}$ that sends the channel funds, held in the funding transaction's output, into a new shared address and 2) The split transaction $\mathtt{TX}_{\mathtt{SP}}$ that splits the channel funds, held in the commit transaction's output, among parties. So, the split transaction has two outputs holding $a$ and $b$ coins owned by $A$ and $B$, respectively.

Now assume $A$ decides to pay $0 < v \leq a$ coins to $B$. To do so, $A$ and $B$ create a new commit and split transactions where the split transaction contains two outputs holding $a - v$ and $b + v$ coins owned by $A$ and $B$, respectively. Since one of the parties may submit a stale state to the blockchain, channel parties need a way to detect and penalize such frauds. So, after each channel update, channel parties exchange revocation secrets that allow the honest party to send all the funds in the stale commit transaction's output to his own address. But we still need a way to guarantee that the malicious party cannot publish the stale commit transaction and spend its output using her counter-party's revocation secret. In the generalized channel, the adaptor signature is leveraged to guarantee that once a party, e.g. $A$, publishes a commit transaction, a secret, called the publishing secret, is revealed to $B$. Thus, once $A$ publishes a stale commit transaction, the honest party $B$ can use $A$'s revocation secret and $A$'s publishing secret to take all the channel funds. Also, to guarantee that the malicious party cannot publish both a stale commit transaction and its corresponding split transaction, the split transaction cannot be published within $T$ rounds since the commit transaction is published on the blockchain. Therefore, the commit transaction has one output that can be spent by: 1) $A$ if she knows $B$'s revocation and publishing secrets, 2) split transaction after $T$ rounds, or 3) $B$ if he knows $A$'s revocation and publishing secrets.

**Adding Privacy to the Channel.** Now assume two parties $U$ and $B$ create a channel like a generalized channel. However, we want the channel update in this channel to be different from the channel update in a generalized channel. Particularly, $B$ in this channel is a blind party, i.e. he is not supposed to see the payment amount. Since the commit transaction contains no data about the payment amount (i.e. $v$ in the previously stated scenario), the two sides can exchange their signatures on the commit transaction like a generalized channel. But since outputs of the split transaction reveal data about the payment amount, $B$ should blindly sign it. However, before signing the split transaction, the unblind party $U$, using the zero-knowledge proofs and without revealing the value of each output in the split transaction, performs amount consistency proof (as informally defined in Section II) and also proves that the transaction that $B$ will blindly sign contains the correct elements, i.e., the correct input and outputs (details of zero knowledge proof used in BlindChannel are provided in the full version [44] (Appendix G)).

By publishing the latest commit and split transaction, $U$ can close the channel. However, since $B$ does not hold the split transaction, it is possible that after publishing the commit transaction, $U$ becomes unresponsive to lock $B$'s funds in the channel and raise a hostage situation. So a new sub-condition is added to the commit transaction's output that allows $B$ to claim the output after $2T$ rounds. Thus, once the commit transaction is published by either of two parties, $U$ has to publish its corresponding split transaction within $2T$ rounds. Otherwise, $B$ would get all channel funds.

**Adding Conditional Payment to the Channel.** In the Blind-

Hub protocol, which we will present later, a payment from $\mathbb{T}$ to $\mathbb{R}$ is performed provided the corresponding payment from $\mathbb{S}$ to $\mathbb{T}$ completes. Correspondingly, BlindChannel parties need to perform conditional payments. Let us provide a high-level overview of the required modifications. Assume that $U$ and $B$ have $a$ and $b$ coins in the channel, respectively, and $U$ wants to conditionally pay $v$ coins to $B$ (for $v < a$). So, $B$ and $U$ create a new commit and split transaction where the split transaction has three outputs: 1) the first output holding $a - v$ coins owned by $U$, 2) the second output holding $b$ coins owned by $B$, and 3) the third output for conditional payment of $v$ coins to the party $B$ where party $B$ has a pre-signature from party $U$ on a transaction called the *adaptor execution delivery* (or briefly *delivery*) transaction $\texttt{TX}_{\texttt{AED}}$ that spends this output and sends its coins to $B$. So if $B$ has the corresponding secret to adapt the pre-signature to a valid signature, he can claim the third output of the split transaction. Otherwise, party $U$, who has $B$'s signature on another transaction called the *timeout* transaction $\texttt{TX}_{\texttt{TO}}$, can claim the output after a timeout.

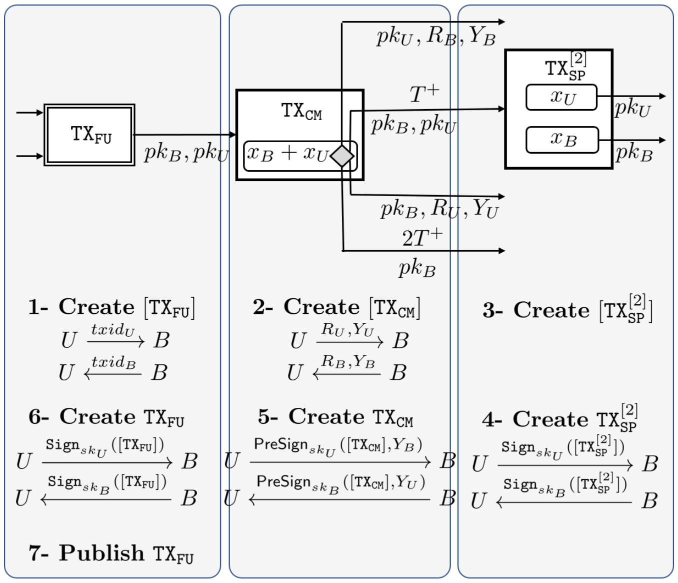
Although $B$ receives $U$'s pre-signature on the delivery transaction, the transaction body itself cannot be given to $B$, as it reveals the payment amount; Only the hash of the transaction body, which is used to create and verify the pre-signature, is given to $B$. However, $B$ should not be able to guess the payment amount by exhaustively searching all possible payment amounts to find the body of the delivery transaction for which the hash value matches. This requirement is satisfied if the delivery transaction's input or equivalently transaction identifier of the split transaction is difficult to guess. To achieve this requirement, $U$ keeps the address that she is using in the first output of the split transaction private. Then, finding the transaction identifier of the split transaction and hence the body of the delivery transaction would be infeasible to $B$. Nevertheless, $U$ uses zero-knowledge proofs to prove that once the split transaction is published on the blockchain, $B$ will learn the currently unknown elements of the delivery transaction, i.e., its input transaction identifier as well as the payment amount. Moreover, since the timeout transaction reveals data about the payment amount, $B$ should blindly sign it. However, before signing the timeout transaction, the unblind party $U$, using the zero-knowledge proofs and without revealing the payment amount and the split transaction identifier, proves the transaction that $B$ will blindly sign is well-structured.

For the case where $U$ is the payee of the payment, everything is the same, but the delivery transaction is signed by the payer using the BAS scheme, introduced in Section IV. Also, $U$'s signature on the timeout transaction is given to the payer without letting him guess the body of the timeout transaction itself.

### C. BlindChannel Protocol Description

The BlindChannel lifetime can be divided into 4 phases, including "create", "update", "close", and "punish". We introduce these phases through the following sub-sections.

**Create.** A blind channel between two parties $B$ and $U$ is created like a generalized channel [2], i.e. through publishing a funding transaction $\texttt{TX}_{\texttt{FU}}$, parties send their coins into a joint account. However, to prevent parties from locking each other's coins into this joint account and raising a hostage situation, they must commit to the initial channel state in advance. So, before signing and publishing the funding transaction, parties create two transactions: 1) a commit transaction $\texttt{TX}_{\texttt{CM}}$ that sends the channel funds into a new joint address and 2) a split transaction $\texttt{TX}_{\texttt{SP}}^{[2]}$ that distributes the channel funds among parties. Fig. 8 summarizes the channel creation phase. Following [2], we use charts to show transaction flows. Each transaction is denoted by a rectangle containing a box for each output. The output value is written inside the box, and the output condition is written above (used for timelocks) and below (used for public keys) the arrow coming out of the output. Outputs with multiple subconditions are denoted by a diamond inside the output box with an arrow corresponding with each subcondition. If a transaction contains a non-zero timelock, it is written inside the transaction rectangle. Let us explain in more detail the different steps of the channel creation phase.



$\texttt{TX}_{\texttt{FU}}$
$pk_B, pk_U$
$\texttt{TX}_{\texttt{CM}}$
$x_B + x_U$
$pk_U, R_B, Y_B$
$T^+$
$pk_B, pk_U$
$pk_B, R_U, Y_U$
$2T^+$
$pk_B$
$\texttt{TX}_{\texttt{SP}}^{[2]}$
$x_U$
$x_B$
$pk_U$
$pk_B$
1- Create $[\texttt{TX}_{\texttt{FU}}]$
$U \xrightarrow{txid_U} B$
$U \xleftarrow{txid_B} B$
2- Create $[\texttt{TX}_{\texttt{CM}}]$
$U \xrightarrow{R_U, Y_U} B$
$U \xleftarrow{R_B, Y_B} B$
3- Create $[\texttt{TX}_{\texttt{SP}}^{[2]}]$
6- Create $\texttt{TX}_{\texttt{FU}}$
$U \xrightarrow{\mathsf{Sign}_{sk_U}([\texttt{TX}_{\texttt{FU}}])} B$
$U \xleftarrow{\mathsf{Sign}_{sk_B}([\texttt{TX}_{\texttt{FU}}])} B$
5- Create $\texttt{TX}_{\texttt{CM}}$
$U \xrightarrow{\mathsf{PreSign}_{sk_U}([\texttt{TX}_{\texttt{CM}}], Y_B)} B$
$U \xleftarrow{\mathsf{PreSign}_{sk_B}([\texttt{TX}_{\texttt{CM}}], Y_U)} B$
4- Create $\texttt{TX}_{\texttt{SP}}^{[2]}$
$U \xrightarrow{\mathsf{Sign}_{sk_U}([\texttt{TX}_{\texttt{SP}}^{[2]}])} B$
$U \xleftarrow{\mathsf{Sign}_{sk_B}([\texttt{TX}_{\texttt{SP}}^{[2]}])} B$
7- Publish $\texttt{TX}_{\texttt{FU}}$

Fig. 8. Creation of a Blind Channel

1) Create $[\texttt{TX}_{\texttt{FU}}]$: At the first step, $B$ and $U$ create the body of the funding transaction $[\texttt{TX}_{\texttt{FU}}]$. To do so, they send each other their funding sources.
2) Create $[\texttt{TX}_{\texttt{CM}}]$: Each party $P \in \{B, U\}$ generates a revocation public/private pair $(R_P, r_P) \leftarrow \mathsf{GenR}$ and a publishing public/secret pair $(Y_P, y_P) \leftarrow \mathsf{GenR}$, and sends the public values $R_P$ and $Y_P$ to the other party. Using the transaction identifier of $\texttt{TX}_{\texttt{FU}}$ and each other's public values, parties create the body of the commit transaction, i.e. $[\texttt{TX}_{\texttt{CM}}]$.
3) Create $[\texttt{TX}_{\texttt{SP}}^{[2]}]$: Using the transaction identifier of $\texttt{TX}_{\texttt{CM}}$, parties create the body of the split transaction $[\texttt{TX}_{\texttt{SP}}^{[2]}]$.

