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

***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.

1. Introduction

Payment Channels (e.g. [3], [17], [46]) 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. [22], [46]) 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) [21], [26], [28], [30], [51] 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 [2], [23], [26], [30], [37], [51][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, [26], [51] 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.

1.1. 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.

1. In existing privacy-preserving PCHs [26], [30], [51], 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 [2], [23], [37] supporting variable values into the PCH model.

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?

1.2. 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 [11] 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 [4]. To analyze the security of BlindHub and inspired by Blind Conditional Signatures (BCS) [26], 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.

1.3. 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 [12], [16], [32], [34], [42], [55], [57], [58]. 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 [8], [10], [24], [31], [39]–[41], [48]–[50], [52], [53], [59]. In the off-chain protocols, BOLT [28] relies on zero-knowledge proofs supported by Zcash, Perun [21], NOCUST [33] and MixCT [20] can only be deployed on Turing complete blockchains (e.g. Ethereum), TeeChain [35] relies on trusted execution environments (e.g. Intel SGX) and Tumblebit [30] and A^2L [51] do not support variable amount payments.

The authors of [30] 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. [51] then introduced A^2L and further gave a formal security notion of synchronization puzzle in the universal composability (UC) framework. The synchronisation puzzle of [51] is more interoperable and efficient compared to the one in TumbleBit. Very recently, Glaeser et al. [26] pointed out that there is a gap in the security model of A^2L, 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 A^2L^+, a modified version of A^2L, and proved that A^2L^+ 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.

2. 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 [23].

System Model. BlindHub, as a payment channel hub (PCH) protocol, is composed of a payment hub (referred

TABLE 1. STATE-OF-THE-ART IN OFF-CHAIN MIXING SERVICES.

	Atomicity	Value Privacy	Relationship Anonymity	Interoperability	Amount Flexibility
BOLT [28]	●	●	●	○ (Zcash)	●
Perun [21]	●	○	○	○ (Ethereum)	●
NOCUST [33]	●	○	○	○ (Ethereum)	●
MixCT [20]	●	●	●	○ (Ethereum)	●
Teechain [35]	●	●	●	◑ (Trusted Hardware)	●
Tumblebit [30]	●	N.A.[b]	●	◑ (HTLC-based currencies)	○
A^2L[a] [51]	●	N.A.[b]	●	● (Digital signature and timelocks)	○
A^2L^+, A^2L^{UC} [26]	●	N.A.[b]	●	● (Digital signature and timelocks)	○
BlindHub	●	●	●	● (Digital signature and timelocks)	●

[a] The model of A^2L is proven to be insecure by [26].
[b] N.A.: not applicable since the amount in these protocols is fixed.

to as Tumbler $\mathbb{T}$) and multiple users, who have established payment channels with $\mathbb{T}$. PCH allows one user (referred to as Sender $\mathbb{S}$) to be able to pay another (referred to as Receiver $\mathbb{R}$) via the Tumbler $\mathbb{T}$. Users have authenticated communication channels with the Tumbler $\mathbb{T}$, 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 one single pair of sender and receiver. It is noted that there might be multiple senders paying multiple receivers at the same time. We assume a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ who can fully control at most two of $\mathbb{S}$, $\mathbb{T}$ or $\mathbb{R}$ for the entire protocol execution. We refer the reader to our supplementary material [1] for more detailed adversary model.

2.1. 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 m coins from $\mathbb{S}$ to $\mathbb{R}$, the PCH should ensure that either $\mathbb{R}$ receives m coins from $\mathbb{T}$ and $\mathbb{T}$ receives 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.

2.2. 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 [51] or Tumblebit [30], 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.

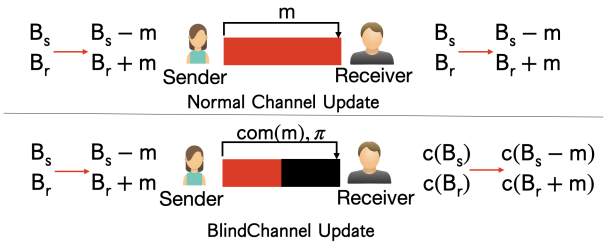

Figure 1. Comparison between a normal channel and a BlindChannel. The black colour represents the party that does not know the balance after the update. $\mathsf{c}(\mathsf{m})$ denotes the commitment of m.

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 B_s and B_r coins. Now one user (sender) wants to pay the other user (receiver) m coins. After the payment, the sender's and receiver's balances become $\mathsf{B}_\mathsf{s} - \mathsf{m}$ coins and $\mathsf{B}_\mathsf{r} + \mathsf{m}$ 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 [15], [43]. 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

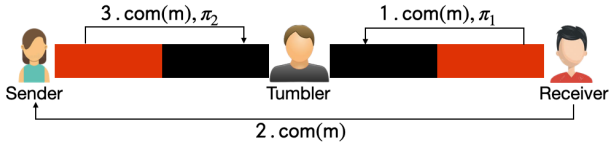

Figure 2. A simple PCH based on BlindChannel.

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 m, com(m), to $\mathbb{S}$ (step 2). On receiving 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^{2}L [47], 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.

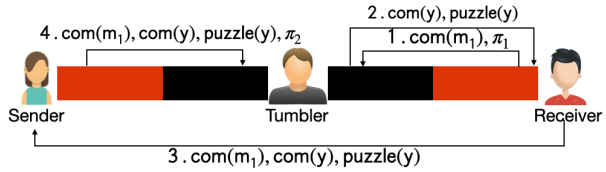

Figure 3. Adding a puzzle to a PCH based on BlindChannel.

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 m_2 which is smaller than 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.

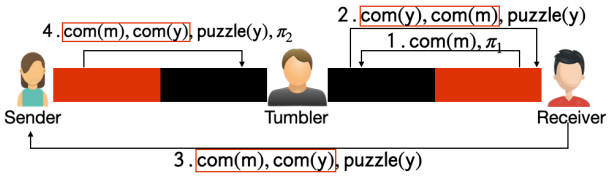

Figure 4. Linking the amount with the puzzle.

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.

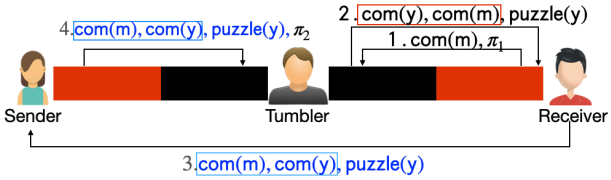

Figure 5. Rerandomize the commitment, the puzzle and the link.

Griefing Resistance: Linking Token with transaction amount. The remaining security goal to achieve is griefing resistance. The idea in A^{2}L [51] 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.

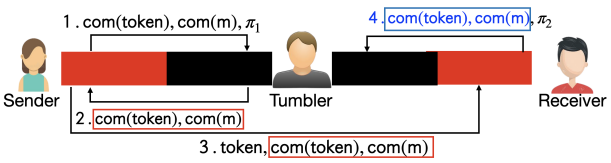

Figure 6. Protocol for the registration phase before initiating the payment. Randomized contents are put in a box with different colours.

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 π_{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 3. Besides, we instantiate the randomizable puzzle as linear-only encryption, which is similar to the approach adopted in A^2L^+ [26]. Also inspired by A^2L and A^2L^+ [26], [51], we leverage adaptor signatures to realize conditional channel update. Detailed descriptions of BlindChannel and BlindHub are provided in Section 5 and 6, respectively.

3. PRELIMINARIES

We denote by 1^λ, 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 λ. 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 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 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 σ composed of two elements. We say a function negl is negligible in λ if it vanishes faster than any polynomial with input λ.

Commitment scheme. Denote message space, randomness space and commitment space as $\mathcal{M}, \mathcal{R}$, and $\mathcal{CM}$, respectively. A commitment scheme Π_{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 GenR that on input 1^λ 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 Π_{NIZK} consists of two PPT algorithms: $\mathsf{P}_{\mathsf{NIZK}}(w, x)$: The prover algorithm that on input a witness w and its statement x, outputs a proof π. $\mathsf{V}_{\mathsf{NIZK}}(x, \pi)$: The verification algorithm that on input the statement x and the proof π, outputs a bit $b \in \{0, 1\}$. The prover can provide a verifier with π 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 Π_{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 λ, outputs a key pair $(\mathsf{ek}, \mathsf{dk})$. $\mathsf{Enc}(\mathsf{ek}, m)$: on input the public key 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 dk and the ciphertext $c \in \mathbb{C}$, outputs a message $m \in \mathbb{M}$. Correctness of a public key encryption scheme Π_{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 [27]. Also, Π_{Enc} is called additively homomorphic if for every $m_1, m_2 \in \mathbb{M}$ and every public key 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) [29] models homomorphic encryption by oracle queries rather than concrete algorithms. The formal description of the oracles modelled by homomorphic encryption can be found in supplementary material [1] (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 Σ usually contains three algorithms: $\mathsf{KeyGen}(1^\lambda)$: inputs a security parameter 1^λ and outputs a secret key/public key pair $(\mathsf{sk}, \mathsf{pk})$. $\mathsf{Sign}_{\mathsf{sk}}(m)$: input a secret key sk and message $m \in \{0, 1\}^*$ and outputs a signature σ. $\mathsf{Vf}_{\mathsf{pk}}(m, \sigma)$: on the input of a public key pk, a signature σ 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 σ is produced using the signing algorithm with input (sk, m), the verification algorithm output 1 on input (pk, σ, 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

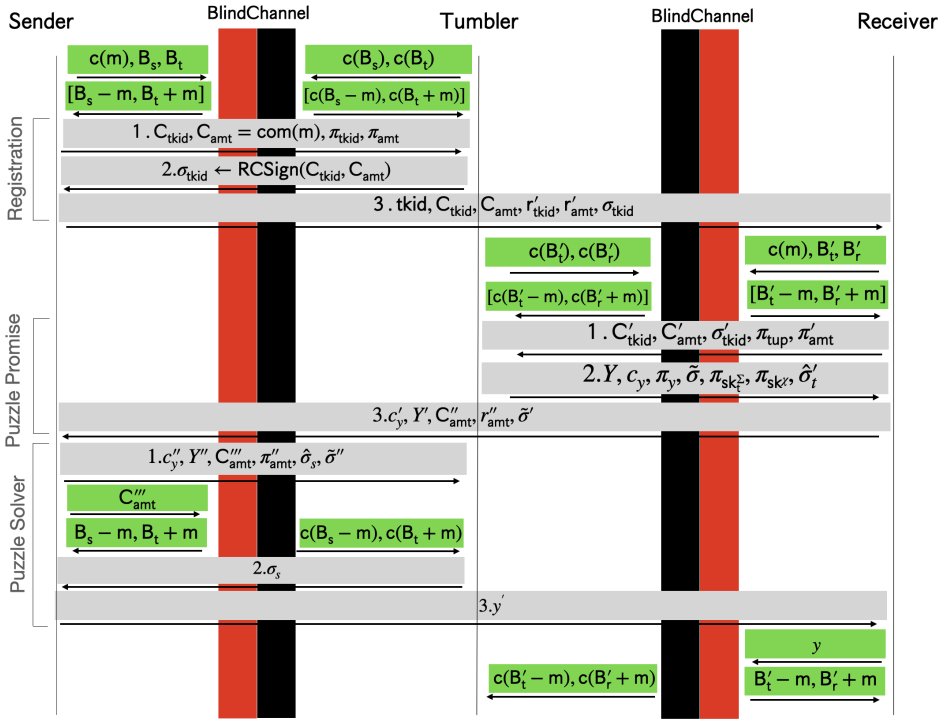

Figure 7. Overview of our solution. Sender pays receiver via tumbler. The overview can be divided into two layers: one layer is the BlindHub protocol, which corresponds to the interactions in the gray box. BlindHub protocol can be divided into three phases: 1) registration, 2) puzzle promise, and 3) 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 6. 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. $\mathsf{B'_s}, \mathsf{B'_t}$ 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.

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 supplementary material [1] (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 [4]. Though in the scheme of [4] 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 supplementary material [1] (Appendix D).

One-more Discrete logarithm (OMDL) problem. OMDL problem [6] 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 supplementary material [1] (Appendix E.2.2).

4. 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 supplementary material [1] (Appendix E.2).

4.1. 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* Π_{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^λ *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* 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 [25].

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 supplementary material [1] (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 Π_{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.

5. 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 2. In the supplementary material [1] (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.

5.1. Security and Privacy Properties

Below we give informal definitions of security and privacy properties required by BlindChannel. Supplementary material [1] (Appendix J.2) provides more details.

The BlindChannel scheme Π_{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.

5.2. 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 [38]. We start by reviewing the generalized channels [3], and then gradually introduce our solution.

Generalized Channel. To create a generalized channel [3], 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 2) 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 supplementary material [1] (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 BlindHub 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 $\mathtt{TX}_{\mathtt{AED}}$ 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 $\mathtt{TX}_{\mathtt{TO}}$, 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 4. Also, U's signature on the timeout transaction is given to the payer without letting him guess the body of the timeout transaction itself.

5.3. 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 [3], i.e. through publishing a funding transaction $\mathtt{TX}_{\mathtt{FU}}$, 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 $\mathtt{TX}_{\mathtt{CM}}$ that sends the channel funds into a new joint address and 2) a split transaction $\mathtt{TX}_{\mathtt{SP}}^{[2]}$ that distributes the channel funds among parties. Fig. 8 summarizes the

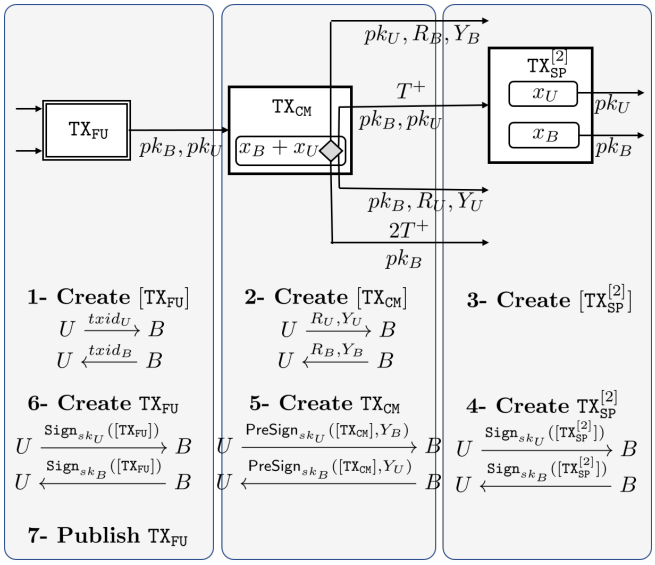

Figure 8. Creation of a Blind Channel

channel creation phase. Following [3], 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.

1) Create $[\mathtt{TX_{FU}}]$: At the first step, B and U create the body of the funding transaction $[\mathtt{TX_{FU}}]$. To do so, they send each other their funding sources.
2) Create $[\mathtt{TX_{CM}}]$: 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 $\mathtt{TX_{FU}}$ and each other's public values, parties create the body of the commit transaction, i.e. $[\mathtt{TX_{CM}}]$.
3) Create $[\mathtt{TX_{SP}^{[2]}}]$: Using the transaction identifier of $\mathtt{TX_{CM}}$, parties create the body of the split transaction $[\mathtt{TX_{SP}^{[2]}}]$.

4-6) Create $\mathtt{TX_{SP}^{[2]}}/\mathtt{TX_{CM}}/\mathtt{TX_{FU}}$: Parties exchange the required signatures to transform $[\mathtt{TX_{SP}^{[2]}}]/[\mathtt{TX_{CM}}]/[\mathtt{TX_{FU}}]$ into $\mathtt{TX_{SP}^{[2]}}/\mathtt{TX_{CM}}/\mathtt{TX_{FU}}$.

7) Publish $\mathtt{TX_{FU}}$: Parties publish $\mathtt{TX_{FU}}$ 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 $[\mathtt{TX_{CM}}]$: The same as step 2 in the create phase.
2) Create $[\mathtt{TX_{SP}^{[3]}}]$: Having the transaction identifier of $\mathtt{TX_{CM}}$, U firstly creates the body of the split transaction $[\mathtt{TX_{SP}^{[3]}}]$, commits it into $\mathsf{com_s}$, and generates a zero knowledge proof π_s to perform amount consistency proof and also prove that $\mathsf{com_s}$ is the commitment on the well-structured split transaction.
3) Create $[\mathtt{TX_{AED}}]$: Having the transaction identifier of $\mathtt{TX_{SP}^{[3]}}$, U generates the body of the adaptor execution delivery (AED) transaction $[\mathtt{TX_{AED}}]$, uses the adaptor statement from the external application (e.g., obtained in the puzzle promise phase of Π_{BH}) to generate a pre-signature $\tilde{\sigma}_a^U$ on $[\mathtt{TX_{AED}}]$, and also generates a zero knowledge proof π_a to prove to B that the pre-signature is on the well-structured delivery transaction $[\mathtt{TX_{AED}}]$. Party U sends the pre-signature $\tilde{\sigma}_a^U$, the hash value $\mathsf{SigHash}([\mathtt{TX_{AED}}])$ and the proof to B.
4) Create $[\mathtt{TX_{TO}}]$: Having the transaction identifier of $\mathtt{TX_{SP}^{[3]}}$, U generates the body of the timeout transaction $[\mathtt{TX_{TO}}]$.
5) Create $\mathtt{TX_{TO}}$: U commits $[\mathtt{TX_{TO}}]$ into $\mathsf{com_t}$, and generates a zero knowledge proof π_t to prove to B that the committed message is the well-structured timeout transaction $[\mathtt{TX_{TO}}]$. After verifying π_t, B generates a blind signature σ_t^B on $\mathtt{TX_{TO}}$ for U.
6) Create $\mathtt{TX_{SP}^{[3]}}$: B generates a blind adaptor signature with Y_P as the adaptor statement on $\mathtt{TX_{SP}}$ for U.
7) Create $\mathtt{TX_{CM}}$: 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 $\mathtt{TX_{AED}}$: B adapts the pre-signature $\tilde{\sigma}_a^U$ into σ_a^U and sends the corresponding witness y_s to U.
10) Create $[\mathtt{TX'_{CM}}]$: the same as step 2 in the create phase.

11-12) Create $[\mathtt{TX_{SP}^{[2]}}]/\mathtt{TX_{SP}^{[2]}}$: Having the transaction identifier of $\mathtt{TX_{CM}}$, U firstly creates the body of the split transaction $[\mathtt{TX_{SP}^{[2]}}]$, commits it into $\mathsf{com_s}$, and generates a zero knowledge proof π_s to perform amount consistency proof and also prove that $\mathsf{com_s}$ is the commitment on the well-structured $\mathtt{TX_{SP}}$. Then, B sends the blind signature on $\mathtt{TX_{SP}^{[2]}}$ to U.

13 Create $\mathtt{TX'_{CM}}$: 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 supplementary material [1] (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 $[\mathtt{TX}_{\mathtt{AED}}]$, adapts U's pre-signature $\tilde{\sigma}_a^U$ into σ_a^U, and finally creates and publishes $\mathtt{TX}_{\mathtt{AED}}$ before a specific timeout or U publishes $\mathtt{TX}_{\mathtt{TO}}$. For the case where U is the payee, either U publishes $\mathtt{TX}_{\mathtt{AED}}$ before a specific timeout or B extracts the split transaction identifier and the payment amount from the published split transaction, creates $\mathtt{TX}_{\mathtt{TO}}$ 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 $\mathtt{TX}_{\mathtt{CM}}$, B uses his own signature in $\mathtt{TX}_{\mathtt{CM}}$ 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 $\mathtt{TX}_{\mathtt{CM}}$'s output.

6. 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 [30], 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.

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

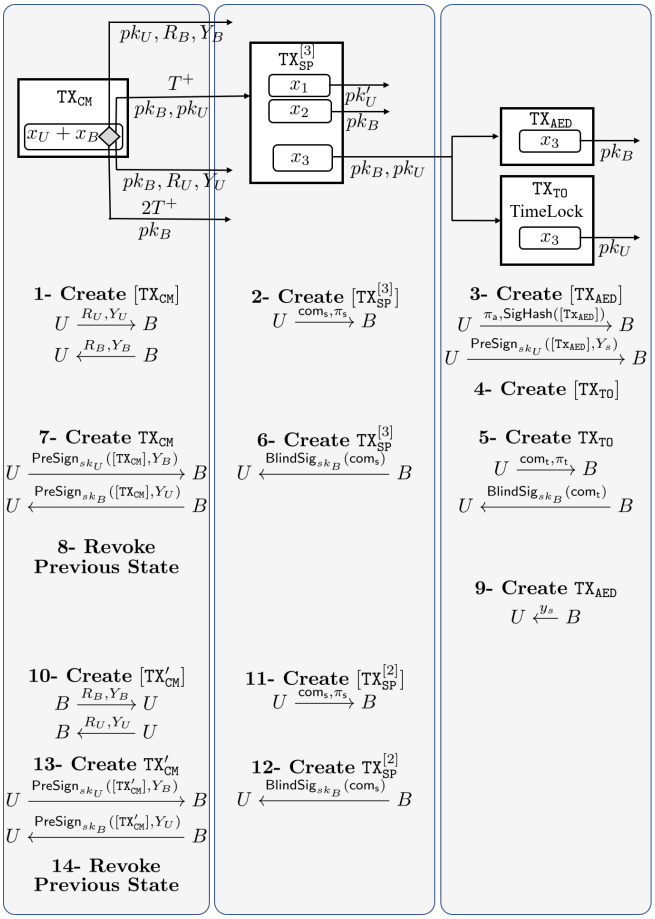

Figure 9. Update of a Blind Channel Initiated by Payer

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

1) On receiving $\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \mathsf{tkid}, r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}$, $\mathbb{R}$ generates a token-uniqueness proof π_{tup} to prove the tkid committed in $\mathsf{C}'_{\mathsf{tkid}}$ has not been used before. In addition, $\mathbb{R}$ generates π'_{amt} to prove knowledge of amt in $\mathsf{C}'_{\mathsf{amt}}$. Then, $\mathbb{R}$ sends $\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}, \pi_{\mathsf{tup}}, \pi'_{\mathsf{amt}}$ to $\mathbb{T}$.
2) $\mathbb{T}$ aborts if one of the followings is not valid: $\sigma'_{\mathsf{tkid}}, \pi_{\mathsf{tup}}$, π'_{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 π_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 $\mathsf{C}_{\mathsf{amt}}$: $\tilde{\sigma} \leftarrow \mathsf{RCSign}(Y, \mathsf{C}_{\mathsf{amt}})$. 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 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 π_y is invalid. Else, $\mathbb{R}$ randomizes $Y, \mathsf{C}'_{\mathsf{amt}}$, $\tilde{\sigma} : (Y', \mathsf{C}''_{\mathsf{amt}}, \tilde{\sigma}', \beta) \leftarrow \mathsf{RCRand}(\mathsf{pp}, Y, \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma})$, and the puzzle $c_y : c'_y \leftarrow \mathsf{PRand}(\beta, c_y)$. Finally, $\mathbb{R}$ sends $c'_y, Y', \mathsf{C}''_{\mathsf{amt}}, \tilde{\sigma}'$ and r''_{amt}, the opening of $\mathsf{C}''_{\mathsf{amt}}$, to $\mathbb{S}$.

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

1) $\mathbb{S}$ firstly randomizes $c'_y, Y', \mathsf{C}''_{\mathsf{amt}}, \tilde{\sigma}'$ received from $\mathbb{R}$ into $c''_y, Y'', \mathsf{C}'''_{\mathsf{amt}}, \tilde{\sigma}''$ to preserve its own anonymity and thwart attacks involving collusion of $\mathbb{T}$ and $\mathbb{R}$. Secondly, $\mathbb{S}$ generates π''_{amt} to prove knowledge of amt in $\mathsf{C}''_{\mathsf{amt}}$. Thirdly, $\mathbb{S}$ generates a proof $\pi_{\mathsf{sk}^{\Sigma}_s}$ to prove that $(\mathsf{sk}^{\Sigma}_s, \mathsf{pk}^{\Sigma}_s)$ is in support of $\Pi_{\mathsf{AS}}.\mathsf{KeyGen}(1^\lambda)$. Then $\mathbb{S}$generates an adaptor signature $\widehat{\sigma}_s$ on the transaction tx' using the randomized adaptor statement $Y'' : \widehat{\sigma}_s \leftarrow \mathsf{PreSign}(\mathsf{sk}^{\Sigma}_s, h', Y'')$, where h' is already proven to be a valid hash value in BlindChannel, as explained before. Finally, $\mathbb{S}$ sends $c''_y, Y'', \mathsf{C}'''_{\mathsf{amt}}, \pi''_{\mathsf{amt}}, \widehat{\sigma}_s, \tilde{\sigma}''$ 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 π''_{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 σ_s, which is then sent to $\mathbb{S}$.
3) $\mathbb{S}$ aborts if the signature σ_s is not valid. Else, $\mathbb{S}$ extracts y'' using the adaptor signature $\widehat{\sigma}_s$ and the valid signature σ_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 σ'_t.

Figures illustrating the above protocols are given in supplementary material [1] (Appendix K).

7. 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 $\mathbb{T}$. 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 $\mathbb{S}$.

8. Flexible Blind Conditional Signature

In a recent work [26], Blind conditional signature (BCS) is proposed to capture the functionality of a synchronization puzzle from [30], [51]. Briefly speaking, synchronization puzzle protocol is a protocol among $\mathbb{S}$, $\mathbb{R}$ and $\mathbb{T}$, where $\mathbb{R}$ and $\mathbb{T}$ execute puzzle solver protocol and generate a puzzle τ, 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 [26] 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 Π_{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 6. 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^λ *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 dk_t, the signing key of the underlying digital signature scheme sk_t^{Σ}, the signing key of* RSoRC *scheme sk^{χ}, and a message m_{TR}) and $\mathbb{R}$ (Receiver) (with inputs the encryption key ek_t, the verification key of the underlying digital signature scheme pk_t^{Σ}, the verification key of* RSoRC *scheme pk^{χ} and a message m_{TR}) and returns $\perp$ to $\mathbb{T}$ and either a puzzle τ 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 sk_s^{Σ}, the encryption key ek_t, the verification key of* RSoRC *scheme pk^{χ}, a message m_{ST}, and a puzzle τ) and $\mathbb{T}$ (Tumbler) (with inputs the decryption key dk_t, the verification key of the underlying digital signature scheme pk_s^{Σ}, the signing key sk^{χ} of Π_{RSoRC} and a message m_{ST}) and returns to both users either a signature σ^* ($\mathbb{S}$ additionally receives a secret s) or $\perp$.*
- $\{\sigma, \perp\} \leftarrow \mathsf{Open}(\tau, s)$: *The open algorithm takes as input a puzzle τ and a secret s and returns a signature σ 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 supplementary material(Appendix F) .

Π_{FBCS} is **correct** if $\mathbb{S}$ and $\mathbb{R}$ are able to obtain the valid signatures on m_{ST} and m_{TR}, respectively after the BlindHub protocol is successfully completed. Π_{FBCS} is **blind** if $\mathbb{T}$ cannot link two promise/solve sessions together. Π_{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. Π_{FBCS} is **unforgeable** if $\mathbb{R}$ could not output a valid signature on m_{TR} before $\mathbb{S}$ successfully complete the solver protocol with $\mathbb{T}$. Π_{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 supplementary material (Appendix G).

Theorem 1. *Let Π_{Enc} be a linear-only homomorphic encryption scheme, Π_{AS} is a secure adaptor signature scheme, Π_{BAS} is a secure BAS scheme, Π_{RSoRC} is a secure signature on randomizable commitments scheme, Π_{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.*

9. 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 Π_{RSoRC}, the biding property of Π_{COM}, the security of Π_{BC} and the soundness of Π_{NIZK}.

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

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 Π_{FBCS} and the privacy of Π_{BC}.

Theorem 3. *Assuming the blindness of Π_{FBCS} and security of the Π_{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 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 4. *Assuming the blindness of Π_{FBCS} and security of Π_{BC}, our PCH system achieves relationship anonymity.*

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 Π_{FBCS} or the security of Π_{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 Π_{FBCS} or the security of Π_{BC}.

Theorem 5. *Assuming the unlockability and unforgeability of Π_{FBCS} and the security of Π_{BC}, our PCH system achieves atomicity.*

10. Performance Analysis

Implementation details. To benchmark the performance of our protocol, we implemented[2] our protocol with Rust programming language, Swanky[3] multiparty computation library, and ZenGo-X/curv[4] 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[5] 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 [4] over the BLS12-381 curve[6]. The linear-only encryption scheme is instantiated using HSM-CL encryption scheme [14]. We instantiate the commitment scheme as Pedersen commitment scheme [44]. We instantiate BlindChannel in the Bitcoin setting, and we instantiate the proof used in Bitcoin-based BlindChannel as garbled circuit-based zero knowledge proof [15], [43]. Concrete instantiation of our whole protocol can be found in the supplementary material [1] (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[7]. 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 supplementary material [1] (Appendix H).

Computation time and communication overhead. The benchmark for the GCZK is taken on a Ryzen 5800H, 8GB RAM laptop computer, where the test data is generated from Bitcoin regtest network[8]. In the local area network (LAN) setting, performing this GCZK itself consumes 11.482 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. We show the computation time and communication overhead by phases in Table 2. 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 2.

Role	Bandwidth (KB)	Comp. (ms)	Optimized Comp. (ms)
Sender	55843	32670	3525
Tumbler	87855	56502	10006
Receiver	32017	25994	8643
Phase	**Bandwidth (KB)**	**Comp. (ms)**	**Optimized Comp. (ms)**
Register	23832	12958	1164
Promise	23834	19915	7714
Solver	23831	14496	2295
Open	16357	10457	157
Total	87860	57751	11330

TABLE 2. BANDWIDTH OVERHEAD AND COMPUTATION COST. THE OPTIMIZED COST IS OUR ESTIMATION OF THE PROTOCOL WITH [56].

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 2 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., [5], [18], [54], [56]) can provide better efficiency. For example, Quicksilver [56] 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 2.

Comparing performance with A^{2}L versions. A^{2}L protocol completes in 3 seconds regarding the implementation results given by the authors [51]. However, as pointed out in [19], the CL encryption parameters[9] chosen in A^{2}L may not be sufficient for security reasons. When, we use the parameters suggested in [19], 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.

11. 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.

2. https://github.com/blind-channel/blind-hub
3. https://github.com/GaloisInc/swanky
4. https://github.com/ZenGo-X/curv
5. https://github.com/YosysHQ/yosys
6. 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 supplementary material [1] (Appendix K).
7. https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki
8. https://developer.bitcoin.org/examples/testing.html
9. 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 [19].

References

[1] "Supplementary material to BlindHub." [Online]. Available: https://github.com/blind-channel/blind-hub

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] O. Blazy, G. Fuchsbauer, D. Pointcheval, and D. Vergnaud, "Signatures on randomizable ciphertexts," in *International Workshop on Public Key Cryptography*. Springer, 2011, pp. 403–422.

[10] 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.

[11] 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.

[12] 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.

[13] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Two-party ecdsa from hash proof systems and efficient instantiations," in *Annual International Cryptology Conference*. Springer, 2019, pp. 191–221.

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

[15] 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.

[16] 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.

[17] 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.

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

[19] 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

[20] 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

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

[36] Y. Lindell, "Fast secure two-party ecdsa signing," in *Annual International Cryptology Conference*. Springer, 2017, pp. 613–644.

[37] 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.

[38] 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.

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

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

[41] 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.

[42] 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.

[43] 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.

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

[45] A. Poelstra, "Scriptless scripts," https://www.sosthene.net/wp-content/uploads/wpforo/default_attachments/1562671969-Poelstra-Scriptless-Scripts.pdf.

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

[47] 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.

[48] 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.

[49] 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.

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

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

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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 [51], 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 2.1.

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 wants 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 $\mathbb{R}$ corrupted, he can send a fake puzzle to $\mathbb{S}$, and any 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. When the $\mathbb{S}$ is corrupted, $\mathbb{T}$ can explorer who is the $\mathbb{R}$ by asking $\mathbb{S}$ to pay $\mathbb{R}$, whose incoming balance is exhausted, so $\mathbb{R}$ is not a possible receiver in the current period, as $\mathbb{T}$ can see. We claim that since BlindHub is built upon BlindChannel and both parties' balances within BlindChannel is hidden from $\mathbb{T}$, it is naturally prevent from the *Tumbler-Sender* collusion attack. Thus when $\mathbb{S}$ or $\mathbb{R}$ colludes with $\mathbb{T}$, *value privacy* and *relationship anonymity* can be compromised.

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.

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.

B.1. 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 4 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* Π_{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_{S} and O_{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.

$\mathsf{wBlindness}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda)$

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 : **return** $(b == b^*)$

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

$\mathsf{baWitExt}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda)$

1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
2 : $(M^*, Y) \leftarrow \mathcal{A}^{O'_{\mathsf{S}}, O'_{\mathsf{pS}}}(\mathsf{pk})$
3 : $\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, Y) \leftrightarrow \mathsf{User}(\mathsf{pk}, M^*, Y)$
4 : $\sigma^* \leftarrow \mathcal{A}^{O_{\mathsf{S}}, O_{\mathsf{pS}}}(\hat{\sigma}, Y)$
5 : $y^* \leftarrow \mathsf{Ext}(Y, \sigma^*, \hat{\sigma})$
6 : **return** $((Y, y^*) \notin R \wedge \mathsf{Vf}(\mathsf{pk}, \sigma^*, M^*) = 1)$

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

Definition 4 (Witness Extractability)**.** *A* BAS *scheme* Π_{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* Π_{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,* $\Pr[\mathsf{omaSignForge}_{\mathcal{A},\Pi_{\mathsf{BAS}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

B.2. 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 [25], [47] 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 λ, 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^λ, signer's public key X and secret key x, adaptor statement Y, adaptor statement y, a and the proof of knowledge π of the witness of Y. The user's input includes security parameter 1^λ, 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 π_h of the pre-image of h. Then the user sends h, π_h to the signer.

2) On receiving h, π_h, the signer verifies the validity of the proof π_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 π_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, π_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 [3]. 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 [7].

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 param, adaptor statement Y and pk to $\mathcal{B}$. $\mathcal{B}$ forwards param, Y and pk to $\mathcal{A}$, and $\mathcal{A}$ outputs $(M_1^*, \ldots, M_n^*)$. When $\mathcal{A}$ queries the signing oracle with h_i, π_{h_i} as input, $\mathcal{B}$ first extract the pre-image M_i^* of h_i from π_{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 9.

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 $\mathsf{C_{amt}}$. Then during the registration protocol, $\mathbb{S}$ sends the commitment of the token $\mathsf{C_{tkid}}$ and $\mathsf{C_{amt}} = \mathsf{com}(m)$ to $\mathbb{T}$. Then, $\mathbb{T}$ generates a randomizable signature $\tilde{\sigma}$ upon $\mathsf{C_{tkid}}$ and $\mathsf{C_{amt}}$ to $\mathbb{S}$, who forwards $(\mathsf{C'_{tkid}}, \mathsf{C'_{amt}}, \tilde{\sigma}')$, which is the randomized version of $(\mathsf{C_{tkid}}, \mathsf{C_{amt}}, \tilde{\sigma})$, 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 $(\mathsf{C'_{tkid}}, \mathsf{C'_{amt}}, \tilde{\sigma}')$ to $\mathbb{T}$, and generates a token-uniqueness proof to prove that the token committed in $\mathsf{C'_{tkid}}$ 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 $\mathsf{C_{amt}}$. But in the channel update phase, $\mathbb{S}$ and $\mathbb{T}$ have reached agreement that the payment amount equals to the amount committed in $\mathsf{C_{amt}}$. 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 $\mathsf{C_{amt}}$. But after receiving a randomizable signature $\tilde{\sigma}$ upon $\mathsf{C_{tkid}}$ and $\mathsf{C_{amt}} = \mathsf{com}(m)$, the adversary ($\mathbb{S}$ or $\mathbb{R}$) forges a new randomizable signature $\tilde{\sigma}'$ upon the new commitment of the token $\mathsf{C_{tkid}} = \mathsf{com}(\mathsf{tkid}')$ and the new commitment of the amount $\mathsf{C'_{amt}} = \mathsf{com}(m')$, where $m' > m$. This voilates the unforgeability of andomizable signatures on randomizable commitments.
3) In the puzzle promise phase, $\mathbb{R}$ shows $(\mathsf{C'_{tkid}}, \mathsf{C'_{amt}}, \tilde{\sigma}')$ 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 ϵ is negligible.*

Proof of Theorem 3 (Sketch). In our PCH system, the information of the value can either be obtained from Π_{FBCS} or Π_{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 Π_{FBCS} or Π_{BC} , which violates the blindness of Π_{FBCS} and the security of Π_{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 ϵ is negligible in λ.*

Proof of Theorem 4 (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 ϵ_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 Π_{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 ϵ_0 is non-negligible. This violates the blindness of flexible blind conditional signature. □

Definition 9 (Atomicity)**.** *Suppose γ_s and γ_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 γ_s, $\mathbb{T}$ pays v coins in γ_r.
2) If $\mathbb{T}$ pays v coins in γ_r, $\mathbb{S}$ pays v coins in γ_s.