4-6) Create $\texttt{TX}_{\texttt{SP}}^{[2]}/\texttt{TX}_{\texttt{CM}}/\texttt{TX}_{\texttt{FU}}$: Parties exchange the required

signatures to transform $[\texttt{TX}_{\texttt{SP}}^{[2]}]/[\texttt{TX}_{\texttt{CM}}]/[\texttt{TX}_{\texttt{FU}}]$ into $\texttt{TX}_{\texttt{SP}}^{[2]}/\texttt{TX}_{\texttt{CM}}/\texttt{TX}_{\texttt{FU}}$.
7) Publish $\texttt{TX}_{\texttt{FU}}$: Parties publish $\texttt{TX}_{\texttt{FU}}$ on the blockchain.

**Update.** Channel update is performed by adding a third output for a conditional payment to the split transaction where the payer in the channel between $\mathbb{S}$ and $\mathbb{T}$ (resp. the channel between $\mathbb{T}$ and $\mathbb{R}$) is $\mathbb{S}$ (resp. $\mathbb{T}$). This third output can be claimed by the payee if having the value of a secret, the payee can adapt a pre-signature on the corresponding delivery transaction to a full signature. Otherwise, after a specific timeout, the payer publishes the corresponding timeout transaction and gets refunded. Fig. 9 Summarizes the channel update phase when the payer is the unblind party. The channel update phase in more details is as follows.

1) Create $[\texttt{TX}_{\texttt{CM}}]$: The same as step 2 in the create phase.
2) Create $[\texttt{TX}_{\texttt{SP}}^{[3]}]$: Having the transaction identifier of $\texttt{TX}_{\texttt{CM}}$, $U$ firstly creates the body of the split transaction $[\texttt{TX}_{\texttt{SP}}^{[3]}]$, commits it into $\mathsf{com_s}$, and generates a zero knowledge proof $\pi_{\mathsf{s}}$ to perform amount consistency proof and also prove that $\mathsf{com_s}$ is the commitment on the well-structured split transaction.
3) Create $[\texttt{TX}_{\texttt{AED}}]$: Having the transaction identifier of $\texttt{TX}_{\texttt{SP}}^{[3]}$, $U$ generates the body of the adaptor execution delivery (AED) transaction $[\texttt{TX}_{\texttt{AED}}]$, uses the adaptor statement from the external application (e.g., obtained in the puzzle promise phase of $\Pi_{\mathsf{BH}}$) to generate a pre-signature $\tilde{\sigma}_a^U$ on $[\texttt{TX}_{\texttt{AED}}]$, and also generates a zero knowledge proof $\pi_{\mathsf{a}}$ to prove to $B$ that the pre-signature is on the well-structured delivery transaction $[\texttt{TX}_{\texttt{AED}}]$. Party $U$ sends the pre-signature $\tilde{\sigma}_a^U$, the hash value $\mathsf{SigHash}([\texttt{TX}_{\texttt{AED}}])$ and the proof to $B$.
4) Create $[\texttt{TX}_{\texttt{TO}}]$: Having the transaction identifier of $\texttt{TX}_{\texttt{SP}}^{[3]}$, $U$ generates the body of the timeout transaction $[\texttt{TX}_{\texttt{TO}}]$.
5) Create $\texttt{TX}_{\texttt{TO}}$: $U$ commits $[\texttt{TX}_{\texttt{TO}}]$ into $\mathsf{com_t}$, and generates a zero knowledge proof $\pi_{\mathsf{t}}$ to prove to $B$ that the committed message is the well-structured timeout transaction $[\texttt{TX}_{\texttt{TO}}]$. After verifying $\pi_{\mathsf{t}}$, $B$ generates a blind signature $\sigma_{\mathsf{t}}^B$ on $\texttt{TX}_{\texttt{TO}}$ for $U$.
6) Create $\texttt{TX}_{\texttt{SP}}^{[3]}$: $B$ generates a blind adaptor signature with $Y_P$ as the adaptor statement on $\texttt{TX}_{\texttt{SP}}$ for $U$.
7) Create $\texttt{TX}_{\texttt{CM}}$: the same as step 5 in the create phase.
8) Revoke: Both parties revoke the previous state by exchanging the corresponding revocation keys.
9) Create $\texttt{TX}_{\texttt{AED}}$: $B$ adapts the pre-signature $\tilde{\sigma}_a^U$ into $\sigma_a^U$ and sends the corresponding witness $y_s$ to $U$.
10) Create $[\texttt{TX}'_{\texttt{CM}}]$: the same as step 2 in the create phase.
11-12) Create $[\texttt{TX}_{\texttt{SP}}^{[2]}]/\texttt{TX}_{\texttt{SP}}^{[2]}$: Having the transaction identifier of $\texttt{TX}_{\texttt{CM}}$, $U$ firstly creates the body of the split transaction $[\texttt{TX}_{\texttt{SP}}^{[2]}]$, commits it into $\mathsf{com_s}$, and generates a zero knowledge proof $\pi_{\mathsf{s}}$ to perform amount consistency proof and also prove that $\mathsf{com_s}$ is the commitment on the well-structured $\texttt{TX}_{\texttt{SP}}$. Then, $B$ sends the blind signature on $\texttt{TX}_{\texttt{SP}}^{[2]}$ to $U$.
13 Create $\texttt{TX}'_{\texttt{CM}}$: the same as step 5 in the create phase.
14 Revoke: Both parties revoke the previous state by exchanging the corresponding revocation keys.

The case where the unblind party is the payee of payment is similar to the above scenario. The main differences are that $B$ uses the BAS scheme to blindly create a pre-signature on delivery transaction for $U$. Moreover, $U$ sends her signature on the timeout transaction to $B$ without sending him the body of the transaction. We refer the corresponding diagram to the full version [44] (Fig. 22, Appendix I) .

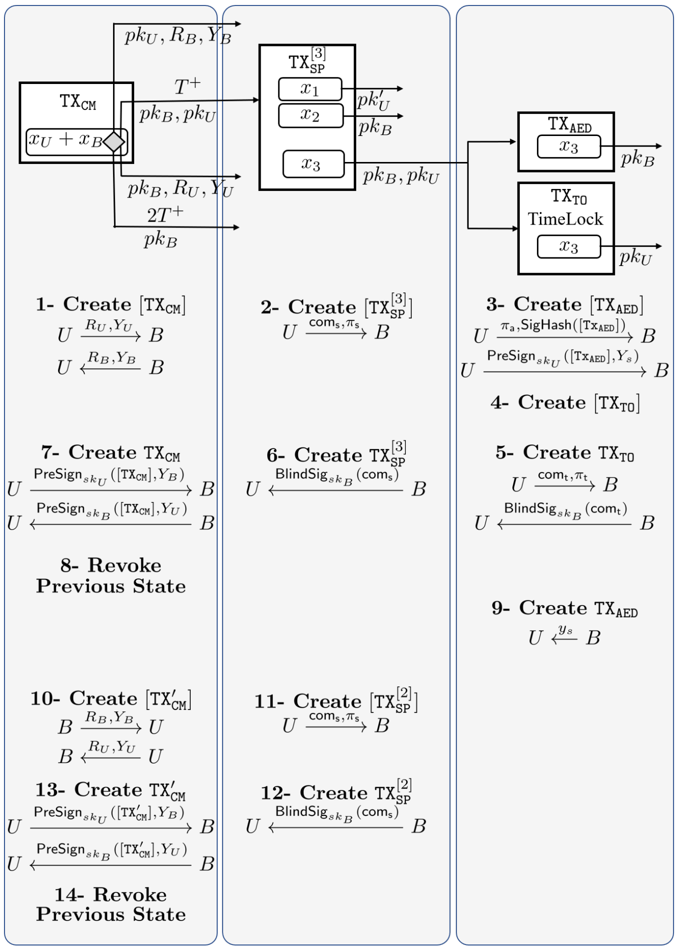
**Close.** To close the channel, party $U$ reveals the value of the latest split transaction and $B$ verifies it. Then, $U$ and $B$ collaboratively create a new transaction that spends the funding transaction's output, and its outputs are the same as the latest split transaction. By publishing this transaction on the blockchain, parties close the channel. Each party $P \in \{B, U\}$ can also non-collaboratively close the channel, given that the other party is unresponsive. To do so, if $P$ is an unblind party, he simply publishes the latest commit and split transaction on the blockchain. If $P$ is a blind party, he publishes the commit transaction. Then, $U$ will have to publish the corresponding split transaction within $2T$ rounds. Given that the published split transaction contains a third output with conditional payment, if $U$ is the payer of that conditional payment, either $B$ extracts the split transaction identifier and the payment amount from the published split transaction, creates $[\texttt{TX}_{\texttt{AED}}]$, adapts $U$'s pre-signature $\tilde{\sigma}_a^U$ into $\sigma_a^U$, and finally creates and publishes $\texttt{TX}_{\texttt{AED}}$ before a specific timeout or $U$ publishes $\texttt{TX}_{\texttt{TO}}$. For the case where $U$ is the payee, either $U$ publishes $\texttt{TX}_{\texttt{AED}}$ before a specific timeout or $B$ extracts the split transaction identifier and the payment amount from the published split transaction, creates $\texttt{TX}_{\texttt{TO}}$ and publishes it on the blockchain.

**Punish.** Once the latest commit transaction is published, if $U$ does not publish its corresponding split transaction within $2T$ rounds, $B$ uses the fourth sub-condition of the commit transaction's output to claim all channel funds. Moreover, if one of the parties, e.g., $U$, publishes an old commit transaction $\texttt{TX}_{\texttt{CM}}$, $B$ uses his own signature in $\texttt{TX}_{\texttt{CM}}$ and its corresponding adaptor statement and pre-signature to extract $U$'s publishing secret $y_U$. Then, having $y_U$ as well as $U$'s revocation secret $r_U$, $B$ claims $\texttt{TX}_{\texttt{CM}}$'s output.

## VI. Description of BlindHub

In this section, we describe the protocol of BlindHub. before the description, we first give the system assumptions.

**System assumptions.** As in TumbleBit [27], we assume the protocols are run in phases and epochs. Each epoch is composed of four phases: (i) *registration* phase, (ii) *puzzle promise* phase, (iii) *puzzle solver* phase, and (iv) *open* phase. We assume that both $\mathbb{S}$ and $\mathbb{R}$ have already carried out the key generation procedure. We assume that communication between honest sender and receiver is unnoticed by $\mathbb{T}$ when exchanging the puzzle and its solution. We further assume that $\mathbb{T}$ will provide NIZK proofs to prove to a user during their first interaction that his encryption key and his verification key of RSoRC scheme are in support of $\Pi_{\mathsf{Enc}}.\mathsf{KeyGen}(1^\lambda)$ and $\Pi_{\mathsf{RSoRC}}.\mathsf{KeyGen}(1^\lambda)$, respectively.



$pk_U, R_B, Y_B$
$\mathtt{TX_{CM}}$
$x_U + x_B$
$T^+$
$pk_B, pk_U$
$pk_B, R_U, Y_U$
$2T^+$
$pk_B$
$\mathtt{TX}_{\mathsf{SP}}^{[3]}$
$x_1$
$x_2$
$x_3$
$pk'_U$
$pk_B$
$pk_B, pk_U$
$\mathtt{TX_{AED}}$
$x_3$
$pk_B$
$\mathtt{TX_{TO}}$
TimeLock
$x_3$
$pk_U$
1- Create $[\mathtt{TX_{CM}}]$
$U \xrightarrow{R_U, Y_U} B$
$U \xleftarrow{R_B, Y_B} B$
2- Create $[\mathtt{TX}_{\mathsf{SP}}^{[3]}]$
$U \xrightarrow{\mathsf{com_s}, \pi_\mathsf{s}} B$
3- Create $[\mathtt{TX_{AED}}]$
$U \xrightarrow{\pi_\mathsf{a}, \mathsf{SigHash}([\mathtt{Tx_{AED}}])} B$
$U \xrightarrow{\mathsf{PreSign}_{sk_U}([\mathtt{Tx_{AED}}], Y_s)} B$
4- Create $[\mathtt{TX_{TO}}]$
7- Create $\mathtt{TX_{CM}}$
$U \xrightarrow{\mathsf{PreSign}_{sk_U}([\mathtt{TX_{CM}}], Y_B)} B$
$U \xleftarrow{\mathsf{PreSign}_{sk_B}([\mathtt{TX_{CM}}], Y_U)} B$
6- Create $\mathtt{TX}_{\mathsf{SP}}^{[3]}$
$U \xleftarrow{\mathsf{BlindSig}_{sk_B}(\mathsf{com_s})} B$
5- Create $\mathtt{TX_{TO}}$
$U \xrightarrow{\mathsf{com_t}, \pi_\mathsf{t}} B$
$U \xleftarrow{\mathsf{BlindSig}_{sk_B}(\mathsf{com_t})} B$
8- Revoke Previous State
9- Create $\mathtt{TX_{AED}}$
$U \xleftarrow{y_s} B$
10- Create $[\mathtt{TX'_{CM}}]$
$B \xrightarrow{R_B, Y_B} U$
$B \xleftarrow{R_U, Y_U} U$
11- Create $[\mathtt{TX}_{\mathsf{SP}}^{[2]}]$
$U \xrightarrow{\mathsf{com_s}, \pi_\mathsf{s}} B$
13- Create $\mathtt{TX'_{CM}}$
$U \xrightarrow{\mathsf{PreSign}_{sk_U}([\mathtt{TX'_{CM}}], Y_B)} B$
$U \xleftarrow{\mathsf{PreSign}_{sk_B}([\mathtt{TX'_{CM}}], Y_U)} B$
12- Create $\mathtt{TX}_{\mathsf{SP}}^{[2]}$
$U \xleftarrow{\mathsf{BlindSig}_{sk_B}(\mathsf{com_s})} B$
14- Revoke Previous State

Fig. 9. Update of a Blind Channel Initiated by Payer

**Protocol of BlindHub.** We now describe the registration, puzzle promise, puzzle solver and open phases.

**Registration.** The Registration Phase is as follows:

1) $\mathbb{S}$ starts by generating a random token identifier $\mathsf{tkid}$ and commits token $\mathsf{tkid}$ and amount $\mathsf{amt}$ to $\mathsf{C_{tkid}}$ and $\mathsf{C_{amt}}$ respectively. Besides, $\mathbb{S}$ generates NIZK proofs $\pi_{\mathsf{tkid}}, \pi_{\mathsf{amt}}$ to prove knowledge of $\mathsf{tkid}$ and $\mathsf{amt}$, respectively. Then $\mathbb{S}$ sends $\mathsf{C_{tkid}}, \mathsf{C_{amt}}, \pi_{\mathsf{tkid}}, \pi_{\mathsf{amt}}$ to $\mathbb{T}$.
2) $\mathbb{T}$ aborts if $\pi_{\mathsf{tkid}}$ or $\pi_{\mathsf{amt}}$ is incorrect. Else, $\mathbb{T}$ generates randomizable signatures on $\mathsf{C_{tkid}}$ and $\mathsf{C_{amt}}$ : $\sigma_{\mathsf{tkid}} \leftarrow \mathsf{RCSign}(\mathsf{C_{tkid}}, \mathsf{C_{amt}})$ and sends $\sigma_{\mathsf{tkid}}$ to $\mathbb{S}$.
3) $\mathbb{S}$ aborts if $\sigma_{\mathsf{tkid}}$ is invalid. Else, $\mathbb{S}$ randomizes $\mathsf{C_{tkid}}, \mathsf{C_{amt}}$, $\sigma_{\mathsf{tkid}}$ to obtain $\mathsf{C'_{tkid}}, \mathsf{C'_{amt}}, \sigma'_{\mathsf{tkid}}$ and sends $\mathsf{C'_{tkid}}, \mathsf{C'_{amt}}, \mathsf{tkid}$, $r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}$ to $\mathbb{R}$, where $r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}$ are the openings of $\mathsf{C'_{tkid}}$ $\mathsf{C'_{amt}}$, respectively.

**Puzzle Promise.** Once the *registration* protocol is completed, the *Puzzle Promise* protocol starts and proceeds as:

1) On receiving $\mathsf{C'_{tkid}}, \mathsf{C'_{amt}}, \mathsf{tkid}, r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}$, $\mathbb{R}$ generates a token-uniqueness proof $\pi_{\mathsf{tup}}$ to prove the $\mathsf{tkid}$ committed in $\mathsf{C'_{tkid}}$ has not been used before. In addition, $\mathbb{R}$ generates $\pi'_{\mathsf{amt}}$ to prove knowledge of $\mathsf{amt}$ in $\mathsf{C'_{amt}}$. Then, $\mathbb{R}$ sends $\mathsf{C'_{tkid}}, \mathsf{C'_{amt}}, \sigma'_{\mathsf{tkid}}, \pi_{\mathsf{tup}}, \pi'_{\mathsf{amt}}$ to $\mathbb{T}$.
2) $\mathbb{T}$ aborts if one of the followings is not valid: $\sigma'_{\mathsf{tkid}}, \pi_{\mathsf{tup}}$, $\pi'_{\mathsf{amt}}$. Else, $\mathbb{T}$ firstly samples the adaptor witness and statement $(Y, y)$, encrypts the witness $y$ into a ciphertext $c_y$, and produces a NIZK proof $\pi_y$ proving that $y$ is a valid solution to puzzle $c_y$. Secondly, $\mathbb{T}$ performs randomizable signatures of randomizable commitments (RSoRC) on adaptor statement $Y$ and the commitment of the amount $\mathsf{C_{amt}}$ : $\tilde{\sigma} \leftarrow \mathsf{RCSign}(Y, \mathsf{C_{amt}})$. After these, $\mathbb{T}$ uses $Y$ as adaptor statement to run the blind adaptor signature protocol with $\mathbb{R}$ to generate $\widehat{\sigma}'_t$ on the transaction for $\mathbb{R}$ : $\widehat{\sigma}'_t \leftarrow \mathsf{BAS.Sign_1}(\mathsf{sk}_t^{\Sigma}, h, Y)$, where $h = H(\mathsf{tx})$(In BlindChannel, $\mathbb{R}$ has sent $h$ to $\mathbb{T}$ and proven knowledge of the pre-image of $h$, which is $\mathsf{tx}$. So $\mathbb{T}$ is convinced that $h$ is a valid hash value). Finally, $\mathbb{T}$ sends $Y, c_y, \pi_y, \tilde{\sigma}, \widehat{\sigma}'_t$ to $\mathbb{R}$.
3) $\mathbb{R}$ aborts if $\pi_y$ is invalid. Else, $\mathbb{R}$ randomizes $Y, \mathsf{C'_{amt}}$, $\tilde{\sigma}$ : $(Y', \mathsf{C''_{amt}}, \tilde{\sigma}', \beta) \leftarrow \mathsf{RCRand}(\mathsf{pp}, Y, \mathsf{C'_{amt}}, \tilde{\sigma})$, and the puzzle $c_y$ : $c'_y \leftarrow \mathsf{PRand}(\beta, c_y)$. Finally, $\mathbb{R}$ sends $c'_y, Y'$, $\mathsf{C''_{amt}}, \tilde{\sigma}'$ and $r''_{\mathsf{amt}}$, the opening of $\mathsf{C''_{amt}}$, to $\mathbb{S}$.

**Puzzle Solver.** The *Puzzle solver* protocol is follows:

1) $\mathbb{S}$ firstly randomizes $c'_y, Y', \mathsf{C''_{amt}}, \tilde{\sigma}'$ received from $\mathbb{R}$ into $c''_y, Y'', \mathsf{C'''_{amt}}, \tilde{\sigma}''$ to preserve its own anonymity and thwart attacks involving collusion of $\mathbb{T}$ and $\mathbb{R}$. Secondly, $\mathbb{S}$ generates $\pi''_{\mathsf{amt}}$ to prove knowledge of $\mathsf{amt}$ in $\mathsf{C''_{amt}}$. Thirdly, $\mathbb{S}$ generates a proof $\pi_{\mathsf{sk}_s^{\Sigma}}$ to prove that $(\mathsf{sk}_s^{\Sigma}$, $\mathsf{pk}_s^{\Sigma})$ is in support of $\Pi_{\mathsf{AS}}.\mathsf{KeyGen}(1^{\lambda})$. Then $\mathbb{S}$generates an adaptor signature $\widehat{\sigma}_s$ on the transaction $\mathsf{tx}'$ using the randomized adaptor statement $Y''$ : $\widehat{\sigma}_s \leftarrow \mathsf{PreSign}(\mathsf{sk}_s^{\Sigma}$, $h', Y'')$, where $h'$ is already proven to be a valid hash value in BlindChannel, as explained before. Finally, $\mathbb{S}$ sends $c''_y, Y'', \mathsf{C'''_{amt}}, \pi''_{\mathsf{amt}}, \widehat{\sigma}_s, \tilde{\sigma}''$ to $\mathbb{T}$.
2) $\mathbb{T}$ aborts if one of the followings is incorrect: the adaptor signature $\widehat{\sigma}_s$($\mathbb{T}$ has obtained $h' = H(\mathsf{tx}')$ in BlindChannel, so $\mathbb{T}$ is able to perform the verification), the randomizable signature $\tilde{\sigma}''$ and the proof $\pi''_{\mathsf{amt}}$. Else, $\mathbb{T}$ decrypts $c''_y$ to obtain the doubly randomized version $y''$ of the value $y$ (i.e., the secret value required by $\mathbb{R}$ to complete the adaptor signature $\widehat{\sigma}'_t$ from puzzle promise). As $y''$ is randomized, $\mathbb{T}$ cannot link it to $\mathbb{R}$ and yet can adapt $\widehat{\sigma}_s$ with $y''$ to generate the full signature $\sigma_s$, which is then sent to $\mathbb{S}$.
3) $\mathbb{S}$ aborts if the signature $\sigma_s$ is not valid. Else, $\mathbb{S}$ extracts $y''$ using the adaptor signature $\widehat{\sigma}_s$ and the valid signature $\sigma_s$, recovers $y'$ by getting rid of one layer of the randomization and shares it with $\mathbb{R}$.