Proof of Theorem 5 (Sketch). Assume the atomicity of our PCH system is broken. There are two cases: 1) $\mathbb{S}$ pays $\mathbb{T}$ v coins in γ_s, but $\mathbb{R}$ cannot receive any coin from $\mathbb{T}$ in γ_r, 2) $\mathbb{T}$ pays v coins in γ_r, but $\mathbb{S}$ only pays $\mathbb{T}$ v' coins in γ_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 γ_r. Below we discuss these two cases separately:

1) $\mathbb{S}$ pays $\mathbb{T}$ v coins in γ_s, but $\mathbb{R}$ cannot receive any coin from $\mathbb{T}$ in γ_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 Π_{FBCS} is broken, and the latter case implies the insecurity of BlindChannel.
2) $\mathbb{T}$ pays v coins in γ_r, but $\mathbb{S}$ only pays $\mathbb{T}$ v' coins in γ_s, where $v' < v$: then, $\mathbb{S}$ and $\mathbb{R}$ are both adversaries. This implies either the unforgeability of Π_{FBCS} is broken, or the insecurity of BlindChannel scheme.

□

Appendix D. Construction of Randomizable Signatures on Randomizable Commitments

D.1. Preliminaries

Bilinear groups. We assume the existence of a probabilistic polynomial-time (p.p.t.) algorithm BGGen that takes as input an integer λ in unary and outputs a description of an (asymmetric) bilinear group $\left(p, \mathbb{G}, G, \hat{\mathbb{G}}, \hat{G}, \mathbb{G}_T, e\right)$ consisting of groups $(\mathbb{G}, +)$ and $(\hat{\mathbb{G}}, +)$, generated by G and $\hat{G}$, resp., and $(\mathbb{G}_T, \cdot)$, all of cardinality a prime number $p \in \left\{2^\lambda, \ldots, 2^{\lambda+1}\right\}$, and a bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$, such that $e(G, \hat{G})$ generates $\mathbb{G}_T$, called pairing.

The *decisional Diffie-Hellman assumption* for BGGen states that no p.p.t. adversary $\mathcal{A}$ can distinguish a triple (dG, rG, drG) for $d, r \leftarrow \mathbb{Z}_p$ from a random triple from $\mathbb{G}^3$ with better than negligible advantage.

Below we firstly give the definition of RSoRC and then give the security definition of RSoRC. Finally, we give a concrete instantiation of RSoRC.

We first give the definition of randomizable commitment and randomizable signatures, following the definition of randomizable ciphertext given in [9]. It is noted that our definition of randomizable signatures is slightly different from the one given in [9] in the way that we remove the part of hash the message m and the corresponding proof of knowledge of m.

Definition 10 (Randomizable Commitment Scheme)**.** *Let Π_{COM} be a commitment scheme with the following additional algorithm:*

- $\mathsf{RndCM}(\mathsf{C}, r')$ *input commitment* C*, using the additional random coins $r' \in \mathcal{R}$, output a new commitment* C'*,*

A commitment scheme is called randomizable if for any $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$*, message $m \in \mathcal{M}$, randomness $r \in \mathcal{R}$, and commitment* $\mathsf{C} = \mathsf{com}(m; r)$*, the following distributions are statistically indistinguishable:* $\mathsf{D}_0 = \{r' \leftarrow\$ \mathcal{R} : \mathsf{com}(m; r')\}$ *and* $\mathsf{D}_1 = \{r' \leftarrow\$ \mathcal{R} : \mathsf{RndCM}(\mathsf{C}; r')\}$

Definition 11 (Randomizable Signature Scheme)**.** *Let* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vf})$ *be a signature scheme, with the following algorithm:*

- $\mathsf{RCRand}(\mathsf{pk}, m, \sigma; r')$: *input message m, signature σ, using the additional randomness $r' \in \mathcal{R}$, outputs a new signature σ.*

A signature scheme is called *randomizable* if for any $\mathsf{param} \leftarrow \mathsf{Setup}(1^\lambda)$, message $m \in \mathcal{M}$, randomness $r \in \mathcal{R}$, and signatures $\sigma = \mathsf{Sign}(m; r)$, the following distributions are statistically indistinguishable: $\mathsf{D}_0 = \{r' \leftarrow\$ \mathcal{R} : \mathsf{Sign}(m; r')\}$ and $\mathsf{D}_1 = \{r' \leftarrow\$ \mathcal{R} : \mathsf{RCRand}(\mathsf{pk}, m, \sigma; r')\}$. We now combine the randomizable signatures and randomizable commitments:

Definition 12 (Randomizable Signatures on randomizable commitments)**.** *A scheme of randomizable signatures on randomizable commitments $\Pi_{\mathsf{RSoRC}} = (\mathsf{Setup}, \mathsf{SKeyGen}, \mathsf{RndCM}, \mathsf{RCSign}, \mathsf{RCRand}, \mathsf{RCVerify})$ with a message space $\mathcal{M}$, a randomness space $\mathcal{R}$, a commitment space $\mathcal{CM}$, a signature space $\mathcal{SIG}$ consists of six algorithms defined as:*
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: *a* PPT *algorithm that on input security parameter 1^λ, outputs public parameters* pp.
$(\mathsf{sk}^\chi, \mathsf{pk}^\chi) \leftarrow \mathsf{SKeyGen}(\mathsf{pp})$: *a* PPT *algorithm that on input public parameters* pp*, outputs signing key/verification key pair* $(\mathsf{sk}, \mathsf{vk})$.
$\mathsf{CM} \leftarrow \mathsf{com}(m)$: *is a* PPT *algorithm that on input public parameters* pp *and a message m, outputs a randomizable commitment* CM.
$\mathsf{CM}' \leftarrow \mathsf{RndCM}(\mathsf{CM}, r)$: *a* PPT *algorithm that on input a randomization factor r and a commitment* CM*, outputs a randomized commitment* CM'.
$\sigma \leftarrow \mathsf{RCSign}_{\mathsf{sk}^\chi}(\mathsf{CM}_1, ..., \mathsf{CM}_n)$: *input a secret key* sk^χ *and a set of randomizable commitment* $\mathsf{CM}_1, ..., \mathsf{CM}_n$*, output a signature σ.*
$(\mathsf{CM}'_1, ..., \mathsf{CM}'_n, \sigma') \leftarrow \mathsf{RCRand}(\mathsf{CM}_1, ..., \mathsf{CM}_n, \sigma, r)$: *a* PPT *algorithm that on input a randomization factor r, a set of randomizable commitments* $\mathsf{CM}_1, ..., \mathsf{CM}_n$ *and a signature σ, outputs randomized commitments* $\mathsf{CM}'_1, ..., \mathsf{CM}'_n$ *and a randomized signature σ'.*
$0/1 \leftarrow \mathsf{RCVerify}(\mathsf{pk}^\chi, \sigma, \mathsf{CM}_1, ..., \mathsf{CM}_n)$: *a* DPT *algorithm that on input public key* pk*, a set of randomizable commitment* $\mathsf{CM}_1, ..., \mathsf{CM}_n$ *and a signature σ, outputs 1 if the signature σ is valid or 0 otherwise.*

We define the equivalence class $[\mathsf{CM}_1, \cdots, \mathsf{CM}_n]$ of a set of commitments $\mathsf{CM}_1, \cdots, \mathsf{CM}_n$ as all randomizations of $\mathsf{CM}_1, \cdots, \mathsf{CM}_n$, i.e., $[\mathsf{CM}'_1, \cdots, \mathsf{CM}'_n] := \{(\mathsf{CM}_1, \cdots, \mathsf{CM}_n) | \exists r \in \mathcal{R} : \mathsf{CM}'_i = \mathsf{RndCM}(\mathsf{CM}_i, r), \forall i \in [n]\}$.

D.2. Correctness and Security Definitions

Correctness of RSoRC requires that the commitment scheme and the signature scheme are correct.

Definition 13. *A* RSoRC *scheme is correct if for all $pp \in \mathcal{PP}$, for all pairs $(\mathsf{sk}^\chi, \mathsf{pk}^\chi)$ in the range of $\mathsf{SKeyGen}(pp)$, respectively, and all $m_i, i = 1, \cdots, n \in \mathcal{M}_{pp}, r_i, i = 1, \cdots, n \in \mathcal{R}_{pp}$ and $\mathsf{CM}_i, i = 1, \cdots, n \in \mathcal{CM}_{pp}$:*

$$\mathsf{Decom}(r, \mathsf{Com}(m_i, r_i)) = m_i, \quad \text{and}$$
$$\sigma = \mathsf{RCSign}(\mathsf{sk}^\chi, \mathsf{CM}_1, \cdots, \mathsf{CM}_n)$$
$$\Pr[\mathsf{Verify}(\mathsf{pk}^\chi, \mathsf{CM}_1, \cdots, \mathsf{CM}_n, \sigma) = 1] = 1.$$

Note that together with signature-adaptation (Def. below), this implies that adapted signatures verify as well.

Class-Hiding states that given a representative of an equivalence class, then a random member of that class is indistinguishable from a random element of the whole space. We give a stronger definition, which we call fully class-hiding (analogously to full anonymity).

Definition 14. *Let game* CL-HID *be as defined in Fig. 13. A* RSoRC *scheme is fully class-hiding if for all p.p.t. adversary $\mathcal{A}$, the following function is negligible in λ:* $\left|\Pr\left[\mathbf{CL} - \mathbf{HID}^{\mathcal{A}}_{\mathsf{RSoRC}}(\lambda) = 1\right]\right|$.

Signature-randomizability requires that signatures that have been randomized are distributed like fresh signatures on the randomized commitments.

Definition 15. *A* RSoRC *scheme is signature-randomizable (under malicious keys) if for all $pp \in \mathcal{PP}$, all $(\mathsf{pk}^\chi, \mathsf{CM}_1, \cdots, \mathsf{CM}_n, \sigma) \in \mathcal{VK}_{pp} \times \mathsf{CM}^n_{pp} \times \mathcal{S}_{pp}$ that satisfy* $\mathsf{Verify}(\mathsf{pk}^\chi, \mathsf{CM}_1, \cdots, \mathsf{CM}_n, \sigma) = 1$ *and for all $r \in \mathcal{R}_{pp}$, the output of* $\mathsf{RCRand}(\sigma, r)$ *is uniformly distributed over the set*

$$\{\sigma' \in \mathcal{S}_{pp} \mid$$
$$\mathsf{RCVerify}\left(\mathsf{pk}^\chi, \mathsf{RndCM}(\mathsf{CM}_1, r), \cdots, \mathsf{RndCM}(\mathsf{CM}_n, r), \sigma'\right) = 1\}.$$

Unforgeability. We present our notion of unforgeability, which is defined w.r.t. equivalence classes. That is, after the adversary queries a signature for $(\mathsf{CM}_1, \mathsf{CM}_2)$, all tuples $(\mathsf{CM}_1, \mathsf{CM}_2)$ with $\mathsf{CM}'_1, \mathsf{CM}'_2 \in [\mathsf{CM}_1, \mathsf{CM}_2]$ are added to a set S of signed objects. The adversary's goal is to produce a signature on a pair $(\mathsf{CM}^\star_1, \mathsf{CM}^\star_2)$ that is not contained in S.

Definition 16 (EUF-CMA)**.** *Let* **UNF** *be the game defined in Fig.14. A* RSoRC *scheme is unforgeable if for all PPT adversary $\mathcal{A}$ the following function is negligible in λ:* $\Pr[\mathsf{UNF}^{\mathcal{A}}_{\mathsf{RSoRC}}(\lambda) = 1]$.

D.3. Construction

Below we give the construction of randomizable signature on randomizable commitment (RSoRC). Our construction is almost the same as the generalized scheme in [4](the generalized case is discussed at the end of the paper), except that we replace all the encryption keys P_i with generators H_i. This is correct, since for a user who does not know the secret key sk, the encryption public key $P = sk \cdot G$ is the same as a normal group element. Thus, the security of our construction directly follows from the security of the generalized case of their scheme.
$\mathsf{Setup}(1^\lambda)$: Define $\mathcal{M}_{pp} := \mathbb{G}, \mathcal{CM}_{pp} := \mathbb{G}$, $\mathcal{R}_{pp} := \mathbb{Z}_p, \mathcal{SK}_{pp} := \left(\mathbb{Z}^*_p\right)^n, \mathcal{VK}_{pp} := \left(\hat{\mathbb{G}}^*\right)^n$ and $\mathcal{S}_{pp} := \mathbb{G} \times \mathbb{G}^* \times \hat{\mathbb{G}}^* \times \mathbb{G}$. Return

$pp = \left(p, \mathbb{G}, G, H_i, i = 1, \cdots, n \in \mathbb{G}, \hat{\mathbb{G}}, \hat{G} \in \hat{\mathbb{G}}, \mathbb{G}_T, e\right) \leftarrow\$ \mathsf{BGGen}\left(1^\lambda\right)$
$\mathsf{KeyGen}(pp)$: pick $x_i, i = 1, \cdots, n \leftarrow\$ \mathbb{Z}_p^*$ and set $\hat{Y}_i = x_i\hat{G}, i = 1, \cdots, n$. Output $(\mathsf{sk} = (x_i, i = 1, \cdots, n), \mathsf{vk} = (\hat{Y}_i, i = 1, \cdots, n)$.
$\mathsf{GenCOM}(m, G, H)$: pick $t \leftarrow\$ \mathbb{Z}_p$, return $\mathsf{CM} = mG + tH$.
$\mathsf{RndCM}(\mathsf{CM} = mG + tH)$: pick $\mu \leftarrow\$ \mathbb{Z}_p$, return $\mathsf{CM}' = mG + tH + \mu H$.
$\mathsf{RCSign}(\mathsf{CM}_0, \cdots, \mathsf{CM}_n, \mathsf{sk}^\chi)$: Given $\mathsf{CM}_0, \cdots, \mathsf{CM}_n \in \mathbb{G}$ of class $[\mathsf{CM}_i], i = 0, \cdots, n$ and secret key $\mathsf{sk}^\chi = (x_0, \cdots, x_n)$, pick $v \leftarrow\$ \mathbb{Z}_p$, output $\sigma = (A, V, \hat{V}, T)$, with $A = \frac{1}{v}(G + \sum_0^n x_i\mathsf{CM}_i), V = vG, \hat{V} = v\hat{G}$, $T = \frac{1}{v}\left(x_0G + \sum_1^n x_iH_i\right)$.
$\mathsf{RCRand}(\mathsf{CM}_0, \cdots, \mathsf{CM}_n, \sigma, \mu, \mathsf{pk}^\chi)$: Given representative $\mathsf{CM}_0, \cdots, \mathsf{CM}_n \in \mathbb{G}, \sigma = (A, V, \hat{V}, T)$, scalar $\mu \in \mathbb{Z}_p^*$ and pk^χ, pick $\psi \leftarrow \mathbb{Z}_p^*$ and return $\left(\mathsf{CM}_i' = \mathsf{CM}_i + \mu H_i, \sigma'\right)$ with $\sigma' \leftarrow (A' = \frac{1}{\psi}(A + \mu T), V' = \psi V, \hat{V}' = \psi\hat{V}, T' = \frac{1}{\psi}T)$.
$\mathsf{RCVerify}(\sigma, \mathsf{CM}_1, \cdots, \mathsf{CM}_n)$: Given representative CM_1, $\cdots, \mathsf{CM}_n \in \mathbb{G}, \sigma = (A, V, \hat{V}, T)$, return 1 if the following conditions hold and 0 otherwise: $e(A, \hat{V}) = e(G, \hat{G}) \prod_{i=0}^n e(\mathsf{CM}_i, \hat{Y}_i) \wedge e(V, \hat{G}) = e(G, \hat{V}) \wedge e(T, \hat{V}) = e(G, \hat{Y}_0)e\prod_{i=1}^n (H_i, \hat{Y}_i)$.

We define the equivalence class $[\mathsf{CM}_1, \cdots, \mathsf{CM}_n]$ of a set of commitment $\mathsf{CM}_i, i = 1, \cdots, n$ as all randomizations of $(\mathsf{CM}_1, \cdots, \mathsf{CM}_n)$:

$$[\mathsf{CM}_1, \cdots, \mathsf{CM}_n] := \{(\mathsf{CM}_1', \cdots, \mathsf{CM}_n') \mid \exists \mu \in \mathcal{R}_{pp} : \mathsf{CM}_i' = \mathsf{RndCM}(\mathsf{CM}_i, \mu)\}.$$

Appendix E. Additional Preliminaries

E.1. Adaptor Signatures

Adaptor signature was firstly introduced as a mean to exercute smart contracts off-chain by Andrew Poelstra [45]. It is also called "scriptless script". Essentially, adaptor signature takes two steps to simulate a contract that authorizes the completeness of a signature with leakage of a secret value: firstly, an "incomplete" signature is generated such that it can only be completed by someone knowing a certain secret. Later this signature is completed, and the secret will be revealed. Here we give a more detailed and formal description of adaptor signatures, following the definitions and security experiments from [3].

Definition 17 (Adaptor signature scheme)**.** *An adaptor signature scheme, w.r.t. a hard relation R and a signature scheme $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of four algorithms $\Xi_{R,\Sigma} = (\mathsf{PreSign}, \mathsf{Adapt}, \mathsf{PreVf}, \mathsf{Ext})$ with the following syntax: $\mathsf{PreSign}(\mathsf{sk}, m, Y)$ is a PPT algorithm that on input a secret key sk, message $m \in \{0,1\}^*$ and statement $Y \in L_R$ outputs a pre-signature $\tilde{\sigma}$. $\mathsf{PreVf}(\mathsf{pk}, m, Y, \hat{\sigma})$ is a DPT algorithm that on input a public key pk, message $m \in \{0,1\}^*$, statement $Y \in L_R$ and pre-signature $\hat{\sigma}$ outputs a bit b. $\mathsf{Adapt}(\hat{\sigma}, y)$ is a DPT algorithm thaton input a pre-signature $\tilde{\sigma}$ and witness y , outputs a signature σ; and $\mathsf{Ext}(\sigma, \hat{\sigma}, Y)$ is a DPT algorithm that on input a signature σ, pre-signature $\tilde{\sigma}$ and statement $Y \in L_R$, outputs witness y, s.t., $(Y, y) \in R$, or $\perp$.*

An adaptor signature scheme $\Xi_{R,\Sigma}$ must satisfy pre-signature correctness stating that for every $m \in \{0,1\}^*$ and every $(Y, y) \in R$, the following holds:

$$\Pr\left[\begin{array}{c|l} \mathsf{PreVf}_{\mathsf{pk}}(m, Y; \tilde{\sigma}) = 1 & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}\left(1^\lambda\right) \\ \wedge & \tilde{\sigma} \leftarrow \mathsf{PreSign}_{\mathsf{sk}}(m, Y) \\ \mathsf{Vf}_{\mathsf{pk}}(m; \sigma) = 1 & \sigma \leftarrow \mathsf{Adapt}_{\mathsf{pk}}(\tilde{\sigma}, y) \\ \wedge & y' := \mathsf{Ext}_{\mathsf{pk}}(\sigma, \tilde{\sigma}, Y) \\ (Y, y') \in R & \end{array}\right] = 1.$$

Definition 18 (Existential unforgeability)**.** *An adaptor signature scheme $\Xi_{R,\Sigma}$ is $\mathsf{aEUF\text{–}CMA}$ secure if for every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function ν such that: $\Pr\left[\mathsf{aSigForge}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1\right] \leq \nu(\lambda)$, where the experiment $\mathsf{aSigForge}_{\mathcal{A},\Xi_{R,\Sigma}}$ is defined as follows:*

Definition 19 (Pre-signature Adaptability)**.** *An adaptor signature scheme $\Xi_{R,\Sigma}$ satisfies pre-signature adaptability if for every message M in the message space, every statement/witness pair $(Y, y) \in L_R$ and for all pre-signature $\hat{\sigma}$, any key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ with $\mathsf{PreVerify}(Y, \mathsf{pk}, \hat{\sigma}, M) = 1$ and $\sigma \leftarrow \mathsf{Adapt}((Y, y), \mathsf{pk}, \hat{\sigma}, M)$, we have $\mathsf{Verify}(\mathsf{pk}, \sigma, M) = 1$.*

Definition 20 (Witness Extractability)**.** *An adaptor signature scheme $\Xi_{R,\Sigma}$ is witness extractable if for every PPT adversary $\mathcal{A}$ running the experiment $\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}$, $\Pr[\mathsf{aWitExt}_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.*

E.2. Blind Signature

Following the definition of [25], we give the formal security notions as follows:

E.2.1. Security Notions. A blind signature scheme BS consists of the following algorithms:

- $\mathsf{BS.Setup}(1^\lambda)$: It takes the security parameter λ and returns public parameters param.
- $\mathsf{BS.KeyGen}(\mathsf{param})$: It takes the public parameters param and returns a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$.
- $(b, \sigma) \leftarrow \langle \mathsf{BS.Sign}(\mathsf{sk}), \mathsf{BS.User}(\mathsf{pk}, m)\rangle$: an interactive protocol is run between the signer with private input a secret key sk and the user with private input a public key pk and a message m. The signer outputs $b = 1$ if the interaction completes successfully and $b = 0$ otherwise, while the user outputs a signature σ if it ends correctly, and $\perp$ otherwise.
- $\mathsf{BS.Verify}(\mathsf{pk}, m, \sigma)$: it takes a public key pk, a message m, and a signature σ, and returns 1 if σ is valid on m under pk and returns 0 otherwise.

For a 2-round protocol the interaction can be realized by the following algorithms:

$$(msg_{U,0}, \text{state}_{U,0}) \leftarrow \mathsf{BS.User}_0(\mathsf{pk}, m)$$
$$(msg_{S,1}, \text{state}_S) \leftarrow \mathsf{BS.Sign}_1(\mathsf{sk}, msg_{U,0})$$
$$(msg_{U,1}, \text{state}_{U,1}) \leftarrow \mathsf{BS.User}_1(\text{state}_{U,0}, msg_{S,1})$$
$$(msg_{S,2}, b) \leftarrow \mathsf{BS.Sign}_2(\text{state}_S, msg_{U,1})$$
$$\sigma \leftarrow \mathsf{BS.User}_2(\text{state}_{U,1}, msg_{S,2})$$

E.2.2. One-more Discrete Logarithm Problem. Here we give a widely-used hardness assumption for blind signature, which is one-more discrete logarithm problem(OMDL), in Fig. 17.

E.3. Randomizable puzzle

Below we give the definition of randomizable puzzzle, which is introduced in [51].

A randomizable puzzle scheme $\mathsf{PRand} = (\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSolve}, \mathsf{PRand})$ with a solution space $\mathcal{S}$ (and a function ϕ acting on $\mathcal{S}$) consists of four algorithms defined as:

$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{PSetup}(1^\lambda)$: is a PPT algorithm that on input security parameter 1^λ, outputs public parameters pp and trapdoors td.

$c_y \leftarrow \mathsf{PGen}(\mathsf{pp}, \zeta)$: is a PPT algorithm that on input public parameters pp and a puzzle solution ζ, outputs a puzzle c_y.

$\zeta := \mathsf{PSolve}(\mathsf{td}, c_y)$: is a DPT algorithm that on input a trapdoor td and puzzle c_y, outputs a puzzle solution ζ.

$c'_y \leftarrow \mathsf{PRand}(c_y, \mathsf{pp})$: a PPT algorithm that on input a randomization factor r and a puzzle c_y, outputs a randomized puzzle c'_y.

A randomizable puzzle (RP) satisfies correctness, security and privacy properties. Correctness property ensures that using the trapdoor we can always recover the correct solution to the puzzle (where a randomized puzzle's solution depends on the randomization factor). Security property ensures that it is infeasible for an adversary that has access only to the puzzle and the public parameters to obtain the underlying solution. Privacy property ensures that it is hard to tell if a puzzle is randomized or not.

E.4. Linear-Only Encryption

The definition of Linear-Only Encryption oracles are given in Figure 18.

Definition 21 (LOE-Encryption Scheme)**.** *Below we give a definition of linear-only encryption(LOE). The LOE-Encryption Scheme is defined with respected to a linearly homomorphic encryption scheme* $\Pi^*_{\mathsf{E}} := (\mathsf{KeyGen}^*, \mathsf{Enc}^*, \mathsf{Dec}^*)$ *over* $\mathbb{Z}_p$ *with additional algorithms:*

- $\mathsf{KeyGen}(1^\lambda)$: *Sample* $(\mathsf{ek}^*, \mathsf{dk}^*) \leftarrow \mathsf{KeyGen}^*(1^\lambda)$ *and some* $\alpha \leftarrow\$ \mathbb{Z}_p$*. Return* $\mathsf{dk} := (\mathsf{dk}^*, \alpha)$ *as the decryption key and* $\mathsf{ek} := (\mathsf{ek}^*, \mathsf{Enc}^*(\mathsf{ek}^*, \alpha))$ *as the encryption key.*
- $\mathsf{Enc}(\mathsf{ek}^*, x)$ *Compute* c *as* $(\mathsf{Enc}^*(\mathsf{ek}^*, x), \mathsf{Enc}^*(\mathsf{ek}^*, \alpha \cdot x))$*, where* $\mathsf{Enc}^*(\mathsf{ek}^*, \alpha \cdot x)$ *is computed homomorphically using* ek.
- $\mathsf{Dec}(\mathsf{dk}^*, c)$*: Parse* c *as* (c_0, c_1) *and compute* $x_0 \leftarrow Dec^*(\mathsf{dk}^*, c_0)$ *and* $x_1 \leftarrow \mathsf{Dec}^*(\mathsf{dk}^*, c_1)$*. If* $x_1 = \alpha \cdot x_0$ *return* x_0*, else return* $\perp$*.*

E.5. Castagnos-Laguillaumie Encryption Scheme

The Castagnos-Laguillaumie encryption scheme, known as CL encryption, is the additive homomorphic encryption with prime order message space, which was raised in 2015 [14]. This scheme works on class group of quadratic imaginary field $K = \mathbb{Q}(\sqrt{D})$ where $D < 0$ and square-free. The discriminant of K is $\Delta_K = D$ if $D \equiv 1(\mod 4)$ and $\Delta_K = 4D$ otherwise. Let O be the subring of K and a free $\mathbf{Z}$-module of rank 2, then O is defined as order. Let O_{Δ_K} be the maximal order of K and $f = [O_{\Delta_K} : O]$ the finite index of any order O in O_{Δ_K}. O could be written as $\mathbf{Z} + f\omega_K\mathbf{Z}$ where $\omega_K = \frac{\Delta_K + \sqrt{\Delta_K}}{2}$. By taking $f = p$ as a prime and $\Delta_K = -pq$ divisible by f where $pq \equiv -1(\mod 4)$, we have $\Delta_p = p^2\Delta_K$ for order O_{Δ_p}. Accordingly, we could pick g_q to generate $G_q = \langle g_q \rangle$ such that $F = \langle f \rangle$ and $G = \langle f \cdot g_q \rangle$. Finding the discrete logarithm in F is easy.

The CL encryption is defined as the algorithm tuple as $(\mathsf{CLSetup}, \mathsf{CLKeyGen}, \mathsf{CLEncrypt}, \mathsf{CLDecrypt}, \mathsf{CLEvalScal}, \mathsf{CLEvalSum})$.

$(\mathsf{param}) \leftarrow \mathsf{CLSetup}(1^\lambda)$ generates public parameter param by security parameter 1^λ.

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{CLKeyGen}(\mathsf{param})$ picks secret key sk in some space and set $\mathsf{pk} \leftarrow g_q^{\mathsf{sk}}$.

$(C_1, C_2) \leftarrow \mathsf{CLEncrypt}(\mathsf{param}, \mathsf{pk}, m)$ picks randomness ρ firstly and computes $C_1 \leftarrow f^m \mathsf{pk}^\rho$ and $C_2 \leftarrow g_q^\rho$.

$(m) \leftarrow \mathsf{CLDecrypt}(\mathsf{param}, \mathsf{sk}, C_1, C_2)$ computes $M \leftarrow C_1/C_2^{\mathsf{sk}}$ and returns the discrete logarithm of M.

$(\hat{C}_1, \hat{C}_2) \leftarrow \mathsf{CLEvalScal}(\mathsf{param}, C_1, C_2, s)$ scalar the encrypted message by s with $\hat{C}_1 \leftarrow C_1^s$ and $\hat{C}_2 \leftarrow C_2^s$.

$(\hat{C}_1, \hat{C}_2) \leftarrow \mathsf{CLEvalSum}(\mathsf{param}, C_1, C_2, C'_1, C'_2)$ computes the ciphertext of the sum of the two corresponding messages by $\hat{C}_1 \leftarrow C_1 C'_1$ and $\hat{C}_2 \leftarrow C_2 C'_2$

Appendix F. Flexible Blind Conditional Signature

Here we give a complete formal definitions and security properties of flexible blind comditional signature(FBCS). We first repeat the definition of FBCS, given in Section 8.

Definition 22 (Flexible Blind Conditional Signature)**.** *A blind conditional signature* $\Pi_{\mathsf{FBCS}} := (\mathsf{Setup}, \mathsf{Promise}, \mathsf{Solver}, \mathsf{Open})$ *is defined with respect to a signature scheme* $\Pi_{\mathsf{DS}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$*,* $\Pi_{\mathsf{RSoRC}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *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^λ *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* dk_t*, the signing key of the underlying digital signature scheme* sk_t^{Σ}*, the signing key of* RSoRC *scheme* sk^{χ}*, and a message* m_{TR}*) and* $\mathbb{R}$ *(Receiver) (with inputs the encryption key* ek_t*, the verification key of the underlying digital signature scheme* pk_t^{Σ}*, the verification key of* RSoRC *scheme* pk^{χ} *and a message* m_{TR}*) and returns* $\perp$ *to* $\mathbb{T}$ *and either a puzzle* τ *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* sk_s^{Σ}*, the encryption key* ek_t*, the verification key of* RSoRC *scheme* pk^{χ}*, a message* m_{ST}*, and a puzzle* τ*) and* $\mathbb{T}$ *(Tumbler) (with inputs the decryption key* dk_t*, the verification key of the underlying digital signature scheme* pk_s^{Σ}*, the signing key* sk^{χ} *of* Π_{RSoRC} *and a message* m_{ST}*) and returns to both users either a signature* σ^* *(*$\mathbb{S}$ *additionally receives a secret* s*) or* $\perp$*.*
- $\{\sigma, \perp\} \leftarrow \mathsf{Open}(\tau, s)$*: The open algorithm takes as input a puzzle* τ *and a secret* s *and returns a signature* σ *or* $\perp$*.*

The security of FBCS is defined as **Correctness**, **Blindness**, **Unlockablity**, and **Unforgeability**, which are defined below.

Definition 23 (Correctness)**.** *A Flexible Blind Conditional Signature* Π_{FBCS} *is correct if for all* $\lambda \in \mathbb{N}$*, all* $(\mathsf{ek}_t, \mathsf{dk}_t)$ *in the support of* $\Pi_{\mathsf{Enc}}.\mathsf{Setup}(1^{\lambda})$*, all* $(\mathsf{pk}^{\chi}, \mathsf{sk}^{\chi})$ *in the support of* $\Pi_{\mathsf{RSoRC}}.\mathsf{Setup}(1^{\lambda})$*, all* $(\mathsf{pk}_t^{\Sigma}, \mathsf{sk}_t^{\Sigma}), (\mathsf{pk}_s^{\Sigma}, \mathsf{sk}_s^{\Sigma})$ *and* $\mathsf{pk}_r^{\Sigma}, \mathsf{sk}_r^{\Sigma}$ *in the support of* $\Pi_{\mathsf{DS}}.\mathsf{KGen}(1^{\lambda})$*, and all messages* $(m_{\mathsf{TR}}, m_{\mathsf{ST}})$*, it holds that* $\Pr\left[\mathsf{Vf}(\mathsf{pk}_t^{\Sigma}, m_{\mathsf{TR}}, \mathsf{Open}(\tau, s)) = 1\right] = 1$*,* $\Pr\left[\mathsf{Vf}(\mathsf{pk}_s^{\Sigma}, m_{\mathsf{ST}}, \sigma^*) = 1\right] = 1$*, where*

- $\tau \leftarrow \mathsf{Promise}\left\langle \begin{matrix} \mathbb{T}\left(\mathsf{dk}_t, \mathsf{sk}_t^{\Sigma}, \mathsf{sk}^{\chi}, \mathsf{com}(m_{\mathsf{TR}})\right) \\ \mathbb{R}\left(\mathsf{ek}_t, \mathsf{pk}_t^{\Sigma}, \mathsf{pk}^{\chi}, m_{\mathsf{TR}}\right) \end{matrix} \right\rangle$ *and*
- $((\sigma^*, s), \sigma^*) \leftarrow \mathsf{Solver}\left\langle \begin{matrix} \mathbb{S}\left(\mathsf{sk}_s^{\Sigma}, \mathsf{ek}_t, \mathsf{pk}^{\chi}, m_{\mathsf{ST}}, \tau\right) \\ \mathbb{T}\left(\mathsf{dk}_t, \mathsf{pk}_s^{\Sigma}, \mathsf{sk}^{\chi}, \mathsf{com}(m_{\mathsf{ST}})\right) \end{matrix} \right\rangle$.

Blindness essentially states that the tumbler cannot link two promise/solve sessions together. It is noted that there are some differences between our blindness game and the one defined in [26]: firstly, the message held by the tumbler is the commitment of the message(transaction) generated by the user. Secondly, in the blindness experiment given in [26], the messages are output by the adversary $\mathcal{A}$, while in our blindness experiment, the messages are sampled in the message space $\mathcal{M}$. The reason for this is that in BlindHub, the amount is allowed to be variable. If it is the tumbler to pick the message(transaction), it is easy for the tumbler to break the blindness by picking transactions with different payment amounts. Although the messages are chosen by the experiment, the keys are still picked by the tumbler, and thus we can still provide anonymity even when the tumbler is malicious and picks its keys adversarially to try to link a sender/receiver pair.

Definition 24 (Blindness)**.** *A Flexible Blind Conditional Signature* Π_{FBCS} *is blind if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*, the following holds:* $\Pr\left[\mathsf{ExpBlnd}^{\mathcal{A}}_{\Pi_{\mathsf{FBCS}}}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$*, where* ExpBlnd *is defined in Figure 19.*

Unlockability requires that it is hard for Tumbler to generate a valid signature on a message from the sender that prevents the receiver from being able to unlock the entire signature in the accompanying promise session. In the unlockability game, we only focus on the security issue and ignore the privacy, so we allow that adversary(tumbler) to choose even the messages used in the game.

Definition 25 (Unlockability)**.** *A Flexible Blind Conditional Signature* Π_{FBCS} *is unlockable if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*, the following holds:* $\Pr\left[\mathsf{ExpUnlock}^{\mathcal{A}}_{\Pi_{\mathsf{FBCS}}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$*, where* ExpUnlock *is defined in Figure 20.*

To define unforgeability, we first define two sets: $\mathcal{S}_{\mathsf{TR}}$ and $\mathcal{S}_{\mathsf{ST}}$. $\mathcal{S}_{\mathsf{TR}}$ contains the amounts sent from the tumbler to the receiver in the puzzle promise phase. $\mathcal{S}_{\mathsf{ST}}$ contains the amounts sent from the sender to the tumbler in the puzzle solver phase. Besides, we define m^{amt} as a transaction m with a payment amount amt. Then we define unforgeability as follows:

Unforgeability requires that sender and receiver could neither produce more than $q-1$ signatures of the tumbler after successfully querying the solving oracle for $q-1$ times, nor $\mathcal{S}_{\mathsf{ST}} \nsubseteq \mathcal{S}_{\mathsf{TR}}$. The former requirement is the same as the definition of unforgeability of BCS [26], while the latter requirement ensures the set of the amounts sent by $\mathbb{S}$ to $\mathbb{T}$ should be the subset of amounts sent by $\mathbb{T}$ to $\mathbb{R}$. This latter requirement captures the fact that in the variable-amount PCH, the sender may cheat by sending to $\mathbb{T}$ an amount which does not match the puzzle. These two requirements implicitly requires that the sender and receiver could not steal the tumbler's coins. The winning requirement b_0 encapsulates the situation in which an adversary forges a signature of the tumbler on a message that has never been used in a promise oracle query. The remaining conditions b_1, b_2, b_3 and b_4 capture following scenarios: 1) adversary outputs q valid distinct key-message-signature tuples but has only queried $q-1$ times for the solutions, 2) the set of the amounts sent from the $\mathbb{S}$ to $\mathbb{T}$ should be a subset of the amounts sent from the $\mathbb{T}$ to $\mathbb{R}$. It is noted that not only the first winning conditions(b_0) and the remaining conditions(b_1, b_2, b_3, b_4) are incomparable, but also b_3, b_4 are incomparable: a successful attack under b_0 does not imply a

successful attack under $b_1, b_2, b_3/b_4$(and vice versa), and a successful attack under b_1, b_2, b_3 does not imply a successful attack under b_1, b_2, b_4 either(and vice versa). This is indeed correct: in the condition b_0, the attacker comes up with a fresh signature that is not the completion of any promise interaction, while in the other conditions, the attacker only manages to query the solving oracle in unexpected ways. In the condition b_3, the attacker tries to queries the solving oracle less than required, whereas in the condition b_4, the attacker uses different set of amounts from the ones used in the puzzle promise phase.