**Open.** $\mathbb{R}$ further removes its part of the randomness from $y'$ and gets the original value $y$, which it uses to adapt the adaptor signature $\widehat{\sigma}'_t$ into a full valid one $\sigma'_t$.

Figures illustrating the above protocols are given in the full version [44] (Appendix K).

## VII. PCH Instantiation

Here we realize a PCH by combining BlindChannel and BlindHub. In particular:

1) **Collateral Setup:** Before the BlindHub registration phase begins, $\mathbb{S}$ updates the channel state of BlindChannel to

the conditional payment state with $\mathbb{T}$ to establish an escrow for the remainder of the protocol between $\mathbb{S}$ and $\mathbb{T}$, where the payment amount is the one committed in $\mathsf{C}_{\mathsf{amt}}$ used in the BlindHub. It is noted here that since it is in the BlindChannel setting, $\mathbb{T}$ has no idea how much the collateral is, but $\mathbb{T}$ is still able to verify if $\mathbb{R}$ has some collateral backed up in the puzzle promise phase. Besides the invisibility, the collateral has two other properties: 1) it can be recovered by $\mathbb{S}$ after the timeout expires unless $\mathbb{S}$ authorizes the spending of it, and 2) it is locked and cannot be reused before the timeout unless $\mathbb{T}$ authorizes the releasing of it.

2) **Payment channel update proposals**: Before the puzzle promise phase of BlindHub starts $\mathbb{T}$ updates the BlindChannel with $\mathbb{R}$ to conditionally pay $m$ coins from the balance of $\mathbb{T}$ to the balance of $\mathbb{R}$, where $m$ is committed in $\mathsf{C}_{\mathsf{amt}}$ used in BlindHub. Here $\mathbb{T}$ cannot see the amount, but $\mathbb{T}$ is able to verify if the same amount of coins has been paid to itself when deciding whether or not to release the coins to $\mathbb{R}$. A similar payment for the same amount of coins is proposed in the BlindChannel between $\mathbb{S}$ and $\mathbb{T}$ before the puzzle solver phase is initiated. As a part of atomicity, here there is an expiration time set for both payments so that the coins can be redeemed by the original owners when the payment is not successful (e.g., one of the parties does not collaborate).
3) **Payment channel update resolutions**: If BlindHub protocol is finally successful, the channel between $\mathbb{S}$ and $\mathbb{T}$ is updated first, and the channel between $\mathbb{T}$ and $\mathbb{R}$ is updated next. On the other hand, if BlindHub protocol fails, the balances of both channels are left as before the start of the execution of the payment.
4) **Collateral release**: At the end of the protocol, the coins locked by $\mathbb{S}$ at the beginning of the payment are released and sent back to $\mathbb{S}$.

## VIII. Flexible Blind Conditional Signature

In a recent work [23], Blind conditional signature (BCS) is proposed to capture the functionality of a synchronization puzzle from [27], [48]. Briefly speaking, synchronization puzzle protocol is a protocol among $\mathbb{S}$, $\mathbb{R}$ and $\mathbb{T}$, where $\mathbb{R}$ and $\mathbb{T}$ execute puzzle promise protocol and generate a puzzle $\tau$, which is used as input in the puzzle solver protocol executed by $\mathbb{S}$ and $\mathbb{T}$, and finally a signature is produced. We refer reader to [23] for the original definitions. Below we propose a variant of BCS to better capture the functionality of BlindHub. We call it Flexible Blind Conditional Signature (FBCS). The main difference between BCS and FBCS is follows: 1) In BCS, the $\mathbb{T}$ and $\mathbb{S}/\mathbb{R}$ can both have access to the transaction, while in FBCS, only $\mathbb{S}/\mathbb{R}$ can have access to the transaction, and $\mathbb{T}$ can only access to the commitment of the transaction. 2) In FBCS, we additionally introduce RSoRC scheme $\Pi_{\mathsf{RSoRC}}$. 3) In BCS, the output of the promise protocol is a puzzle $\tau := (Y, c_y)$, while in FBCS, the output of the promise protocol includes $\tau := (Y, c_y, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma})$, where $Y, c_y, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma}$ are as defined in Section VI. It is noted that since synchronization puzzle only covers promise and solver protocol, the parts related to registration phase in the promise protocol of BlindHub(the token tkid, randomizable signature $\tilde{\sigma}$ and the token-uniqueness proof) are removed from the promise algorithm of FBCS.

**Definition 2** (Flexible Blind Conditional Signature)**.** *A blind conditional signature* $\Pi_{\mathsf{FBCS}} := (\mathsf{Setup}, \mathsf{Promise}, \mathsf{Solver}, \mathsf{Open})$ *is defined with respect to two signature schemes* $\Pi_{\mathsf{DS}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf}), \Pi_{\mathsf{BAS}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf}),\ \Pi_{\mathsf{RSoRC}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *and consists of the following efficient algorithms.*

- $(\mathsf{ek}_t, \mathsf{dk}_t) \leftarrow \mathsf{Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter* $1^\lambda$ *and outputs a key pair* $(\mathsf{ek}_t, \mathsf{dk}_t)$*.*
- $(\perp, \{\tau, \perp\}) \leftarrow \mathsf{Promise}\left\langle \begin{matrix} \mathbb{T}\left(\mathsf{dk}_t, \mathsf{sk}_t^{\Sigma}, \mathsf{sk}^{\chi}, \mathsf{com}(m_{\mathsf{TR}})\right) \\ \mathbb{R}\left(\mathsf{ek}_t, \mathsf{pk}_t^{\Sigma}, \mathsf{pk}^{\chi}, m_{\mathsf{TR}}\right) \end{matrix} \right\rangle$*: The puzzle promise algorithm is an interactive protocol between two users* $\mathbb{T}$ *(Tumbler) (with inputs the decryption key* $\mathsf{dk}_t$*, the signing key of the underlying digital signature scheme* $\mathsf{sk}_t^{\Sigma}$*, the signing key of* RSoRC *scheme* $\mathsf{sk}^{\chi}$*, and a message* $m_{\mathsf{TR}}$*) and* $\mathbb{R}$ *(Receiver) (with inputs the encryption key* $\mathsf{ek}_t$*, the verification key of the underlying digital signature scheme* $\mathsf{pk}_t^{\Sigma}$*, the verification key of* RSoRC *scheme* $\mathsf{pk}^{\chi}$ *and a message* $m_{\mathsf{TR}}$*) and returns* $\perp$ *to* $\mathbb{T}$ *and either a puzzle* $\tau$ *or* $\perp$ *to* $\mathbb{R}$*.*
- $(\{(\sigma^*, s), \perp\}, \{\sigma^*, \perp\}) \leftarrow \mathsf{Solver}\left\langle \begin{matrix} \mathbb{S}\left(\mathsf{sk}_s^{\Sigma}, \mathsf{ek}_t, \mathsf{pk}^{\chi}, m_{\mathsf{ST}}, \tau\right) \\ \mathbb{T}\left(\mathsf{dk}_t, \mathsf{pk}_s^{\Sigma}, \mathsf{pk}^{\chi}, \mathsf{com}(m_{\mathsf{ST}})\right) \end{matrix} \right\rangle$*: The puzzle solving algorithm is an interactive protocol between two users* $\mathbb{S}$ *(Sender) (with inputs the signing key of the underlying digital signature scheme* $\mathsf{sk}_s^{\Sigma}$*, the encryption key* $\mathsf{ek}_t$*, the verification key of* RSoRC *scheme* $\mathsf{pk}^{\chi}$*, a message* $m_{\mathsf{ST}}$*, and a puzzle* $\tau$*) and* $\mathbb{T}$ *(Tumbler) (with inputs the decryption key* $\mathsf{dk}_t$*, the verification key of the underlying digital signature scheme* $\mathsf{pk}_s^{\Sigma}$*, the signing key* $\mathsf{sk}^{\chi}$ *of* $\Pi_{\mathsf{RSoRC}}$ *and a message* $m_{\mathsf{ST}}$*) and returns to both users either a signature* $\sigma^*$ *(*$\mathbb{S}$ *additionally receives a secret* $s$*) or* $\perp$*.*
- $\{\sigma, \perp\} \leftarrow \mathsf{Open}(\tau, s)$*: The open algorithm takes as input a puzzle* $\tau$ *and a secret* $s$ *and returns a signature* $\sigma$ *or* $\perp$*.*

The security of FBCS is defined as **Correctness**, **Blindness**, **Unlockablity**, and **Unforgeability**. Below we only present the informal definitions of these security properties. We leave the complete formal definitions in the full version [44](Appendix F) .

$\Pi_{\mathsf{FBCS}}$ is **correct** if $\mathbb{S}$ and $\mathbb{R}$ are able to obtain the valid signatures on $m_{\mathsf{ST}}$ and $m_{\mathsf{TR}}$, respectively after the BlindHub protocol is successfully completed. $\Pi_{\mathsf{FBCS}}$ is **blind** if $\mathbb{T}$ cannot link two promise/solve sessions together. $\Pi_{\mathsf{FBCS}}$ is **unlockable** if it is hard for $\mathbb{T}$ to generate a valid signature on a message from $\mathbb{S}$ that prevents $\mathbb{R}$ from being able to unlock the entire signature in the accompanying promise session. $\Pi_{\mathsf{FBCS}}$ is **unforgeable** if $\mathbb{R}$ could not output a valid signature on $m_{\mathsf{TR}}$ before $\mathbb{S}$ successfully complete the solver protocol with

$\mathbb{T}$. $\Pi_{\mathsf{FBCS}}$ is secure if it is correct, blind, unlockable, and unforgeable.

Finally, we can give our main theorem in this section. The corresponding proofs can be found in the full version [44] (Appendix G).

**Theorem 1.** *Let $\Pi_{\mathsf{Enc}}$ be a linear-only homomorphic encryption scheme, $\Pi_{\mathsf{AS}}$ is a secure adaptor signature scheme, $\Pi_{\mathsf{BAS}}$ is a secure BAS scheme, $\Pi_{\mathsf{RSoRC}}$ is a secure signature on randomizable commitments scheme, $\Pi_{\mathsf{NIZK}}$ is a sound proof system. Assuming the hardness of one-more discrete logarithm problem, the BlindHub protocol is a secure flexible blind conditional signature scheme.*

## IX. Security Analysis of PCH

We now analyze the security of our PCH system. More formal security definitions and proofs of the following theorems can be found in Appendix C .

**Griefing Resistance.** It requires a user to prove that the payment request is previously backed by some locked coins during the payment procedure. In our construction, the authenticity is enforced by $\mathbb{T}$ performing $\tilde{\sigma} \leftarrow \mathsf{RCSign}(\mathsf{C}_{\mathsf{tkid}}, \mathsf{C}_{\mathsf{amt}})$ and securely updating the BlindChannel during the registration phase, and $\mathbb{R}$ proving the uniqueness of the token committed in $\mathsf{C}_{\mathsf{tkid}}$ as well as securely updating the BlindChannel in the promise phase. The security depends on the unforgeability of $\Pi_{\mathsf{RSoRC}}$, the biding property of $\Pi_{\mathsf{COM}}$, the security of $\Pi_{\mathsf{BC}}$ and the soundness of $\Pi_{\mathsf{NIZK}}$.

**Theorem 2.** *Assuming the security of $\Pi_{\mathsf{BC}}$, the unforgeability of $\Pi_{\mathsf{RSoRC}}$ , the biding of $\Pi_{\mathsf{COM}}$ and the soundness of $\Pi_{\mathsf{NIZK}}$, our PCH system achieves griefing resistance.*

**Atomicity** ensures that the money received by $\mathbb{R}$ from $\mathbb{T}$ should be the same as that received by $\mathbb{T}$ from $\mathbb{S}$. Below we analyze the balance security of $\mathbb{T}$ and $\mathbb{S}$ respectively. $\mathbb{T}$ loses money if the money it paid to the $\mathbb{R}$ is more than that received from $\mathbb{S}$. This will violate either the unforgeability of $\Pi_{\mathsf{FBCS}}$ or the security of $\Pi_{\mathsf{BC}}$. $\mathbb{S}$ loses money if at the end of the puzzle solver protocol $\mathbb{T}$ receives money, but $\mathbb{R}$ does not get paid. This will violate either the unlockability of $\Pi_{\mathsf{FBCS}}$ or the security of $\Pi_{\mathsf{BC}}$.

**Theorem 3.** *Assuming the unlockability and unforgeability of $\Pi_{\mathsf{FBCS}}$ and the security of $\Pi_{\mathsf{BC}}$, our PCH system achieves atomicity.*

**Value Privacy** requires that the payment value could not be leaked to $\mathbb{T}$. This is enforced by committing the amounts throughout the protocol. The security depends on the blindness of $\Pi_{\mathsf{FBCS}}$ and the privacy of $\Pi_{\mathsf{BC}}$.

**Theorem 4.** *Assuming the blindness of $\Pi_{\mathsf{FBCS}}$ and security of the $\Pi_{\mathsf{BC}}$, our PCH system achieves value privacy.*

**Relationship anonymity** requires that the relationship of $\mathbb{S}$ and $\mathbb{R}$ should be hidden from $\mathbb{T}$. This is enforced as follows. Firstly, we assumed that all protocols are phase- and epoch-coordinated, which eliminates timing attacks in which $\mathbb{T}$ purposefully delays or accelerates its interactions with another party. Secondly, we assumed that $\mathbb{S}$ and $\mathbb{R}$ communicate through a secure and anonymous communication channel, so $\mathbb{T}$ cannot eavesdrop and utilize the network information to link $\mathbb{S}$ and $\mathbb{R}$. Thirdly, the elements generated in the registration/promise phase will be randomized before being used in the promise/solver phase. Fourthly, our PCH system achieves value privacy, which prevents the tumbler to learn the relationship from the payment value.

**Theorem 5.** *Assuming the blindness of $\Pi_{\mathsf{FBCS}}$ and security of $\Pi_{\mathsf{BC}}$, our PCH system achieves relationship anonymity.*

## X. Performance Analysis

**Implementation details.** To benchmark the performance of our protocol, we implemented[3] our protocol with Rust programming language, Swanky[4] multiparty computation library, and ZenGo-X/curv[5] curve library, where ZenGo-X/curv is a pretty wrapper of the real Bitcoin curve library. The GC circuit files are generated with Verilog hardware description language and the Yosys[6] tool.

We instantiate adaptor signature scheme and blind adaptor signature scheme as ECDSA-based adaptor signature and ECDSA-based blind adaptor signature, respectively, and both instantiations are over the elliptic curve *secp256k1*, which is used on Bitcoin. We instantiate the RSoRC scheme using the generalized version of the scheme introduced in [3] over the BLS12-381 curve[7]. The linear-only encryption scheme is instantiated using HSM-CL encryption scheme [11]. We instantiate the commitment scheme as Pedersen commitment scheme [41]. We instantiate BlindChannel in the Bitcoin setting, and we instantiate the proof used in Bitcoin-based BlindChannel as garbled circuit-based zero knowledge proof [12], [40]. Concrete instantiation of our whole protocol can be found in the full version [44] (Appendix K).

We leverage garbled circuit-based zero-knowledge proof (GCZK) to prove the correctness of the committed transaction. In a nutshell, the unblind party is going to prove the following statements to the blind party: 1) the committed message is a correct transaction, 2) the payment amount written in the transaction is the one committed in a given commitment, 3) the message is committed according to the transaction digest algorithm defined in BIP-0143[8]. We design a circuit to capture the transaction digest algorithm. The circuit for proving this statement uses 33 SHA-256 modules and consequently reaches 721054 AND gates. More details about the GCZK can be found in the full version [44] (Appendix H).

**Computation time and communication overhead.** The benchmark for the GCZK is taken on a Ryzen 5800H, 8GB

[3]https://github.com/blind-channel/blind-hub

[4]https://github.com/GaloisInc/swanky

[5]https://github.com/ZenGo-X/curv

[6]https://github.com/YosysHQ/yosys

[7]To enable RSoRC over BLS12-381 curve to sign on group element over secp256k1 curve, we leverage an equality proof to bridge the gap between these two different curves. Detailed discussions about this can be found in the full version [44] (Appendix K).

[8]https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki

| Role | Bandwidth (KB) | Comp. (ms) | Optimized.Comp. (ms) |
|---|---|---|---|
| Sender | 55843 | 8172 | 3083 |
| Tumbler | 87855 | 16245 | 8160 |
| Receiver | 32017 | 9063 | 6067 |

| Phase | Bandwidth (KB) | Comp. (ms) | Optimized Comp. (ms) |
|---|---|---|---|
| Register | 23832 | 2687 | 594 |
| Promise | 23834 | 7887 | 5773 |
| Solver | 23831 | 4381 | 2267 |
| Open | 16357 | 2284 | 520 |
| Total | 87860 | 17239 | 9154 |

TABLE II
BANDWIDTH OVERHEAD AND COMPUTATION COST. THE OPTIMIZED COST IS OUR ESTIMATION OF THE PROTOCOL WITH [53].

RAM laptop computer, where the test data is generated from Bitcoin regtest network[9]. In the local area network (LAN) setting, performing this GCZK itself consumes 1.716 seconds. Besides, the circuit needs additional 5 seconds to be read from the disk, but such a time cost could be stripped since the circuit could be preloaded into RAM in the real world. In addition, we use the tc tool (in Linux) to emulate the WAN setting with 250mbps bandwidth and 200ms latency, and achieve an overall running time of $\sim$31 s for one payment. We show the computation time and communication overhead by phases in Table II. Observe that the required computation time for different parties varies even in the same phase, which occurs when one party finishes all the required computations and sends messages to the other one while the other party needs to do verification and other computation. And accordingly, the phase-based computation time would contain some idle time. To capture this, we also show the performance by role in Table II.

**Discussion.** We note that our implementation is a proof-of-concept realization of our work, and the bandwidth and computational cost can be improved by using alternative zero knowledge proofs. Our implementation results given in Table II show that our protocol is feasible and relatively practical. However, replacing the GCZK protocol we used to realize proofs for circuits, with other zero-knowledge proof techniques (e.g., [4], [15], [51], [53]) can provide better efficiency. For example, Quicksilver [53] can prove boolean circuits at a speeding of 7.7 million AND gates per second and 1bit/gate in bandwidth. We conjecture that we can finish the proof in less than 1 second and have 128 times improvement in bandwidth usage by utilizing Quicksilver. In this regard, we estimate the time and overhead of our solution leveraging quicksilver and add them as optimized computation time in Table II.

**Comparing performance with A$^2$L versions.** A$^2$L protocol completes in 3 seconds regarding the implementation results given by the authors [48]. However, as pointed out in [16], the CL encryption parameters[10] chosen in A$^2$L may not be sufficient for security reasons. When, we use the parameters suggested in [16], the A$^2$L protocol is expected to run in 6 seconds, which is in the same order of computational cost of our protocol with the optimized implementation.

[9]https://developer.bitcoin.org/examples/testing.html

[10]The discriminant, which is a parameter of unknown order group, used in A$^2$L is only around 1800 bits, which is lower than the required length of 3800-bit pointed out in [16].

## XI. CONCLUSION

We present the first Bitcoin-compatible PCH that achieves value privacy, relationship anonymity and supports variable amounts for payment. To achieve this, we propose BlindChannel, a new payment channel protocol for privacy preserving payment, and BlindHub, a novel three party protocol to synchronize the payment channel update in the PCH. We formally analyze their security and give an implementation to show their practicality.

### A. Acknowledgements

We thank Erkan Tairi, Xinyu Li, Ron Steinfeld, Raphael Phan, Shabnam Kasra Kermanshahi, Sushmita Ruj and anonymous reviewers for useful discussions and suggestions. This work is supported by the Australian Research Council (ARC) under project DE210100019 and project DP220101234, the Research Grant Council of Hong Kong (GRF project 106220117).

## REFERENCES

[1] L. Aumayr, K. Abbaszadeh, and M. Maffei, “Thora: Atomic and privacy-preserving multi-channel updates,” *IACR Cryptol. ePrint Arch.*, p. 317, 2022.

[2] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, “Generalized channels from limited blockchain scripts and adaptor signatures,” in *ASIACRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 13091. Springer, 2021, pp. 635–664.

[3] B. Bauer and G. Fuchsbauer, “Efficient signatures on randomizable ciphertexts,” in *International Conference on Security and Cryptography for Networks*. Springer, 2020, pp. 359–381.

[4] C. Baum, A. J. Malozemoff, M. B. Rosen, and P. Scholl, “Mac’n’cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions,” in *Annual International Cryptology Conference*. Springer, 2021, pp. 92–122.

[5] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, “The one-more-rsa-inversion problems and the security of chaum’s blind signature scheme.” *Journal of Cryptology*, vol. 16, no. 3, 2003.

[6] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova, “On the (in) security of ros,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 33–53.

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

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

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

[10] T. Cao, J. Yu, J. Decouchant, X. Luo, and P. Veríssimo, “Exploring the monero peer-to-peer network,” in *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, ser. Lecture Notes in Computer Science, J. Bonneau and N. Heninger, Eds., vol. 12059. Springer, 2020, pp. 578–594.

[11] G. Castagnos and F. Laguillaumie, “Linearly homomorphic encryption from ddh,” in *Cryptographers’ Track at the RSA Conference*. Springer, 2015, pp. 487–505.

[12] M. Chase, C. Ganesh, and P. Mohassel, “Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials,” in *Annual International Cryptology Conference*. Springer, 2016, pp. 499–530.