Definition 26 (Unforgeability)**.** *A Flexible Blind Conditional Signature* Π_{FBCS} *is* unforgeable *if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$ *and all PPT adversaries* $\mathcal{A}$*, the following holds:* $\Pr\left[\mathsf{ExpUnforg}^{\mathcal{A}}_{\Pi_{\mathsf{FBCS}}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$*, where* ExpUnforg *is defined in Figure 21.*

The security of FBCS is defined as follows:

Definition 27 (Security)**.** *A Flexible Blind Conditional Signature* Π_{FBCS} *is secure if it is blind, unlockable, and unforgeable.*

Theorem 7. *Let* Π_E *be an LOE scheme,* Π_{AS} *is a secure adaptor signature scheme,* Π_{BAS} *is a secure blind adaptor signature scheme,* Π_{RSoRC} *is a secure signature on randomizable commitments scheme,* Π_{NIZK} *is a sound proof system. Assuming the hardness of OMDL, the BlindHub protocol is a secure blind conditional signature scheme.*

The proof for Theorem 7 can be found in Appendix G.

Appendix G. Security Analysis of BlindHub

G.1. OM-CCA-BlindHub

In this section we present our security results. Before proving our main theorem, we define a property which is going to be useful for our analysis.

Definition 28 (OM-CCA-BlindHub)**.** *An encryption scheme* Π_{E} *is one-more CCA-BlindHub-secure (OM-CCA-BlindHub) if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, all polynomials* $q = q(\lambda)$*, and all PPT adversaries* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\textit{OM-CCA-BlindHub}^{\mathcal{A}}_{\Pi_{\mathsf{E}},q}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

where OM-CCA-BlindHub is defined in Figure 22.

The following technical lemma shows that an LOE scheme satisfies this property, assuming the hardness of the OMDL problem.

Lemma 1. *Let* Π_{E} *be an LOE scheme. Assuming the hardness of OMDL,* Π_{E} *is OM-CCA-BlindHub secure.*

Proof. We proof by reduction. Let $\mathcal{A}$ be a PPT adversary that can win the OM-CCA-BlindHub game with non-negligible advantage. We construct another adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of OMDL.

Firstly, $\mathcal{B}$ is given $(Y_1, \cdots, Y_{q+1}) = (g^{y_1}, \cdots, g^{y_{q+1}})$ by the OMDL game. Then, it samples $q+1$ uniform $\lambda-$ bit strings $(c^*_{y,1}, \cdots, c^*_{y,q+1})$, which are identically distributed to outputs of $\mathcal{O}^{\mathsf{Enc}}$. It enters $\left(X_1, c^*_{y,1}\right), \ldots, \left(X_{q+1}, c^*_{y,q+1}\right)$ into a table M, where the X_i are random variables. Afterwards, $\mathcal{B}$ samples a random group elements as a commitment of the amount $\mathsf{C}_{\mathsf{amt}}$. Then it signs the Y_i and $\mathsf{C}_{\mathsf{amt}} : \tilde{\sigma}_i \leftarrow \mathsf{RCSign}(Y_i, \mathsf{C}_{\mathsf{amt}}), \in [q+1]$. Now it sends $(c^*_{y,1}, Y_1, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma}_1), \cdots, (c^*_{y,q+1}, Y_{q+1}, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma}_{q+1})$ to $\mathcal{A}$. For $\mathcal{A}$'s oracle queries to the encryption scheme oracles ($\mathcal{O}^{\mathsf{Gen}}$, $\mathcal{O}^{\mathsf{Enc}}, \mathcal{O}^{\mathsf{Dec}}, \mathcal{O}^{\mathsf{Add}}$), $\mathcal{B}$ just forwards them to the oracle and returns the oracle replies unchanged, but records them in M. When $\mathcal{A}$ makes some query $(\mathsf{pk}^{\Sigma}_{s,i}, \mathsf{pk}^{\chi}, m_i, \hat{\sigma}_i, Y'_i, c_{y,i}, \mathsf{C}_{\mathsf{amt},i}, \tilde{\sigma}_i)$, $\mathcal{B}$ first check that if pk^{Σ}_s is in the support if $\Pi_{\mathsf{BAS}}.\mathsf{KGen}(1^\lambda)$ and if the signature $\tilde{\sigma}_i$ on Y_i and $\mathsf{C}_{\mathsf{amt},i}$ is valid. After this, there are the following four cases:

1) If $c_{y,i} = c^*_{y,j}$ and $Y'_i = Y^*_j$ for some j, it checks $\mathsf{PreVf}(\mathsf{pk}^{\Sigma}_{s,i}, m_i, Y'_i, \hat{\sigma}_i) = 1 \wedge \mathsf{RCVerify}(\mathsf{pk}^{\chi}, Y'_i, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma}_i) = 1$. If not, return $\perp$. Otherwise, it queries the discrete log oracle on Y'_i to get y_i and returns $\Pi_{\mathsf{BAS}}.\mathsf{Adapt}(\hat{\sigma}_i, y_i)$ to $\mathcal{A}$.
2) if $c_{y,i} = c^*_{y,j}$ but $Y'_i \neq Y^*_j$: return $\perp$.
3) if $(\cdot, c_{y,i}) \notin M$: return $\perp$.
4) Otherwise, let p_i be the plaintext entry corresponding to $c_{y,i}$ in M. Since we are in the linear-only encryption(LOE) model, p_i must be a polynomial in $X_1, \cdots, X_{q+1}$ with $\mathsf{deg}(p_i) \leq 1$.
   a) if $\mathsf{deg}(p_i) = 0$, which means p_i is just a constant value x_i. In this case, $\mathcal{B}$ uses x_i to run the normal $\mathcal{O}^{\mathsf{BlindHub}}$ and sends its output to $\mathcal{A}$.
   b) if $\mathsf{deg}(p_i) = 1$, define $p_i := \alpha_0 + \alpha_1 X_1 + \ldots + \alpha_n X_{q+1}$. If $Y'_i = g^{\alpha_0} \prod_{t=1}^{q+1} Y_t^{\alpha_t} = g^{p_i}$ and $\mathsf{PreVf}(\mathsf{pk}^{\Sigma}_s{}_i, m_i, Y'_i, \hat{\sigma}_i) = 1$, $\mathcal{B}$ queries the discrete log oracle on Y'_i and uses the oracle replies to adapt $\hat{\sigma}_i$ and sends the full signature to $\mathcal{A}$. Otherwise, returns $\perp$ to $\mathcal{A}$.

Finally, if $\mathcal{A}$ outputs $(y_1, \cdots, y_{q+1})$, $\mathcal{B}$ returns them as the answers in the OMDL game. Since $\mathcal{A}$ wins with non-negligible probability, $\mathcal{B}$ wins the OMDL game with non-negligible probability. The theorem follows. □

G.2. Proof of Theorem 7

Lemma 2 (Blindness)**.** *Assuming* Π_{NIZK} *is sound,* $\Pi_{\tilde{\sigma}}$ *achieves class hiding,* Π_{COM} *is randomizable, the BlindHub scheme achieves blindness in the LOE model.*

Proof. Fix any two Promise executions, we now show, via a series of hybrid experiments, that the cases of b = 0 and b = 1 are computationally close.
Hybrid $\mathcal{H}_0$: Run ExpBlnd with $b = 0$.
Hybrid $\mathcal{H}_1$: the same as $\mathcal{H}_0$ except in both runs of Solver, $\tilde{\sigma}'' \leftarrow \mathsf{RCSign}(\mathsf{sk}^{\chi}, Y'', \mathsf{C}''_{\mathsf{amt}})$

Hybrid $\mathcal{H}_2$: the same as $\mathcal{H}_1$ except in both runs of Solver, sample $y \leftarrow\$ \mathbb{Z}_q$, and set $Y'' \leftarrow g^y$ and $c''_y \leftarrow \Pi_{\mathsf{Enc}}(\mathsf{ek}_t, y)$, $\tilde{\sigma}'' \leftarrow \mathsf{RCSign}(\mathsf{sk}^\chi, Y'', \mathsf{C}''_{\mathsf{amt}})$
Hybrid $\mathcal{H}_3$: the same as $\mathcal{H}_2$ except that in both runs of Solver, sample $r_{\mathsf{amt}} \leftarrow\$ \mathbb{Z}_p$, compute $\mathsf{C}''_{\mathsf{amt}} \leftarrow \mathsf{com}(\mathsf{amt}, r_{\mathsf{amt}}), \tilde{\sigma}'' \leftarrow \mathsf{RCSign}(\mathsf{sk}^\chi, Y'', \mathsf{C}''_{\mathsf{amt}})$
Hybrid $\mathcal{H}_4$: Compute c''_y, Y'' honestly using τ_1 in the first run of Solver and τ_0 in the second run of Solver.
Hybrid $\mathcal{H}_5$: Compute $\mathsf{C}''_{\mathsf{amt}}$ honestly using τ_1 in the first run of Solver and τ_0 in the second run of Solver.
Hybrid $\mathcal{H}_6$: Compute $\tilde{\sigma}''$ honestly using τ_1 in the first run of Solver and τ_0 in the second run of Solver.
Hybrid $\mathcal{H}_7$: Run ExpBlnd with $b = 1$.

Claim 1. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_0,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_1,\mathcal{A}}$$

Proof. The difference between $\mathcal{H}_0$ and $\mathcal{H}_1$ is that in $\mathcal{H}_1$, rather than computing $\tilde{\sigma}''$ by $(\tilde{\sigma}'', Y'', \mathsf{C}''_{\mathsf{amt}}) \leftarrow \mathsf{RCRand}(\mathsf{pk}^\chi, Y', \mathsf{C}'_{\mathsf{amt}}, \alpha)$ for some randomness α, we first randomize $Y', \mathsf{C}'_{\mathsf{amt}}$ using the randomness α, than compute $\tilde{\sigma}'' \leftarrow \mathsf{RCSign}(\mathsf{sk}^\chi, Y', \mathsf{C}'_{\mathsf{amt}})$. The secret key sk^χ is obtained by extracting the witness from the NIZK proof given by the tumbler: $\pi_{\mathsf{sk}^\chi} \leftarrow \mathsf{P}_{\mathsf{NIZK}}\{(\mathsf{sk}^\chi, \mathsf{pk}^\chi)| \in \mathsf{Supp}(\Pi_{\mathsf{RSoRC}}.\mathsf{KGen}(1^\lambda))\}$. Then the indistinguishability of $\mathcal{H}_0$ and $\mathcal{H}_1$ follows from the class hiding property of Π_{RSoRC}. □

Claim 2. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_1,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A}}$$

Proof. Y'' are g raised to a uniform element and c''_y is an encryption of the same uniform element in both experiments, conditioned on the ciphertext provided by the Hub being well-formed. Thus, any distinguishing advantage necessarily corresponds to a violation of the soundness property of Π_{NIZK}. It follows that the executions are statistically indistinguishable. □

Claim 3. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A}}$$

Proof. Since we sample r_{amt} uniformly, due to the homomorphic property of Π_{COM}, the indistinguishability holds unconditionally. □

Claim 4. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_4,\mathcal{A}}$$

Proof. This holds by the same logic as Claim 2. □

Claim 5. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_4,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_5,\mathcal{A}}$$

Proof. This holds by the same logic as Claim 3. □

Claim 6. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_5,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_6,\mathcal{A}}$$

Proof. This holds by the same logic as Claim 1. □

Claim 7. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_6,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_7,\mathcal{A}}$$

Proof. The change is only syntactical and the executions are identical. □

Hence, the cases of $b = 0$ and $b = 1$ are computationally indistinguishable. □

Lemma 3 (Unlockability)**.** *Assuming that Π_{AS} and Π_{BAS} are witness extractable, pre-signature adaptable, and unforgeable, the BlindHub scheme is unlockable.*

Proof. the same logic as that in [26]. □

Lemma 4 (Unforgeability)**.** *Assuming the hardness of OMDL and that Π_{AS} is witness extractable and unforgeable, the BlindHub scheme is unforgeable in the LOE model.*

Proof. We give a series of hybrid experiments, show they are indistinguishable, and prove by reduction to OM-CCA-BlindHub that no adversary exists with non-negligible advantage against the final hybrid.
Hybrid $\mathcal{H}_0$: This is the normal game ExpUnforg.
Hybrid $\mathcal{H}_1$: the same as $\mathcal{H}_0$ except that all NIZK proofs are simulated using $\Pi_{\mathsf{NIZK}}.\mathsf{Sim}$.
Hybrid $\mathcal{H}_2$: the same as $\mathcal{H}_1$ except that if $\exists i \in [q]$ such that $\mathsf{Vf}(\mathsf{pk}^\Sigma_{t,i}, m_i, \sigma_i) = 1$ and $(\mathsf{pk}^\Sigma_{t,i}, \cdot) \in \mathcal{L}$ but $\left(\mathsf{pk}^\Sigma_{t,i}, m_i\right) \notin \mathcal{L}$, return 0.
Hybrid $\mathcal{H}_3$: the same as $\mathcal{H}_2$ except that if $\exists i \in [q]$ such that $\mathsf{Vf}(\mathsf{pk}^\Sigma_{t,i}, m_i, \sigma_i) = 1$ and $g^{\mathsf{Ext}(\tilde{\sigma}_i, \sigma_i)} \neq Y_i$, return 0.
Hybrid $\mathcal{H}_4$: the same as $\mathcal{H}_3$ except that if $\mathcal{T} \notin \{\mathsf{amt}_1, \cdots, \mathsf{amt}_q\}$, return 0. □

Claim 8. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_0,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_1,\mathcal{A}}$$

Proof. This follows directly from zero-knowledge of Π_{NIZK}. □

Claim 9. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_1,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A}}$$

Proof. The hybrids differ only in the case where the attacker returns a valid signature on a message that was not part of the transcript. By the unforgeability of the adaptor signature, this happens only with negligible probability. □

Claim 10. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A}}$$

Proof. suppose the distinguishing advantage is non-negligible. Then there exists an adversary that can output tuples $(\mathsf{pk}^\Sigma_{t,i}, m_i, \sigma_i)$ such that $\mathsf{Vf}(\mathsf{pk}^\Sigma_{t,i}, m_i, \sigma_i) = 1$ and $g^{\mathsf{Ext}(\tilde{\sigma}_i, \sigma_i)} \neq Y_i$ with non-negligible probability. Then we

can give a reduction to the witness-extractability of Π_{AS}. The proof for this is similar to the one given in claim 6, lemma C.3 in [26], and we omit it here. □

Claim 11. *For all PPT adversaries $\mathcal{A}$,*

$$\mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A}} \approx \mathsf{EXEC}_{\mathcal{H}_4,\mathcal{A}}$$

Proof. Suppose there exists an adversary $\mathcal{A}$ with non-negligible success probability in $\mathcal{H}_3$. We can construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the unforgeability of the RSoRC scheme. $\mathcal{B}$ is constructed as follows: at the beginning, given pk^{χ} from the RSoRC challenger $\mathcal{C}$, $\mathcal{B}$ set it be the RSoRC signing key and forwards it to $\mathcal{A}$. When $\mathcal{A}$ queries the puzzle promise oracle $\mathcal{O}\mathsf{PP}$, $\mathcal{B}$ computes everything the same as the protocol, except following: 1) rather than computing RSoRC signature by itself, $\mathcal{B}$ computes RSoRC signature by forwards commitment of amount $\mathsf{C}_{\mathsf{amt}}$ and adaptor statement Y to the signing oracle of RSoRC and sends the oracle replies $\tilde{\sigma}$ to $\mathcal{A}$. 2)$\mathcal{B}$ records $(y, Y, \mathsf{amt}, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma})$ in the list $\mathcal{Y}$. Finally, $\mathcal{A}$ terminates and outputs $\{(\mathsf{pk}_{t,i}^{\Sigma}, m_i^{[\mathsf{amt}_i]}, \sigma_i)\}_{\forall i \in [q]}$ which satisfy conditions b_1, b_2, b_4. Since $\mathcal{L} \notin \{\mathsf{amt}_1, \cdots, \mathsf{amt}_q\}$, $\mathcal{B}$ is able to figure out the difference between $\mathcal{L}$ and $\{\mathsf{amt}_1, \cdots, \mathsf{amt}_q\}$: $\mathcal{S}^- := \{\mathsf{amt}_{\mathsf{ST},1}, \cdots, \mathsf{amt}_{\mathsf{ST},q'}\} - \{\mathsf{amt}_{\mathsf{ST},1}, \cdots, \mathsf{amt}_{\mathsf{ST},q'}\} \cap \{\mathsf{amt}_1, \cdots, \mathsf{amt}_q\}$. If $\exists \mathsf{amt}_{\mathsf{ST},i^*} \in \mathcal{S}^-$ such that $\mathsf{amt}_{\mathsf{ST},i^*} \notin \{\mathsf{amt}_1, \cdots, \mathsf{amt}_q\}$, then $Y'^*, \mathsf{C}'^*_{\mathsf{amt}}, \tilde{\sigma}'^*$, which satisfies $(\mathsf{amt}_{\mathsf{ST},i^*}, Y'^*, \cdot, \mathsf{C}'^*_{\mathsf{amt}}, \tilde{\sigma}'^*) \in \mathcal{U}$, is a valid forgery. If $\forall \mathsf{amt} \in \mathcal{S}^{-1}$, $\mathsf{amt} \in \{\mathsf{amt}_1, \cdots, \mathsf{amt}_q\}$, $\mathcal{B}$ returns $(Y'^{**}, \mathsf{C}'^{**}_{\mathsf{amt}}, \tilde{\sigma}'^{**})$ as a forgery, where $(Y'^{**}, \mathsf{C}'^{**}_{\mathsf{amt}}, \tilde{\sigma}'^{**})$ satisfies the following requirements: 1)$\exists \mathsf{amt}^{**}_{\mathsf{ST}} \in \mathcal{S}^{-1}, c_y{}'^{**}, s.t., (\mathsf{amt}^{**}_{\mathsf{ST}}, Y'^{**}, c'^{**}_y, \mathsf{C}'^{*}_{\mathsf{amt}}, \tilde{\sigma}'^{**}) \in \mathcal{U}$, 2) $\exists (y, Y, \mathsf{amt}, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma}) \in \mathcal{Y}$, s.t., $\mathsf{amt} = \mathsf{amt}^{**}_{\mathsf{ST}}, (\mathsf{C}'^{**}_{\mathsf{amt}}, Y'^{**}, \tilde{\sigma}'^{**}) = \mathsf{RCRand}(\mathsf{C}_{\mathsf{amt}}, Y, \tilde{\sigma}, \alpha)$, where $\alpha = y^{**}/y$ and y^{**} is the solution of c'^{**}_y.

□

Now we give a reduction from hybrid $\mathcal{H}_4$ to OM-CCA-BlindHub.

Suppose there exists an adversary $\mathcal{A}$ with non-negligible success probability in $\mathcal{H}_4$. We give a reduction that uses $\mathcal{A}$ to win the OM-CCA-BlindHub game. The reduction is given $(c_{y,1}, g^{y_1}), \ldots, (c_{y,q+1}, g^{y_{q+1}})$. It generates $(\mathsf{ek}, \mathsf{dk}) \leftarrow \Pi_{\mathsf{Enc}}.\mathsf{KGen}(1^\lambda), (\mathsf{pk}^{\chi}, \mathsf{sk}^{\chi}) \leftarrow \Pi_{\mathsf{RSoRC}}.\mathsf{KGen}(1^\lambda)$ and $(\mathsf{pk}_t^{\Sigma}, \mathsf{sk}_t^{\Sigma}) \leftarrow \mathsf{KGen}(1^\lambda)$ as in $\mathcal{H}_3$ and starts $\mathcal{A}$ on input ek. For the promise oracle $\mathcal{O}\mathsf{PP}$ queries, the reduction follows the same steps as $\mathcal{H}_3$ except it uses a group element g^{y_i}(given by the challenger in the OM-CCA-BlindHub game) as adaptor statement each time to generate a pre-signature. When $\mathcal{A}$ queries solver oracle $\mathcal{O}\mathsf{PS}$, the reduction computes the completed signature σ_{ts} as the output of $\mathcal{O}^{\mathsf{BlindHub}}$ run on $\mathcal{A}$'s outputs $(\mathsf{pk}^{\Sigma}_{ts}, m_{\mathsf{ST}}, \sigma, c''_y, Y'', \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma})$. Note that since $\mathcal{A}$ makes at most q non-$\perp$ queries to $\mathcal{O}\mathsf{PS}$, the reduction also makes at most q non- queries to $\mathcal{O}^{\mathsf{BlindHub}}$, as the oracles return in exactly the same cases.

Once $\mathcal{A}$ returns $q+1$ tuples $(\mathsf{pk}^{\Sigma}_{t,i}, m_i, \sigma_i)$, , the reduction computes $y_i \leftarrow \Pi_{\mathsf{BAS}}.\mathsf{Ext}(\mathsf{pk}^{\Sigma}_{t,i}, m_i, \hat{\sigma}_i, \sigma_j, Y_i), \forall i, j \in [q+1]$ until it has $q+1$ non-$\perp$ values y_i and outputs those values. By the definition of $\mathcal{H}_3$, we have $g^{y_i} = Y_i$, $\forall i \in [q+1]$. By assumption, the reduction wins the OM-CCA-BlindHub game with non-negligible probability. This violates OM-CCA-BlindHub-security of Π_{Enc}. The theorem follows.

Appendix H. Instantiation of BlindChannel based on Bitcoin

In this section, we show how to realize BlindChannel for Bitcoin. Recall that in the BlindChannel model (described in Section 5), when the unblind party updates the channel with the blind party, the splitt transaction ($\mathtt{TX}_{\mathtt{SP}}$), the adaptor execution delivery transaction ($\mathtt{TX}_{\mathtt{AED}}$) and the timeout transaction ($\mathtt{TX}_{\mathtt{TO}}$) should be hidden from the blind party, otherwise the amounts on the transactions will be revealed. However, the unblind party requires the signature on the transaction from the blind party. To convince blind party that the unblind party is honest, the unblind party should prove the correctness of the hidden transaction before the blind party blindly signing the protocol. Recall that in ECDSA signature, the signing algorithm used in Bitcoin, the message to be signed is hashed by a cryptographic hash function. It means that the unblind party needs to prove knowledge of pre-image of a hash value used in the signing algorithm. But for different transactions ($\mathtt{TX}_{\mathtt{SP}}/\mathtt{TX}_{\mathtt{TO}}/\mathtt{TX}_{\mathtt{AED}}$), the corresponding proofs are different. Below we present details of the proof for each transaction.

H.1. Relations for Zero Knowledge Proof

Before we dive into the details, we first recall some background knowledge of blockchain transactions. According to [3], a transaction tx can be formally defined as a tuple of the form $(\mathsf{txid}, \mathsf{In}, \mathsf{Out}, \mathsf{Witness})$, where $\mathsf{txid} \in \{0,1\}^*$ is the unique identifier of tx and is calculated as $\mathsf{txid} := H(\mathsf{tx})$, where H is a hash function and $[\mathsf{tx}]$ is the body of the transaction defined as $[\mathsf{tx}] := (\mathsf{In}, \mathsf{Out})$; $\mathsf{In} = ((tid_1, ind_1), \ldots, (tid_n, ind_n))$ is a vector of strings identifying all transaction inputs, where tid is the transaction id of previous transactions and ind is the output index of them; $\mathsf{Out} = (\theta_1, \ldots, \theta_n)$ is a vector of new outputs, where each output has two attributes: the value $\mathsf{cash} \in \mathbb{Z}^*$ representing the amount of coins and the function $\varphi : \{0,1\}^* \to \{0,1\}$ defining the spending condition; $\mathsf{Witness} \in \{0,1\}^*$ contains the witness to spend the transaction inputs. The signature hash of a transaction can be formalized as $h_s := H_{\mathsf{sig}}(\mathsf{tx})$, where H_{sig} is the cryptographic hash function mapping the transaction to a hash value used in the signing algorithm.

For the ease of presentation, we use the notation in Fig. 9 and Fig. 24.

1) $\mathtt{TX}_{\mathtt{SP}}$: We parse $\mathtt{TX}_{\mathtt{SP}}$ as $(\mathsf{txid}, \mathsf{In} = (tid_{\mathtt{TXCM}}, 0), \mathsf{Out} = (\theta_1 = (x_1, \mathsf{pk}'_U), \theta_2 = (x_2, \mathsf{pk}_B), \theta_3 = (x_3, (\mathsf{pk}_U, \mathsf{pk}_B))), \mathsf{Witness})$. In can be revealed because $\mathtt{TX}_{\mathtt{CM}}$ can be accessed by both parties. But amounts in Out should be hidden and set as relation witness. All data in $\mathtt{TX}_{\mathtt{SP}}$ could be revealed and become public inputs except amounts and pk'_U.

2) $\mathtt{TX}_{\mathtt{AED}}$: We parse $\mathtt{TX}_{\mathtt{AED}}$ as $(\mathsf{txid}, \mathsf{In} = (tid_{\mathtt{TX}_{\mathtt{SP}}}, 0), \mathsf{Out} = (\theta = (x_3, \varphi)), \mathsf{Witness})$. $tid_{\mathtt{TX}_{\mathtt{SP}}}$ in the $\mathtt{TX}_{\mathtt{AED}}$ can not be revealed, x_3 in Out should be hidden and set as the relation witness, otherwise the payment amount will be leaked by brute-force. The correctness of $tid_{\mathtt{TX}_{\mathtt{SP}}}$ relies on the garbled circuit internal wires. We could compute the $\mathtt{TX}_{\mathtt{SP}}$.txid in the garbled circuit and fill up $tid_{\mathtt{TX}_{\mathtt{SP}}}$ and let the circuit output h_{AED}.
3) $\mathtt{TX}_{\mathtt{TO}}$: We parse $\mathtt{TX}_{\mathtt{TO}}$ as $(\mathsf{txid}, \mathsf{In} = (tid_{\mathtt{TX}_{\mathtt{SP}}}, 0), \mathsf{Out} = (\theta = (x_3, \varphi)), \mathsf{Witness})$. The $\mathtt{TX}_{\mathtt{TO}}$ works indeed the same as $\mathtt{TX}_{\mathtt{AED}}$ except it contains a relative timelock, which ensures $\mathtt{TX}_{\mathtt{AED}}$ cannot be invalidated immediately. Besides, the timelock needs to be a public input.

We merge all the three ZKGC relations into one as the following. We remark that it's essential to prevent overflow of $\sum \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_i.\mathsf{cash}$ when checking the balance since all the amounts are denoted in 64-bit unsigned number. Otherwise, unblind party could cheat by setting $\sum \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_i.\mathsf{cash} = (1 << 63) + \mathtt{TX}_{\mathtt{CM}}.\mathsf{Out.cash}$.

$$\mathcal{R}_{\mathsf{all}} = \left\{ \begin{pmatrix} \mathtt{TX}_{\mathtt{SP}}.\mathsf{txid}, \\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_1.\mathsf{cash}, \\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_2.\mathsf{cash}, \\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_3.\mathsf{cash}, \\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_1.\varphi \\ \mathtt{TX}_{\mathtt{AED}}.\mathsf{In}.tid, \\ \mathtt{TX}_{\mathtt{AED}}.\mathsf{Out}_1.\mathsf{cash} \\ \mathtt{TX}_{\mathtt{TO}}.\mathsf{In}.tid, \\ \mathtt{TX}_{\mathtt{TO}}.\mathsf{Out}_1.\mathsf{cash} \end{pmatrix} \middle| \begin{array}{l} \delta \text{ holds.} \\ \wedge\, \mathsf{C}_1 = \mathsf{com}(\mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_1.\mathsf{cash}) \\ \wedge\, \mathsf{C}_2 = \mathsf{com}(\mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_2.\mathsf{cash}) \\ \wedge\, \mathsf{C}_{\mathsf{amt}} = \mathsf{com}(\mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_3.\mathsf{cash}) \end{array} \right\}$$

where the condition δ is defined as

$$\delta = \begin{cases} \mathtt{TX}_{\mathtt{SP}}.\mathsf{In}.tid = \mathtt{TX}_{\mathtt{CM}}.\mathsf{txid} \\ \wedge \sum \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_i.\mathsf{cash} = \mathtt{TX}_{\mathtt{CM}}.\mathsf{Out.cash} \\ \wedge\, h_{\mathsf{S}} = H_{\mathsf{sig}}(\mathtt{TX}_{\mathtt{SP}}) \\ \wedge\, \mathtt{TX}_{\mathtt{AED}}.\mathsf{In}.tid = \mathtt{TX}_{\mathtt{SP}}.\mathsf{txid} \\ \wedge\, \mathtt{TX}_{\mathtt{AED}}.\mathsf{Out}_1.\mathsf{cash} = \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_3.\mathsf{cash} \\ \wedge\, h_{\mathsf{AED}} = H_{\mathsf{sig}}(\mathtt{TX}_{\mathtt{AED}}) \\ \wedge\, \mathtt{TX}_{\mathtt{TO}}.\mathsf{In}.tid = \mathtt{TX}_{\mathtt{SP}}.\mathsf{txid} \\ \wedge\, \mathtt{TX}_{\mathtt{TO}}.\mathsf{Out}_1.\mathsf{cash} = \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_3.\mathsf{cash} \\ \wedge\, h_{\mathsf{TO}} = H_{\mathsf{sig}}(\mathtt{TX}_{\mathtt{TO}}) \end{cases}$$

H.2. Transaction digest algorithm

According to BIP-0143[10], the transaction digest algorithm is defined as Double SHA256 of the serialization of:

1) nVersion of the transaction (4-byte little endian)
2) hashPrevouts (32-byte hash)
3) hashSequence (32-byte hash)
4) outpoint (32-byte hash + 4-byte little endian)
5) scriptCode of the input (serialized as scripts inside CTxOuts)
6) value of the output spent by this input (8-byte little endian)
7) nSequence of the input (4-byte little endian)
8) hashOutputs (32-byte hash)
9) nLocktime of the transaction (4-byte little endian)
10) sighash type of the signature (4-byte little endian)

10. https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki

From the above we see that our task is to prove the knowledge of pre-image of hash values output from multiple layers of hash functions. To perform such a zero knowledge proof, we use *garbled circuits based zero knowledge proof*, specifically the technique introduced in [43].

H.3. Garbled-Circuits Based Zero Knowledge Proof

Before we give our GCZK protocol, we first give some priliminaries of garbled circuits and committed oblivious transfer. We follow the definitions of garbled circuits and committed oblivious transfer from [43].

H.3.1. Garbled Circuits.

Definition 29 (Garbled Circuits)**.** *A garbled circuit scheme is a triple $\mathcal{GC}$ of ppt algorithms $\mathcal{GC} = (\mathsf{Garble}, \mathsf{Eval}, \mathsf{Verify})$ where:*

- $\mathsf{Garble}(1^\lambda, f) \rightarrow (\mathsf{GC}, \{L_0^{(i)}, L_1^{(i)}\}_{i\in[m]}, \{Z_0^{(i)}, Z_1^{(i)}\}_{i\in[n]})$ *on input λ and the description of a function $f : \{0,1\}^m \rightarrow \{0,1\}^n$, outputs a garbled circuit, a set of m input label pairs and a set of n output label pairs.*
- $\mathsf{Eval}(\mathsf{GC}, \{L^{(i)}\}_{i\in[m]}) \rightarrow \{Z^{(i)}\}_{i\in[n]}$ *on input a garbled circuit and a set of input labels, outputs a set of output labels.*
- $\mathsf{Verify}(\mathsf{GC}, \{L_0^{(i)}, L_1^{(i)}\}_{i\in[m]}, \{Z_0^{(i)}, Z_1^{(i)}\}_{i\in[n]}) \rightarrow \{0, 1\}$ *on input a garbled circuit, a function and a set of input label pairs and a set of output label pairs, outputs a bit (1 for acceptance and 0 for rejection).*

We say a garbled circuit scheme is *correct* if for all polynomial-size in λ functions $f : \{0,1\}^m \rightarrow \{0,1\}^n$ and all $x \in \{0,1\}^m$, for $y = f(x)$, the following probability is overwhelming in λ,

$$\Pr\left[\begin{array}{l} (\mathrm{GC}, \{L_0^{(i)}, L_1^{(i)}\}_{i\in[m]}, \{Z_0^{(i)}, Z_1^{(i)}\}_{i\in[n]}) \\ \leftarrow \mathsf{Garble}(1^\lambda, f), \{\hat{Z}^{(i)}\}_{i\in[n]} \leftarrow \mathrm{Eval}(\mathrm{GC}, \{L_{x_i}^{(i)}\}_{i\in[m]}) : \\ \hat{Z}^{(i)} = Z_{y_i}^{(i)} \forall i \in [n] \end{array}\right]$$

We say a garbled circuit scheme has *authenticity* if for all polynomial-size in λ functions $f : \{0,1\}^m \rightarrow \{0,1\}^n$ and all $x \in \{0,1\}^m$, for $y = f(x)$, and for all ppt adversaries $\mathcal{A}$, the following probability is negligible in λ,

$$\Pr\left[\begin{array}{l} (\mathrm{GC}, \{L_0^{(i)}, L_1^{(i)}\}_{i\in[m]}, \{Z_0^{(i)}, Z_1^{(i)}\}_{i\in[n]}) \\ \leftarrow \mathsf{Garble}(1^\kappa, f), (Z^*, i^*) \leftarrow \mathcal{A}(f, x, \mathrm{GC}, \{L_{x_i}^{(i)}\}_{i\in[m]}) : \\ Z^* = Z_{\neg y_{i^*}}^{(i^*)}. \end{array}\right]$$

We say a garbled circuit scheme is *verifiable* if for all polynomial-size in λ functions $f : \{0,1\}^m \rightarrow \{0,1\}^n$ and all $x \in \{0,1\}^m$, for $y = f(x)$, and for all ppt

adversaries $\mathcal{A}$, the following probability is negligible in λ,

$$\Pr\left[\begin{array}{c}(\mathrm{GC},\{L_0^{(i)},L_1^{(i)}\}_{i\in[m]},\{Z_0^{(i)},Z_1^{(i)}\}_{i\in[n]})\leftarrow\mathcal{A}(1^\kappa,f)\\ \{\hat{Z}^{(i)}\}_{i\in[n]}\leftarrow\mathsf{Eval}(\mathrm{GC},\{L_{x_i}^{(i)}\}_{i\in[m]}):\\ \mathsf{Verify}(\mathrm{GC},f,\{L_0^{(i)},L_1^{(i)}\}_{i\in[m]},\{Z_0^{(i)},Z_1^{(i)}\}_{i\in[n]})=1\\ \wedge\exists i\in[n],\hat{Z}^{(i)}\neq Z_{y_i}^{(i)}\end{array}\right].$$

H.3.2. Committing oblivious transfer. Below we describe the ideal functionality associated to committing oblivious transfer $\mathcal{F}_{\mathsf{COT}}$. We assume a fixed $n\in\mathbb{N}$:

1) *Choose:* on input (choose, $\{b_i\}_{i\in[n]}$) from the receiver, where $b_i\in\{0,1\},\forall i\in[n]$, inform receiver that a choice was received.
2) *Transfer:* on input (transfer, $\left\{m_0^{(i)},m_1^{(i)}\right\}_{i\in[n]}$) from the sender, send messages $\left\{m_{b_i}^{(i)}\right\}_{i\in[n]}$ to the receiver.
3) *Open*: on input (open-all) from the sender, reveal all pairs $\left\{m_0^{(i)},m_1^{(i)}\right\}_{i\in[n]}$ to the receiver and halt.

We propose the overall GCZK protocol in Fig. 23. The protocol is roughly the same as the one in [43], except that we finally generate 3 different commitments respectively for 3 different cashs in $\mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}$. We give the definitions of circuit public input PubIn and circuit functionality f as below, in which we use " $-$ " to denote set difference, i.e. $A-B=\{x|x\in A\wedge x\notin B\}$. We remark that although the circuit functionality contains no $\mathtt{TX}_{\mathtt{SP}}.\mathsf{txid}$ as input, this value could be computed from $\mathtt{TX}_{\mathtt{SP}}.\mathsf{In}$ and $\mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}$. Accordingly, $\mathtt{TX}_{\mathtt{AED}}.\mathsf{In}.tid$ and $\mathtt{TX}_{\mathtt{TO}}.\mathsf{In}.tid$ could be filled internally using the inner-computed value, the circuit is able to successfully compute the signature hash for both transactions. We also note that the changes of the GCZK does not affect the security of our protocol, thus the security of our protocol is also guaranteed by the proof given in [43]. We refer readers to [43] for the proof.

$$\mathbf{PubIn}=\begin{Bmatrix}\mathtt{TX}_{\mathtt{CM}}.\mathsf{Out}_1.\mathsf{cash}\\ \mathtt{TX}_{\mathtt{SP}}\\ \mathtt{TX}_{\mathtt{AED}}\\ \mathtt{TX}_{\mathtt{TO}}\\ h_{\mathsf{S}}\\ h_{\mathsf{AED}}\\ h_{\mathsf{TO}}\end{Bmatrix}-\begin{Bmatrix}\mathtt{TX}_{\mathtt{SP}}.\mathsf{txid}\\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_1.\mathsf{cash}\\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_2.\mathsf{cash}\\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_3.\mathsf{cash}\\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_1.\varphi\\ \mathtt{TX}_{\mathtt{AED}}.\mathsf{In}.tid\\ \mathtt{TX}_{\mathtt{AED}}.\mathsf{Out}_1.\mathsf{cash}\\ \mathtt{TX}_{\mathtt{TO}}.\mathsf{In}.tid\\ \mathtt{TX}_{\mathtt{TO}}.\mathsf{Out}_1.\mathsf{cash}\end{Bmatrix}$$

$$f_{\mathbf{PubIn}}(m_1,m_2,m_3,\mathsf{pk}_U)=\begin{cases}(m_1,m_2,m_3) & \text{, if } \delta' \text{ holds.}\\ 0 & \text{, otherwise.}\end{cases}$$

, where condition δ' is defined as

$$\delta'=\begin{cases}\delta \text{ holds when setting}\\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_1.\mathsf{cash}:=\mathsf{m}_1\\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_2.\mathsf{cash}:=\mathsf{m}_2\\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_3.\mathsf{cash}:=\mathsf{m}_3\\ \mathtt{TX}_{\mathtt{SP}}.\mathsf{Out}_1.\varphi=\mathsf{pk}_U\\ \mathtt{TX}_{\mathtt{AED}}.\mathsf{In}.tid:=H(\mathtt{TX}_{\mathtt{SP}})\\ \mathtt{TX}_{\mathtt{TO}}.\mathsf{In}.tid:=H(\mathtt{TX}_{\mathtt{SP}})\end{cases}$$

Appendix I. Complementary to BlindChannel Protocol

Figure 24 describes the process to update a BlindChannel when the unblind party is payee.

Appendix J. UC Model of BlindChannel

J.1. Universal Composability Framework

To capture the concurrent executions, we employ the Universal Composability (UC) framework with gloable setup [11] to model the security and privacy of BlindChannel. Let $P=\{P_1,\ldots,P_n\}$ be the set of Turing machine that are executing the protocol. We assume that all the participated Turing machines are under static corruptions, which are fully controlled by the adversary $\mathcal{A}$ for the entire protocol execution. In order to capture the execution rounds, we also assume synchronous communication among participants and authenticated communication channels between users with guaranteed delivery.

Let E be the environment including everything outside the execution of the protocol. We denote Π a real protocol and $\mathcal{A}$ an adversary, $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ be the execution of Π by $\mathcal{A}$. We also denote $\mathcal{F}$ the ideal functionality, $\mathcal{S}$ the simulator, and $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ the distribution ensemble of the execution of $\mathcal{F}$ by $\mathcal{S}$. We then give the definition of Universal Composability in Definition 30.

Definition 30 (Universal Composability)**.** *A protocol Π UC-realizes an ideal functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{E}$ the ensembles $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

J.2. Ideal Functionality

For BlindChannel, there are unblind and blind parties in a channel. Namely, one party (the unblind) knows the transaction amount in each state within their channel, while the other (the blind) knows nothing about these transactions' amounts. We denote the unblind party by U and the blind party by B.

Blind Channel Syntax. Let γ be the BlindChannel with a tuple of attributes $(\gamma.id,\gamma.users,\gamma.cash,\gamma.st)$, where

$\gamma.id \in \{0,1\}^*$ is the channel identifier, $\gamma.user \in \{U, B\}$ defines the identities of channel users, $\gamma.cash \in \mathbb{R}^{\leq 0}$ represents the capacity of γ, and $\gamma.st = (\theta_1, \ldots, \theta_n)$ is the state of γ composed of a list of *outputs*. Each outputs θ has two attributes: the value $\theta.cash \in \mathbb{R}^{\leq 0}$ representing the amount of coins and the function $\theta.\varphi : \{0,1\}^* \to \{0,1\}$ defining the spending condition.

Before the formalization, we introduce some security properties required by BlindChannel. BlindChannel inherits all properties form Generalized Channel with an additional properties Privacy.

Consensus on creation. A BlindChannel, γ, is successfully created only if both parties $\gamma.users$ in the channel agree with the creation.

Consensus on update. A blind channel γ is successfully updated only if both parties in $\gamma.users$ agree with the update. Moreover, parties in $\gamma.users$ reach agreement whether the update is successful or not after an a-priori bounded number of rounds.

Optimistic update. If both parties in $\gamma.users$ are honest, the update procedure takes a constant number of round (independent of the blockchain delay Δ).

Finality with punish. An honest party $P \in \gamma.users$ 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 $\gamma.cash$ coins.

Privacy. Only the unblind party knows the amount of each payment, while the blind party knows nothing but the amount is correct.

We assume our protocol is executed among a fixed set $\mathcal{P} = \{P_1, \ldots, P_n\}$. Let $\mathcal{A}$ be an adversary who can fully control a party $P_i \in \mathcal{P}$ for the entire protocol execution. All participants are connected to the environment $\mathcal{E}$, which represents anything "external" to the current protocol execution. We also assume a synchronized network, so that protocol is executed in round with the an authenticated communication channel and guaranteed delivery. We assume that local computation and any additional communication (such as messages conveyed between $\mathcal{A}$ and $\mathcal{E}$) take zero round.

Since Blind Channel are constructed based on Generalized Channel [3], our model inherits some notations from them. First, we denote the global ideal ledger functionality $\mathcal{L}(\Delta, \Sigma)$, where Δ is an upper bound of the number of blocks that takes to publish a transaction, Σ is the signature scheme. To generalize BlindChannel's construction, we use T to denote the upper bounds the maximal number of consecutive off-chain communication rounds between channel users, and k be the number of ways the channel state $\gamma.st$ can be published on the ledger. Let P be one of the channel parties and Q be the other one, where $P \in \{U, B\}$ and $Q \in \{U, B\} \setminus P$.

(Simplified) Ideal Functionality $\mathcal{L}(\Delta, \Sigma)$

This functionality keeps the list of (the public keys of) the parties in PKI, and stores all the published transactions in the ledger. A transaction is validated with respect to the owners signature via Σ, and possible other conditions. Once a valid transaction is posted, it takes at most Δ rounds to be added in the ledger.

Register: Upon $(\text{register}, \mathsf{pk}_P) \overset{\tau_0}{\hookleftarrow} P$, add (pk_P, P) to PKI.

Post a transaction: Upon $(\text{post}, \mathsf{tx}) \overset{\tau_0}{\hookleftarrow} P$ where $P \in \mathsf{PKI}$ and tx is valid, publish tx on the ledger $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + \Delta$.

As a global setup, we employ a simplified version of global UC framework (GUC) expanding the basic UC framework to explicitly model the security of our architecture on off-chain channels. To capture the desired functionalities, we construct the ideal functionality of BlindChannel $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$. The core construction of $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ is extended from [3], since our model is constructed based on them.

Create. A transaction tx is expected to be recorded on $\mathcal{L}$ within time T after $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ received messages $(\texttt{CREATE}, \gamma, \mathsf{tid}_\mathsf{P}) \overset{\tau_0}{\hookleftarrow} \mathsf{P}$ from both channel parties $\gamma.\mathsf{users}$. tx spends sources from both channel parties, and contains at least one output $\gamma.cash$. If it is true, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ stores the channel γ and transaction tx in Γ and informs both parties about the successful channel creation via the message $\texttt{CREATED}$. Since a successfully channel creation requires $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ to receive the $\texttt{CREATE}$ messages from both parties, the property "consensus on creation" can be guaranteed.

Close. We consider two scenarios of channel closure:

First, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ received $(\texttt{CLOSE}, \mathsf{id})$ from both parties within T, which indicates that both channel parties agree to close the channel identifier id. This scenario is regarded as the "peaceful closure". If a transaction tx_1, whose input is $\mathsf{tx}.\mathsf{txid}$ and output is the latest channel state $\gamma.st$, appears on ledger within $\tau_0 + \Delta$, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ sends $(\texttt{CLOSED}, \mathsf{id})$ to both parties; otherwise, sends $(\texttt{ERROR})$ to both parties.

Second, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ received only one $(\texttt{CLOSE}, \mathsf{id})$ message within T, the $\texttt{ForceClose}$ subprocedure will be executed in round $\tau_0 + T$. Let $S := \{\gamma.\mathsf{st} \mid \gamma \in X\}$, if tx has been spent and there is a transaction, whose output contains S, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ sends $(\texttt{CLOSED}, \mathsf{id})$ to both parties.

Update. There are two phases in a channel update: 1) prepare phase, and 2) revocation phase[11]. Prepare phase is for both parties to negotiate some auxiliary information on a payment, such as the estimated time for processing a payment and proof on the current payment to B, since he cannot see the payment amount. The entire update are processed as follows:

BlindChannel update is always triggered by U by sending a message $(\texttt{UPDATE}, \mathsf{id}, \vec{\theta}, t_{stp}, (\mathsf{Y}_\gamma, \mathsf{w}_\gamma))$ to $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$

11. Actually, in the formal BlindChannel protocol, the update follows the lock-then-pay rule, thus we have four phases to update a channel: 1) preparing for the state i, 2) revoking for the state $i-1$, 3) preparing for the state $i+1$, and 4) revoking for the state i)

at time τ_0, where id is the identifier of the channel to be updated, $\vec{\theta}$ is the newest state, t_{stp} is the required number of round for this update among channel parties, and $(\mathsf{Y}_\gamma, \mathsf{w}_\gamma)$ is a pair of statement and witness satisfying the relation $\mathcal{R}$.

Prepare phase. The prepare phase then starts when $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ receives a vector of transaction identifiers $\vec{\mathsf{tid}} = (\mathsf{tid}_1, \ldots, \mathsf{tid}_k)$ from the simulator $\mathcal{S}$, where k denotes the number of ways the channel state $\gamma.\mathsf{st}$ can be published on the ledger. Functionality $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ then sends $(\texttt{UPDATE–REQ}, \mathsf{id}, t_{stp})$ to B and $(\texttt{SETUP}, \mathsf{id}, \vec{\mathsf{tid}}, \vec{\theta})$ to U. If received $(\texttt{SETUP–OK}, \mathsf{id})$ from U within $\tau_0 + T + t_{stp}$, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ sends $(\texttt{SETUP–OK}, \mathsf{id})$ to B, else B can stop and reject this close channel request or $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ stores the two channel states $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{tx})$ and $\texttt{ForceClose}(\mathsf{id})$. Since the blind party B cannot access the channel state in the BlindChannel setting, we should take some additional steps to guarantee the Privacy property: B sends a message $(\texttt{VERIFY–REQ}, \mathsf{id}, \mathsf{Y}'_\gamma)$ to $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, who checks if $\mathsf{Y}'_\gamma = \mathsf{Y}'_\gamma$ and $\mathcal{R}(\mathsf{Y}_\gamma, \mathsf{w}_\gamma)$ holds, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ sends $(\texttt{VERIFIED}, \mathsf{id})$ to B. Otherwise, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ sends $(\texttt{NOT–VERIFIED}, \mathsf{id})$ to B. If $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ received $(\texttt{UPDATE–OK}, \mathsf{id})$ from B, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ forwards the same message $(\texttt{UPDATE–OK}, \mathsf{id})$ to U. Otherwise, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ stops this channel update or executes the $\texttt{ForceClose}$. In the optimistic case, U receives an $\texttt{UPDATE–OK}$ message from $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ within $5T + t_{stp}$.

Revocation phase. Revocation phase is also started by $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ receiving a $(\texttt{REVOKE}, \mathsf{id})$ message from U, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ then sends $(\texttt{REVOKE–REQ}, \mathsf{id})$ to B. If $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ did not receive $(\texttt{REVOKE}, \mathsf{id})$ message from U until $\tau_0 + 4T$, $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ sets $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{tx})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop. If received $(\texttt{REVOKE}, \mathsf{id})$ from B, sets $\Gamma(\mathsf{id}) := (\{\gamma'\}, \mathsf{tx})$ and sends $(\texttt{UPDATED}, \mathsf{id})$ to both parties within $\tau_0 + 5T$. Then channel γ is updated to the new channel state. Since a BlindChannel can only be updated successfully if $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ received messages $(\texttt{REVOKE}, \mathsf{id})$ from both parties, the "consensus on update" is guaranteed.

Punish. To prevent the honest party from losing coins, there is also a *punishment* mechanism in BlindChannel model for satisfying the property "instant finality with punish". $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ checks at each round whether the funding transaction (on-chain) is spent or not. If so, one of the following is anticipated to occur: 1) a punish transaction, which allocates all coins within the channel to the honest party, appears on $\mathcal{L}$, or 2) a transaction, which allocates coins according to the latest channel state $\gamma.\mathsf{st}$, appears on $\mathcal{L}$. 3) the message $(\texttt{ERROR})$ returns to both parties. The security property "instant finality with punish" can be guaranteed, if no $\texttt{ERROR}$ was returned.

Ideal Functionality of BlindChannel $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$

Upon $(\texttt{CREATE}, \gamma, \mathsf{tid}_\mathsf{P}) \overset{\tau_0}{\hookleftarrow} \mathsf{P}$, distinguish:

Both agreed: If already received $(\texttt{CREATE}, \gamma, \mathsf{tid}_\mathsf{Q}) \overset{\tau}{\hookleftarrow} \mathsf{Q}$, where $\tau_0 - \tau \leq T$: If tx s.t. $\mathsf{tx.In} = (\mathsf{tid}_\mathsf{U}, \mathsf{tid}_\mathsf{B})$ and $\mathsf{tx.Out} = (\gamma.\mathsf{cash}, \varphi)$, for some φ, appears on $\mathcal{L}$ in round $\tau_1 \leq \tau + \Delta + T$, set $\Gamma(\gamma.\mathsf{id}) := (\{\gamma\}, \mathsf{tx})$ and $(\texttt{CREATED}, \gamma.\mathsf{id}) \overset{\tau_1}{\hookrightarrow} \gamma.\mathsf{users}$. Else stop.

Wait for Q**:** Else wait if $(\texttt{CREATE}, \mathsf{id}) \overset{\tau \leq \tau_0 + T}{\hookleftarrow} Q$ (in that case "Both agreed" option is executed). If such message is not received, stop.

Upon $(\texttt{UPDATE}, \mathsf{id}, \vec{\theta}, t_{stp}, (\mathsf{Y}_\gamma, \mathsf{w}_\gamma)) \overset{\tau_0}{\hookleftarrow} U$:

1) Parse $(\{\gamma\}, \mathsf{tx}) := \Gamma(\mathsf{id})$, set $\gamma' := \gamma$, $\gamma'.\mathsf{st} := \vec{\theta}$;
2) In round $\tau_1 \leq \tau_0 + T$, let $\mathcal{S}$ define $\vec{\mathsf{tid}}$ s.t. $|\vec{\mathsf{tid}}| = k$. Then $(\texttt{UPDATE–REQ}, \mathsf{id}, \vec{\mathsf{tid}}, t_{stp}) \overset{\tau_1}{\hookrightarrow} B$ and $(\texttt{SETUP}, \mathsf{id}, \vec{\mathsf{tid}}, \vec{\theta}) \overset{\tau_1}{\hookrightarrow} U$.
3) If $(\texttt{SETUP–OK}, \mathsf{id}) \overset{\tau_2 \leq \tau_1 + t_{stp}}{\hookleftarrow} U$, then $(\texttt{SETUP–OK}, \mathsf{id}) \overset{\tau_3 \leq \tau_2 + T}{\hookrightarrow} B$. Else stop.
4) The verification procedure is parameterized by a binary relation $\mathcal{R}$:
   If $(\texttt{VERIFY–REQ}, \mathsf{id}, \mathsf{Y}'_\gamma) \overset{\tau_3}{\hookleftarrow} B$:
   - If $\mathsf{Y}'_\gamma = \mathsf{Y}_\gamma$ and $\mathcal{R}(\mathsf{Y}_\gamma, \mathsf{w}_\gamma)$ holds, $(\texttt{VERIFIED}, \mathsf{id}) \overset{\tau_4 \leq \tau_3 + T}{\hookrightarrow} B$.
   - Else $(\texttt{NOT–VERIFIED}, \mathsf{id}) \overset{\tau_4 \leq \tau_3 + T}{\hookrightarrow} B$ and stop.

   Else stop.
5) If $(\texttt{UPDATE–OK}, \mathsf{id}) \overset{\tau_4}{\hookleftarrow} B$, $(\texttt{UPDATE–OK}, \mathsf{id}) \overset{\tau_5 \leq \tau_4 + T}{\hookrightarrow} U$. Else distinguish:
   - If B honest or if instructed by $\mathcal{S}$, stop *(reject)*.
   - Else set $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{tx})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop.
6) If $(\texttt{REVOKE}, \mathsf{id}) \overset{\tau_5}{\hookleftarrow} U$, send $(\texttt{REVOKE–REQ}, \mathsf{id}) \overset{\tau_6 \leq \tau_5 + T}{\hookrightarrow} B$. Else set $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{tx})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop.
7) If $(\texttt{REVOKE}, \mathsf{id}) \overset{\tau_5}{\hookleftarrow} B$, set $\Gamma(\mathsf{id}) := (\{\gamma'\}, \mathsf{tx})$, $(\texttt{UPDATED}, \mathsf{id}) \overset{\tau_6 \leq \tau_5 + T}{\hookrightarrow} \gamma.\mathsf{users}$ and stop (accept). Else set $\Gamma(\mathsf{id}) := (\{\gamma, \gamma'\}, \mathsf{tx})$, run $\texttt{ForceClose}(\mathsf{id})$ and stop.

Upon $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau_0}{\hookleftarrow} P$, distinguish:

Both agreed: If already received $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau}{\hookleftarrow} Q$, where $\tau_0 - \tau \leq T$, run $\texttt{ForceClose}(\mathsf{id})$ unless both parties are honest. In this case, let $(\{\gamma\}, \mathsf{tx}) := \Gamma(\mathsf{id})$ and distinguish:

- If tx_1, with $\mathsf{tx}_1.\mathsf{In} = \mathsf{tx.txid}$ and $\mathsf{tx}_1.\mathsf{Out} = \gamma.\mathsf{st}$ appears on $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + \Delta$, set $\Gamma(\mathsf{id}) := \bot$, send $(\texttt{CLOSED}, \mathsf{id}) \overset{\tau_1}{\hookrightarrow} (\gamma.\mathsf{users})$ and stop.
- Else output $(\texttt{ERROR}) \overset{\tau_0 + \Delta}{\hookrightarrow} \gamma.\mathsf{users}$ and stop.

Wait for Q**:** Else wait if $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau \leq \tau_0 + T}{\hookleftarrow} Q$ (in that case "Both agreed" option is executed). If such message is not received, run $\texttt{ForceClose}(\mathsf{id})$ in round

$\tau_0 + T$.

At the end of every round τ_0: For each $\mathsf{id} \in \{0,1\}^*$ s.t. $(X, \mathsf{tx}) := \Gamma(\mathsf{id}) \neq \bot$,

- For U, check if $\mathcal{L}$ contains tx_1 such that $\mathsf{tx}_1.\mathsf{In} = \mathsf{tx}.\mathsf{txid}$.
- For B, check if $\mathcal{L}$ contains tx_1 such that $\mathsf{tx}_1.\mathsf{In} = \mathsf{tx}.\mathsf{txid}$, or tx_2 such that $\mathsf{tx}_2.\mathsf{In} = \mathsf{tx}_1.\mathsf{txid}$.

If yes, then define $S := \{\gamma.\mathsf{st} \mid \gamma \in X\}$, $\tau := \tau_0 + 2\Delta$ and distinguish:
Close: If tx_3 s.t. $\mathsf{tx}_3.\mathsf{In} = \mathsf{tx}_2.\mathsf{txid}$ appears on $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + \Delta$, where $\mathsf{tx}_3.\mathsf{Out} \in S$, set $\Gamma(\mathsf{id}) := \bot$ and $(\mathtt{CLOSED}, \mathsf{id}) \overset{\tau_1}{\hookrightarrow} (\gamma.P, \gamma.Q)$ if not sent yet.

Punish: If tx_2 s.t. $\mathsf{tx}_2.\mathsf{In} = \mathsf{tx}_1.\mathsf{txid}$ and $\mathsf{tx}_2.\mathsf{Out} = (\gamma.\mathsf{cash}, \mathtt{One\text{–}Sig}_{\mathsf{pk_P}})$ appears on $\mathcal{L}$ in round $\tau_1 \leq \tau$, for P honest, set $\Gamma(\mathsf{id}) := \bot$, $(\mathtt{PUNISHED}, \mathsf{id}) \overset{\tau_1}{\hookrightarrow} P$ and stop.

Error: Else $(\mathtt{ERROR}) \overset{\tau}{\hookrightarrow} \gamma.\mathsf{users}$.
$\mathtt{ForceClose}(\mathsf{id})$:
Let τ_0 be the current round and $(X, \mathsf{tx}) := \Gamma(\mathsf{id})$. If within Δ rounds tx is still unspent on $\mathcal{L}$, then $(\mathtt{ERROR}) \overset{\tau_0+\Delta}{\hookrightarrow} \gamma.\mathsf{users}$ and stop. Note that otherwise, message $m \in \{\mathtt{CLOSED}, \mathtt{PUNISHED}, \mathtt{ERROR}\}$ is output latest in round $\tau_0 + 4 \cdot \Delta + t_{timeout}$.

J.3. Formal Description of BlindChannel

The party $P \in \gamma_b.\mathsf{users}$ maintains a set Γ^P that contains data about all party P's channels. Party P also maintains a set Θ^P that contains all revoked commit transactions along with their corresponding revocation secrets.

Blind Channel Protocol π

Create

Party P upon $(\mathtt{INTRO}, \gamma, tid_P) \overset{t_0}{\hookleftarrow} \mathcal{E}$

1) Set $id := \gamma.\mathsf{id}$, generate $(R_P, r_P) \leftarrow \mathsf{Gen}$ and $(Y_P, y_P) \leftarrow \mathsf{Gen}$ and send $(\mathsf{createInfo}, id, tid_P, R_P, Y_P) \overset{t_0}{\hookrightarrow} Q$.
2) If $(\mathsf{createInfo}, id, tid_Q, R_Q, Y_Q) \overset{t_0+1}{\hookleftarrow} Q$, create:

$$[\mathtt{TX_{FU}}] := \mathtt{GenFund}((tid_P, tid_Q), \gamma)$$
$$[\mathtt{TX_{CM}}] := \mathtt{GenCommit}([\mathtt{TX_{FU}}].\mathsf{txid} \| 1, I_P, I_Q)$$
$$[\mathtt{TX}^{[2]}_{\mathtt{SP}}] := \mathtt{GenSplit2}([\mathtt{TX_{CM}}].\mathsf{txid} \| 1, \gamma.\mathsf{st})$$

for $I_P := (pk_P, R_P, Y_P)$ and $I_Q := (pk_Q, R_Q, Y_Q)$. Else stop.

3) Compute $\sigma^P_s := \mathsf{Sign}_{sk_P}([\mathtt{TX}^{[2]}_{\mathtt{SP}}])$ and $\hat{\sigma}^P_c := \mathsf{PreSign}_{sk_P}([\mathtt{TX_{CM}}], Y_Q)$ and send $(\mathsf{createCom}, id, \sigma^P_s, \hat{\sigma}^P_c) \overset{t_0+1}{\hookrightarrow} Q$.
4) If $(\mathsf{createCom}, id, \sigma^Q_s, \hat{\sigma}^Q_c)) \overset{t_0+2}{\hookleftarrow} Q$ s.t. $\mathsf{PreVf}_{pk_Q}([\mathtt{TX_{CM}}], Y_P; \hat{\sigma}^Q_c) = 1$ and $\mathsf{Vf}_{pk_Q}([\mathtt{TX}^{[2]}_{\mathtt{SP}}]; \sigma^Q_s) = 1$, compute $\sigma^P_f := \mathtt{Sign}_{sk_P}([\mathtt{TX_{FU}}])$ and send $(\mathsf{createFund}, id, \sigma^P_f) \overset{t_0+2}{\hookrightarrow} Q$. Else stop.
5) If $(\mathsf{createFund}, id, \sigma^Q_f) \overset{t_0+3}{\hookleftarrow} Q$ s.t. $\mathsf{Vf}_{pk_Q}([\mathtt{TX_{FU}}]; \sigma^Q_f) = 1$, set $\mathtt{TX_{FU}} := ([\mathtt{TX_{FU}}], \{\sigma^P_f, \sigma^Q_f\})$ and post $(\mathsf{post}, \mathtt{TX_{FU}}) \overset{t_0+3}{\hookrightarrow} \mathcal{L}$. Else stop.
6) If $\mathtt{TX_{FU}}$ is accepted by $\mathcal{L}$ in round $t_1 \leq t_0 + 3 + \Delta$, set $\mathtt{TX_{CM}} := ([\mathtt{TX_{CM}}], \{\mathsf{Sign}_{sk_P}([\mathtt{TX_{CM}}]), \mathsf{Adapt}(\hat{\sigma}^Q_c, y_P)\})$ and $\mathtt{TX}^{[2]}_{\mathtt{SP}} = ([\mathtt{TX}^{[2]}_{\mathtt{SP}}], \{\sigma^P_s, \sigma^Q_s\})$, store $\Gamma^U(\gamma.id) := (\gamma, \mathtt{TX_{FU}}, (\mathtt{TX_{CM}}, r_U, R_B, Y_B, \hat{\sigma}^U_c), \mathtt{TX}^{[2]}_{\mathtt{SP}}, \bot)$ if $P = U$ and $\Gamma^B(\gamma.id) := (\gamma, \mathtt{TX_{FU}}, (\mathtt{TX_{CM}}, r_B, R_U, Y_U, \hat{\sigma}^B_c), \bot, \bot)$ otherwise and send $(\mathtt{CREATED}, id) \overset{t_1}{\hookrightarrow} \mathcal{E}$. Else stop.

Update

Party U upon $(\mathtt{UPDATE}, id, v, t_{timeout}) \overset{t_0}{\hookleftarrow} \mathcal{E}$

1) Generate $(R_U, r_U) \leftarrow \mathsf{Gen}$ and $(Y_U, y_U) \leftarrow \mathsf{Gen}$ and send $(\mathsf{updateReq}, id, R_U, Y_U, t_{timeout}) \overset{t_0}{\hookrightarrow} B$.

Party B upon $(\mathsf{updateReq}, id, R_U, Y_U, t_{timeout}) \overset{\tau_0}{\hookleftarrow} U$

2) Generate $(R_B, r_B) \leftarrow \mathsf{Gen}$ and $(Y_B, y_B) \leftarrow \mathsf{Gen}$ and send $(\mathsf{updateInfo}, id, R_B, Y_B) \overset{\tau_0}{\hookrightarrow} U$.

Party U upon $(\mathsf{updateInfo}, id, R_B, Y_B) \overset{t_0+2}{\hookleftarrow} B$

3) Extract γ and $\mathtt{TX_{FU}}$ from $\Gamma^U(id)$ and create:

$$[\mathtt{TX_{CM}}] := \mathtt{GenCommit}([\mathtt{TX_{FU}}].\mathsf{txid} \| 1, I_U, I_B)$$
$$[\mathtt{TX}^{[3]}_{\mathtt{SP}}] := \mathtt{GenSplit3}([\mathtt{TX_{CM}}].\mathsf{txid} \| 1, \gamma.\mathsf{st}, v, I)$$
$$[\mathtt{TX_{AED}}] := \mathtt{GenAED}([\mathtt{TX}^{[3]}_{\mathtt{SP}}].\mathsf{txid} \| 3, |v|, pk)$$
$$[\mathtt{TX_{TO}}] := \mathtt{GenTO}([\mathtt{TX}^{[3]}_{\mathtt{SP}}].\mathsf{txid} \| 3, |v|, t_{timeout}, pk')$$

for $I = (pk_B, pk_U, pk'_U)$, $I_B := (pk_B, R_B, Y_B)$ and $I_U := (pk_U, R_U, Y_U)$ and with $pk := pk_B$ and $pk' := pk_U$ if $v > 0$ (or equivalently if U is the payer) and $pk := pk_U$ and $pk' := pk_B$ otherwise (or equivalently if U is the payee). Else stop.
4) Compute $(\mathsf{com_s}, \mathsf{decom_s}) \leftarrow \mathsf{P_{COM}}([\mathtt{TX}^{[3]}_{\mathtt{SP}}])$. If U is the payer, compute $(\mathsf{com_t}, \mathsf{decom_t}) \leftarrow \mathsf{P_{COM}}([\mathtt{TX_{TO}}])$. Otherwise, compute $(\mathsf{com_a}, \mathsf{decom_a}) \leftarrow \mathsf{P_{COM}}([\mathtt{TX_{AED}}])$.
5) Compute π_s, π_a and π_t. If U is the payer, compute $\hat{\sigma}^U_a := \mathsf{PreSign}_{sk_U}([\mathtt{TX_{AED}}], Y_s)$ and $h = \mathsf{SigHash}([\mathtt{TX_{AED}}])$, and send $(\mathsf{updateProofU}, id, \pi_\mathsf{s},$

$\pi_{\mathsf{a}}, \pi_{\mathsf{t}}, \hat{\sigma}_a^U, h, \mathsf{com}_s, \mathsf{com}_t))$ $\xleftrightarrow{t_0+2}$ B . Otherwise, compute $\sigma_t^U := \mathsf{Sign}_{sk_U}([\mathtt{TX_{TO}}])$ and $h = \mathsf{SigHash}([\mathtt{TX_{TO}}])$ and send $(\text{updateProofU}, id, \pi_{\mathsf{s}}, \pi_{\mathsf{a}}, \pi_{\mathsf{t}}, \sigma_t^U, h, \mathsf{com}_s, \mathsf{com}_a) \xleftrightarrow{t_0+2} B$.

Party B

6) Extract γ and $\mathtt{TX_{FU}}$ from $\Gamma^B(id)$ and create:

$$[\mathtt{TX_{CM}}] := \mathtt{GenCommit}([\mathtt{TX_{FU}}].\mathsf{txid} \| 1, I_U, I_B) \tag{1}$$

for $I_B := (pk_B, R_B, Y_B)$ and $I_U := (pk_U, R_U, Y_U)$. If B is the payee and $(\text{updateProofU}, id, \pi_{\mathsf{s}}, \pi_{\mathsf{a}}, \pi_{\mathsf{t}}, \hat{\sigma}_a^U, h, \mathsf{com}_s, \mathsf{com}_t) \xleftrightarrow{\tau_0+2} U$, s.t. $\pi_{\mathsf{s}}, \pi_{\mathsf{a}}, \pi_{\mathsf{t}}, \hat{\sigma}_a^U$ are valid, or B is the payer and $(\text{updateProofU}, id, \pi_{\mathsf{s}}, \pi_{\mathsf{a}}, \pi_{\mathsf{t}}, \sigma_t^U, h, \mathsf{com}_s, \mathsf{com}_a) \xleftrightarrow{\tau_0+2} U$, s.t. $\pi_{\mathsf{s}}, \pi_{\mathsf{a}}, \pi_{\mathsf{t}}, \sigma_t^U$ are valid, then create $\hat{\sigma}_c^B := \mathsf{PreSign}_{sk_B}([\mathtt{TX_{CM}}], Y_U)$, $\sigma_s^{*B} := \mathsf{BlindSig}_{sk_B}(\mathsf{com}_s)$. Else stop. If B is the payee, compute $\sigma_t^{*B} := \mathsf{BlindSig}_{sk_B}(\mathsf{com}_{\mathsf{t}})$ and send $(\text{updateComB}, id, \hat{\sigma}_c^B, \sigma_s^{*B}, \sigma_t^{*B}) \xleftrightarrow{\tau_0+2} U$. Else compute $\hat{\sigma}_a^B := \mathsf{BAS.Sign}_{1sk_B}(\mathsf{com}_{\mathsf{a}}, Y_s)$ and send $(\text{updateComB}, id, \hat{\sigma}_c^B, \sigma_s^{*B}, \hat{\sigma}_a^B) \xleftrightarrow{\tau_0+2} U$.

Party U

7) If U is the payer and $(\text{updateComB}, id, \hat{\sigma}_c^B, \sigma_s^{*B}, \sigma_t^{*B}) \xleftrightarrow{t_0+4} B$ s.t. $\mathsf{PreVf}_{pk_B}([\mathtt{TX_{CM}}], Y_U; \hat{\sigma}_c^B) = 1$ and $\sigma_s^{*B}, \sigma_t^{*B}$ are valid, or U is the payee and $(\text{updateComB}, id, \hat{\sigma}_c^B, \sigma_s^{*B}, \hat{\sigma}_a^B) \xleftrightarrow{t_0+4} B$ s.t. $\mathsf{PreVf}_{pk_B}([\mathtt{TX_{CM}}], Y_U; \hat{\sigma}_c^B) = 1$ and $\sigma_s^{*B}, \hat{\sigma}_a^B$ are valid, then output $(\mathtt{UPDATE-OK}, id) \xleftrightarrow{t_0+4} \mathcal{E}$. Else stop.
8) If $(\mathtt{UPDATEB}, id) \xleftrightarrow{t_0+4} \mathcal{E}$, continue. Else send $(\text{updateNotOK}, id, r_U) \xleftrightarrow{t_0+4} B$ and stop.
9) If U is the payer, then sign $\sigma_t^U := \mathsf{Sign}_{sk_U}([\mathtt{TX_{TO}}])$, compute $\sigma_t^B := \mathsf{UnBlindSig}(\sigma_t^{*B}, \mathsf{decom}_{\mathsf{t}})$, set $\mathtt{TX_{TO}} := ([\mathtt{TX_{TO}}], \{\sigma_t^U, \sigma_t^B\})$. Else, sign $\sigma_a^U := \mathsf{Sign}_{sk_U}([\mathtt{TX_{AED}}])$.
10) Sign $\sigma_s^U := \mathsf{Sign}_{pk_U}([\mathtt{TX_{SP}^{[3]}}])$, compute $\sigma_s^B = \mathsf{UnBlindSig}(\sigma_s^{*B}, \mathsf{decom}_{\mathsf{s}})$, set $\mathtt{TX_{SP}^{[3]}} := ([\mathtt{TX_{SP}^{[3]}}], \{\sigma_s^U, \sigma_s^B\})$, sign $\sigma_c^U := \mathsf{Sign}_{sk_U}([\mathtt{TX_{CM}}])$, compute $\sigma_c^B := \mathsf{Adapt}(\hat{\sigma}_c^B, y_U)$, set $\mathtt{TX_{CM}} := ([\mathtt{TX_{CM}}], \{\sigma_c^U, \sigma_c^B\})$, sign $\hat{\sigma}_c^U := \mathsf{PreSign}_{sk_U}([\mathtt{TX_{CM}}], Y_B)$ and send $(\text{updateComU}, id, \hat{\sigma}_c^U) \xleftrightarrow{t_0+4} B$.

Party B

11) In round $\tau_0 + 4$, distinguish the following cases:
- If $(\text{updateComU}, id, \hat{\sigma}_c^U) \xleftrightarrow{\tau_0+4} U$, s.t. $\mathsf{PreVf}_{pk_U}([\mathtt{TX_{CM}}], Y_B; \hat{\sigma}_c^U) = 1$, then sign $\sigma_c^B := \mathsf{Sign}_{sk_B}([\mathtt{TX_{CM}}])$, compute $\sigma_c^U := \mathsf{Adapt}(\hat{\sigma}_c^U, y_B)$, set $\mathtt{TX_{CM}} := ([\mathtt{TX_{CM}}], \{\sigma_c^U, \sigma_c^B\})$ and output $(\mathtt{UPDATE-OK}, id) \xleftrightarrow{\tau_0+4} \mathcal{E}$.
- If $(\text{updateNotOK}, id, r_U) \xleftrightarrow{\tau_0+4} B$, s.t. $(R_U, r_U) \in R$, then add $\Theta^B(id) := \Theta^B(id) \cup ([\mathtt{TX_{CM}}], r_U, Y_U, \hat{\sigma}_c^B)$ and stop.
- Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.

12) If $(\mathtt{REVOKE}, id) \xleftrightarrow{\tau_0+4} \mathcal{E}$, then parse $\Gamma^B(id)$ as $(\gamma, \mathtt{TX_{FU}}, (\overline{\mathtt{TX}}_{\mathtt{CM}}, \overline{r}_B, \overline{R}_U, \overline{Y}_U, \overline{\hat{\sigma}}_c^B), \bot, \bot)$, update the channel space as $\Gamma^B(\gamma.id) := (\gamma, \mathtt{TX_{FU}}, (\mathtt{TX_{CM}}, r_B, R_U, Y_U, \hat{\sigma}_c^B), \sigma, t_{timeout})$ with $\sigma := \hat{\sigma}_a^U$ if B is the payee and $\sigma := \sigma_t^U$ otherwise and send $(\text{revokeB}, id, \overline{r}_B) \xleftrightarrow{\tau_0+4} U$. Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.