[13] J. O. M. Chervinski, D. Kreutz, and J. Yu, "Analysis of transaction flooding attacks against monero," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2021, Sydney, Australia, May 3-6, 2021*. IEEE, 2021, pp. 1–8.

[14] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[15] S. Dittmer, Y. Ishai, and R. Ostrovsky, "Line-point zero knowledge and its applications," *Cryptology ePrint Archive*, 2020.

[16] S. Dobson and S. D. Galbraith, "Trustless groups of unknown order with hyperelliptic curves," *IACR Cryptol. ePrint Arch.*, p. 196, 2020. [Online]. Available: https://eprint.iacr.org/2020/196

[17] J. Du, Z. Ge, Y. Long, Z. Liu, S. Sun, X. Xu, and D. Gu, "Mixct: Mixing confidential transactions from homomorphic commitment," Cryptology ePrint Archive, Paper 2022/951, 2022, https://eprint.iacr.org/2022/951. [Online]. Available: https://eprint.iacr.org/2022/951

[18] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 106–123.

[19] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 949–966.

[20] O. Ersoy, J. Decouchant, S. Prabhu Kimble, and S. Roos, "Syncpcn/psyncpcn: Payment channel networks without blockchain synchrony," *arXiv e-prints*, pp. arXiv–2207, 2022.

[21] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *International conference on the theory and application of cryptology and information security*. Springer, 2019, pp. 649–678.

[22] G. Fuchsbauer, A. Plouviez, and Y. Seurin, "Blind schnorr signatures and signed elgamal encryption in the algebraic group model," in *EUROCRYPT 2020*, ser. Lecture Notes in Computer Science, A. Canteaut and Y. Ishai, Eds., vol. 12106. Springer, 2020, pp. 63–95.

[23] N. Glaeser, M. Maffei, G. Malavolta, P. Moreno-Sanchez, E. Tairi, and S. A. Thyagarajan, "Foundations of coin mixing services," Cryptology ePrint Archive, Paper 2022/942, 2022, https://eprint.iacr.org/2022/942. [Online]. Available: https://eprint.iacr.org/2022/942

[24] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[25] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 473–489.

[26] J. Groth, "Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems," in *TCC*, ser. Lecture Notes in Computer Science, vol. 2951. Springer, 2004, pp. 152–170.

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

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

[29] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, "An empirical analysis of anonymity in zcash," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, W. Enck and A. P. Felt, Eds. USENIX Association, 2018, pp. 463–477.

[30] R. Khalil, A. Gervais, and G. Felley, "Nocust-a securely scalable commit-chain," *Cryptology ePrint Archive, Report 2018/642*, 2018.

[31] A. Kumar, C. Fischer, S. Tople, and P. Saxena, "A traceability analysis of monero's blockchain," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 153–173.

[32] J. Lind, O. Naor, I. Eyal, F. Kelbert, P. Pietzuch, and E. G. Sirer, "Teechain: Reducing storage costs on the blockchain with offline payment channels," in *Proceedings of the 11th ACM International Systems and Storage Conference*, 2018, pp. 125–125.

[33] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *CCS*. ACM, 2017, pp. 455–471.

[34] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," *Cryptology ePrint Archive*, 2018.

[35] G. Maxwell, "Coinswap: Transaction graph disjoint trustless trading," *CoinSwap: Transaction graph disjoint trustless trading (October 2013)*, 2013.

[36] S. Meiklejohn and R. Mercer, "Möbius: Trustless tumbling for transaction privacy," 2018.

[37] I. Miers, "Bolt: Private payment channels," https://electriccoin.co/blog/bolt-private-payment-channels/.

[38] P. Moreno-Sanchez, T. Ruffing, and A. Kate, "Pathshuffle: Credit mixing and anonymous payments for ripple." *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 3, p. 110, 2017.

[39] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin, "An empirical analysis of traceability in the monero blockchain," *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 3, pp. 143–163, 2018.

[40] K. Nguyen, M. Ambrona, and M. Abe, "Wi is almost enough: Contingent payment all over again," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 641–656.

[41] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.

[42] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," https://lightning.network/lightning-network-paper.pdf, 2016.

[43] X. Qin, C. Cai, and T. H. Yuen, "One-more unforgeability of blind ecdsa," in *European Symposium on Research in Computer Security*. Springer, 2021, pp. 313–331.

[44] X. Qin, S. Pan, A. Mirzaei, Z. Sui, O. Ersoy, A. Sakzad, M. F. Esgin, J. K. Liu, J. Yu, and T. H. Yuen, "Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts," Cryptology ePrint Archive, 2022.

[45] T. Ruffing and P. Moreno-Sanchez, "Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 133–154.

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

[47] ——, "P2p mixing and unlinkable bitcoin transactions," *Cryptology ePrint Archive*, 2016.

[48] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "$A^2l$: Anonymous atomic locks for scalability in payment channel hubs," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1834–1851.

[49] M. Tran, L. Luu, M. S. Kang, I. Bentov, and P. Saxena, "Obscuro: A bitcoin mixer using trusted execution environments," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 692–701.

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

[51] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1074–1091.

[52] D. A. Wijaya, J. K. Liu, R. Steinfeld, D. Liu, and J. Yu, "On the unforkability of monero," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*, S. D. Galbraith, G. Russello, W. Susilo, D. Gollmann, E. Kirda, and Z. Liang, Eds. ACM, 2019, pp. 621–632.

[53] K. Yang, P. Sarkar, C. Weng, and X. Wang, "Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2986–3001.

[54] J. Yu, M. H. A. Au, and P. J. E. Veríssimo, "Re-thinking untraceability in the cryptonote-style blockchain," in *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019, pp. 94–107.

[55] Z. Yu, M. H. Au, J. Yu, R. Yang, Q. Xu, and W. F. Lau, "New empirical traceability analysis of cryptonote-style blockchains," in *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, ser. Lecture Notes in Computer Science, I. Goldberg and T. Moore, Eds., vol. 11598. Springer, 2019, pp. 133–149.

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

## Appendix A
## Discussion Regarding Privacy

Here, we discuss additional aspects of our PCH pertaining to privacy limitations. As mentioned in A$^2$L [48], these limitations are inherent in the PCH settings, and thus affecting our construction as well.

**Epoch anonymity.** Our payment channel hub protocol runs in epoch. The anonymity level of the PCH system running in epoch depends on the number of the participants within the epoch. Say, during an epoch, if there are $k$ payments completed successfully, the anonymity set is of size $k$(i.e., $k$-anonymity), since there exists $k$ compatible interaction graphs, as defined in Section II-A.

However, the above $k$-anonymity can only be achieved if there is no intersection between sender-receiver pairs from different epochs. Otherwise, tumbler can further eliminate the anonymity sets.

**Intersection attack.** The tumbler can correlate the information across phases and epochs to eliminate the anonymity sets. Namely, tumbler can observe which pairs of senders and receivers fall in the intersection of different epochs and break their anonymity. This is called *intersection attack*.

We observe that fix-amount PCH system are vulnerable to intersection attack. Since unless the amount a sender is going to pay the receiver is exactly the same as the one fixed in the system, he needs to run the protocol for multiple times for one payment, which means that he needs to interact with tumbler for multiple epochs in order to complete the payment. As a result, the tumbler can easily link the sender and receiver by observing which pair of sender-receiver appear continuously. That is, the tumbler can break their anonymity by a simple intersection attack. In contrast, BlindHub allows variable-amount payment. It means that users only need to run the protocol once, which protect them from the aforementioned intersection attack.

**Tumbler-Sender(Receiver) Collusion**. It is possible for the tumbler $\mathbb{T}$ to figure out who the sender $\mathbb{S}$ (receiver $\mathbb{R}$) is if $\mathbb{R}$ ($\mathbb{S}$) is corrupted. During some anonymous payments, $\mathbb{S}$ or $\mathbb{R}$ does not want to reveal his real identity, $\mathbb{T}$ can leverage the corrupted party to make some fake payment and explore the real identity of the uncorrupted party. For example, when $\mathbb{R}$ is corrupted, he can send a fake puzzle to $\mathbb{S}$, and anyone who is asking $\mathbb{T}$ to solve the puzzle will be identified as $\mathbb{S}$ in this payment. This can be mitigated by requiring $\mathbb{R}$ to prove that the given puzzle is randomized from the original one. When $\mathbb{S}$ is corrupted, $\mathbb{T}$ can explore who the $\mathbb{R}$ is by asking $\mathbb{S}$ not to send the witness to $\mathbb{R}$. The one who fails to provide the witness and claim coins from $\mathbb{T}$ is the potential $\mathbb{R}$.

**Attacks with Auxiliary Information**. The privacy properties are discussed without auxiliary information from the outside. Otherwise, they cannot be guaranteed as well. For example, when $\mathbb{T}$ knows that there is a certain $\mathbb{S}$ pays a milk company every week at a certain time, and $\mathbb{T}$ can observe such periodical payments and infer the users identify with higher possibility. This can be mitigated by making the unregular payment.

```
$\mathsf{wBlindness}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda)$
1 : $(\mathsf{pk}, m_0, m_1) \leftarrow \mathcal{A}(1^\lambda)$
2 : $b \leftarrow\$ \{0,1\}$
3 : $b^* \leftarrow \mathcal{A}^{\mathsf{User}(\mathsf{pk},m_b),\mathsf{User}(\mathsf{pk},m_{1-b})}(1^\lambda)$
4 : $\mathbf{return}\ (b == b^*)$
```

Fig. 10. Experiment $\mathsf{wBlindness}_{\mathcal{A},\prod_{R,\Sigma}}$.

## Appendix B
## Security Model and Concrete Construction of Blind Adaptor Signature

Here, we give detailed security models, security analysis and concrete construction of blind adaptor signature.

### A. Security Model

Suppose that there are $N_s$ (resp. $N_p$) interactions by the signer in the $\mathsf{Sign}(\mathsf{sk})$ (resp. $\mathsf{PreSign}(\mathsf{sk}, Y)$) algorithm. We use $(m', st_1) \leftarrow \mathsf{Sign}_1(\mathsf{sk}, m)$ (resp. $(m', st_1) \leftarrow \mathsf{PreSign}_1(\mathsf{sk}, Y, m)$) to represent the first interaction, where $m$ is the message received by the signer, $m'$ is the message output and $st_1$ is the internal state. Similarly, $(m', st_i) \leftarrow \mathsf{Sign}_i(st_{i-1}, m)$ (resp. $(m', st_1) \leftarrow \mathsf{PreSign}_i(st_{i-1}, m)$) denotes the $i$-th interaction, for $i \in [2, N_s - 1]$ (resp. $i \in [2, N_p - 1]$ ), and $(m', b) \leftarrow \mathsf{Sign}_{N_s}(st_{N_s-1}, m)$ (resp. $(m', b) \leftarrow \mathsf{PreSign}_{N_p}(st_{N_p-1}, m)$) denotes the last interaction, where $b$ is a bit. Below we give the formal definitions of the properties of BAS given in Section IV and give the one-more unforgeability game of BAS in Fig. 12, weak blindness game in Fig. 10 and witness extractability game in Fig. 11.