Party U

13) Parse $\Gamma^U(id)$ as $(\gamma, \mathtt{TX_{FU}}, (\overline{\mathtt{TX}}_{\mathtt{CM}}, \overline{r}_U, \overline{R}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U), \bot, \bot)$. If $(\text{revokeB}, id, \overline{r}_B) \xleftrightarrow{t_0+6} B$, s.t. $(\overline{R}_B, \overline{r}_B) \in R$, $(\mathtt{REVOKE-REQ}, id) \overset{t_0+6}{\hookrightarrow} \mathcal{E}$. Else execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
14) If $(\mathtt{REVOKE}, id) \xleftrightarrow{t_0+6} \mathcal{E}$, continue. Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
15) Set $\Theta^U(id) := \Theta^U(id) \cup ([\overline{\mathtt{TX}}_{\mathtt{CM}}], \overline{r}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U)$. If U is the payer, then set $\Gamma^U(\gamma.id) := (\gamma, \mathtt{TX_{FU}}, (\mathtt{TX_{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \mathtt{TX_{SP}^{[3]}}, \mathtt{TX_{TO}})$. Otherwise, set $\Gamma^U(\gamma.id) := (\gamma, \mathtt{TX_{FU}}, (\mathtt{TX_{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \mathtt{TX_{SP}^{[3]}}, ([\mathtt{TX_{AED}}], \{\sigma_a^U, \hat{\sigma}_a^B\})$. Send $(\text{revokeU}, id, \overline{r}_U) \xleftrightarrow{t_0+6} B$, $(\mathtt{UPDATED}) \xleftrightarrow{t_0+6} \mathcal{E}$.

Party B

16) If $(\text{revokeU}, id, \overline{r}_U) \xleftrightarrow{\tau_0+6} U$, s.t. $(\overline{R}_U, \overline{r}_U) \in R$, set $\Theta^B(id) := \Theta^B(id) \cup ([\overline{\mathtt{TX}}_{\mathtt{CM}}], \overline{r}_U, \overline{Y}_U, \overline{\hat{\sigma}}_c^B)$ and send $(\mathtt{UPDATED}, id) \xleftrightarrow{\tau_0+6} \mathcal{E}$. Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.
17) For the case where B is the payee, if $(\mathtt{ADAPT}, id) \xleftrightarrow{\tau_1} \mathcal{E}$ with $\tau_0 + 6 \leq \tau_1 \leq \tau_0 + T_1$, then send $(\text{adapt}, y_s, id) \xleftrightarrow{\tau_1} U$. Else set $\tau_1 = \tau_0 + T_1$ and continue. For the case where B is the payer, continue.

Party U

18) For the case where U is the payer, continue. For the case where U is the payee, if $(\mathtt{ADAPT}, id) \xleftrightarrow{t_1} \mathcal{E}$ with $t_0 + 6 \leq t_1 \leq t_0 + T_1'$, then $(\text{updateReq}, id, y_s) \xleftrightarrow{t_1} B$. Else set $t_1 = t_0 + T_1'$ and continue.

Party B

19) For the case where B is the payee, continue. For the case where B is the payer, if $(\text{updateReq}, id, y_s) \xleftrightarrow{\tau_1} U$ with $\tau_0 + 6 \leq \tau_1 \leq \tau_0 + T_1'$ s.t. $(Y_s, y_s) \in R$, continue. Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
20) For the case where B is the payee, continue. For the case where B is the payer, if $(\mathtt{UPDATE}, id) \xleftrightarrow{\tau_1} \mathcal{E}$, send $(\text{updateReq}, id) \xleftrightarrow{\tau_1} U$. Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.

Party U

21) For the case where U is the payer, if $(\text{adapt}, y_s, id) \xleftarrow{t_1} \mathcal{B}$ with $t_0 + 8 \leq t_1 \leq t_0 + 2 + T_1$, s.t. $(Y_s, y_s) \in R$ continue. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop. For the case where U is the payee, continue.
22) For the case where U is the payer, if $(\texttt{UPDATE}, id, \vec{\theta}) \xleftarrow{t_1} \mathcal{E}$, then set $t_2 := t_1$ and continue. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop. For the case where U is the payee, if $(\text{updateReq}, id) \xleftarrow{t_1+2} B$, then set $t_2 := t_1 + 2$ and continue. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop.
23) Generate $(R_U, r_U) \leftarrow \mathsf{Gen}$ and $(Y_U, y_U) \leftarrow \mathsf{Gen}$ and send $(\text{updateReq}, id, R_U, Y_U) \xhookrightarrow{t_2} B$.

Party B

24) If $(\text{updateReq}, id, R_U, Y_U) \xleftarrow{\tau_1+2} U$, Send $(\texttt{UPDATE} - \texttt{REQ}, id) \xhookrightarrow{\tau_1+2} \mathcal{E}$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
25) If $(\texttt{UPDATE} - \texttt{REQ} - \texttt{OK}, id) \xleftarrow{\tau_1+2} \mathcal{E}$, then Generate $(R_B, r_B) \leftarrow \mathsf{Gen}$ and $(Y_B, y_B) \leftarrow \mathsf{Gen}$ and send $(\text{updateInfo}, id, R_B, Y_B) \xhookrightarrow{\tau_1+2} U$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.

Party U

26) If $(\text{updateInfo}, id, R_B, Y_B) \xleftarrow{t_2+2} B$, create:

$$[\texttt{TX}_{\texttt{CM}}] := \texttt{GenCommit}([\texttt{TX}_{\texttt{FU}}].\texttt{txid} \| 1, I_U, I_B)$$
$$[\texttt{TX}_{\texttt{SP}}^{[2]}] := \texttt{GenSplit2}([\texttt{TX}_{\texttt{CM}}].\texttt{txid} \| 1, \vec{\theta})$$

for $I_B := (pk_B, R_B, Y_B)$ and $I_U := (pk_U, R_U, Y_U)$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
27) Compute $(\mathsf{com_s}, \mathsf{decom_s}) \leftarrow \mathsf{P_{COM}}([\texttt{TX}_{\texttt{SP}}^{[2]}])$.
28) Compute π_{s} and send $(\text{updateProofU}, id, \pi_{\mathsf{s}}, \mathsf{com_s}) \xhookrightarrow{t_2+2} \mathsf{B}$.

Party B

29) Create:

$$[\texttt{TX}_{\texttt{CM}}] := \texttt{GenCommit}([\texttt{TX}_{\texttt{FU}}].\texttt{txid} \| 1, I_U, I_B) \quad (2)$$

for $I_B := (pk_B, R_B, Y_B)$ and $I_U := (pk_U, R_U, Y_U)$. If $(\text{updateProofU}, id, \pi_{\mathsf{s}}, \mathsf{com_s}) \xleftarrow{\tau_1+4} \mathsf{U}$, s.t. $\pi_{\mathsf{s}}, \mathsf{com_s}$ are valid, then create $\sigma_s^{*B} := \mathsf{BlindSig}_{sk_B}(\mathsf{com_s})$ and $\hat{\sigma}_c^B := \mathsf{PreSign}_{sk_B}([\texttt{TX}_{\texttt{CM}}], Y_U)$ and send $(\text{updateComB}, id, \sigma_s^{*B}, \hat{\sigma}_c^B) \xhookrightarrow{\tau_1+4} U$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.

Party U

30) If $(\text{updateComB}, id, \sigma_s^{*B}, \hat{\sigma}_c^B) \xleftarrow{t_2+4} B$ s.t. σ_s^{*B}, $\hat{\sigma}_c^B$ are valid, output $(\texttt{UPDATE} - \texttt{OK}, id) \xhookrightarrow{t_1+4} \mathcal{E}$. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop.
31) If $(\texttt{UPDATE}, id) \xleftarrow{t_2+4} \mathcal{E}$, then create $\hat{\sigma}_c^U := \mathsf{PreSign}_{sk_U}([\texttt{TX}_{\texttt{CM}}], Y_B)$ and send $(\text{updateComU}, id, \hat{\sigma}_c^U) \xhookrightarrow{t_2+4} B$. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop.

Party B

32) If $(\text{updateComU}, id, \hat{\sigma}_c^U) \xleftarrow{\tau_1+6} U$, s.t. $\hat{\sigma}_c^U$ is valid, then output $(\texttt{UPDATE} - \texttt{OK}, id) \xhookrightarrow{\tau_1+6} \mathcal{E}$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
33) If $(\texttt{REVOKE}, id) \xleftarrow{\tau_1+6} \mathcal{E}$, then set $\texttt{TX}_{\texttt{CM}} := ([\texttt{TX}_{\texttt{CM}}], \{\mathsf{Sign}_{sk_B}([\texttt{TX}_{\texttt{CM}}]), \mathsf{Adapt}(\hat{\sigma}_c^U, y_B)\})$, parse $\Gamma^B(id)$ as $(\gamma, \texttt{TX}_{\texttt{FU}}, (\overline{\texttt{TX}}_{\texttt{CM}}, \overline{r}_B, \overline{R}_U, \overline{Y}_U, \overline{\hat{\sigma}}_c^B), \overline{\sigma}, \overline{t_{timeout}})$, update the channel space as $\Gamma^B(id) := (\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_B, R_U, Y_U, \hat{\sigma}_c^B), \perp, \perp)$ and send $(\text{revokeB}, id, \overline{r}_B) \xhookrightarrow{\tau_1+6} U$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.

Party U

34) If U is the payer, then Parse $\Gamma^U(id)$ as $(\gamma, \texttt{TX}_{\texttt{FU}}, (\overline{\texttt{TX}}_{\texttt{CM}}, \overline{r}_U, \overline{R}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U), \overline{\texttt{TX}}_{\texttt{SP}}^{[3]}, \overline{\texttt{TX}}_{\texttt{TO}})$. Else, Parse $\Gamma^U(id)$ as $(\gamma, \texttt{TX}_{\texttt{FU}}, (\overline{\texttt{TX}}_{\texttt{CM}}, \overline{r}_U, \overline{R}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U), \overline{\texttt{TX}}_{\texttt{SP}}^{[3]}, ([\overline{\texttt{TX}}_{\texttt{AED}}], \{\overline{\sigma}_a^U, \overline{\hat{\sigma}}_a^B\}))$. If $(\text{revokeB}, id, \overline{r}_B) \xleftarrow{t_2+6} B$, s.t. $(\overline{R}_B, \overline{r}_B) \in R$, $(\texttt{REVOKE} - \texttt{REQ}, id) \xhookrightarrow{t_2+6} \mathcal{E}$. Else execute the procedure $\texttt{ForceClose}^U(id)$ and stop.
35) If $(\texttt{REVOKE}, id) \xleftarrow{t_2+6} \mathcal{E}$, set $\Theta^U(id) := \Theta^U(id) \cup ([\overline{\texttt{TX}}_{\texttt{CM}}], \overline{r}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U)$. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop. Set $\texttt{TX}_{\texttt{CM}} := ([\texttt{TX}_{\texttt{CM}}], \{\mathsf{Sign}_{sk_U}([\texttt{TX}_{\texttt{CM}}]), \mathsf{Adapt}(\hat{\sigma}_c^B, y_U)\})$, Sign $\sigma_s^U := \mathsf{Sign}_{pk_U}([\texttt{TX}_{\texttt{SP}}^{[2]}])$, compute $\sigma_s^B = \mathsf{UnBlindSig}(\sigma_s^{*B}, \mathsf{decom_s})$, set $\texttt{TX}_{\texttt{SP}}^{[2]} := ([\texttt{TX}_{\texttt{SP}}^{[2]}], \{\sigma_s^U, \sigma_s^B\})$, set $\Gamma^U(id) := (\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \texttt{TX}_{\texttt{SP}}, \perp)$. Send $(\text{revokeU}, id, \overline{r}_U) \xhookrightarrow{t_2+6} B$, $(\texttt{UPDATED}) \xhookrightarrow{t_2+6} \mathcal{E}$ and stop.

Party B

36) If $(\text{revokeU}, id, \overline{r}_U) \xleftarrow{\tau_1+8} U$, s.t. $(\overline{R}_U, \overline{r}_U) \in R$, set $\Theta^B(id) := \Theta^B(id) \cup ([\overline{\texttt{TX}}_{\texttt{CM}}], \overline{r}_U, \overline{Y}_U, \overline{\hat{\sigma}}_c^B)$ and send $(\texttt{UPDATED}, id) \xhookrightarrow{\tau_1+8} \mathcal{E}$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.

Closure

Party U upon $(\texttt{CLOSE}, id) \xleftarrow{t_0} \mathcal{E}$

1) Extract $\texttt{TX}_{\texttt{FU}}$ and $\texttt{TX}_{\texttt{SP}}^{[2]}$ from $\Gamma^U(id)$ and create $[\overline{\texttt{TX}}_{\texttt{SP}}]$ as follows:

$$[\overline{\texttt{TX}}_{\texttt{SP}}] := \texttt{GenSplit2}(\texttt{TX}_{\texttt{FU}}.\texttt{txid} \| 1, \texttt{TX}_{\texttt{SP}}^{[2]}.\texttt{Output})$$

Compute $\overline{\sigma}_s^U := \mathsf{Sign}_{sk_P}([\overline{\texttt{TX}}_{\texttt{SP}}])$ and send

$(\text{closeReq}, \mathtt{TX}_{\mathtt{SP}}^{[2]}, [\overline{\mathtt{TX}}_{\mathtt{SP}}], \overline{\sigma}_s^U) \xhookrightarrow{t_0} B.$

Party B upon $(\mathtt{CLOSE}, id) \xhookleftarrow{\tau_0} \mathcal{E}$

2) Extract $\mathtt{TX}_{\mathtt{FU}}$ and $\mathtt{TX}_{\mathtt{CM}}$ from $\Gamma^B(id)$. If $(\text{closeReq}, \mathtt{TX}_{\mathtt{SP}}^{[2]}, [\overline{\mathtt{TX}}_{\mathtt{SP}}], \overline{\sigma}_s^U) \xhookleftarrow{\tau_0+1} U$ s.t. $\mathsf{Vf}_{pk_U}([\overline{\mathtt{TX}}_{\mathtt{SP}}]; \overline{\sigma}_s^U) = 1$, $[\overline{\mathtt{TX}}_{\mathtt{SP}}].\mathsf{Output} = \mathtt{TX}_{\mathtt{SP}}^{[2]}.\mathsf{Output}$, $\mathtt{TX}_{\mathtt{SP}}^{[2]}.\mathsf{Input} = \mathtt{TX}_{\mathtt{CM}}.\mathsf{txid} \| 1$ and $[\overline{\mathtt{TX}}_{\mathtt{SP}}].\mathsf{Input} = \mathtt{TX}_{\mathtt{FU}}.\mathsf{txid} \| 1$, continue. Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.
3) Parse $\mathtt{TX}_{\mathtt{SP}}^{[2]}$ as $([\mathtt{TX}_{\mathtt{SP}}^{[2]}], \{\sigma_s^U, \sigma_s^B\})$. If $\mathsf{Vf}_{pk_B}([\mathtt{TX}_{\mathtt{SP}}^{[2]}], \sigma_s^B) = 1$, then Sign $\overline{\sigma}_s^B := \mathsf{Sign}_{sk_B}([\overline{\mathtt{TX}}_{\mathtt{SP}}])$ set $\overline{\mathtt{TX}}_{\mathtt{SP}} := ([\overline{\mathtt{TX}}_{\mathtt{SP}}], \{\overline{\sigma}_s^U, \overline{\sigma}_s^B\})$, send $(\text{closeRes}, \overline{\sigma}_s^B) \xhookrightarrow{\tau_0+1} U$ and post $(\text{post}, \overline{\mathtt{TX}}_{\mathtt{SP}}) \xhookrightarrow{\tau_0+2} \mathcal{L}$. Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.
4) If in round $\tau_1 \leq \tau_0 + 2 + \Delta$, the transaction $\overline{\mathtt{TX}}_{\mathtt{SP}}$ is accepted by $\mathcal{L}$, set $\Gamma^B(id) := \perp$, $\Theta^B(id) := \perp$ and send $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

Party U

5) If $(\text{closeRes}, \overline{\sigma}_s^B) \xhookleftarrow{t_0+2} B$ s.t. $\mathsf{Vf}_{pk_B}([\overline{\mathtt{TX}}_{\mathtt{SP}}]; \overline{\sigma}_s^B) = 1$, then set $\overline{\mathtt{TX}}_{\mathtt{SP}} := ([\overline{\mathtt{TX}}_{\mathtt{SP}}], \{\overline{\sigma}_s^U, \overline{\sigma}_s^B\})$ and post $(\text{post}, \overline{\mathtt{TX}}_{\mathtt{SP}}) \xhookrightarrow{t_0+2} \mathcal{L}$. Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
6) If in round $t_1 \leq t_0 + 2 + \Delta$, the transaction $\overline{\mathtt{TX}}_{\mathtt{SP}}$ is accepted by $\mathcal{L}$, set $\Gamma^U(id) := \perp$, $\Theta^U(id) := \perp$ and send $(\mathtt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

Punish

Party P upon $(\mathtt{PUNISH}) \xhookleftarrow{t_0} \mathcal{E}$

For each $id \in \{0,1\}^*$, s.t. $\Gamma^P(id) \neq \perp$, extract $\Gamma^P(id)$ as $(\gamma, \mathtt{TX}_{\mathtt{FU}}, (\mathtt{TX}_{\mathtt{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \mathtt{TX}_{\mathtt{SP}}, x)$ if $P = U$ or $(\gamma, \mathtt{TX}_{\mathtt{FU}}, (\mathtt{TX}_{\mathtt{CM}}, r_B, R_U, Y_U, \hat{\sigma}_c^B), \sigma)$ otherwise. Check if $\mathtt{TX}_{\mathtt{FU}}$ is spent by any transaction $\mathtt{TX}$ s.t. $[\mathtt{TX}] \neq [\mathtt{TX}_{\mathtt{CM}}]$. If yes:

- Parse $\Theta^P(id) := \{([\mathtt{TX}_{\mathtt{CM}}^{(i)}], r_Q^{(i)}, Y_Q^{(i)}, \hat{\sigma}_c^{P(i)})\}_{i \in m}$ and find i s.t. $[\mathtt{TX}_{\mathtt{CM}}^{(i)}] = [\mathtt{TX}]$. Then, parse the witness as $(\sigma_c^P, \sigma_c^Q) := \mathtt{TX}.\mathsf{Witness}$ and set $y_Q^{(i)} := \mathsf{Ext}(\sigma_c^P, \hat{\sigma}_c^{P(i)}, Y_Q^{(i)})$.
- Define the body of the punishment transaction $[\mathtt{TX}_{\mathtt{PU}}]$ as:

$$\mathtt{TX}_{\mathtt{PU}}.\mathsf{Input} := \mathtt{TX}.\mathsf{txid} \| 1$$
$$\mathtt{TX}_{\mathtt{PU}}.\mathsf{Output} := \{(\gamma.\mathsf{cash}, pk_P)\}$$

- Compute $\sigma_p^y \leftarrow \mathsf{Sign}_{y_Q^{(i)}}([\mathtt{TX}_{\mathtt{PU}}])$, $\sigma_p^r \leftarrow \mathsf{Sign}_{r_Q^{(i)}}([\mathtt{TX}_{\mathtt{PU}}])$, $\sigma_p^P \leftarrow \mathsf{Sign}_{sk_P}([\mathtt{TX}_{\mathtt{PU}}])$, set $\mathtt{TX}_{\mathtt{PU}} := \{[\mathtt{TX}_{\mathtt{PU}}], \{\sigma_p^y, \sigma_p^r, \sigma_p^P\}\}$ and (post, $\mathtt{TX}_{\mathtt{PU}}$) $\xhookrightarrow{t_0} \mathcal{L}$.
- Let $\mathtt{TX}_{\mathtt{PU}}$ be accepted by $\mathcal{L}$ in round $t_1 \leq t_0 + \Delta$. Set $\Theta^P(id) := \perp$, $\Gamma^P(id) := \perp$ and output $(\mathtt{PUNISHED}, id) \xhookrightarrow{t_1} \mathcal{E}$.

If no:

Party B

If $\mathtt{TX}.\mathsf{Output}$ is still unspent in round $t_1 = t_0 + 2T$, then

- Define the body of the punishment transaction $[\mathtt{TX}_{\mathtt{PU}}]$ as:

$$\mathtt{TX}_{\mathtt{PU}}.\mathsf{Input} := \mathtt{TX}.\mathsf{txid} \| 1$$
$$\mathtt{TX}_{\mathtt{PU}}.\mathsf{Output} := \{(\gamma.\mathsf{cash}, pk_B)\}$$

Compute $\sigma_p^B \leftarrow \mathsf{Sign}_{sk_B}([\mathtt{TX}_{\mathtt{PU}}])$, set $\mathtt{TX}_{\mathtt{PU}} := \{[\mathtt{TX}_{\mathtt{PU}}], \sigma_p^B\}$ and $(\text{post}, \mathtt{TX}_{\mathtt{PU}}) \xhookrightarrow{t_0+2T} \mathcal{L}$.
- Let $\mathtt{TX}_{\mathtt{PU}}$ be accepted by $\mathcal{L}$ in round $t_1 \leq t_0 + 2T + \Delta$. Set $\Theta^B(id) := \perp$, $\Gamma^B(id) := \perp$ and output $(\mathtt{PUNISHED}, id) \xhookrightarrow{t_1} \mathcal{E}$.

Close

Party U

For each $id \in \{0,1\}^*$, s.t. $\Gamma^U(id) \neq \perp$, extract $\Gamma^U(id)$ as $(\gamma, \mathtt{TX}_{\mathtt{FU}}, (\mathtt{TX}_{\mathtt{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \mathtt{TX}_{\mathtt{SP}}, x)$. If $\mathtt{TX}_{\mathtt{FU}}$ is spent by a transaction $\mathtt{TX}$ s.t. $[\mathtt{TX}] = [\mathtt{TX}_{\mathtt{CM}}]$, then

- Post $(\text{post}, \mathtt{TX}_{\mathtt{SP}}) \xhookrightarrow{t_0+T} \mathcal{L}$.
- Let $\mathtt{TX}_{\mathtt{SP}}$ be accepted by $\mathcal{L}$ in round $t_1 \leq t_0 + T + \Delta$. If $x = \perp$, then set $\Theta^P(id) := \perp$, $\Gamma^P(id) := \perp$ and output $(\mathtt{CLOSED}, id) \xhookrightarrow{t_1} \mathcal{E}$ and stop.
- If U is the payer, set $\mathtt{TX}_{\mathtt{TO}} := x$ and $t_{timeout} := \mathtt{TX}_{\mathtt{TO}}.\mathsf{nLT}$ and post $(\text{post}, \mathtt{TX}_{\mathtt{TO}}) \xhookrightarrow{t_{timeout}} \mathcal{L}$. Else, parse x as $([\mathtt{TX}_{\mathtt{AED}}], \{\sigma_a^U, \hat{\sigma}_a^B\})$, possibly adapt $\hat{\sigma}_a^B$ to σ_a^B, set $\mathtt{TX}_{\mathtt{AED}} := ([\mathtt{TX}_{\mathtt{AED}}], \{\sigma_a^U, \sigma_a^B\})$ and post $(\text{post}, \mathtt{TX}_{\mathtt{AED}}) \xhookrightarrow{t_1} \mathcal{L}$.
- Let $\mathtt{TX}_{\mathtt{SP}}.\mathsf{Output}[2]$ (the HTLC output) be spent in round $t_2 \leq t_{timeout} + \Delta$. Output $(\mathtt{CLOSED}, id) \xhookrightarrow{t_2} \mathcal{E}$ and stop.

Party B

For each $id \in \{0,1\}^*$, s.t. $\Gamma^B(id) \neq \perp$, extract $\Gamma^B(id)$ as $(\gamma, \mathtt{TX}_{\mathtt{FU}}, (\mathtt{TX}_{\mathtt{CM}}, r_B, R_U, Y_U, \hat{\sigma}_c^B), \sigma, t_{timeout})$. If $\mathtt{TX}_{\mathtt{FU}}$ is spent by a transaction $\mathtt{TX}$ s.t. $[\mathtt{TX}] = [\mathtt{TX}_{\mathtt{CM}}]$, then

- If $\mathtt{TX}.\mathsf{Output}$ is spent in round $t_1 \leq t_0 + 2T$ by a transaction $\mathtt{TX}'$, then if $\sigma \neq \perp$:
  - If B is the payee, set $\theta := [\mathtt{TX}'].\mathsf{Output}[2]$ and $v := \theta.\mathsf{Cash}$ and create:

$$[\mathtt{TX}_{\mathtt{AED}}] := \mathtt{GenAED}([\mathtt{TX}'].\mathsf{txid} \| 3, v, pk_B) \quad (3)$$

, set $\hat{\sigma}_a^U := \sigma$ and possibly adapt $\hat{\sigma}_a^U$ to σ_a^U, sign $\sigma_a^B := \mathsf{Sign}_{sk_B}([\mathtt{TX}_{\mathtt{AED}}])$, create $\mathtt{TX}_{\mathtt{AED}} := ([\mathtt{TX}_{\mathtt{AED}}], \{\sigma_a^U, \sigma_a^B\})$, and post (post, $\mathtt{TX}_{\mathtt{AED}}$) $\xhookrightarrow{t_2 < t_{timeout} - \Delta} \mathcal{L}$.

– If B is the payer, set $\theta := [\mathtt{TX}'].\mathsf{Output}[2]$ and $v := \theta.\mathsf{Cash}$ and create:

$$[\mathtt{TX}_{\mathtt{TO}}] := \mathtt{GenTO}([\mathtt{TX}'].\mathsf{txid} \| 3, v, t_{timeout}, pk_B) \tag{4}$$

, set $\sigma_t^U := \sigma$, sign $\sigma_t^B := \mathsf{Sign}_{sk_B}([\mathtt{TX}_{\mathtt{TO}}])$, create $\mathtt{TX}_{\mathtt{TO}} := ([\mathtt{TX}_{\mathtt{TO}}], \{\sigma_t^U, \sigma_t^B\})$, and post (post, $\mathtt{TX}_{\mathtt{TO}}$) $\xhookrightarrow{t_{timeout}} \mathcal{L}$.

– Let $\mathtt{TX}_{\mathtt{SP}}.\mathsf{Output}[2]$ (the HTLC output) be spent in round $t \leq t_{timeout} + \Delta$. Output ($\mathtt{CLOSED}$, id) $\xhookrightarrow{t} \mathcal{E}$ and stop.

Else, Output $(\mathtt{CLOSED}, id) \xhookrightarrow{t_1} \mathcal{E}$ and stop.

Subprocedures

$\mathtt{GenFund}((tid_P, tid_Q), \gamma)$:

Return $[\mathtt{TX}_{\mathtt{FU}}]$ where $[\mathtt{TX}_{\mathtt{FU}}].\mathsf{Input} := (tid_P, tid_Q)$ and $[\mathtt{TX}_{\mathtt{FU}}].\mathsf{Output} := (\gamma.\mathsf{Cash}, \mathtt{Multi} - \mathtt{Sig}_{\gamma.\mathsf{Users}})$

$\mathtt{GenCommit}(tid, (pk_U, R_U, Y_U), (pk_B, R_B, Y_B))$:

Let $(c, \mathtt{Multi} - \mathtt{Sig}_{pk_U, pk_B}) := [\mathtt{TX}_{\mathtt{FU}}].\mathsf{Output}$ and denote:

$\varphi_1 := \mathtt{Multi} - \mathtt{Sig}_{R_B, Y_B, pk_U}$

$\varphi_2 := \mathtt{Multi} - \mathtt{Sig}_{R_U, Y_U, pk_B}$

$\varphi_3 := \mathtt{CheckRelative}_{\mathsf{T}} \wedge \mathtt{Multi} - \mathtt{Sig}_{pk_U, pk_B}$

$\varphi_4 := \mathtt{CheckRelative}_{\mathsf{2T}} \wedge \mathtt{Sig}_{pk_B}$

Return $[\mathtt{TX}]$ with $[\mathtt{TX}].\mathsf{Input} := tid$, $[\mathtt{TX}].\mathsf{Output} := (c, \varphi_1 \vee \varphi_2 \vee \varphi_3 \vee \varphi_4)$.

$\mathtt{GenSplit2}(tid, \vec{\theta})$:

Return $[\mathtt{TX}]$ where $\mathtt{TX}.\mathsf{Input} := tid$ and $\mathtt{TX}.\mathsf{Output} := \vec{\theta}$.

$\mathtt{GenSplit3}(tid, \vec{\theta}, v, (pk_B, pk_U, pk'_U))$:

Return $[\mathtt{TX}]$ where $\mathtt{TX}.\mathsf{Input} := tid$ and $\mathtt{TX}.\mathsf{Output}[0] := (\vec{\theta}[0].\mathsf{Cash} - v, \mathtt{One} - \mathtt{Sig}_{pk'_U})$, $\mathtt{TX}.\mathsf{Output}[1] := (\vec{\theta}[1].\mathsf{Cash}, \mathtt{One} - \mathtt{Sig}_{pk_B})$ and $\mathtt{TX}.\mathsf{Output}[2] := (v, \mathtt{Multi} - \mathtt{Sig}_{pk_B, pk_U})$ if $v > 0$ and $\mathtt{TX}.\mathsf{Input} := tid$ and $\mathtt{TX}.\mathsf{Output}[0] := (\vec{\theta}[0].\mathsf{Cash}, \mathtt{One} - \mathtt{Sig}_{pk'_U})$, $\mathtt{TX}.\mathsf{Output}[1] := (\vec{\theta}[1].\mathsf{Cash} + v, \mathtt{One} - \mathtt{Sig}_{pk_B})$ and $\mathtt{TX}.\mathsf{Output}[2] := (-v, \mathtt{Multi} - \mathtt{Sig}_{pk_B, pk_U})$ otherwise.

$\mathtt{GenAED}(tid, v, pk_P)$:

Return $[\mathtt{TX}]$ where $\mathtt{TX}.\mathsf{Input} := tid$ and $\mathtt{TX}.\mathsf{Output} := (v, \mathtt{One} - \mathtt{Sig}_{pk_P})$.

$\mathtt{GenTO}(tid, v, t_{timeout}, pk_P)$:

Return $[\mathtt{TX}]$ where $\mathtt{TX}.\mathsf{Input} := tid$, $\mathtt{TX}.\mathsf{Output} := (v, \mathtt{One} - \mathtt{Sig}_{pk_P})$ and $\mathtt{TX}.\mathsf{nLT} := t_{timeout}$.

$\mathtt{ForceClose}^P(id)$:

Let t_0 be the current round. Extract $\mathtt{TX}_{\mathtt{CM}}$ from $\Gamma^P(id)$.

- Post (post, $\mathtt{TX}_{\mathtt{CM}}^P$) $\xhookrightarrow{t_0} \mathcal{L}$.

(Publishing the corresponding split transaction takes place in the Punish phase.)

J.4. Simulator

We provide the code for a simulator below. It simulates the protocol of BlindChannel in the ideal world having access to the functionalities of $\mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$. The main challenge in providing a simulation in UC proofs usually arises from the fact that the simulator is not given the secret inputs of the parties in the protocol, which makes it difficult to provide a simulated transcript that is indistinguishable to a transcript of a real protocol execution. As both channel parties do not obtain any secret inputs, and only receive commands from environment, and hence the only challenge that arises during the simulation is handling different behavior of malicious parties. For this reason, we omit the case that both channel parties are honest.

Simulator $\mathcal{S}$ for Create

Let $T_1 = 3$.

Case P is honest and Q is corrupted

Upon P sending $(\mathtt{CREATE}, \gamma, \mathsf{tid_P}) \xhookrightarrow{\tau_0} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, if Q does not send $(\mathtt{CREATE}, \gamma, \mathsf{tid_Q}) \xhookrightarrow{\tau} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, where $|\tau_0 - \tau| \leq T_1$, then distinguish the following cases:

1) If Q sends $(\mathsf{createInfo}, id, tid_Q, R_Q, Y_Q) \xhookrightarrow{\tau_0} P$, then sends $(\mathtt{CREATE}, \gamma, \mathsf{tid_P}) \xhookrightarrow{\tau_0} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of Q;
2) Otherwise stop.

Do the following:

1) Set $id := \gamma.\mathsf{id}$, generate $(R_P, r_P) \leftarrow \mathsf{Gen}$ and $(Y_P, y_P) \leftarrow \mathsf{Gen}$ and send $(\text{createInfo}, id, tid_P, R_P, Y_P) \overset{\tau_0}{\hookrightarrow} Q$.
2) If $(\text{createInfo}, id, tid_Q, R_Q, Y_Q) \overset{\tau_0+1}{\hookrightarrow} P$, create:

$$[\mathtt{TX_{FU}}] := \mathtt{GenFund}((tid_P, tid_Q), \gamma)$$
$$[\mathtt{TX_{CM}}] := \mathtt{GenCommit}([\mathtt{TX_{FU}}].\mathsf{txid} \| 1, I_P, I_Q)$$
$$[\mathtt{TX_{SP}}] := \mathtt{GenSplit}([\mathtt{TX_{CM}}].\mathsf{txid} \| 1, \gamma.\mathsf{st})$$

for $I_P := (pk_P, R_P, Y_P)$ and $I_Q := (pk_Q, R_Q, Y_Q)$. Else stop.
3) Compute $\sigma_s^P := \mathsf{Sign}_{sk_P}([\mathtt{TX_{SP}}])$ and $\hat{\sigma}_c^P := \mathsf{PreSign}_{sk_P}([\mathtt{TX_{CM}}], Y_Q)$ and send $(\text{createCom}, id, \sigma_s^P, \hat{\sigma}_c^P) \overset{\tau_0+1}{\hookrightarrow} Q$.
4) If $(\text{createCom}, id, \sigma_s^Q, \tilde{\sigma}_c^Q)) \overset{\tau_0+2}{\hookleftarrow} Q$ s.t. $\mathsf{PreVf}_{pk_Q}([\mathtt{TX_{CM}}], Y_P; \tilde{\sigma}_c^Q) = 1$ and $\mathsf{Vf}_{pk_Q}([\mathtt{TX_{SP}}]; \sigma_s^Q) = 1$, compute $\sigma_f^P := \mathsf{Sign}_{sk_P}([\mathtt{TX_{FU}}])$ and send $(\text{createFund}, id, \sigma_f^P) \overset{\tau_0+2}{\hookrightarrow} Q$. Else stop.
5) If $(\text{createFund}, id, \sigma_f^Q) \overset{\tau_0+3}{\hookleftarrow} Q$ s.t. $\mathsf{Vf}_{pk_Q}([\mathtt{TX_{FU}}]; \sigma_f^Q) = 1$, set $\mathtt{TX_{FU}} := ([\mathtt{TX_{FU}}], \{\sigma_f^P, \sigma_f^Q\})$ and post $(\text{post}, \mathtt{TX_{FU}}) \overset{\tau_0+3}{\hookrightarrow} \mathcal{L}$. Else stop.
6) If $\mathtt{TX_{FU}}$ is accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + 3 + \Delta$, set $\mathtt{TX_{CM}} := ([\mathtt{TX_{CM}}], \{\mathsf{Sign}_{sk_P}([\mathtt{TX_{CM}}]), \mathsf{Adapt}(\tilde{\sigma}_c^Q, y_P)\})$ and $\mathtt{TX_{SP}} = ([\mathtt{TX_{SP}}], \{\sigma_s^P, \sigma_s^Q\})$, store $\Gamma^P(\gamma.id) := (\gamma, \mathtt{TX_{FU}}, (\mathtt{TX_{CM}}, r_U, R_B, Y_B, \tilde{\sigma}_c^U), \mathtt{TX_{SP}}, \bot)$ if $P = U$ and $\Gamma^B(\gamma.id) := (\gamma, \mathtt{TX_{FU}}, (\mathtt{TX_{CM}}, r_B, R_U, Y_U, \tilde{\sigma}_c^B), \bot, \bot, \bot)$ otherwise and send $(\mathtt{CREATED}, id) \overset{\tau_1}{\hookrightarrow} \mathcal{E}$. Else stop.

Simulator for Update

Let $T_1 = 2$ and $T_2 = 1$ and let $|\vec{tid}| = 1$.

Case B is honest and U is corrupted

Upon B receives $(\text{updateInfo}, id, R_B, Y_B) \overset{\tau_0}{\hookleftarrow} U$, sending $(\mathtt{UPDATE}, \mathsf{id}, t_{stp}, (\mathsf{Y}_\gamma, \mathsf{w}_\gamma)) \overset{\tau}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of U, if U not sends, where $\tau \leq \tau_0 + T_1$, and let $\mathcal{S}$ define $\vec{\mathsf{tid}}$ s.t. $|\vec{\mathsf{tid}}| = k$. If received $(\mathtt{UPDATE\text{–}REQ}, \mathsf{id}, \vec{\mathsf{tid}}, t_{stp}) \overset{\tau_0+2}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, proceed as follows:

1) Generate $(R_B, r_B) \leftarrow \mathsf{Gen}$ and $(Y_B, y_B) \leftarrow \mathsf{Gen}$ and send $(\text{updateInfo}, id, R_B, Y_B) \overset{\tau_0+2}{\hookrightarrow} U$.
2) If received $(\mathtt{SETUP\text{–}OK}, \mathsf{id}) \overset{\tau_0+2}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, then sends $(\mathtt{VERIFY\text{–}REQ}, \mathsf{id}, \mathsf{Y}'_\gamma) \overset{\tau_3}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$.
3) Extract γ and $\mathtt{TX_{FU}}$ from $\Gamma^B(id)$ and create:

$$[\mathtt{TX_{CM}}] := \mathtt{GenCommit}([\mathtt{TX_{FU}}].\mathsf{txid} \| 1, I_U, I_B) \tag{5}$$

for $I_B := (pk_B, R_B, Y_B)$ and $I_U := (pk_U, R_U, Y_U)$.
Distinguishing the following cases:
If B is the payee and $(\text{updateProofU}, id, \pi_\mathsf{s}, \pi_\mathsf{a}, \pi_\mathsf{t}, \tilde{\sigma}_a^U, h, e, (\pi, \mathsf{com}), (\pi', \mathsf{com}')) \overset{\tau_0+2}{\hookleftarrow} U$, s.t. $\pi_\mathsf{s}, \pi_\mathsf{a}, \pi_\mathsf{t}, \hat{\sigma}_a^U$, or B is the payer and $(\text{updateProofU}, id, \pi_\mathsf{s}, \pi_\mathsf{a}, \pi_\mathsf{t}, \sigma_t^U, h, e, \mathsf{com}_s, \mathsf{com}_a) \overset{\tau_0+2}{\hookleftarrow} U$, s.t. $\pi_\mathsf{s}, \pi_\mathsf{a}, \pi_\mathsf{t}, \sigma_t^U$, then create $\tilde{\sigma}_c^B := \mathsf{PreSign}_{sk_B}([\mathtt{TX_{CM}}], Y_U)$, $\tilde{\sigma}_s^{*B} := \mathsf{BAS.Sign}_{1_{sk_B}}(\mathsf{com}, Y'_U)$. Else stop. If B is the payee, compute $\sigma_t^{*B} := \mathsf{BlindSig}_{sk_B}(\mathsf{com}_\mathsf{t})$ and send $(\text{updateComB}, id, \hat{\sigma}_c^B, \hat{\sigma}_s^B, \sigma_t^{*B}) \overset{\tau_0+2}{\hookrightarrow} U$. Else compute $\tilde{\sigma}_a^{*B} := \mathsf{BAS.Sign}_{1_{sk_B}}(\mathsf{com}', Y'_s)$ and send $(\text{updateComB}, id, \tilde{\sigma}_c^B, \tilde{\sigma}_s^{*B}, \tilde{\sigma}_a^{*B}) \overset{\tau_0+2}{\hookrightarrow} U$.
4) In round τ_0+4, if $(\mathtt{VERIFIED}) \overset{\tau_0+4+T}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, distinguish the following cases:

- If received $(\text{updateComU}, id, \tilde{\sigma}_c^U) \overset{\tau_0+4}{\hookleftarrow} U$, s.t. $\mathsf{PreVf}_{pk_U}([\mathtt{TX_{CM}}], Y_B; \tilde{\sigma}_c^U) = 1$, then sign $\sigma_c^B := \mathsf{Sign}_{sk_B}([\mathtt{TX_{CM}}])$, compute $\sigma_c^U := \mathsf{Adapt}(\tilde{\sigma}_c^U, y_B)$, set $\mathtt{TX_{CM}} := ([\mathtt{TX_{CM}}], \{\sigma_c^U, \sigma_c^B\})$ and output $(\mathtt{UPDATE-OK}, id) \overset{\tau_0+4}{\hookrightarrow} \mathcal{E}$ and $(\mathtt{UPDATE\text{–}OK}) \overset{\tau_0+4}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$.
- If $(\text{updateNotOK}, id, r_U) \overset{\tau_0+4}{\hookleftarrow} B$, s.t. $(R_U, r_U) \in R$, then add $\Theta^B(id) := \Theta^B(id) \cup ([\mathtt{TX_{CM}}], r_U, Y_U, \tilde{\sigma}_c^B)$ and stop.
- Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.

Otherwise stop.

5) If received $(\mathtt{REVOKE}, id) \overset{\tau_0+4}{\hookleftarrow} \mathcal{E}$, then sends $(\mathtt{REVOKE}, \mathsf{id}) \overset{\tau_6 \leq \tau_5+T}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of U, if U not sends.
6) If received $(\mathtt{REVOKE\text{–}REQ}, \mathsf{id}) \overset{\tau_0+4}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, parse $\Gamma^B(id)$ as $(\gamma, \mathtt{TX_{FU}}, (\overline{\mathtt{TX}}_{\mathtt{CM}}, \overline{r}_B, \overline{R}_U, \overline{Y}_U, \overline{\tilde{\sigma}}_c^B), \bot, \bot)$, update the channel space as $\Gamma^B(\gamma.\mathsf{id}) := (\gamma, \mathtt{TX_{FU}}, (\mathtt{TX_{CM}}, r_B, R_U, Y_U, \tilde{\sigma}_c^B), Y'_U, \tilde{\sigma}_s^{*B}, e, \sigma)$ with $\sigma := \tilde{\sigma}_a^U$ if B is the payee and $\sigma := \sigma_t^U$, and send $(\text{revokeB}, id, \overline{r}_B) \overset{\tau_0+4}{\hookrightarrow} U$. Otherwise stop.
7) If received $(\text{revokeU}, id, \overline{r}_U) \overset{\tau_0+6}{\hookleftarrow} U$, set $\Theta^B(id) := \Theta^B(id) \cup ([\overline{\mathtt{TX}}_{\mathtt{CM}}], \overline{r}_U, \overline{Y}_U, \overline{\tilde{\sigma}}_c^B)$ and send $(\mathtt{UPDATED}, id) \overset{\tau_0+6}{\hookrightarrow} \mathcal{E}$, and send $(\mathtt{REVOKE}, \mathsf{id}) \overset{\tau_0+6}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, s.t. $(\overline{R}_U, \overline{r}_U) \in R$. Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.
8) For the case where B is the payee, if $(\mathtt{ADAPT}, id) \overset{\tau_1}{\hookleftarrow} \mathcal{E}$ with $\tau_0 + 6 \leq \tau_1 \leq \tau_0 + T_1$, then send $(\text{adapt}, y_s, id) \overset{\tau_1}{\hookrightarrow} U$. Else set $\tau_1 = \tau_0 + T_1$ and continue. For the case where B is the payer, continue.

9) For the case where B is the payee, continue. For the case where B is the payer, if $(\text{updateReq}, id, y'_s) \xleftarrow{\tau_1} U$ with $\tau_0 + 6 \leq \tau_1 \leq \tau_0 + T'_1$ s.t. $(Y'_s, y'_s) \in R$, continue. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop.
10) For the case where B is the payee, continue. For the case where B is the payer, if $(\texttt{UPDATE}, id, \vec{\theta}) \xleftarrow{\tau_1} \mathcal{E}$, send $(\text{updateReq}, id) \xhookrightarrow{\tau_1} U$. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop.
11) If received $(\text{updateReq}, id, R_U, Y_U) \xleftarrow{\tau_1+2} U$, send $(\texttt{UPDATE}, \mathsf{id}, \theta, t_{stp}, (\mathsf{Y}_\gamma, \mathsf{w}_\gamma)) \xhookrightarrow{\tau_1+2} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of U, if U not send at τ_1+2. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop.
12) If $(\texttt{UPDATE–REQ}, \mathsf{id}, t_{stp}) \xhookleftarrow{\tau_1+2} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, send $(\texttt{UPDATE}-\texttt{REQ}, id) \xrightarrow{\tau_1+2} \mathcal{E}$.
13) If $(\texttt{UPDATE}-\texttt{REQ}-\texttt{OK}, id) \xleftarrow{\tau_1+2} \mathcal{E}$, then generates $(R_B, r_B) \leftarrow \mathsf{Gen}$ and $(Y_B, y_B) \leftarrow \mathsf{Gen}$ and sends $(\text{updateInfo}, id, R_B, Y_B) \xhookrightarrow{\tau_1+2} U$, and sends $(\texttt{SETUP–OK}, \mathsf{id}) \xhookrightarrow{t_0+2} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of U, if U not sends at t_0+2. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
14) If received $(\texttt{SETUP–OK}, \mathsf{id}) \xhookleftarrow{\tau_0+2} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, countine. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
15) Create:

$$[\texttt{TX}_{\texttt{CM}}] := \texttt{GenCommit}([\texttt{TX}_{\texttt{FU}}].\mathsf{txid} \| 1, I_U, I_B) \tag{6}$$

for $I_B := (pk_B, R_B, Y_B)$ and $I_U := (pk_U, R_U, Y_U)$. If received $(\text{updateProofU}, id, \pi_\mathsf{s}, \mathsf{com_s}) \xleftarrow{\tau_1+4} \mathsf{U}$, s.t. $\pi_\mathsf{s}, \mathsf{com_s}$ are valid, then create $\sigma_s^{*B} := \mathsf{BlindSig}_{sk_B}(\mathsf{com_s})$ and $\hat{\sigma}_c^B := \mathsf{PreSign}_{sk_B}([\texttt{TX}_{\texttt{CM}}], Y_U)$ and send $(\text{updateComB}, id, \sigma_s^{*B}, \hat{\sigma}_c^B) \xhookrightarrow{\tau_1+4} U$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
16) If received $(\text{updateComU}, id, \hat{\sigma}_c^U) \xleftarrow{\tau_1+6} U$, s.t. $\hat{\sigma}_c^U$ is valid, or received $(\texttt{VERIFIED}) \xhookleftarrow{\tau_4 \leq \tau_3+T} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, send $(\texttt{UPDATE}-\texttt{OK}, id) \xhookrightarrow{\tau_1+6} \mathcal{E}$ and $(\texttt{UPDATE}-\texttt{OK}, id) \xhookrightarrow{\tau_1+6} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
17) If $(\texttt{REVOKE}, id) \xleftarrow{\tau_1+6} \mathcal{E}$, then set $\texttt{TX}_{\texttt{CM}} := ([\texttt{TX}_{\texttt{CM}}], \{\mathsf{Sign}_{sk_B}([\texttt{TX}_{\texttt{CM}}]), \mathsf{Adapt}(\hat{\sigma}_c^U, y_B)\})$, parse $\Gamma^B(id)$ as $(\gamma, \texttt{TX}_{\texttt{FU}}, (\overline{\texttt{TX}}_{\texttt{CM}}, \overline{r}_B, \overline{R}_U, \overline{Y}_U, \overline{\hat{\sigma}}_c^B), \overline{t_{timeout}})$, update the channel space as $\Gamma^B(id) := (\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_B, R_U, Y_U, \hat{\sigma}_c^B), \bot, \bot)$ and send $(\text{revokeB}, id, \overline{r}_B) \xhookrightarrow{\tau_1+6} U$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
18) Sends $(\texttt{REVOKE}, \mathsf{id}) \xhookrightarrow{\tau_1+6} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of U, if U not sends.
19) If $(\texttt{REVOKE}-\texttt{REQ}, id) \xleftarrow{\tau_1+6} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, then parse $\Gamma^B(id)$ as $(\gamma, \texttt{TX}_{\texttt{FU}}, (\overline{\texttt{TX}}_{\texttt{CM}}, \overline{r}_B, \overline{R}_U, \overline{Y}_U, \overline{\hat{\sigma}}_c^B), \overline{Y}'_U, \overline{\hat{\sigma}}_s^{*B}, \overline{e}, \overline{\sigma})$, update the channel space as $\Gamma^B(id) := (\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_B, R_U, Y_U, \hat{\sigma}_c^B), \bot, \bot, \bot, \bot)$ and send $(\text{revokeB}, id, \overline{r}_B) \xhookrightarrow{\tau_1+6} U$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
20) If received $(\text{revokeU}, id, \overline{r}_U) \xleftarrow{\tau_1+8} U$ or $(\texttt{UPDATED}, \mathsf{id}) \xhookleftarrow{\tau_6 \leq \tau_5+T} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, s.t. $(\overline{R}_U, \overline{r}_U) \in R$, set $\Theta^B(id) := \Theta^B(id) \cup ([\overline{\texttt{TX}}_{\texttt{CM}}], \overline{r}_U, \overline{Y}_U, \overline{\hat{\sigma}}_c^B)$ and send $(\texttt{UPDATED}, id) \xhookrightarrow{\tau_1+8} \mathcal{E}$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.

Case U is honest and B is corrupted

Upon U receiving $(\texttt{UPDATE}, id, \vec{\theta}) \xleftarrow{\tau_0} \mathcal{E}$, sends $(\texttt{UPDATE}, id, \vec{\theta}, (\mathsf{Y}_\gamma, \mathsf{w}_\gamma)) \xhookrightarrow{\tau_0} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\cdot,\pm,\mathcal{V})}$, parses $(\{\gamma\}, \mathsf{tx}) := \Gamma^U(\mathsf{id})$, set $\gamma' := \gamma$, $\gamma'.\mathsf{st} := \vec{\theta}$. If received $(\texttt{SETUP}, \mathsf{id}, \vec{\mathsf{tid}}) \xhookleftarrow{\tau_0} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, then distinguish the following cases:

1) Generate $(R_U, r_U) \leftarrow \mathsf{Gen}$, $(Y_U, y_U) \leftarrow \mathsf{Gen}$ and $(Y'_U, y'_U) \leftarrow \mathsf{Gen}$.
2) If received $(\text{updateInfo}, id, R_B, Y_B) \xleftarrow{\tau_0+2} B$, create:

$$[\texttt{TX}_{\texttt{CM}}] := \texttt{GenCommit}([\texttt{TX}_{\texttt{FU}}].\mathsf{txid} \| 1, I_U, I_B)$$
$$[\texttt{TX}_{\texttt{SP}}] := \texttt{GenSplit}([\texttt{TX}_{\texttt{CM}}].\mathsf{txid} \| 1, \vec{\theta})$$
$$[\texttt{TX}_{\texttt{AED}}] := \texttt{GenAED}([\texttt{TX}_{\texttt{SP}}].\mathsf{txid} \| 3, \vec{\theta}, pk)$$
$$[\texttt{TX}_{\texttt{TO}}] := \texttt{GenTO}([\texttt{TX}_{\texttt{SP}}].\mathsf{txid} \| 3, \vec{\theta}, t_{timeout}, pk')$$

for $I_B := (pk_B, R_B, Y_B)$ and $I_U := (pk_U, R_U, Y_U)$ and with $pk := pk_B$ and $pk' := pk_U$ if U is the payer and $pk := pk_U$ and $pk' := pk_B$ otherwise. Then sends $(\texttt{SETUP–OK}, \mathsf{id}) \xhookrightarrow{\tau_0+2} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ as U. Else stop.
3) If $(\text{updateInfo}, id, R_B, Y_B) \xleftarrow{\tau_2+2} B$, sends $(\texttt{VERIFY–REQ}, \mathsf{id}, \mathsf{Y}'_\gamma) \xhookrightarrow{\tau_2+2} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of B, of B not sends in τ_0+2. Compute $(\mathsf{com_s}, \mathsf{decom_s}) \leftarrow \mathsf{P}_{\mathsf{COM}}([\texttt{TX}_{\texttt{SP}}])$. If U is the payer, compute $(\mathsf{com_t}, \mathsf{decom_t}) \leftarrow \mathsf{P}_{\mathsf{COM}}([\texttt{TX}_{\texttt{TO}}])$. Otherwise, compute $(\mathsf{com_a}, \mathsf{decom_a}) \leftarrow \mathsf{P}_{\mathsf{COM}}([\texttt{TX}_{\texttt{AED}}])$.
4) Compute π_s, π_a and π_t. If U is the payer, compute $\hat{\sigma}_a^U := \mathsf{PreSign}_{sk_U}([\texttt{TX}_{\texttt{AED}}], Y'_s)$, $h = \mathsf{SigHash}([\texttt{TX}_{\texttt{AED}}])$, and $ct := \mathsf{Enc}_{\mathsf{y}'_\mathsf{U}}([\texttt{TX}_{\texttt{AED}}])$ and send $(\text{updateProofU}, id, \pi_\mathsf{s}, \pi_\mathsf{a}, \pi_\mathsf{t}, \hat{\sigma}_a^U, h, ct, (\pi, \mathsf{com}), (\pi', \mathsf{com}')) \xhookrightarrow{\tau_0+2} B$. Otherwise, compute $\sigma_t^U := \mathsf{Sign}_{sk_U}([\texttt{TX}_{\texttt{TO}}])$, $h = \mathsf{SigHash}([\texttt{TX}_{\texttt{TO}}])$ and $ct := \mathsf{Enc}_{\mathsf{y}'_\mathsf{U}}([\texttt{TX}_{\texttt{TO}}])$ and send $(\text{updateProofU}, id, \pi_\mathsf{s}, \pi_\mathsf{a}, \pi_\mathsf{t}, \sigma_t^U, h, ct, \mathsf{com}_s, \mathsf{com}_t)) \xhookrightarrow{t_0+2} B$.

5) If received $(\text{updateComB}, id, \hat{\sigma}_c^B, \hat{\sigma}_s^{*B}, \sigma_t^{*B}) \xleftarrow{\tau_0+4} B$:
   - For the case U is the payer and $\mathsf{PreVf}_{pk_B}([\mathtt{TX}_{\mathtt{CM}}], Y_U; \hat{\sigma}_c^B) = 1$, and σ_s^{*B}, σ_t^{*B} are valid;
   - For the case U is the payee and $\mathsf{PreVf}_{pk_B}([\mathtt{TX}_{\mathtt{CM}}], Y_U; \hat{\sigma}_c^B) = 1$, $\sigma_s^{*B}, \hat{\sigma}_a^B$.

   - Send $(\mathtt{UPDATE\text{–}OK}) \overset{\tau_0+4}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of B, if he does not send. If received $(\mathtt{UPDATE\text{–}OK}) \overset{\tau_0+4}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, then outputs $(\mathtt{UPDATE-OK}, id) \xhookrightarrow{\tau_0+4} \mathcal{E}$.
6) If $(\mathtt{UPDATEB}, id) \xleftarrow{t_0+4} \mathcal{E}$, continue.; Else send $(\text{updateNotOK}, id, r_U) \xhookrightarrow{\tau_0+4} B$, and stop.
7) If U is the payer, then sign $\sigma_t^U := \mathsf{Sign}_{sk_U}([\mathtt{TX}_{\mathtt{TO}}])$, compute $\sigma_t^B := \mathsf{UnBlindSig}(\sigma_t^{*B}, \mathsf{decom}')$, set $\mathtt{TX}_{\mathtt{TO}} := ([\mathtt{TX}_{\mathtt{TO}}], \{\sigma_t^U, \sigma_t^B\})$. Else, sign $\sigma_a^U := \mathsf{Sign}_{sk_U}([\mathtt{TX}_{\mathtt{AED}}])$, compute $\hat{\sigma}_a^B := \mathsf{UnBlindSig}(\hat{\sigma}_a^{*B}, \mathsf{decom}')$.
8) Sign $\sigma_s^U := \mathsf{Sign}_{pk_U}([\mathtt{TX}_{\mathtt{SP}}^{[3]}])$, compute $\sigma_s^B = \mathsf{UnBlindSig}(\sigma_s^{*B}, \mathsf{decom_s})$, set $\mathtt{TX}_{\mathtt{SP}}^{[3]} := ([\mathtt{TX}_{\mathtt{SP}}^{[3]}], \{\sigma_s^U, \sigma_s^B\})$, sign $\sigma_c^U := \mathsf{Sign}_{sk_U}([\mathtt{TX}_{\mathtt{CM}}])$, compute $\sigma_c^B := \mathsf{Adapt}(\hat{\sigma}_c^B, y_U)$, set $\mathtt{TX}_{\mathtt{CM}} := ([\mathtt{TX}_{\mathtt{CM}}], \{\sigma_c^U, \sigma_c^B\})$, sign $\hat{\sigma}_c^U := \mathsf{PreSign}_{sk_U}([\mathtt{TX}_{\mathtt{CM}}], Y_B)$ and send $(\text{updateComU}, id, \hat{\sigma}_c^U) \xhookrightarrow{\tau_0+4} B$.

9) Parse $\Gamma^U(id)$ as $(\gamma, \mathtt{TX}_{\mathtt{FU}}, (\overline{\mathtt{TX}}_{\mathtt{CM}}, \overline{r}_U, \overline{R}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U), \bot, \bot)$. If received $(\text{revokeB}, id, \overline{r}_B) \xleftarrow{\tau_0+6} B$, s.t. $(\overline{R}_B, \overline{r}_B) \in R$, sends $(\mathtt{REVOKE}, \mathsf{id}) \overset{\tau_0+6}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ and $(\mathtt{REVOKE-REQ}, id) \overset{\tau_0+6}{\hookrightarrow} \mathcal{E}$. Else execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
10) If $(\mathtt{REVOKE}, id) \xleftarrow{\tau_0+6} \mathcal{E}$, continue. Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
11) If $(\mathtt{UPDATED}, \mathsf{id}) \overset{\tau_0+6}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, set $\Theta^U(id) := \Theta^U(id) \cup ([\overline{\mathtt{TX}}_{\mathtt{CM}}], \overline{r}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U)$. If U is the payer, then set $\Gamma^U(\gamma.id) := (\gamma, \mathtt{TX}_{\mathtt{FU}}, (\mathtt{TX}_{\mathtt{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \mathtt{TX}_{\mathtt{SP}}, \mathtt{TX}_{\mathtt{TO}})$. Otherwise, set $\Gamma^U(\gamma.id) := (\gamma, \mathtt{TX}_{\mathtt{FU}}, (\mathtt{TX}_{\mathtt{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \mathtt{TX}_{\mathtt{SP}}, ([\mathtt{TX}_{\mathtt{AED}}], \{\sigma_a^U, \hat{\sigma}_a^B\})$. Send $(\text{revokeU}, id, \overline{r}_U) \xhookrightarrow{\tau_0+6} B$ and $(\mathtt{UPDATED}) \xhookrightarrow{\tau_0+6} \mathcal{E}$. Also, sends $(\mathtt{REVOKE}, \mathsf{id}) \overset{\tau_0+6}{\hookrightarrow} \mathcal{E}$ on behalf of B, if he does not send.
12) For the case where U is the payer, continue.
    For the case where U is the payee:
    - if $(\mathtt{ADAPT}, id) \xleftarrow{\tau_1} \mathcal{E}$ with $\tau_0+6 \le \tau_1 \le \tau_0+T_1'$, and $(\text{updateReq}, id, y_s') \overset{t_1}{\hookrightarrow} B$;
    - Else set $t_1 = t_0 + T_1'$ and continue.

13) If received $(\mathtt{UPDATE\text{–}REQ}, \mathsf{id}, t_{stp}) \overset{\tau_1}{\hookleftarrow} B$, then
    - For the case where U is the payer, if $(\text{adapt}, y_s', id) \overset{t_1}{\hookleftarrow} \mathcal{B}$ with $t_0+8 \le t_1 \le t_0+2+T_1$, s.t. $(Y_s', y_s') \in R$, continue.
    - Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
14) If received $(\mathtt{UPDATE}, id, \vec{\theta}) \xleftarrow{\tau_1} \mathcal{E}$, sends $(\mathtt{UPDATE}, \mathsf{id}, \vec{\theta}, t_{stp}, (\mathsf{Y}_\gamma, \mathsf{w}_\gamma)) \overset{\tau_1}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, set $\tau_2 := \tau_1$ and continue; Otherwise, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
15) For the case where U is the payee:
    - if $(\text{updateReq}, id) \xleftarrow{\tau_1+2} B$, then set $\tau_2 := \tau_1+2$ and continue.
    - Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
16) If received $(\mathtt{SETUP}, \mathsf{id}, \vec{\mathsf{tid}}, \vec{\theta}) \overset{\tau_1}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$: Generate $(R_U, r_U) \leftarrow \mathsf{Gen}$ and $(Y_U, y_U) \leftarrow \mathsf{Gen}$ and send $(\text{updateReq}, id, R_U, Y_U) \overset{\tau_2}{\hookrightarrow} B$, sends $(\mathtt{SETUP\text{–}OK}, \mathsf{id}) \overset{\tau_2 \le \tau_1 + t_{stp}}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$. Otherwise, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
17) If $(\text{updateInfo}, id, R_B, Y_B) \xleftarrow{\tau_2+2} B$, sends $(\mathtt{VERIFY\text{–}REQ}, \mathsf{id}, \mathsf{Y}_\gamma') \overset{\tau_2+2}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$on behalf of B, of B not sends in τ_2+2, then proceed as follows:
    - create:

      $[\mathtt{TX}_{\mathtt{CM}}] := \mathtt{GenCommit}([\mathtt{TX}_{\mathtt{FU}}].\mathsf{txid} \| 1, I_U, I_B)$

      $[\mathtt{TX}_{\mathtt{SP}}] := \mathtt{GenSplit}([\mathtt{TX}_{\mathtt{CM}}].\mathsf{txid} \| 1, \vec{\theta})$

      for $I_B := (pk_B, R_B, Y_B)$ and $I_U := (pk_U, R_U, Y_U)$. Else, execute the procedure $\mathtt{ForceClose}^B(id)$ and stop.
    - Computes $(\mathsf{com_s}, \mathsf{decom_s}) \leftarrow \mathsf{P_{COM}}([\mathtt{TX}_{\mathtt{SP}}])$.
    - Computes π_{s} and send $(\text{updateProofU}, id, \pi_{\mathsf{s}}, \mathsf{com_s}) \xhookrightarrow{\tau_2+2} \mathsf{B}$.

18) If $(\text{updateComB}, id, \sigma_s^{*B}, \hat{\sigma}_c^B) \xleftarrow{\tau_2+4} B$ s.t. σ_s^{*B}, $\hat{\sigma}_c^B$ are valid, outputs $(\mathtt{UPDATE-OK}, id) \xhookrightarrow{\tau_2+4} \mathcal{E}$. Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.

19) If $(\mathtt{UPDATE}, id) \xleftarrow{\tau_2+4} \mathcal{E}$, then create $\hat{\sigma}_c^U := \mathsf{PreSign}_{sk_U}([\mathtt{TX}_{\mathtt{CM}}], Y_B)$ and send $(\text{updateComU}, id, \hat{\sigma}_c^U) \xhookrightarrow{\tau_2+4} B$. Else, execute the procedure $\mathtt{ForceClose}^U(id)$ and stop.
20) If received $(\mathtt{UPDATE\text{–}OK}) \overset{\tau_2+6}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$:
    - For the case U is the payer, then parse $\Gamma^U(id)$ as $(\gamma, \mathtt{TX}_{\mathtt{FU}}, (\overline{\mathtt{TX}}_{\mathtt{CM}}, \overline{r}_U, \overline{R}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U), \overline{\mathtt{TX}}_{\mathtt{SP}}^{[3]}, \overline{\mathtt{TX}}_{\mathtt{TO}})$.
    - for the case U is the payer, parse $\Gamma^U(id)$ as $(\gamma, \mathtt{TX}_{\mathtt{FU}}, (\overline{\mathtt{TX}}_{\mathtt{CM}}, \overline{r}_U, \overline{R}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U), \overline{\mathtt{TX}}_{\mathtt{SP}}^{[3]}, ([\overline{\mathtt{TX}}_{\mathtt{AED}}], \{\overline{\sigma}_a^U, \overline{\hat{\sigma}}_a^B\}))$, s.t. $(\overline{R}_B, \overline{r}_B) \in R$, $(\mathtt{REVOKE-REQ}, id) \overset{\tau_2+6}{\hookrightarrow} \mathcal{E}$.

Else execute the procedure $\texttt{ForceClose}^U(id)$ and stop.

21) If $(\texttt{REVOKE}, id) \xleftarrow{\tau_2+6} \mathcal{E}$, send $(\texttt{REVOKE}, \mathsf{id}) \overset{\tau_2+6}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ and set $\Theta^U(id) := \Theta^U(id) \cup ([\overline{\texttt{TX}}_{\texttt{CM}}], \overline{r}_B, \overline{Y}_B, \overline{\hat{\sigma}}_c^U)$. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop. Set $\texttt{TX}_{\texttt{CM}} := ([\texttt{TX}_{\texttt{CM}}], \{\mathsf{Sign}_{sk_U}([\texttt{TX}_{\texttt{CM}}]), \mathsf{Adapt}(\hat{\sigma}_c^B, y_U)\})$, Sign $\sigma_s^U := \mathsf{Sign}_{pk_U}([\texttt{TX}_{\texttt{SP}}^{[2]}])$, compute $\sigma_s^B = \mathsf{UnBlindSig}(\sigma_s^{*B}, \mathsf{decom_s})$, set $\texttt{TX}_{\texttt{SP}}^{[2]} := ([\texttt{TX}_{\texttt{SP}}^{[2]}], \{\sigma_s^U, \sigma_s^B\})$, set $\Gamma^U(id) := (\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \texttt{TX}_{\texttt{SP}}, \perp)$.

22) Once received $(\texttt{UPDATED}, \mathsf{id}) \overset{\tau_2+6}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, sends $(\text{revokeU}, id, \overline{r}_U) \xhookrightarrow{\tau_2+6} B$, $(\texttt{UPDATED}) \xhookrightarrow{\tau_2+6} \mathcal{E}$ and stop.

Simulator for Closure

Let $T_1 = 1$.

Case B is honest and U is corrupted

Upon B receiving $(\texttt{CLOSE}, id) \xleftarrow{\tau_0} \mathcal{E}$, sends $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, and sends $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of U, if U does not send, where $\tau_0 - \tau \leq T_1$, proceed as follow:

1) Extract $\texttt{TX}_{\texttt{FU}}$ and $\texttt{TX}_{\texttt{CM}}$ from $\Gamma^B(id)$. If $(\text{closeReq}, \texttt{TX}_{\texttt{SP}}, [\overline{\texttt{TX}}_{\texttt{SP}}], \overline{\sigma}_s^U) \xleftarrow{\tau_0+1} U$ s.t. $\mathsf{Vf}_{pk_U}([\overline{\texttt{TX}}_{\texttt{SP}}]; \overline{\sigma}_s^U) = 1$, $[\overline{\texttt{TX}}_{\texttt{SP}}].\mathsf{Output} = \texttt{TX}_{\texttt{SP}}.\mathsf{Output}$, $\texttt{TX}_{\texttt{SP}}.\mathsf{Input} = \texttt{TX}_{\texttt{CM}}.\mathsf{txid} \| 1$ and $[\overline{\texttt{TX}}_{\texttt{SP}}].\mathsf{Input} = \texttt{TX}_{\texttt{FU}}.\mathsf{txid} \| 1$, continue. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
2) Parse $\texttt{TX}_{\texttt{SP}}$ as $([\texttt{TX}_{\texttt{SP}}], \{\sigma_s^U, \sigma^B, s\})$. If $\mathsf{Vf}_{pk_B}([\texttt{TX}_{\texttt{SP}}], \sigma_s^B)$, then Sign $\overline{\sigma}_s^B := \mathsf{Sign}_{sk_B}([\overline{\texttt{TX}}_{\texttt{SP}}])$ set $\overline{\texttt{TX}}_{\texttt{SP}} := ([\overline{\texttt{TX}}_{\texttt{SP}}], \{\overline{\sigma}_s^U, \overline{\sigma}_s^B\})$, send $(\text{closeRes}, \overline{\sigma}_s^B) \xhookrightarrow{\tau_0+1} B$ and post $(\text{post}, \overline{\texttt{TX}}_{\texttt{SP}}) \xhookrightarrow{\tau_0+2} \mathcal{L}$. Else, execute the procedure $\texttt{ForceClose}^B(id)$ and stop.
3) If in round $\tau_1 \leq \tau_0 + 2 + \Delta$, the transaction $\overline{\texttt{TX}}_{\texttt{SP}}$ is accepted by $\mathcal{L}$, set $\Gamma^B(id) := \perp$, $\Theta^B(id) := \perp$ and send $(\texttt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

Case U us honest and B is corrupted

Upon U receiving $(\texttt{CLOSE}, id) \xleftarrow{t_0} \mathcal{E}$, then sends $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, and sends $(\texttt{CLOSE}, \mathsf{id}) \overset{\tau_0}{\hookrightarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$ on behalf of B, if B does not send, where $\tau_0 - \tau \leq T_1$, proceed as follows:

1) If $(\text{closeRes}, \overline{\sigma}_s^B) \xleftarrow{t_0+2} B$ s.t. $\mathsf{Vf}_{pk_B}([\overline{\texttt{TX}}_{\texttt{SP}}]; \overline{\sigma}_s^B) = 1$, then set $\overline{\texttt{TX}}_{\texttt{SP}} := ([\overline{\texttt{TX}}_{\texttt{SP}}], \{\overline{\sigma}_s^U, \overline{\sigma}_s^B\})$ and post $(\text{post}, \overline{\texttt{TX}}_{\texttt{SP}}) \xhookrightarrow{t_0+2} \mathcal{L}$. Else, execute the procedure $\texttt{ForceClose}^U(id)$ and stop.
2) If in round $t_1 \leq t_0 + 2 + \Delta$, the transaction $\overline{\texttt{TX}}_{\texttt{SP}}$ is accepted by $\mathcal{L}$, set $\Gamma^U(id) := \perp$, $\Theta^U(id) := \perp$ and send $(\texttt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$.

Simulator for Close

Case U is honest and B is corrupted

Party U upon received $(\texttt{CLOSE}) \xleftarrow{\tau_0} \mathcal{E}$.

For each $id \in \{0,1\}^*$, s.t. $\Gamma^P(id) \neq \perp$, extract $\Gamma^P(id)$ as $(\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \texttt{TX}_{\texttt{SP}}, x)$. If $\texttt{TX}_{\texttt{FU}}$ is spent by a transaction $\texttt{TX}$ s.t. $[\texttt{TX}] = [\texttt{TX}_{\texttt{CM}}]$, then

- Post $(\text{post}, \texttt{TX}_{\texttt{SP}}) \xhookrightarrow{\tau_0+T} \mathcal{L}$.
- Let $\texttt{TX}_{\texttt{SP}}$ be accepted by $\mathcal{L}$ in round $t_1 \leq \tau_0 + T + \Delta$. If $x = \perp$, then set $\Theta^P(id) := \perp$, $\Gamma^P(id) := \perp$ and output $(\texttt{CLOSED}, id) \xhookrightarrow{\tau_1} \mathcal{E}$ and stop.
- If U is the payer, set $\texttt{TX}_{\texttt{TO}} := x$ and post $(\text{post}, \texttt{TX}_{\texttt{TO}}) \xhookrightarrow{\tau_{timeout}} \mathcal{L}$. Else, parse x as $([\texttt{TX}_{\texttt{AED}}], \{\sigma_a^U, \hat{\sigma}_a^B\})$, possibly adapt $\hat{\sigma}_a^B$ to σ_a^B, set $\texttt{TX}_{\texttt{AED}} := ([\texttt{TX}_{\texttt{AED}}], \{\sigma_a^U, \sigma_a^B\})$ and post $(\text{post}, \texttt{TX}_{\texttt{AED}}) \xhookrightarrow{\tau_1 < \tau_{timeout} - \Delta} \mathcal{L}$.
- Let $\texttt{TX}_{\texttt{SP}}.\mathsf{Output}[2]$ (the HTLC output) be spent in round $\tau_2 \leq \tau_{timeout} + \Delta$.
- If received $(\texttt{CLOSED}, \mathsf{id}) \overset{\tau_1}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, Output $(\texttt{CLOSED}, id) \xhookrightarrow{\tau_2} \mathcal{E}$ and stop.

Case B is honest and U is corrupted

Party B upon received $(\texttt{CLOSE}) \xleftarrow{\tau_0} \mathcal{E}$.