**Definition 3** (Weak Blindness)**.** *A* BAS *scheme* $\Pi_{\mathsf{BAS}}$ *achieves weak blindness if for every PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{wBlindness}_{\mathcal{A},\Pi_{\mathsf{BAS}}}$ *defined in Fig. 10,* $\Pr[\mathsf{wBlindness}_{\mathcal{A},\Pi_{\mathsf{BAS}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

Witness Extractability of BAS is different from that of the adaptor signature. It is because the message signed by the oracles $O_{\mathsf{S}}$ and $O_{\mathsf{pS}}$ (as shown in Fig. 12) is unknown to the challenger. Hence, we have to change the definition of the oracles to avoid giving a full signature/pre-signature to the adversary.

**Definition 4** (Witness Extractability)**.** *A* BAS *scheme* $\Pi_{\mathsf{BAS}}$ *is witness extractable if for every PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{baWitExt}_{\mathcal{A},\Pi_{\mathsf{BAS}}}$ *defined in Fig. 11,* $\Pr[\mathsf{baWitExt}_{\mathcal{A},\Pi_{\mathsf{BAS}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

**Definition 5** (One-more Unforgeability)**.** *A* BAS *scheme* $\Pi_{\mathsf{BAS}}$ *is* omaEUF–CMA *secure if for every PPT adversary* $\mathcal{A}$ *running the experiment* $\mathsf{omaSignForge}_{\mathcal{A},\Pi_{\mathsf{BAS}}}$ *defined in Fig. 12,* $\Pr[\mathsf{omaSignForge}_{\mathcal{A},\Pi_{\mathsf{BAS}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

```
$\mathsf{baWitExt}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$(M^*, Y) \leftarrow \mathcal{A}^{O'_{\mathsf{S}}, O'_{\mathsf{pS}}}(\mathsf{pk})$
$\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, Y) \leftrightarrow \mathsf{User}(\mathsf{pk}, M^*, Y)$
$\sigma^* \leftarrow \mathcal{A}^{O_{\mathsf{S}}, O_{\mathsf{pS}}}(\hat{\sigma}, Y)$
$y^* \leftarrow \mathsf{Ext}(Y, \sigma^*, \hat{\sigma})$
$\mathbf{return}\ ((Y, y^*) \notin R \wedge \mathsf{Vf}(\mathsf{pk}, \sigma^*, M^*) = 1)$
```

Fig. 11. Experiment $\mathsf{baWitExt}_{\mathcal{A},\prod_{R,\Sigma}}$. $O'_{\mathsf{S}}, O'_{\mathsf{pS}}$ are the same as $O_{\mathsf{S}}, O_{\mathsf{pS}}$ in Algorithm 12 except that $O_{\mathsf{S}}(\cdot, N_s, \cdot), O_{\mathsf{pS}}(\cdot, \cdot, N_p, \cdot)$ are not allowed.

## B. Concrete Construction

We now construct a provably secure blind adaptor signature scheme based on ECDSA digital signatures that are commonly used by blockchains. Although in the literature, there are already blind signatures schemes [22], [43] that are compatible with Bitcoin, their blindness property is the strong one, which requires that a signer cannot link a message/signature pair to a particular execution of the signing protocol. Here the blind adaptor ECDSA we propose satisfies weak blindness.

**Blind Adaptor ECDSA Construction.**

Setup. On input a security parameter $\lambda$, it runs $(p, \mathbb{G}, G) \leftarrow \mathsf{GpGen}(1^{\lambda})$ and picks a cryptographic hash function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$. It returns $\mathsf{par} = (p, \mathbb{G}, G, H)$.

KeyGen. On input par, it picks $\mathsf{sk} := x \leftarrow\$\, \mathbb{Z}_p$ and computes $\mathsf{pk} := X = xG$.

Sign. The signer's input includes security parameter $1^{\lambda}$, signer's public key $X$ and secret key $x$, adaptor statement $Y$, adaptor statement $y$, a and the proof of knowledge $\pi$ of the witness of $Y$. The user's input includes security parameter $1^{\lambda}$, signer's public key $X$ and the message to be signed $m$.

1) The user hashes the message $m$: $h = H(m)$, and generates a proof-of-knowledge $\pi_h$ of the pre-image of $h$. Then the user sends $h, \pi_h$ to the signer.
2) On receiving $h, \pi_h$, the signer verifies the validity of the proof $\pi_h$ and aborts if the proof is not correct. Then the signer samples $k_a$ from $\mathbb{Z}_p \backslash \{0\}$, then computes $\hat{R} = k_a Y = (r, \cdot), R = k_a G$, and gives a zero knowledge proof $\pi_a$ to prove that $\hat{R}$ and $R$ share the same discrete log under $Y$ and $G$ respectively. finally, the signer generates $s' = {k_a}^{-1}(h + rx)$, and sends $\hat{R}, R$, $\pi_a, s'$ to the user.
3) The user aborts if $s' = 0$ or $\mathsf{PreVf}(m, I_Y; s') = 0$ or $\mathsf{NIZKVerify}(\pi_a) = 0$. Else, the user returns $(r, s')$ as the adaptor ECDSA signature.

Verify. To verify an adaptor signature $\hat{\sigma} = (r, s, Y)$ for a message $m$ and a public key $X$, where $Y$ is a adaptor statement, it computes $R = Y(\frac{H(m)}{s}G + \frac{r}{s}X)$. It returns 1 if $r = f(R)$, or returns 0 otherwise.

*Security Analysis.* Now we analyze the security of the blind adaptor ECDSA proposed above. Firstly, the protocol achieves weak blindness trivially, since the user only forwards the hash of the message to the signer. Below we prove the one-more unforgeability of our protocol by reducing it to the EUF-CMA security of ECDSA adaptor signature, which is introduced in [2]. It is noted that the one-more unforgeability of our blind adaptor ECDSA scheme does not rely on ROS (Random inhomogeneities in a Overdetermined Solvable system of linear equations) assumption, and thus not affected by the ROS attack proposed recently [6].

```
$\mathsf{omaSignForge}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda)$
$\mathcal{S}_s := \emptyset, \mathcal{S}_p := \emptyset, k_{s_1} := 0, k_{p_1} := 0, k_{s_2} := 0$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$
$(M_1^*, \ldots, M_n^*) \leftarrow \mathcal{A}^{O_{\mathsf{S}}, O_{\mathsf{pS}}}(\mathsf{pk})$
$(Y, y) \leftarrow \mathsf{GenR}(1^{\lambda})$
$\hat{\sigma}_i \leftarrow \mathsf{PreSign}((\mathsf{pk}, \mathsf{sk}), Y, M_i^*), \forall i \in [1, n]$
$(\sigma_1^*, \ldots, \sigma_n^*) \leftarrow \mathcal{A}^{O_{\mathsf{S}}, O_{\mathsf{pS}}}(\hat{\sigma}_1, \ldots, \hat{\sigma}_n, Y)$
$\mathbf{return}\ (k_{s_2} < n \wedge M_i^* \neq M_j^*, \forall i \neq j \in [1, n] \wedge$
$\mathsf{Vf}(\mathsf{pk}, M_i^*, \sigma_i^*) = 1, \forall i \in [1, n]$

$O_{\mathsf{S}}(M, i, j)$
$\mathbf{if}\ i = 1:$
    $k_{s_1} = k_{s_1} + 1$
$(M', st_{k_{s_1},1}) \leftarrow \mathsf{Sign}_1(\mathsf{sk}, M)$
    $\mathcal{S}_s = \mathcal{S}_s \cup \{k_{s_1}\}$
    $\mathbf{return}\ (k_{s_1}, M')$
$\mathbf{if}\ i = N_s:$
    $\mathbf{if}\ j \notin \mathcal{S}_s : \mathbf{return}\ \bot$
    $(M', b) \leftarrow \mathsf{Sign}_{N_s}(st_{j,N_s}, M)$
    $\mathbf{if}\ b = 1 : \mathcal{S}_s = \mathcal{S}_s \setminus \{j\}, k_{s_2} = k_{s_2} + 1$
    $\mathbf{return}\ M'$
$\mathbf{if}\ i \in [2, N_s - 1]:$
    $\mathbf{if}\ j \notin \mathcal{S}_s : \mathbf{return}\ \bot$
    $(M', st_{j,i}) \leftarrow \mathsf{Sign}_i(st_{j,i-1}, M)$
    $\mathbf{return}\ M'$
$\mathbf{return}\ \bot$

$O_{\mathsf{pS}}(M, Y, i, j)$
$\mathbf{if}\ i = 1:$
    $k_{p_1} = k_{p_1} + 1$
    $(M', st_{k_{p_1},1}) \leftarrow \mathsf{PreSign}_1(\mathsf{sk}, Y, M)$
    $\mathcal{S}_p = \mathcal{S}_p \cup \{k_{p_1}\}$
    $\mathbf{return}\ (k_{p_1}, M')$
$\mathbf{if}\ i = N_p:$
    $\mathbf{if}\ j \notin \mathcal{S}_p : \mathbf{return}\ \bot$
    $(M', b) \leftarrow \mathsf{Sign}_{N_p}(st_{j,N_s}, M)$
    $\mathbf{if}\ b = 1 : \mathcal{S}_p = \mathcal{S}_p \setminus \{j\}, k_2 = k_2 + 1$
    $\mathbf{return}\ M'$
$\mathbf{if}\ i \in [2, N_s - 1]:$
    $\mathbf{if}\ j \notin \mathcal{S}_p : \mathbf{return}\ \bot$
    $(M', st_{j,i}) \leftarrow \mathsf{PreSign}_i(st_{j,i-1}, M)$
    $\mathbf{return}\ M'$
$\mathbf{return}\ \bot$
```

Fig. 12. Experiment $\mathsf{omaSignForge}_{\mathcal{A},\Pi_{\mathsf{BAS}}}$

**Theorem 6.** *Assuming the unforgeability of ECDSA adaptor signature and the soundness of non-interactive zero knowledge proof, our blind adaptor ECDSA protocol achieves one-more unforgeability.*

*Proof.* Assume there is an adversary $\mathcal{A}$ that can break the one-more unforgeability of our blind adaptor ECDSA protocol, we construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the unforgeability of ECDSA adaptor signature. First, the challenger $\mathcal{C}$ of the ECDSA adaptor signature gives $\mathsf{param}$, adaptor statement $Y$ and $\mathsf{pk}$ to $\mathcal{B}$. $\mathcal{B}$ forwards $\mathsf{param}$, $Y$ and $\mathsf{pk}$ to $\mathcal{A}$, and $\mathcal{A}$ outputs $(M_1^*, \ldots, M_n^*)$. When $\mathcal{A}$ queries the signing oracle with $h_i$, $\pi_{h_i}$ as input, $\mathcal{B}$ first extract the pre-image $M_i^*$ of $h_i$ from $\pi_{h_i}$. Then $\mathcal{B}$ forwards $(Y, M_i^*)$ to $\mathcal{C}$ and returns the oracle reply to $\mathcal{A}$. After running $n-1$ queries to the signing oracle, $\mathcal{A}$ outputs $n$ distinct message-signature pairs $(M_j, \sigma_j = (s_j, r_j))$ for $j \in [1, n]$ such that $R_j = Y(\frac{H(m_j)}{s_j}G + \frac{r_j}{s_j}X)$ and $r_j = f(R_j)$. Then $\mathcal{B}$ forwards $(M_{i_0}^*, \sigma_{i_0}^*)$ as a forgery to $\mathcal{C}$, where $M_{i_0}^*$ has not been queried in the signing oracle. □

## Appendix C
## Formal Security Analysis of PCH

Here, we formally define security properties of our PCH and the proofs for the theorems given in Section IX.