For each $id \in \{0,1\}^*$, s.t. $\Gamma^P(id) \neq \perp$, extract $\Gamma^P(id)$ as $(\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_B, R_U, Y_U, \hat{\sigma}_c^B), Y'_U, \hat{\sigma}_s^{*B}, e, \sigma)$. If $\texttt{TX}_{\texttt{FU}}$ is spent by a transaction $\texttt{TX}$ s.t. $[\texttt{TX}] = [\texttt{TX}_{\texttt{CM}}]$, then

- If $\texttt{TX}.\mathsf{Output}$ is spent in round $\tau_1 \leq \tau_0 + 2T$ by a transaction $\texttt{TX}'$, then if $e \neq \perp$:
  - Parse witness of $\texttt{TX}'$ as $(\sigma_s^U, \sigma_s^B) := \texttt{TX}'.\mathsf{Witness}$ and set $y'_U := \mathsf{Ext}(\sigma_s^B, \hat{\sigma}_s^B, Y'_U)$. If B is the payee $([\texttt{TX}_{\texttt{AED}}]) := \mathsf{Dec}_{y'_U}(\mathsf{e})$, set $\hat{\sigma}_a^U := \sigma$ and possibly adapt $\hat{\sigma}_a^U$ to σ_a^U, sign $\sigma_a^B := \mathsf{Sign}_{sk_B}([\texttt{TX}_{\texttt{AED}}])$, create $\texttt{TX}_{\texttt{AED}} := ([\texttt{TX}_{\texttt{AED}}], \{\sigma_a^U, \sigma_a^B\})$, and post $(\text{post}, \texttt{TX}_{\texttt{AED}}) \xhookrightarrow{\tau_2 < \tau_{timeout} - \Delta} \mathcal{L}$. If B is the payer, $([\texttt{TX}_{\texttt{TO}}]) := \mathsf{Dec}_{y'_U}(\mathsf{e})$, set $\sigma_t^U := \sigma$, sign $\sigma_t^B := \mathsf{Sign}_{sk_B}([\texttt{TX}_{\texttt{TO}}])$, create $\texttt{TX}_{\texttt{TO}} := ([\texttt{TX}_{\texttt{TO}}], \{\sigma_t^U, \sigma_t^B\})$, and post $(\text{post}, \texttt{TX}_{\texttt{TO}}) \xhookrightarrow{\tau_{timeout}} \mathcal{L}$.
  - Let $\texttt{TX}_{\texttt{SP}}.\mathsf{Output}[2]$ (the HTLC output) be spent in round $\tau_2 \leq \tau_{timeout} + \Delta$. If received

$(\texttt{CLOSED}, \mathsf{id}) \overset{\tau_1}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, output $(\texttt{CLOSED}, id) \overset{\tau_2}{\hookrightarrow} \mathcal{E}$ and stop.

Else, if received $(\texttt{CLOSED}, \mathsf{id}) \overset{\tau_1}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, output $(\texttt{CLOSED}, id) \overset{\tau_1}{\hookrightarrow} \mathcal{E}$ and stop.

Simulator for Punish

Case P is honest and Q is corrupted

Party P upon received $(\texttt{PUNISH}) \overset{\tau_0}{\hookleftarrow} \mathcal{E}$. For each $id \in \{0,1\}^*$, s.t. $\Gamma^P(id) \neq \perp$, extract $\Gamma^P(id)$ as $(\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_U, R_B, Y_B, \hat{\sigma}_c^U), \texttt{TX}_{\texttt{SP}}, x)$ if $P = U$ or $(\gamma, \texttt{TX}_{\texttt{FU}}, (\texttt{TX}_{\texttt{CM}}, r_B, R_U, Y_U, \hat{\sigma}_c^B), Y'_U, \hat{\sigma}_s^{*B}, e, \sigma)$ otherwise. Check if $\texttt{TX}_{\texttt{FU}}$ is spent by any transaction $\texttt{TX}$ s.t. $[\texttt{TX}] \neq [\texttt{TX}_{\texttt{CM}}]$. If yes:

- Parse $\Theta^P(id) := \{([\texttt{TX}_{\texttt{CM}}^{(i)}], r_Q^{(i)}, Y_Q^{(i)}, \hat{\sigma}_c^{P(i)})\}_{i \in m}$ and find i s.t. $[\texttt{TX}_{\texttt{CM}}^{(i)}] = [\texttt{TX}]$. Then, parse the witness as $(\sigma_c^P, \sigma_c^Q) := \texttt{TX}.\mathsf{Witness}$ and set $y_Q^{(i)} := \mathsf{Ext}(\sigma_c^P, \hat{\sigma}_c^{P(i)}, Y_Q^{(i)})$.
- Define the body of the punishment transaction $[\texttt{TX}_{\texttt{PU}}]$ as:

$$\texttt{TX}_{\texttt{PU}}.\mathsf{Input} := \texttt{TX}.\mathsf{txid} \| 1$$
$$\texttt{TX}_{\texttt{PU}}.\mathsf{Output} := \{(\gamma.\mathsf{cash}, pk_P)\}$$

- Compute $\sigma_p^y \leftarrow \mathsf{Sign}_{y_Q^{(i)}}([\texttt{TX}_{\texttt{PU}}])$, $\sigma_p^r \leftarrow \mathsf{Sign}_{r_Q^{(i)}}([\texttt{TX}_{\texttt{PU}}])$, $\sigma_p^P \leftarrow \mathsf{Sign}_{sk_P}([\texttt{TX}_{\texttt{PU}}])$, set $\texttt{TX}_{\texttt{PU}} := \{[\texttt{TX}_{\texttt{PU}}], \{\sigma_p^y, \sigma_p^r, \sigma_p^P\}\}$ and $(\text{post}, \texttt{TX}_{\texttt{PU}}) \overset{\tau_0}{\hookrightarrow} \mathcal{L}$.
- Let $\texttt{TX}_{\texttt{PU}}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + \Delta$. Set $\Theta^P(id) := \perp$, $\Gamma^P(id) := \perp$ and output $(\texttt{PUNISHED}, id) \overset{\tau_1}{\hookrightarrow} \mathcal{E}$.

If no and P is B:

If $\texttt{TX}.\mathsf{Output}$ is still unspent in round $\tau_1 = \tau_0 + 2T$, then

- Define the body of the punishment transaction $[\texttt{TX}_{\texttt{PU}}]$ as:

$$\texttt{TX}_{\texttt{PU}}.\mathsf{Input} := \texttt{TX}.\mathsf{txid} \| 1$$
$$\texttt{TX}_{\texttt{PU}}.\mathsf{Output} := \{(\gamma.\mathsf{cash}, pk_B)\}$$

  Compute $\sigma_p^B \leftarrow \mathsf{Sign}_{sk_B}([\texttt{TX}_{\texttt{PU}}])$, set $\texttt{TX}_{\texttt{PU}} := \{[\texttt{TX}_{\texttt{PU}}], \sigma_p^B\}$ and $(\text{post}, \texttt{TX}_{\texttt{PU}}) \overset{\tau_0+2T}{\longrightarrow} \mathcal{L}$.
- Let $\texttt{TX}_{\texttt{PU}}$ be accepted by $\mathcal{L}$ in round $\tau_1 \leq \tau_0 + 2T + \Delta$. Set $\Theta^B(id) := \perp$.
- If received $(\texttt{PUNISHED}, \mathsf{id}) \overset{\tau_1}{\hookleftarrow} \mathcal{F}_{\mathsf{BC}}^{\mathcal{L}(\Delta,\Sigma,\mathcal{V})}$, then set $\Gamma^B(id) := \perp$ and output $(\texttt{PUNISHED}, id) \overset{\tau_1}{\hookrightarrow} \mathcal{E}$.

Simulator for ForceClose(id)

1) Extract $\texttt{TX}_{\texttt{CM}}$ and $\texttt{TX}_{\texttt{SP}}$ from $\Gamma(id)$.
2) Send $(\text{post}, \texttt{TX}_{\texttt{CM}}) \overset{\tau_0}{\hookrightarrow} \mathcal{L}$.
3) Let $\tau_1 \leq \tau_0 + \Delta$ be the round in which $\texttt{TX}_{\texttt{CM}}$ is accepted by the blockchain. Wait for Δ rounds to $(\text{post}, \texttt{TX}_{\texttt{SP}}) \overset{\tau_2+\Delta}{\hookrightarrow} \mathcal{L}$.
4) Once $\texttt{TX}_{\texttt{SP}}$ is accepted by the blockchain in round $\tau_3 \leq \tau_2 + 2\Delta$, set $\Theta^P(\mathsf{id}) = \perp$ and $\Gamma^P(\mathsf{id}) = \perp$.

Appendix K. ECDSA-based Instantiation of BlindHub

Here we present a concrete instantiation of our protocol that is compatible with Bitcoin. The procedures are roughly the same as described in section 6, here we mainly focus on explaining the intritions behind some technical details and concrete instantiations of the underlying primitives.

Let $\mathbb{G}$ be an elliptic curve group of prime order q with a generator g. Let $(p, \bar{\mathbb{G}}, \bar{g}, \bar{h}_1, \bar{h}_2, \hat{\bar{\mathbb{G}}}, \hat{\bar{g}}, \mathbb{G}_T, e)$ be an (asymmetric) bilinear group, consisting of groups $(\bar{\mathbb{G}}, +)$ and $(\hat{\bar{\mathbb{G}}}, +)$ generated by $\bar{g}/\bar{h}$ and $\hat{\bar{g}}$, resp., and $(\mathbb{G}_T, \cdot)$, and a bilinear map $e : \bar{\mathbb{G}} \times \hat{\bar{\mathbb{G}}} \to \mathbb{G}_T$ such that $e(\bar{\mathbb{G}}, \hat{\bar{\mathbb{G}}})$ generates $\mathbb{G}_T$. Let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a collision resistant hash function. We instantiate the underlying cryptographic primitives that are specific to our protocol as follows. For the commitment scheme Π_{COM}, we instantiate it as Pedersen commitment. For the basic signing algorithm Σ used to sign the transaction, we instantiate it as ECDSA-based 2-of-2 threshold signatures, as described in [13], [36]. For the randomizable puzzle scheme, we set it to Castagnos-Laguillaumie (CL) encryption scheme [14] on the group (B, n, t, s, g, f, G, F) with the message space $\mathbb{Z}_t$, where t is a prime number larger than p^2q. For the RSoRC scheme Π_{RSoRC}, we use the generalized version of scheme described in [4](page 14), where $n = 2$.

For the ease of presentation, we assume that before running the protocol, parties have already generated a joint public key for the threshold signatures. That is, they will share $\mathsf{pk}_{tr}^{\Sigma}$ in the puzzle promise protocol and $\mathsf{pk}_{ts}^{\Sigma}$ in the puzzle solver protocol. Besides, we assume that they have already agreed on a committed transaction $h = H(\mathsf{tx})$. The reason why this assumption is reasonable is that, as mentioned in the section of BlindChannel, the unblind party will prove the pre-image of h is indeed a valid transaction. For the sake of simplicity, we omit this detail in our concrete instantiation and just assume the correctness of h. The concrete instantiation of registration, puzzle promise and puzzle solver protocols are shown in Fig. 29, Fig. 30 and Fig.31 respectively. To ease the understanding of Fig. 29, Fig. 30 and Fig.31, we provide figures Fig.25, Fig.26 and Fig.27 for the general construction of BlindHub described in Section 6 as comparisons.

Registration Protocol: At the beginning, $\mathbb{S}$ generates two commitments: commitment of the token $\mathsf{C}_{\mathsf{tkid}}$ and com-

mitment of the amount $\mathsf{C}_{\mathsf{amt}}$. Since later they will be signed by $\mathbb{T}$ using RSoRC, which is a signing protocol based on bilinear group, we instantiate $\mathsf{C}_{\mathsf{tkid}}$ and $\mathsf{C}_{\mathsf{amt}}$ as Pedersen commitments on group $\bar{\mathbb{G}}$ with the generator $\bar{g}, \bar{h}$(line 6-9). To fix in the structure of RSoRC, we additionally generate R_0 as a placeholder(line 7). Besides, $\mathbb{S}$ generates two NIZK proof to prove the knowledge of $\mathsf{C}_{\mathsf{tkid}}$ and $\mathsf{C}_{\mathsf{amt}}$(line 10, 11). Then $\mathbb{S}$ sends $R_0, \mathsf{C}_{\mathsf{tkid}}, \mathsf{C}_{\mathsf{amt}}, \pi_{\mathsf{tkid}}, \pi_{\mathsf{amt}}$ to $\mathbb{T}$. On receiving $R_0, \mathsf{C}_{\mathsf{tkid}}, \mathsf{C}_{\mathsf{amt}}, \pi_{\mathsf{tkid}}, \pi_{\mathsf{amt}}$, $\mathbb{T}$ firstly checks the validity of $\pi_{\mathsf{tkid}}, \pi_{\mathsf{amt}}$. If both of them are valid, $\mathbb{T}$ performs RSoRC on $\mathsf{C}_{\mathsf{tkid}}$ and $\mathsf{C}_{\mathsf{amt}}$ and sends the signature to $\mathbb{R}$, who aborts if it is invalid(line 15). Finally, $\mathbb{S}$ randomizes $R_0, \mathsf{C}_{\mathsf{tkid}}, \mathsf{C}_{\mathsf{amt}}, \sigma_{\mathsf{tkid}}$ to obtain $R'_0, \mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}$ and sends $R'_0, \mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}$ to $\mathbb{R}$, where $r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}$ are the openings of $\mathsf{C}'_{\mathsf{tkid}}$ $\mathsf{C}'_{\mathsf{amt}}$ respectively(line 12-13).

Puzzle Promise Protocol: On receiving $R'_0, \mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}$, $\mathbb{R}$ generates a NIZK proof π'_{amt} to prove knowledge of $\mathsf{C}'_{\mathsf{amt}}$(line 6), and then a token-uniqueness proof π_{tup}(line 7). The idea of the token-uniqueness proof is follows. Since the commitment of the token $\mathsf{C}'_{\mathsf{tkid}} = \bar{g}^{\mathsf{tkid}}\bar{h}^{r'_{\mathsf{tkid}}}$ is randomizable(it is a Pedersen commitment, and we can easily randomize it by multiplying it by a $h^{r'}$ for some random r'), it cannot be used to prove the uniqueness of the token. Then we generate another commitment of tkid $D = g^{\mathsf{tkid}}$, and perform an equality proof to prove that $\mathsf{C}'_{\mathsf{tkid}}$ and D share the same committed message. Since g^{tkid} is perfectly binding to tkid, tumbler can just store g^{tkid} as a tag to the token tkid. In this manner, the tumbler can detect if the receiver has reused the token by observing if there are two identical tags.

After generating the token-uniqueness proof, $\mathbb{R}$ sends $R'_0, \mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}, \pi'_{\mathsf{amt}}$ to $\mathbb{T}$, who will abort if one of the followings is not valid: $\sigma'_{\mathsf{tkid}}, \pi_{\mathsf{tup}}, \pi'_{\mathsf{amt}}$(line 8-12). If they are all valid, the tumbler samples the adaptor witness and statement (Y, y), generates the randomizable puzzle c_y. Meanwhile, $\mathbb{T}$ starts running the two-party ECDSA protocol and randomizable signatures on randomizable commitments(RSoRC) with $\mathbb{R}$. For the two-party ECDSA protocol, they firstly execute the coin-tossing protocol on the randomness $r = k_t \cdot k_r \cdot y$, where y is unknown to $\mathbb{R}$(line 14-25). Once the randomness is computed, $\mathbb{T}$ performs its side of two-party ECDSA scheme using $c_{\mathsf{sk}_r^{\Sigma}}$, which is the encryption of the sk_r^{Σ} and the additive homomorphic properties of the encryption scheme. It is noted that tumbler does not embed the adaptor witness(i.e., y^{-1}) into the signature(line 23-25). Then the receiver is able to to compute the pre-signature by decrypting the ciphertext received from the $\mathbb{T}$ and performing his part of the signature(line 28-34).

Besides running two-party ECDSA protocol, $\mathbb{T}$ generates randomizale signatures on adaptor statement Y, a group element R_1 as a placeholder and commitment of the amount $\mathsf{C}'_{\mathsf{amt}}$. Here the adaptor statement can also be seen as a randomizable commitment: the committed message is zero and the adaptor witness is the randomness, and $\bar{g}$ is the generator for the randomness. However, there is a technical challenge to overcome for running the RSoRC protocol. Since the scheme in [4] is based on bilinear group$\left(p, \overline{\mathbb{G}}, \bar{g}, \bar{h}, \hat{\bar{\mathbb{G}}}, \hat{\bar{g}}, \mathbb{G}_T, e\right)$, which is different from the group$(\mathbb{G}, g, q)$(on the elliptic curve *secp256k1*) used in the ECDSA protocol, we cannot directly perform RSoRC on the adaptor statement, which is a group element on $(\mathbb{G}, g, q)$. We overcome this challenge by using an equality proof to bridge these two different groups. Specifically, we generate $\bar{Y} \leftarrow \bar{g}^y$ on group $\bar{\mathbb{G}}$, and then we perform an equality proof to prove that Y and $\bar{Y}$ share the same discrete logarithm. In this way, $\mathbb{T}$ can perform RSoRC on $(\bar{Y}, \mathsf{C}'_{\mathsf{amt}})$ rather than $(Y, \mathsf{C}'_{\mathsf{amt}})$(line 18) and get $\tilde{\sigma}$, and sends it to $\mathbb{R}$, who will check the validity of $\tilde{\sigma}$ and abort if it is invalid. Recall that in our general protocol, we also require the tumbler to prove that y is a valid solution to puzzle c_y. This is also important, since otherwise the tumbler can launch the following attack to break the unlinkability, which is also mentioned in $\mathrm{A}^2\mathrm{L}$ [51]: firstly the tumbler $\mathbb{T}$ picks a receiver $\mathbb{R}^*$ to attack. Then tumbler generates (Y^*, y^*) and c_y^*, where the solution to the puzzle c_y^* is different from y^* and sends them to $\mathbb{R}$, and performs the following protocol. Later in the puzzle solver phase when the sender $\mathbb{S}^*$ gives out the randomization of Y^* and c_y^*, the tumbler will find that they are unmatched. Since tumbler only behaves maliciously to $\mathbb{R}^*$, he can easily link $\mathbb{S}^*$ and $\mathbb{R}^*$.

From the above we know that now the tumbler needs to prove that there is an integer $y \in \mathbb{Z}_{min\{q,p,t\}}$, where q is the order of $\mathbb{G}$, p is the order of $\bar{\mathbb{G}}$ and $t > p^2q$ is the size of the message space of CL encryption, such that $\bar{Y} = \bar{g}^y \wedge Y = g^y \wedge c_y = \mathsf{Enc}(y)$ (line 17 in Fig. 30). Assume $c_y = (\tilde{f}^y \tilde{h}^v, \tilde{g}^v)$ for some v, where $\tilde{h} = \tilde{g}^{\mathsf{dk}}$, and H is a random oracle, and $q < p$, such a proof can be provided as follows:

1) sample $r \in [q^2, t], t \in B$ and compute $h_1 = g^r, h_2 = \bar{g}^r, h_3 = \tilde{g}^r \tilde{h}^t$, where B is the security parameter of CL encryption.
2) compute the challenge $e \leftarrow H(h_1, h_2, h_3) \in \mathbb{Z}_q$
3) $z_1 \leftarrow r + ey, z_2 \leftarrow t + ev (z_1, z_2$ are integers without modulo any order)

Besides, the prover performs a range proof π_{range} to prove that the solution in c_y is smaller than q. The verifier aborts if the following does not hold: $g^{z_1} = h_1 Y^e \wedge \bar{g}^{z_1} = h_2 \bar{Y}^e \wedge \tilde{g}^{z_1}\tilde{h}^{z_2} = h_3 c_y^e \wedge \mathsf{NIZKVerify}(\pi_{\mathsf{range}}) = 1$. The security analysis for the above proof is follows. The completeness is straightforward. For the soundness, we perform the classic rewinding technique to obtain two challenges e, e', such that $z_1 = r + ey, z'_1 = r + e'y, z_2 = t + ev$, $z'_2 = t + e'v$, $g^{z_1} = h_1 Y^e \wedge \bar{g}^{z_1} = h_2 \bar{Y}^e \wedge \tilde{g}^{z_1}\tilde{h}^{z_2} = h_3 c_y^e$ and $g^{z'_1} = h_1 Y^{e'} \wedge \bar{g}^{z'_1} = h_2 \bar{Y}^{e'} \wedge \tilde{g}^{z'_1}\tilde{h}^{z'_2} = h_3 c_y^{e'}$, and thus we have $g^{\frac{z_1 - z'_1}{e - e'} \bmod q} = Y, \bar{g}^{\frac{z_1 - z'_1}{e - e'} \bmod p} = \bar{Y}$, $\tilde{g}^{\frac{z_1 - z'_1}{e - e'}} \tilde{h}^{\frac{z_2 - z'_2}{e - e'}} = c_y$ and the witness extraction follows. For zero knowledge, since $r \geq q^2$ is larger than $ey < q^2$, z_1, z_2 would not leak information about the witness y. Finally, the range proof π_{range} guarantees that $y \in \mathbb{Z}_q$. This concludes the proof.

Puzzle Solver Protocol: At the beginning, $\mathbb{T}$ starts the coin tossing protocol similar to the one performed in the puzzle promise protocol(line 4-11). Besides, when

$\mathbb{S}$ performs its coin tossing, he randomizes $c''_y, Y'', \bar{Y}''$, $\mathsf{C}''_{\mathsf{amt}}, \tilde{\sigma}''$ and sends the randomized ones together with R'_s, $\pi_{k'_s}$ to $\mathbb{T}$(line 10-11). $\mathbb{T}$ checks the validity of them and aborts if the signature $\tilde{\sigma}$ or the proof $\pi_{k'_s}$ is invalid. Afterwards, $\mathbb{T}$ solves the puzzle c''_y to obtain y'', and aborts if $Y'' \neq g^{y''} \vee \bar{Y}'' \neq \bar{g}^{y''}$. Essentially, this checks whether the amount and the adaptor witness are still linked to each other. Recall that in our BlindHub protocol, we are using RSoRC to link the commitment of the amount $\mathsf{C}_{\mathsf{amt}}$ and the adaptor statement Y so that we can link the amount and the adaptor witness. But here in order to be compatible with Bitcoin, we generate Y on the curve *Secp256k1*, which is different from the one required by RSoRC scheme (RSoRC requires bilinear group). As a result, RSoRC can only link $\bar{Y}''$ to $\mathsf{C}''_{\mathsf{amt}}$, rather than link Y'' to $\mathsf{C}''_{\mathsf{amt}}$. But the tumbler can still checks whether $\bar{Y}''$ is linked to Y'' by checking whether the puzzle solution y'' is the witness of both $\bar{Y}''$ and Y''. Since if $\bar{Y}''$ does not link to Y'', there will be a solution that can fit $\bar{Y}''$ and Y'' simultaneously. The rest of the protocol is similar to that in the puzzle promise protocol, where they compute a common randomness and perform two-party ECDSA signature. Finally, $\mathbb{T}$ completes the adaptor signature $\widehat{\sigma}_s$ and share it with $\mathbb{S}$, who extracts the witness from it and sends it to $\mathbb{R}$.

$\mathsf{omaSignForge}_{\mathcal{A},\prod_{R,\Sigma}}(\lambda)$

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

$O_{\mathsf{S}}(M, i, j)$

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

$O_{\mathsf{pS}}(M, Y, i, j)$

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

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

Game CL-HID$^{\mathcal{A}}_{\mathsf{RSoRC}}(\lambda)$

$pp \leftarrow\$ \mathsf{Setup}(1^\lambda)$
$\mathsf{CM}_1, \cdots, \mathsf{CM}_n \leftarrow\$ \mathcal{A}(pp)$
$\mathsf{CM}_{01}, \cdots, \mathsf{CM}_{0n} \leftarrow\$ \mathcal{CM}_{pp}$
$\mu \leftarrow\$ \mathcal{R}_{pp}, b \leftarrow\$ \{0,1\}$
$\mathsf{CM}_{1i} := \mathsf{RndCM}(\mathsf{CM}_{0i}, \mu),$
$b' \leftarrow\$ \mathcal{A}(\mathsf{CM}_{bi}, i = 1, \cdots, n)$
return $(b' == b)$

Figure 13. Games for class-hiding

Game UNF$^{\mathcal{A}}_{\mathsf{RSoRC}}(\lambda)$

$\mathsf{par} \leftarrow \mathsf{RSoRC.Setup}(1^\lambda)$
$(\mathsf{sk}^\chi, \mathsf{pk}^\chi) \leftarrow \mathsf{RSoRC.KeyGen}(\mathsf{par})$
$S := \emptyset$
$(\mathsf{CM}^*_1, \cdots, \mathsf{CM}^*_n, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{RCSign}}}(\mathsf{par}, \mathsf{pk}^\chi)$
return $\mathsf{Verify}(\mathsf{CM}^*_1, \cdots, \mathsf{CM}^*_n, \sigma^*) \wedge$
$\{[\mathsf{CM}^*_1, \cdots, \mathsf{CM}^*_n]\} \notin S$

$\mathcal{O}_{\mathsf{RCSign}}(\mathsf{CM}_1, \cdots, \mathsf{CM}_n)$

$\sigma \leftarrow \mathsf{RCSign}(\mathsf{sk}^\chi, \mathsf{CM}_1, \cdots, \mathsf{CM}_n)$
$S := S \cup \{[\mathsf{CM}_1, \cdots, \mathsf{CM}_n]\}$
return $\mathsf{Sign}(\mathsf{sk}^\chi, \mathsf{CM}_1, \cdots, \mathsf{CM}_n)$

Figure 14. The unforgeability game for RSoRC.

aSigForge$_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$

$\mathcal{Q} := \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(Y, y) \leftarrow \mathsf{GenR}(\lambda)$
$M^* \leftarrow \mathcal{A}_1^{O_\mathsf{S}, O_{\mathsf{pS}}}(\mathsf{pk}, Y)$
$\hat{\sigma} \leftarrow \mathsf{PreSign}((\mathsf{pk}, \mathsf{sk}), Y, M^*)$
$\sigma \leftarrow \mathcal{A}_2^{O_\mathsf{S}, O_{\mathsf{pS}}}(\hat{\sigma}, Y)$
$a_1 \leftarrow (M^* \notin \mathcal{Q})$
$a_2 \leftarrow \mathsf{Vf}(\mathsf{pk}, \sigma, M^*)$
return $(a_1 \wedge a_2)$

$\mathcal{O}_S(m)$

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, M)$
$\mathcal{Q} := \mathcal{Q} \cup \{M\}$
return σ

$\mathcal{O}_{pS}(m, Y)$

$\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, Y, M)$
$\mathcal{Q} := \mathcal{Q} \cup \{M\}$
return σ

Figure 15. Experiment aSigForge$_{\mathcal{A},\prod_{R,\Sigma}}$.

aWitExt$_{\mathcal{A},\Xi_{R,\Sigma}}(\lambda)$

$\mathcal{Q} := \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(M^*, Y) \leftarrow \mathcal{A}^{O_\mathsf{S}, O_{\mathsf{pS}}}(\mathsf{pk})$
$\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, Y, M^*)$
$\sigma^* \leftarrow \mathcal{A}^{O_\mathsf{S}, O_{\mathsf{pS}}}(\hat{\sigma})$
$y^* \leftarrow \mathsf{Ext}(Y, \sigma^*, \hat{\sigma})$
$a_1 \leftarrow (M^* \notin \mathcal{Q})$
$a_2 \leftarrow (Y, y^*) \notin L_R$
$a_3 \leftarrow \mathsf{Vf}(\mathsf{pk}, \sigma^*, M^*) = 1)$
return $(a_1 \wedge a_2 \wedge a_3)$

$\mathrm{O}_S(m)$

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, M)$
$\mathcal{Q} := \mathcal{Q} \cup \{M\}$
return σ

$\mathrm{O}_{pS}(m, Y)$

$\hat{\sigma} \leftarrow \mathsf{PreSign}(\mathsf{sk}, Y, M)$
$\mathcal{Q} := \mathcal{Q} \cup \{M\}$
return σ

Figure 16. Experiment aWitExt$_{\mathcal{A},\prod_{R,\Sigma}}$.

Game OMDL$^{\mathcal{A}}_{\mathsf{Setup},n,q}(\lambda)$

$(p, \mathbb{G}, G) \leftarrow \mathsf{Setup}(1^\lambda); S := \emptyset;$
for $i = 1, \ldots, n+1,$
$x_i \leftarrow\$ \mathbb{Z}_p; X_i := x_i G$
$x'_i, \forall i \in [n+1] \leftarrow \mathcal{A}^{\mathrm{DLO}}(p, \mathbb{G}, G, X_1, \ldots, X_{n+1})$
return $(x_i == x'_i, \forall i \in [n+1])$

Oracle DLO(W)

if $i \in S$ or $i \notin [n]$, **return** $\perp$
$S := S \cup \{i\}$
return $\log(W)$

Figure 17. The OMDL problem.

$\mathcal{O}^{\mathsf{Gen}}(i)$

1 : $ek_i \leftarrow\$ \{0,1\}^n$
2 : Either (i, ek_i) into table K
3 : **return** ek_i

$\mathcal{O}^{\mathsf{Enc}}(ek_i, m)$

1 : $c_j \leftarrow\$ \{0,1\}^n$
2 : Either (m, c_j) into table M_i
3 : **return** c_j

$\mathcal{O}^{\mathsf{Dec}}(ek_i, c)$

1 : if $(\cdot, c) \notin M_i$ then return $\perp$
2 : **else**
3 : Look up m corresponding to c in M_i
4 : **return** m

$\mathcal{O}^{\mathsf{Add}}(ek_i, c_0, c_1)$

1 : Look up (m_0, m_1) corresponding to c_0, c_1 in table M_i
2 : $\tilde{c} \leftarrow\$ \{0,1\}^n$
3 : Enter $(m_0 + m_1, \tilde{c})$ into table M_i
4 : **return** $\tilde{c}$

Figure 18. Oracles of Linear-Only Encryption

$\mathsf{ExpBlnd}^{\mathcal{A}}_{\Pi_{\mathsf{FBCS}}}(\lambda)$

$(m_{\mathsf{TR},i}, m_{\mathsf{ST},i}) \leftarrow\$ \mathcal{M}, i = 0, 1$
$(\mathsf{ek}_t, \mathsf{pk}^{\chi}, \mathsf{pk}^{\Sigma}_{t,0}, \mathsf{pk}^{\Sigma}_{t,1}) \leftarrow \mathcal{A}(1^{\lambda})$
$(\mathsf{pk}^{\Sigma}_{s,i}, \mathsf{sk}^{\Sigma}_{s,i}) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^{\lambda}), i = 0, 1$
for $i = 0, 1 : \tau_i \leftarrow$
$\mathsf{Promise}\left\langle \mathcal{A}(\mathsf{pk}^{\Sigma}_{s,0}, \mathsf{pk}^{\Sigma}_{s,1}, \mathsf{com}(m_{\mathsf{TR},i})), \mathbb{R}(\mathsf{ek}_t, \mathsf{pk}^{\chi}, \mathsf{pk}^{\Sigma}_{t,i}, m_{\mathsf{TR},i}) \right\rangle$
$b \leftarrow \{0,1\}$
for $i = 0, 1 : (\sigma^*_i, s_i) \leftarrow$
$\mathsf{Solver}\left\langle \mathbb{S}\left(\mathsf{sk}^{\Sigma}_{s,i}, \mathsf{ek}_t, \mathsf{pk}^{\chi}, m_{\mathsf{ST},i}, \tau_{i\oplus b}\right), \mathcal{A}\left(\mathsf{com}(m_{\mathsf{ST},i})\right) \right\rangle$
if $(\sigma^*_0 = \perp) \vee (\sigma^*_1 = \perp) \vee (\tau_0 = \perp) \vee (\tau_1 = \perp)$
  $\sigma_0 := \sigma_1 := \perp$
else
  for $i = 0, 1 : \sigma_{i\oplus b} \leftarrow \mathsf{Open}(\tau_{i\oplus b}, s_i)$
$b' \leftarrow \mathcal{A}(\sigma_0, \sigma_1)$
return $(b = b')$

Figure 19. Blindness experiment

$\mathsf{ExpUnlock}^{\mathcal{A}}_{\Pi_{\mathsf{FBCS}}}(\lambda)$

$(\mathsf{ek}_t, \mathsf{pk}^{\Sigma}_t, \mathsf{pk}^{\chi}, m_{\mathsf{TR}}, m_{\mathsf{ST}}) \leftarrow \mathcal{A}(1^{\lambda})$
$(\mathsf{pk}^{\Sigma}_s, \mathsf{sk}^{\Sigma}_s) \leftarrow \Pi_{\mathsf{DS}}.\mathsf{KGen}(1^{\lambda})$
$\tau \leftarrow \mathsf{Promise}\left\langle \mathcal{A}(\mathsf{pk}^{\Sigma}_s), \mathbb{R}(\mathsf{ek}_t, \mathsf{pk}^{\Sigma}_t, \mathsf{pk}^{\chi}, m_{\mathsf{TR}}) \right\rangle$
if $\tau = \perp$:
  $(\hat{\sigma}, \hat{m}) \leftarrow \mathcal{A}, b_0 := (\mathsf{Vf}(\mathsf{pk}^{\Sigma}_s, \hat{\sigma}, \hat{m}) = 1)$
if $\tau \neq \perp$:
  $(\sigma^*, s) \leftarrow \mathsf{Solver}\left\langle \mathbb{S}\left(\mathsf{sk}^{\Sigma}_s, \mathsf{ek}_t, \mathsf{pk}^{\chi}, m_{\mathsf{ST}}, \tau\right), \mathcal{A} \right\rangle$
  $(\hat{\sigma}, \hat{m}) \leftarrow \mathcal{A}$
  $b_1 := (\mathsf{Vf}(\mathsf{pk}^{\Sigma}_s, \hat{\sigma}, \hat{m}) = 1) \wedge (\hat{m} \neq m_{\mathsf{ST}})$
  $b_2 := (\mathsf{Vf}(\mathsf{pk}^{\Sigma}_s, \sigma^*, m_{\mathsf{ST}}) = 1)$
  $b_3 := (\mathsf{Vf}(\mathsf{pk}^{\Sigma}_t, m_{\mathsf{ST}}, \mathsf{Open}(\tau, s)) \neq 1)$
return $b_0 \vee b_1 \vee (b_2 \wedge b_3)$

Figure 20. Unlockability experiment

$\mathsf{ExpUnforg}^{\mathcal{A}}_{\Pi_{\mathsf{FBCS}}}(\lambda)$

$\mathcal{L} := \emptyset, \mathcal{T} := \emptyset, \mathcal{U} := \emptyset, \mathcal{M} := \emptyset, Q := 0$
$(\mathsf{ek}_t, \mathsf{dk}_t) \leftarrow \mathsf{Setup}(1^{\lambda})$
$(\mathsf{pk}^{\chi}, \mathsf{sk}^{\chi}) \leftarrow \Pi_{\mathsf{RSoRC}}.\mathsf{KGen}(1^{\lambda})$
$\{(\mathsf{pk}^{\Sigma}_{t,i}, m_i^{[\mathsf{amt}_i]}, \sigma_i)\}_{\forall i \in [q]} \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{PP}(\cdot), \mathcal{O}\mathsf{PS}(\cdot)}(\mathsf{ek}_t, \mathsf{pk}^{\chi})$
$b_0 := \exists i \in [q]$ s.t. $(\mathsf{pk}^{\Sigma}_{t,i}, \cdot) \in \mathcal{L} \wedge (\mathsf{pk}^{\Sigma}_{t,i}, m_i^{[\mathsf{amt}_i]}) \notin \mathcal{L}$
  $\wedge\, \mathsf{Vf}(\mathsf{pk}^{\Sigma}_{t,i}, m_i^{[\mathsf{amt}_i]}, \sigma_i) = 1$
$b_1 := \forall i \in [q], (\mathsf{pk}^{\Sigma}_{t,i}, m_i^{[\mathsf{amt}_i]}) \in \mathcal{L} \wedge \mathsf{Vf}(\mathsf{pk}^{\Sigma}_{t,i}, m_i^{[\mathsf{amt}_i]}, \sigma_i) = 1$
$b_2 := \bigwedge_{i,j \in [q], i \neq j} (\mathsf{pk}^{\Sigma}_{t,i}, m_i^{[\mathsf{amt}_i]}, \sigma_i) \neq (\mathsf{pk}^{\Sigma}_{t,j}, m_j^{[\mathsf{amt}_j]}, \sigma_j)$
$b_3 := (Q \leq q - 1)$
$b_4 := (\mathcal{T} \notin \{\mathsf{amt}_1, \cdots, \mathsf{amt}_q\})$
return $b_0 \vee (b_1 \wedge b_2 \wedge (b_3 \vee b_4))$

$\mathcal{O}\mathsf{PP}(m_{\mathsf{TR}}^{[\mathsf{amt}_{\mathsf{TR}}]}, \mathsf{C}_{\mathsf{amt}})$

$(\mathsf{pk}^{\Sigma}_t, \mathsf{sk}^{\Sigma}_t) \leftarrow \Pi_{\mathsf{BAS}}.\mathsf{KGen}(1^{\lambda})$
$\mathcal{L} := \mathcal{L} \cup \{(\mathsf{pk}^{\Sigma}_t, m_{\mathsf{TR}}^{[\mathsf{amt}_{\mathsf{TR}}]})\}$
$(\tilde{\sigma}, Y, \mathsf{C}_{\mathsf{amt}}, \cdot) \leftarrow \mathsf{Promise}\langle \mathbb{T}(\mathsf{dk}, \mathsf{sk}^{\Sigma}_t, \mathsf{sk}^{\chi}, m_{\mathsf{TR}}^{[\mathsf{amt}_{\mathsf{TR}}]}), \mathcal{A}(\mathsf{pk}^{\Sigma}_t) \rangle$