**Definition 6** (Griefing Resistance). *A PCH system achieves griefing resistance if for all PPT senders and receivers, at any time t, the locked coins of the tumbler is no larger than the locked coins to be sent to the tumbler.*

Before proving Theorem 2, we first recall that in our PCH construction, the griefing resistance is enforced as follows: firstly, before the registration phase, the sender $\mathbb{S}$ and the tumbler $\mathbb{T}$ updates their BlindChannel to a conditional payment state, where the payment amount is $m$ committed in the commitment of the amount $\mathsf{C}_{\mathsf{amt}}$. Then during the registration protocol, $\mathbb{S}$ sends the commitment of the token $\mathsf{C}_{\mathsf{tkid}}$ and $\mathsf{C}_{\mathsf{amt}} = \mathsf{com}(m)$ to $\mathbb{T}$. Then, $\mathbb{T}$ generates a randomizable signature $\tilde{\sigma}$ upon $\mathsf{C}_{\mathsf{tkid}}$ and $\mathsf{C}_{\mathsf{amt}}$ to $\mathbb{S}$, who forwards $(\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma}')$, which is the randomized version of $(\mathsf{C}_{\mathsf{tkid}}, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma})$, to the receiver $\mathbb{R}$. Before the puzzle promise protocol, $\mathbb{R}$ and $\mathbb{T}$ updates their BlindChannel to a conditional payment state with payment amount $m$. Later $\mathbb{R}$ shows $(\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma}')$ to $\mathbb{T}$, and generates a token-uniqueness proof to prove that the token committed in $\mathsf{C}'_{\mathsf{tkid}}$ has never been used before.

*Proof of Theorem 2 (Sketch).* Assume there is an adversary $\mathcal{A}$ that can break the griefing resistance of our PCH system. It means that $\mathcal{A}$ is able to lock more coins of $\mathbb{T}$ than those of $\mathbb{S}$. Then, there are three cases:

1) Before the registration phase, $\mathbb{S}$ and $\mathbb{T}$ updates their BlindChannel to a conditional payment state, where the payment amount is $m$. But $m < m'$, where $m'$ is committed in $\mathsf{C}_{\mathsf{amt}}$. But in the channel update phase, $\mathbb{S}$ and $\mathbb{T}$ have reached agreement that the payment amount equals to the amount committed in $\mathsf{C}_{\mathsf{amt}}$. This violates the security of BlindChannel, which guarantees that the BlindChannel is successfully updated only if both parties in the channel agree with the update.
2) $\mathbb{S}$ and $\mathbb{T}$ have updated their BlindChannel to a conditional payment state, where the payment amount is $m$ committed in $\mathsf{C}_{\mathsf{amt}}$. But after receiving a randomizable signature $\tilde{\sigma}$ upon $\mathsf{C}_{\mathsf{tkid}}$ and $\mathsf{C}_{\mathsf{amt}} = \mathsf{com}(m)$, the adversary ($\mathbb{S}$ or $\mathbb{R}$) forges a new randomizable signature $\tilde{\sigma}'$ upon the new commitment of the token $\mathsf{C}_{\mathsf{tkid}} = \mathsf{com}(\mathsf{tkid}')$ and the new commitment of the amount $\mathsf{C}'_{\mathsf{amt}} = \mathsf{com}(m')$, where $m' > m$. This voilates the unforgeability of andomizable signatures on randomizable commitments.
3) In the puzzle promise phase, $\mathbb{R}$ shows $(\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma}')$ to $\mathbb{T}$, and provides a zero knowledge proof to prove the token is unique, and the proof passes the verification of $\mathbb{T}$. But the token is not unique. This violates the soundness of the zero knowledge proof.

□

**Value Privacy Game**: Let $\mathbb{T}$ chooses two payment values $v_0$ and $v_1$ for a payment pair of sender and receiver $(\mathbb{S}, \mathbb{R})$, where the both channels (with $\mathbb{T}$) have sufficient capacities for both values. Let $b \in \{0, 1\}$ be chosen randomly. Let $pay_b$ be the corresponding payment with payment value $v^b$. In case of successful payment of $pay_b$, $\mathcal{A}$ wins the game by guessing the value of $b$: $\Pr_{\mathsf{VP}} := \Pr\left[b' = b : b' \leftarrow \mathcal{A}^{v_0, v_1}, b \xleftarrow{R} \{0, 1\}\right]$.

**Definition 7** (Value Privacy). *A PCH satisfies value privacy if for every PPT tumbler $\mathbb{T}$, the probability of winning Value Privacy Game is $Pr_{\mathsf{VP}} = 1/2 + \epsilon$, where $\epsilon$ is negligible.*

*Proof of Theorem 4 (Sketch).* In our PCH system, the information of the value can either be obtained from $\Pi_{\mathsf{FBCS}}$ or $\Pi_{\mathsf{BC}}$ . If there is an adversary $\mathcal{A}$ ($\mathbb{T}$) that can win the value privacy game, i.e., guess the payment value with more than 1/2 probability, it implies that $\mathcal{A}$ is able to obtain useful information of the value from $\Pi_{\mathsf{FBCS}}$ or $\Pi_{\mathsf{BC}}$ , which violates the blindness of $\Pi_{\mathsf{FBCS}}$ and the security of $\Pi_{\mathsf{BC}}$. □

**Relationship Anonymity Game**: Let $\mathbb{T}$ chooses two candidate senders $\mathbb{S}_0$, $\mathbb{S}_1$ and receivers $\mathbb{R}_0$, $\mathbb{R}_1$. Let $b \in \{0, 1\}$ be chosen randomly. If $b = 0$, then $pay_i = (\mathbb{S}_i, \mathbb{R}_i)$, otherwise $pay_i = (\mathbb{S}_i, \mathbb{R}_{1-i})$ for $i = 0, 1$. Let $M_i$ be the message(s) that $\mathbb{T}$ exchanged (sent and received) with the corresponding parties for the payment pairs $pay_i$ for $i = 0, 1$. In case of simultaneous successful payments of the pairs $pay_0$ and $pay_1$, $\mathbb{T}$ wins the game by guessing the value of $b$: $\Pr_{\mathsf{RA}} := \Pr\left[b' = b : b' \leftarrow \mathbb{T}^{M_0, M_1}, b \xleftarrow{R} \{0, 1\}\right]$.

**Definition 8** (Relationship Anonymity). *A PCH satisfies relationship anonymity if for every PPT tumbler $\mathbb{T}$, the probability of winning the Relationship Anonymity Game is $Pr_{\mathsf{RA}} = 1/2 + \epsilon$ where $\epsilon$ is negligible in $\lambda$.*

*Proof of Theorem 5 (Sketch).* Assume there is an adversary $\mathcal{A}$ ($\mathbb{T}$) that can break the relation anonymity and guess the right payment pairs with probability of $1/2 + \epsilon_0$, where $\epsilon_0$ is non-negligible. It means the adversary can get information of the relationship from either BlindChannel or BlindHub. The only information in BlindChannel can reveal the information of the relationship is the payment amount. If the adversary $\mathcal{A}$ can get information of the payment amount from BlindChannel, it violates the security of $\Pi_{\mathsf{BC}}$. Otherwise, since $\mathbb{T}$ can link the parties with the corresponding promise/solver sessions trivially, it means the adversary $\mathcal{A}$ can link the promise-solver sessions with probability of $1/2 + \epsilon_0$, where $\epsilon_0$ is non-negligible. This violates the blindness of flexible blind conditional signature. □

**Definition 9** (Atomicity)**.** *Suppose $\gamma_s$ and $\gamma_r$ represent the channels shared by $(\mathbb{S}, \mathbb{T})$ and $(\mathbb{T}, \mathbb{R})$, respectively. Let pay$^v$ be the payment of $(\mathbb{S}, \mathbb{R})$ with payment value $v$. A PCH satisfies atomicity if for every PPT sender $\mathbb{S}$, PPT $\mathbb{T}$ and PPT $\mathbb{R}$, for any payment pay$^v$ of $\mathbb{S}$ and $\mathbb{R}$ with any values $v$, the following conditions hold:*

1) *If $\mathbb{S}$ pays $v$ coins in $\gamma_s$, $\mathbb{T}$ pays $v$ coins in $\gamma_r$.*
2) *If $\mathbb{T}$ pays $v$ coins in $\gamma_r$, $\mathbb{S}$ pays $v$ coins in $\gamma_s$.*

*Proof of Theorem 3 (Sketch).* Assume the atomicity of our PCH system is broken. There are two cases: 1) $\mathbb{S}$ pays $\mathbb{T}$ $v$ coins in $\gamma_s$, but $\mathbb{R}$ cannot receive any coin from $\mathbb{T}$ in $\gamma_r$, 2) $\mathbb{T}$ pays $v$ coins in $\gamma_r$, but $\mathbb{S}$ only pays $\mathbb{T}$ $v'$ coins in $\gamma_s$, where $v' < v$. It is noted that since in BlindChannel, the amount is picked by the unblind party($\mathbb{S}/\mathbb{R}$). Assuming the unblind party is rational, $\mathbb{T}$ can only pay $v$ coins or $0$ coins in $\gamma_r$. Below we discuss these two cases separately:

1) $\mathbb{S}$ pays $\mathbb{T}$ $v$ coins in $\gamma_s$, but $\mathbb{R}$ cannot receive any coin from $\mathbb{T}$ in $\gamma_r$: in this case, $\mathbb{T}$ is the adversary $\mathcal{A}$. This means either $\mathcal{A}$ is able to generate a valid signature on a message from the sender that prevents the receiver from being able to unlock the entire signature on the message from the $\mathbb{T}$, or $\mathbb{R}$ cannot still claim coins from the channel even if $\mathbb{R}$ obtain the full signature on the message from the $\mathbb{T}$. The former case implies the unlockability of $\Pi_{\mathsf{FBCS}}$ is broken, and the latter case implies the insecurity of BlindChannel.
2) $\mathbb{T}$ pays $v$ coins in $\gamma_r$, but $\mathbb{S}$ only pays $\mathbb{T}$ $v'$ coins in $\gamma_s$, where $v' < v$: then, $\mathbb{S}$ and $\mathbb{R}$ are both adversaries. This implies either the unforgeability of $\Pi_{\mathsf{FBCS}}$ is broken, or the insecurity of BlindChannel scheme.

□