$\mathcal{O}\mathsf{PS}(\mathsf{pk}^{\Sigma}_s, Y', c'_y, \tilde{\sigma}', \mathsf{C}'_{\mathsf{amt}}, m_{\mathsf{ST}}^{[\mathsf{amt}_{\mathsf{ST}}]})$

$\sigma^* \leftarrow \mathsf{Solver}\langle \mathcal{A}, \mathbb{T}(\mathsf{dk}_t, \mathsf{sk}^{\chi}, \mathsf{pk}^{\Sigma}_s, m_{\mathsf{ST}}^{[\mathsf{amt}_{\mathsf{ST}}]}) \rangle$
if $\sigma^* \neq \perp$ **then** $Q := Q + 1, \mathcal{T} := \mathcal{T} \cup \{\mathsf{amt}_{\mathsf{ST}}\}$,
$\mathcal{U} := \mathcal{U} \cup \{(\mathsf{amt}_{\mathsf{ST}}, Y', c'_y, \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma}')\}$

Figure 21. Unforgeability experiment

$\mathrm{OM\text{-}CCA\text{-}BlindHub}^{\mathcal{A}}_{\Pi_{\mathsf{E}},q}$

$Q := 0$
$(\mathsf{ek}, \mathsf{dk}) \leftarrow \Pi_{\mathsf{E}}.\mathsf{KGen}(1^\lambda)$
$(\mathsf{pk}^\chi, \mathsf{sk}^\chi) \leftarrow \Pi_{\mathsf{RSoRC}}.\mathsf{KGen}(1^\lambda)$
$y_1, \ldots, y_{q+1} \leftarrow\!\!\$\ \{0,1\}^\lambda, Y_i \leftarrow g^{y_i}, i = 1, \cdots, q+1$
$\mathsf{C}_{\mathsf{amt}} \leftarrow \mathcal{O}^{\Pi_{\mathsf{COM}}}, \tilde{\sigma}_i \leftarrow \mathsf{RCSign}(\mathsf{sk}^\chi, \mathsf{C}_{\mathsf{amt}}, Y_i), i = 1, \cdots, q+1$
$c_{y,i} \leftarrow \Pi_{\mathsf{E}}.\mathsf{Enc}(\mathsf{ek}, y_i)$
$(y'_1, \ldots, y'_{q+1}) \leftarrow$
$\mathcal{A}^{\mathcal{O}^{\mathsf{BlindHub}}_{\mathsf{dk},\Pi_{\mathsf{E}},\Pi_{\mathsf{BAS}},\Pi_{\mathsf{RSoRC}}}}(\mathsf{ek}, \mathsf{pk}^\chi, (c_{y,i}, g^{y_i}, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma}_i), \forall i \in [q+1])$
if $y'_i = y_i\ \forall i \in 1, \ldots, q+1 \wedge Q \leq q$ **then return** 1
else return 0

$\mathcal{O}^{\mathsf{BlindHub}}_{\mathsf{dk},\Pi_{\mathsf{E}},\Pi_{\mathsf{BAS}},\Pi_{\mathsf{RSoRC}}}(\mathsf{pk}^\Sigma_s, \mathsf{pk}^\chi, m, \hat{\sigma}, Y, c_y, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma})$

check if
$\mathsf{vk} \in \mathsf{Supp}(\Pi_{\mathsf{BAS}}.\mathsf{KGen}(1^\lambda)) \wedge \mathsf{RCVerify}(\mathsf{pk}^\chi, Y, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma}) = 1 :$
  $y \leftarrow \Pi_{\mathsf{E}}.\mathsf{Dec}(\mathsf{dk}, c_y)$
if $\Pi_{\mathsf{BAS}}.\mathsf{PreVf}(\mathsf{pk}^\Sigma_s, m, Y, \tilde{\sigma}) = 1 \wedge g^y = Y$
  $Q := Q + 1$
  return $\sigma' \leftarrow \Pi_{\mathsf{BAS}}.\mathsf{Adapt}(\hat{\sigma}, y)$
else return $\perp$

Figure 22. OM-CCA-BlindHub game

$\mathsf{Prover}(m_1, m_2, m_3, \mathsf{pk}_{U'}; \mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3, f_{\mathbf{PubIn}})$		$\mathsf{Verifier}(\mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3, f_{\mathbf{PubIn}})$
$s = (m_1, m_2, m_3, \mathsf{pk}_U)$	$\mathcal{F}_{\mathsf{COT}}(\mathsf{choose}, \{s_i\}_{i\in[3\alpha+\beta]]})$	$\mathsf{Garble}(1^k, f_{\mathbf{PubIn}}) \rightarrow$
		$(\mathsf{GC}, \{L_0^{(i)}, L_1^{(i)}\}_{i\in[3\alpha+\beta]}, \{Z_0^{(i)}, Z_1^{(i)}\}_{i\in[3\alpha]})$
		$\forall i \in [3\alpha], z_0^{(i)} \xleftarrow{\$} \mathbb{Z}_p$and set $z_1^{(i)} := z_0^{(i)} + \delta$
$\{\hat{Z}_{s_i}^{(i)}\}_{i\in[3\alpha]} \leftarrow \mathsf{Eval}(\mathsf{GC}, \{L'^{(i)}\}_{i\in[3\alpha+\beta]})$	$\{c_0^{(i)}, c_1^{(i)}\}_{i\in[3\alpha]}$ $\mathsf{GC}, \mathcal{F}_{\mathsf{COT}}(\mathsf{transfer}, \{L_{s_i}^{(i)}\}_{i\in[3\alpha+\beta]})$	$\forall b \in \{0,1\}, \forall i \in [3\alpha], c_b^{(i)} := z_b^{(i)} + H(Z_b^{(i)})$
$\forall i \in [3\alpha]$, set $\hat{z}_{s_i}^{(i)} := c_{s_i}^{(i)} - H(\hat{Z}_{s_i}^{(i)})$		
set $\hat{t_1} \xleftarrow{\$} \mathbb{Z}_p$; compute $\hat{B_1} = \sum_{i=1}^{\alpha} 2^{i-1}\hat{z}_{s_i}^{(i)} G + \hat{t_1} H$		
set $\hat{t_2} \xleftarrow{\$} \mathbb{Z}_p$; compute $\hat{B_2} = \sum_{i=\alpha+1}^{2\alpha} 2^{i-1}\hat{z}_{s_i}^{(i)} G + \hat{t_2} H$		
set $\hat{t_3} \xleftarrow{\$} \mathbb{Z}_p$; compute $\hat{B_3} = \sum_{i=2\alpha+1}^{3\alpha} 2^{i-1}\hat{z}_{s_i}^{(i)} G + \hat{t_3} H$	$\hat{B_1}, \hat{B_2}, \hat{B_3}$	compute $B_1 := \delta^{-1}(\hat{B_1} - \sum_{i=1}^{\alpha} 2^{i-1} z_0^{(i)})$
		compute $B_2 := \delta^{-1}(\hat{B_2} - \sum_{i=\alpha+1}^{2\alpha} 2^{i-1} z_0^{(i)})$
	$\mathcal{F}_{\mathsf{COT}}(\mathsf{open\_all}), \{Z_0^{(i)}, Z_1^{(i)}\}_{i\in[3\alpha]}, \delta$	compute $B_3 := \delta^{-1}(\hat{B_3} - \sum_{i=2\alpha+1}^{3\alpha} 2^{i-1} z_0^{(i)})$
set $\hat{z}_{\neg s_i}^{(i)} = \hat{z}_{s_i}^{(i)} + (-1)^{s_i}\delta, \forall i \in [3\alpha]$ set $t_1' := \delta^{-1}\hat{t_1}, t_2' := \delta^{-1}\hat{t_2}, t_3' := \delta^{-1}\hat{t_3}$ abort if $\mathsf{Verify}(\mathsf{GC}, f, \{L_0^{(i)}, L_1^{(i)}\}, \{Z_0^{(i)}, Z_1^{(i)}\}) = 0$ or $\exists i \in [3\alpha], b \in \{0,1\} : c_b^{(i)} \neq_p z_b^{(i)} + H(Z_b^{(i)})$		
	PoK consistency for $(B_i, \hat{B_i})_{i\in[3]}$	
		accept iff PoK is accepting

Figure 23. Description of our protocol for hybrid statements for $\mathcal{R}_{\mathsf{all}}$. The amount length is α and public key length is β.

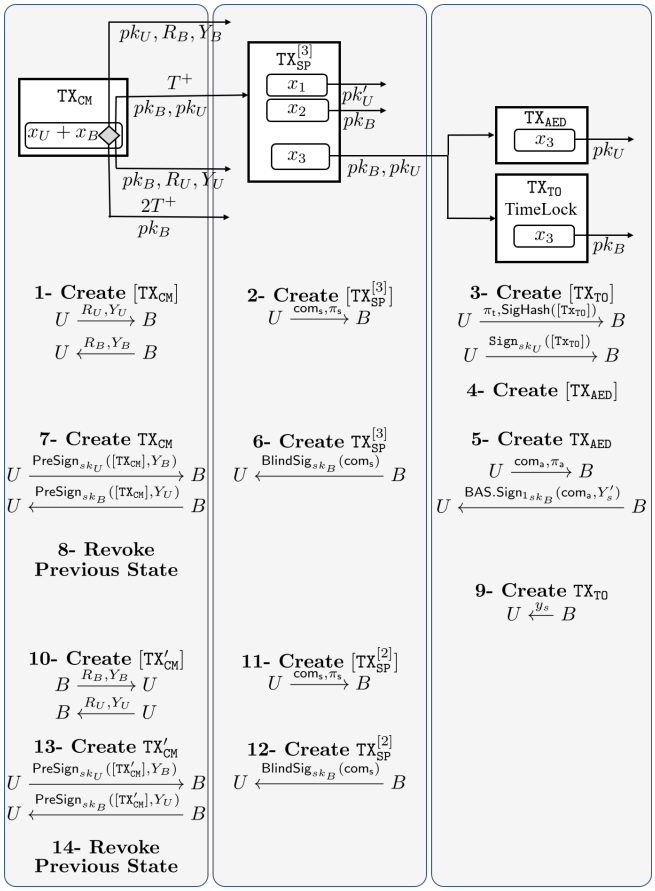

Figure 24. Update of a Blind Channel Initiated by Payee

	$\mathbb{S}(\mathsf{pk}^{\mathcal{X}}, \mathsf{amt})$		$\mathbb{T}(\mathsf{sk}^{\mathcal{X}}, \mathsf{pk}^{\mathcal{X}})$
2 :	Sample a token tkid and randomness $r_{\mathsf{tkid}}, r_{\mathsf{amt}}$		
3 :	$\mathsf{C}_{\mathsf{tkid}} = \mathsf{com}(\mathsf{tkid}, r_{\mathsf{tkid}})$		
4 :	$\mathsf{C}_{\mathsf{amt}} = \mathsf{com}(\mathsf{amt}, r_{\mathsf{amt}})$		
5 :	$\pi_{\mathsf{tkid}} \leftarrow \{\mathsf{tkid}, r_{\mathsf{tkid}} \vert \mathsf{C}_{\mathsf{tkid}} = \mathsf{com}(\mathsf{tkid}, r_{\mathsf{tkid}})\}$		
6 :	$\pi_{\mathsf{amt}} \leftarrow \{\mathsf{amt}, r_{\mathsf{amt}} \vert \mathsf{C}_{\mathsf{amt}} = \mathsf{com}(\mathsf{amt}, r_{\mathsf{amt}})\}$		
7 :		$\xrightarrow{\mathsf{C}_{\mathsf{tkid}}, \mathsf{C}_{\mathsf{amt}}, \pi_{\mathsf{tkid}}, \pi_{\mathsf{amt}}}$	Abort if:
8 :			$\mathsf{NIZKVerify}(\pi_{\mathsf{tkid}}) \neq 1 \vee$
9 :			$\mathsf{NIZKVerify}(\pi_{\mathsf{amt}}) \neq 1$
10 :		$\xleftarrow{\sigma_{\mathsf{tkid}}}$	$\sigma_{\mathsf{tkid}} \leftarrow \mathsf{RCSign}(\mathsf{sk}^{\mathcal{X}}, \mathsf{C}_{\mathsf{tkid}}, \mathsf{C}_{\mathsf{amt}})$
11 :	**if** $\mathsf{RCVerify}(\mathsf{pk}^{\mathcal{X}}, \sigma_{\mathsf{tkid}}) \neq 1$ then abort		
12 :	$\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}} \leftarrow \mathsf{RCRand}(\mathsf{C}_{\mathsf{tkid}}, \mathsf{C}, \sigma_{\mathsf{tkid}})$		
13 :	Send $(\mathsf{tkid}, \mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}})$ to $\mathbb{R}$		
14 :	**return** $(\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \mathsf{tkid}, r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}})$		**return** $\top$

Figure 25. Registration Protocol.

	$\mathbb{T}(\mathsf{sk}_t^{\Sigma}, \mathsf{sk}^{\mathcal{X}}, \mathsf{dk}_t, \mathsf{pk}_r^{\Sigma}, h = H(\mathsf{tx}))$		$\mathbb{R}(\mathsf{sk}_r^{\Sigma}, \mathsf{pk}_t^{\Sigma}, \mathsf{pk}^{\mathcal{X}}, h = H(\mathsf{tx}))$
2 :			$\pi'_{\mathsf{amt}} \leftarrow$
3 :			$\mathsf{NIZK}\{(\mathsf{amt}, r'_{\mathsf{amt}}) \vert \mathsf{C}'_{\mathsf{amt}} = \mathsf{com}(\mathsf{amt}, r'_{\mathsf{amt}})\}$
4 :		$\xleftarrow{\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}, \pi_{\mathsf{tup}}, \pi'_{\mathsf{amt}}}$	$\pi_{\mathsf{tup}} \leftarrow \mathsf{NIZK}_{\mathsf{tup}}(\mathsf{C}'_{\mathsf{tkid}}, \mathsf{tkid})$
5 :	Abort if :		
6 :	$\mathsf{RCVerify}(\mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}}) \neq 1 \vee$		
7 :	$\mathsf{NIZKVerify}(\pi'_{\mathsf{amt}}) \neq 1$		
8 :	$\mathsf{NIZKVerify}(\pi_{\mathsf{tup}}) \neq 1$		
9 :	Else add π_{tup} into $\mathcal{T}$		
10 :	$\mathsf{GenR} \rightarrow (Y, y); \mathsf{Enc}(\mathsf{dk}_t, y) \rightarrow c_y$		
11 :	$\pi_y \leftarrow \mathsf{P}_{\mathsf{NIZK}}\{\exists y \vert (Y, y) \in L_R \wedge c_y = \mathsf{Enc}(y)\}$		
12 :	$\pi_{\mathsf{dk}_t} \leftarrow \mathsf{P}_{\mathsf{NIZK}}\{\mathsf{dk}_t \vert (\mathsf{ek}_t, \mathsf{dk}_t) \in \mathsf{Supp}(\Pi_{\mathsf{Enc}}.\mathsf{KGen}(1^{\lambda}))\}$		
13 :	$\pi_{\mathsf{sk}^{\mathcal{X}}} \leftarrow \mathsf{P}_{\mathsf{NIZK}}\{\mathsf{sk}^{\mathcal{X}} \vert (\mathsf{sk}^{\mathcal{X}}, \mathsf{pk}^{\mathcal{X}}) \in \mathsf{Supp}(\Pi_{\mathsf{RSoRC}}.\mathsf{KGen}(1^{\lambda}))\}$		
14 :	$\tilde{\sigma} \leftarrow \mathsf{RCSign}(Y, \mathsf{C}'_{\mathsf{amt}})$		
15 :	$\widehat{\sigma}'_t \leftarrow \mathsf{BAS.Sign}_1(\mathsf{sk}_t^{\Sigma}, h, Y)$	$\xrightarrow{c_y, \pi_y, \pi_{\mathsf{dk}_t}, \pi_{\mathsf{sk}^{\mathcal{X}}}, \widehat{\sigma}'_t, \tilde{\sigma}}$	
16 :			Abort if:
17 :			$\mathsf{NIZKVerify}(\pi_y, (c_y, Y)) \neq 1 \vee$
18 :			$\mathsf{NIZKVerify}(\pi_{\mathsf{dk}_t}) \neq 1 \vee$
19 :			$\mathsf{NIZKVerify}(\pi_{\mathsf{sk}^{\mathcal{X}}}) \neq 1 \vee$
20 :			$\mathsf{PreVf}(\mathsf{pk}_t^{\Sigma}, Y, \hat{\sigma}'_t) \neq 1 \vee$
21 :			$\mathsf{RCVerify}(\mathsf{pk}^{\mathcal{X}}, \tilde{\sigma}) \neq 1$
22 :			$\left(Y', \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma}', \beta\right) \leftarrow \mathsf{RCRand}(Y, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma})$
23 :			$c'_y \leftarrow \mathsf{PRand}(c_y, \beta)$
24 :			Let $\mathsf{C}'_{\mathsf{amt}} = \mathsf{com}(\mathsf{amt}, r')$
25 :			Send $c'_y, Y', \mathsf{C}'_{\mathsf{amt}}, r', \tilde{\sigma}'$ to Sender
26 :			Set $\Pi := \left(\beta, \left(\mathsf{pk}_t^{\Sigma}, \mathsf{pk}_r^{\Sigma}\right), \mathsf{tx}(\mathsf{amt}), \left(\hat{\sigma}'_t, \sigma'_r\right)\right)$
27 :	**return** $\top$		**return** $\left(\Pi, \left(c_y, Y, \mathsf{C}_{\mathsf{amt}}, \tilde{\sigma}, c'_y, Y', \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma}'\right)\right)$

Figure 26. The Puzzle Promise Phase

	$\mathbb{S}(\mathsf{sk}_s^{\Sigma}, Y', c'_y, \tilde{\sigma}', \mathsf{C}'_{\mathsf{amt}}, \mathsf{pk}_t^{\Sigma})$		$\mathbb{T}(\mathsf{sk}_t^{\Sigma}, \mathsf{pk}_s^{\Sigma}, h' = H(\mathsf{tx}'))$
2 :	$\left(Y'', \mathsf{C}''_{\mathsf{amt}}, \tilde{\sigma}'', \alpha\right) \leftarrow \mathsf{RCRand}(\mathsf{pp}, Y', \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma}')$		
3 :	$c''_y \leftarrow \mathsf{PRand}(c'_y, \alpha), Y'' \leftarrow Y'^{\alpha}$		
4 :	$\pi''_{\mathsf{amt}} \leftarrow \mathsf{P}_{\mathsf{NIZK}}\{(\mathsf{amt}, r''_{\mathsf{amt}}) \mid \mathsf{C}''_{\mathsf{amt}} = \mathsf{com}(\mathsf{amt}, r''_{\mathsf{amt}})\}$		
5 :	$\pi_{\mathsf{sk}_s^{\Sigma}} \leftarrow \mathsf{P}_{\mathsf{NIZK}}\{\mathsf{sk}_s^{\Sigma} \mid (\mathsf{sk}_s^{\Sigma}, \mathsf{pk}_s^{\Sigma}) \in \mathsf{Supp}(\Pi_{\mathsf{AS}}.\mathsf{KGen}(1^{\lambda}))\}$		
6 :	$\widehat{\sigma}_s \leftarrow \mathsf{PreSign}(\mathsf{sk}_s^{\Sigma}, \mathsf{tx}', Y'')$	$\xrightarrow{c''_y, Y'', \mathsf{C}''_{\mathsf{amt}}, \tilde{\sigma}'', \pi''_{\mathsf{amt}}, \pi_{\mathsf{sk}_s^{\Sigma}}, \widehat{\sigma}_s}$	Abort if:
7 :			$\mathsf{PreVf}(\widehat{\sigma}_s, h', Y'') \neq 1 \vee$
8 :			$\mathsf{NIZKVerify}(\pi_{\mathsf{sk}_s^{\Sigma}}) \neq 1 \vee$
9 :			$\mathsf{NIZKVerify}(\pi''_{\mathsf{amt}}) \neq 1 \vee$
10 :			$\mathsf{RCVerify}(\tilde{\sigma}'', Y'', \mathsf{C}''_{\mathsf{amt}}) \neq 1$
11 :			$y'' := \mathsf{Dec}(\mathsf{sk}_t^{\Sigma}, c''_y)$
12 :		$\xleftarrow{\sigma_s}$	$\sigma_s \leftarrow \mathsf{Adapt}(\widehat{\sigma}_s, y'')$
13 :	Abort if: $\mathsf{Vf}(\mathsf{pk}_s^{\Sigma}, \sigma_s) \neq 1$		
14 :	$y'' := \mathsf{Ext}(\widehat{\sigma}_s, \sigma_s, Y'')$ and abort if $y'' = \perp$		
15 :	$y' := y''\alpha^{-1}$ and send y' to $\mathbb{R}$		
16 :	**return** y'		**return** $\top$

Figure 27. The Puzzle Solver Phase

$\mathbb{R}(\Pi, y')$
Parse Π as $\left(\beta, \left(\mathsf{pk}_t^{\Sigma}, \mathsf{pk}_r^{\Sigma}\right), \mathsf{tx}, \left(\hat{\sigma}'_t, \sigma'_r\right)\right)$
$y \leftarrow y' \cdot \beta^{-1}$
$\sigma'_t := \mathsf{Adapt}\left(\hat{\sigma}'_t, y\right)$
return (σ'_t, σ'_r)

Figure 28. The Open Phase

Public parameters: $(\mathbb{G}, g, q), (p, \bar{\mathbb{G}}, \bar{g}, \bar{h}_1, \bar{h}_2, \hat{\mathbb{G}}, \hat{g}, \mathbb{G}_T, e)$

Common Input: $\mathsf{pk}_t^{\mathcal{X}} = (\hat{Y}_i = \hat{g}^{x_i}, i = 0, 1, 2)$

Tumbler's private Input: $\mathsf{sk}_t^{\mathcal{X}} = (x_0, x_1, x_2)$.

Sender's private Input:amt

$\mathbb{S}$		$\mathbb{T}$
$\mathsf{tkid}, r_0, r_{\mathsf{tkid}}, r_{\mathsf{amt}} \leftarrow\$ \mathbb{Z}_p$		
$R_0 = \bar{g}^{r_0}$		
$\mathsf{C}_{\mathsf{tkid}} = \bar{g}^{\mathsf{tkid}} \bar{h}_1^{r_{\mathsf{tkid}}}$		
$\mathsf{C}_{\mathsf{amt}} = \bar{g}^{\mathsf{amt}} \bar{h}_2^{r_{\mathsf{amt}}}$		
$\pi_{\mathsf{tkid}} \leftarrow \{\mathsf{tkid}, r_{\mathsf{tkid}} \mid \mathsf{C}_{\mathsf{tkid}} = \bar{g}^{\mathsf{tkid}} \bar{h}_1^{r_{\mathsf{tkid}}}\}$		
$\pi_{\mathsf{amt}} \leftarrow \{\mathsf{amt}, r_{\mathsf{amt}} \mid \mathsf{C}_{\mathsf{amt}} = \bar{g}^{\mathsf{amt}} \bar{h}_2^{r_{\mathsf{amt}}}\}$		
	$\xrightarrow{R_0, \mathsf{C}_{\mathsf{tkid}}, \mathsf{C}_{\mathsf{amt}}, \pi_{\mathsf{tkid}}, \pi_{\mathsf{amt}}}$	Abort if:
		$\mathsf{NIZKVerify}(\pi_{\mathsf{tkid}}) \neq 1 \vee$
		$\mathsf{NIZKVerify}(\pi_{\mathsf{amt}}) \neq 1$
RCVerify: Abort if the following does not hold: $e(A, \hat{U}) = e(\bar{g}, \hat{\bar{g}}) e(R_0, \hat{Y}_0) e(\mathsf{C}_{\mathsf{tkid}}, \hat{Y}_1) e(\mathsf{C}_{\mathsf{amt}}, \hat{Y}_2)$ $\wedge e(U, \hat{\bar{g}}) = e(\bar{g}, \hat{U}) \wedge$ $e(T, \hat{U}) = e(\bar{g}, \hat{Y}_0) e(\bar{h}_1, \hat{Y}_1) e(\bar{h}_2, \hat{Y}_2)$	$\xleftarrow{\sigma_{\mathsf{tkid}}}$	RSoRC: $\mathrm{u} \leftarrow\$ \mathbb{Z}_p, A = (\bar{g} R_0^{x_0} \mathsf{C}_{\mathsf{tkid}}^{x_1} \mathsf{C}_{\mathsf{amt}}^{x_2})^{\frac{1}{u}}, T = (\bar{g}^{x_0} \bar{h}_1^{x_1} \bar{h}_2^{x_2})^{\frac{1}{u}}$ $\sigma_{\mathsf{tkid}} \leftarrow (A, U = \bar{g}^u, \hat{U} = \hat{\bar{g}}^u, T)$
RCRand: $\mu, \delta \leftarrow \mathbb{Z}_p, r'_{\mathsf{amt}} \leftarrow r_{\mathsf{amt}} + \mu, r'_{\mathsf{tkid}} \leftarrow r_{\mathsf{tkid}} + \mu$ $\mathsf{C}'_{\mathsf{amt}} \leftarrow \bar{g}^{\mathsf{amt}} \bar{h}_2^{r'_{\mathsf{amt}}}, \mathsf{C}'_{\mathsf{tkid}} \leftarrow \bar{g}^{\mathsf{tkid}} \bar{h}_1^{r'_{\mathsf{tkid}}}$ $A' \leftarrow (A T^{\mu})^{\frac{1}{\delta}}, T' \leftarrow T^{\frac{1}{\delta}}, U' \leftarrow U^{\delta}, \hat{U}' \leftarrow \hat{U}^{\delta}$ $\sigma'_{\mathsf{tkid}} \leftarrow (A', U', \hat{U}', T')$		
Send $(R_0, \mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \mathsf{tkid}, r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}})$ to $\mathbb{R}$		
return $(R_0, \mathsf{C}'_{\mathsf{tkid}}, \mathsf{C}'_{\mathsf{amt}}, \mathsf{tkid}, r'_{\mathsf{tkid}}, r'_{\mathsf{amt}}, \sigma'_{\mathsf{tkid}})$		**return** $\top$

Figure 29. Registration Protocol of ECDSA-based Construction

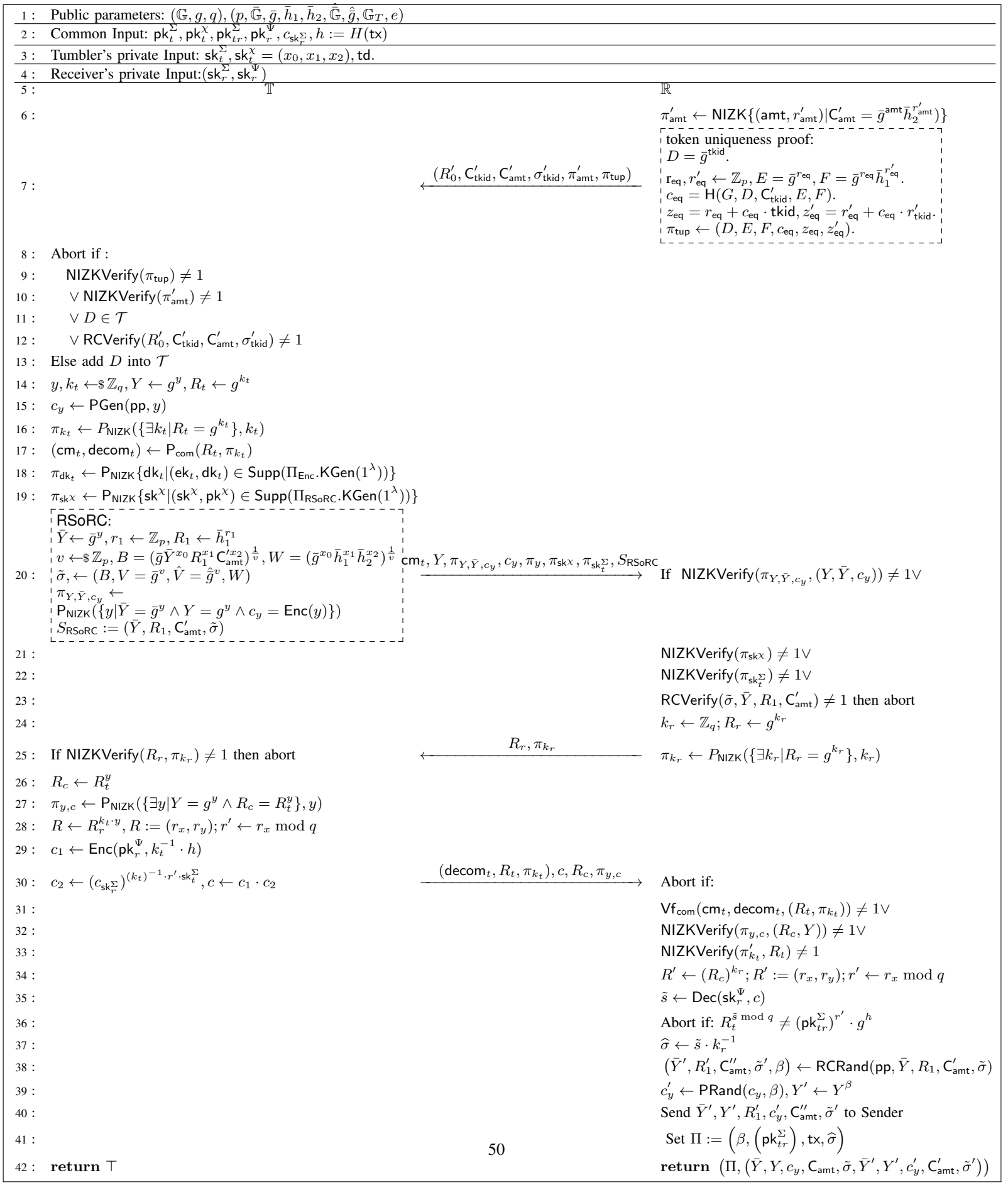

Figure 30. Puzzle Promise Protocol of ECDSA-based Construction

1 :	Public parameters: group description$(\mathbb{G}, g, q)$		
2 :	Common Input: $\mathsf{ek}_s, c_{\mathsf{sk}_s^\Sigma}, h' = \mathsf{H}(\mathsf{tx}')$		
3 :	$\mathbb{S}((\mathsf{sk}_s^\Sigma, \mathsf{dk}_s), \mathsf{pp}, Y', \bar{Y}', c'_y, \tilde{\sigma}')$		$\mathbb{T}(\mathsf{sk}_t^\Sigma)$
4 :			$k'_t \leftarrow\$ \mathbb{Z}_q, R'_t \leftarrow g^{k'_t}$
5 :			$\pi_{k'_t} \leftarrow P_{\mathsf{NIZK}}(\{\exists k'_t \vert R_t = g^{k'_t}\}, k'_t)$
6 :			$(\mathsf{cm}'_t, \mathsf{decom}'_t) \leftarrow \mathsf{P}_{\mathsf{com}}(R'_t, \pi_{k'_t})$
7 :		$\xleftarrow{\quad \mathsf{cm}'_t \quad}$	
8 :	$k'_s \leftarrow\$ \mathbb{Z}_q, R'_s \leftarrow g^{k'_s}$		
9 :	$\pi_{k'_s} \leftarrow P_{\mathsf{NIZK}}(\{\exists k'_s \vert R'_s = g^{k'_s}\})$		
10 :	$\left(\bar{Y}'', R''_1, \mathsf{C}''_{\mathsf{amt}}, \tilde{\sigma}'', \alpha\right) \leftarrow \mathsf{RCRand}(\mathsf{pp}, \bar{Y}', R'_1, \mathsf{C}'_{\mathsf{amt}}, \tilde{\sigma}')$		
11 :	$c''_y \leftarrow \mathsf{PRand}(c'_y, \alpha), Y'' \leftarrow Y'^{\alpha}$		
12 :	$\pi''_{\mathsf{amt}} \leftarrow \mathsf{P}_{\mathsf{NIZK}}\{(\mathsf{amt}, r''_{\mathsf{amt}}) \vert \mathsf{C}''_{\mathsf{amt}} = \mathsf{com}(\mathsf{amt}, r''_{\mathsf{amt}})\}$		
13 :	$\pi_{\mathsf{sk}_s^\Sigma} \leftarrow \mathsf{P}_{\mathsf{NIZK}}\{\mathsf{sk}_s^\Sigma \vert (\mathsf{sk}_s^\Sigma, \mathsf{pk}_s^\Sigma) \in \mathsf{Supp}(\Pi_{\mathsf{AS}}.\mathsf{KGen}(1^\lambda))\}$	$\xrightarrow{\quad R'_s, \pi_{k'_s}, c''_y, Y'', \bar{Y}'', \mathsf{C}''_{\mathsf{amt}}, \tilde{\sigma}'', \pi''_{\mathsf{amt}}, \pi_{\mathsf{sk}_s^\Sigma} \quad}$	Abort if:
14 :			$\mathsf{NIZKVerify}(\pi_{k'_s}, R'_s) \neq 1 \vee$
15 :			$\mathsf{NIZKVerify}(\pi''_{\mathsf{amt}}) \neq 1 \vee$
16 :			$\mathsf{NIZKVerify}(\pi_{\mathsf{sk}_s^\Sigma}) \neq 1 \vee$
17 :			$\mathsf{RCVerify}(\tilde{\sigma}'', \bar{Y}'', R''_1, \mathsf{C}''_{\mathsf{amt}}) \neq 1$
18 :			$y'' := \mathsf{Dec}(\mathsf{sk}_t^\Sigma, c''_y), R'_c \leftarrow R_t^{y''}$
19 :			Abort if: $Y'' \neq g^{y''} \vee \bar{Y}'' \neq \bar{g}^{y''}$
20 :			$\pi'_{y''} \leftarrow \mathsf{P}_{\mathsf{NIZK}}(\{\exists y'' \vert Y = g^{y''} \wedge R'_c = R_t^{y''}\}, y'')$
21 :			$R' \leftarrow (R'_s)^{k'_t \cdot y''}, R' := (r_x, r_y); r' \leftarrow r_x \bmod q$
22 :			$c'_1 \leftarrow \mathsf{Enc}(\mathsf{ek}_s, (k'_t)^{-1} \cdot h')$
23 :			$c'_2 \leftarrow (c_{\mathsf{sk}_s^\Sigma})^{(k'_t)^{-1} \cdot r' \cdot \mathsf{sk}_t^\Sigma}$
24 :			$c' = c'_1 \cdot c'_2$
25 :		$\xleftarrow{\quad (\mathsf{decom}'_t, R'_t, \pi_{k'_t}), c', R'_c, \pi'_{y''} \quad}$	
26 :	Abort if:		
27 :	$\quad \mathsf{Vf}_{\mathsf{com}}(\mathsf{cm}'_t, \mathsf{decom}'_t, (R'_t, \pi_{k'_t})) \neq 1 \vee$		
28 :	$\quad \mathsf{NIZKVerify}(\pi''_y, (R'_c, Y')) \neq 1 \vee$		
29 :	$\quad \mathsf{NIZKVerify}(\pi_{k'_t}, R'_t) \neq 1 \vee$		
30 :	$R' \leftarrow (R'_c)^{k'_s}; R' := (r_x, r_y); r' \leftarrow r_x \bmod q$		
31 :	$s' \leftarrow \mathsf{Dec}(\mathsf{dk}_s, c')$		
32 :	Abort if: $R_t^{s' \bmod q} \neq (\mathsf{pk}_{ts}^\Sigma)^{r'} \cdot g^{h'}$		
33 :	$\widehat{s} \leftarrow s' \cdot (k'_s)^{-1} \bmod q$	$\xrightarrow{\quad \widehat{s} \quad}$	$s \leftarrow \widehat{s} \cdot (y'')^{-1}$
34 :			$g^k \leftarrow (g^{h'} (\mathsf{pk}_{ts}^\Sigma)^{r'})^{\frac{1}{s}}, g^k := (r'', \cdot)$
35 :			If $r'' \neq r'$ then abort
36 :			Else publish signature (r', s)
37 :	$y'' := (s \cdot (\widehat{s})^{-1})^{-1}$		
38 :	$y' := y''\alpha^{-1}$ and send y' to $\mathbb{R}$		
39 :	**return** y'		**return** $\top$

Figure 31. Puzzle Solver Protocol of ECDSA-based Construction

$\mathsf{Open}(\Pi, \alpha')$

Parse Π as $\left(\beta, \mathsf{pk}_{tr}^{\Sigma}, \mathsf{tx}, \widehat{\sigma}\right)$

Set $\alpha \leftarrow \alpha' \cdot \beta^{-1}$

Set $s \leftarrow s' \cdot \alpha^{-1}$

return (r', s)

Figure 32. Open and verify of ECDSA-based Construction

$\mathsf{Verify}(\Pi, \sigma)$

Parse Π as $\left(\beta, \mathsf{pk}_{tr}^{\Sigma}, \mathsf{tx}, \widehat{\sigma}\right)$

return $\mathsf{Vf}_{\mathsf{ECDSA}}(\mathsf{pk}_{tr}^{\Sigma}, \mathsf{tx}, \sigma)$

Figure 33. Open and verify of ECDSA-based Construction