Warning! The Timeout T Cannot Protect You From Losing Coins

PipeSwap: Forcing the Timely Release of a Secret for Atomic Swaps Across All Blockchains

***Abstract*—Atomic cross-chain swap is a critical functionality that enables the distrusting users to atomically exchange coins, thereby facilitating inter-currency exchange and trading. While numerous atomic swaps protocols based on Hash Timelock Contracts have been applied and deployed in practice, they are substantially far from universality due to the inherent dependence of rich scripting language supported by the underlying blockchains. The recently proposed Universal Atomic Swaps protocol [IEEE S&P'22] takes a novel approach to scriptless cross-chain swaps by ingeniously delegating scripting functionality to cryptographic lock mechanisms, particularly the adaptor signatures and timed commitment schemes designed to guarantee *atomicity*. However, in this work, we discover a new form of attack called *double-claiming* attack, such that the honest user would lose coins with overwhelming probability and *atomicity* would be directly broken. Moreover, this attack is easy to carry out and can be naturally generalized to other scriptless cross-chain swaps protocols as well as the payment channel networks, highlighting the general challenges in designing universal atomic swaps protocol.**

We present PipeSwap, a cross-chain swaps protocol that satisfies both security and practical universality. To resist the *double-claiming* attack, specifically to protect the frozen coins from being double-claimed, PipeSwap introduces a novelly designed paradigm of pipelined coins flow leveraging the *two-hop swap* and *two-hop refund* techniques. PipeSwap also achieves *universality* by not relying on any specific script language, aside from the basic ability to verify signatures. In addition to a detailed security analysis in the Universal Composability framework, we develop a proof-of-concept implementation of PipeSwap with Schnorr/ECDSA signatures, and conduct extensive experiments to evaluate the overhead. The experimental results show that PipeSwap can be performed in less than 1.7 seconds and requires less than 7 kb of communication overhead on commodity machines, highlighting its high efficiency.

I. Introduction

With a multitude of diverse blockchain systems coexisting today, it is unrealistic to envision each one evolving in isolation, especially given the explosive development of cryptocurrencies such as Bitcoin [1], Ethereum [2], Ripple [3] and Monero [4]. This highlights an extremely urgent demand of deploying financial operations to securely exchange one currency for another. The atomic cross-chain swaps protocol [5] is introduced as a mechanism to facilitate secure coins exchange between two mutually distrusting users, each holding some coins on distinct blockchains. The fundamental security property of *atomicity* guarantees that the honest user cannot lose coins. In slightly more detail, *atomicity* refers to the entire execution process of swaps protocol being atomic, which means it either succeeds in exchanging coins or fails completely without any intermediate state. Conventionally, the timeout parameter T, which is predefined specifically for each frozen coin, serves as the crux of describing *atomicity* such that each frozen coin is either successfully claimed by the intended receiver before timeout T, or definitely refunded to the original owner after timeout T.

Most classic efforts focus on studying the Hash Timelock Contracts (HTLC)-style protocols [6–10], which rely on the rich scripting languages supported by the underlying blockchains to enforce some specific spending behaviors, namely, when and how the locked coins can be claimed by the intended receiver or refunded back to the original owner. At a high level, the user Alice can use an HTLC script with the hash function H to temporarily lock some coins until timeout T as follows: The HTLC specifies a hash value $h := H(x)$ such that if Bob can present x before timeout T, he can obtain the locked coins; otherwise, these locked coins are definitely refunded to Alice after timeout T.

Unsurprisingly, these HTLC-style proposals are incompatible with a wide range of cryptocurrencies that do not support such scripts or contracts (e.g., Bitcoin [1], Monero [4], Mimblewimble [11], Ripple [3] and Zcash [12]), and far from the universal solutions. Additionally, due to the extensive resources required for constructing, verifying and updating the special scripts and contracts on the underlying blockchains, these protocols result in significantly higher transaction fees for the users swapping their coins. Moreover, they lack on-chain privacy and undermine the *fungibility* property, as the transactions employing special scripts are clearly distinguishable from the general transactions that only require signature verification scripts. Therefore, it is not only of practically relevant but also theoretically interesting to investigate the minimal scripting functionalities necessary to design a secure cross-chain swaps protocol and ultimately, present a scriptless solution of the HTLC-style protocols.

A Desideratum for Achieving Atomic Cross-Chain Swaps in the Absence of Custom Scripts. Universal Atomic Swaps protocol [13] is the closest solution for a universal proposal, which aims to ensure *atomicity* like the HTLC-style protocols even in the presence of malicious users. Noticeably, instead of relying on on-chain scripts to describe the locked coins and their corresponding unlock conditions, Universal Atomic Swaps only require the scripts to verify digital signatures from the underlying blockchains. This is achieved by leveraging the adaptor signature scheme [14] and verifiable

timed discrete logarithm (VTD) scheme [15]. Specifically, the witness extractability of adaptor signature facilitates a successful swap, where once the user holding witness y posts a valid swap transaction, the witness y is subsequently released to the other user to complete his swap operation. Additionally, VTD ensures that in the event of a failed swap, the locked coins are refunded to their original owner after a predefined timeout T. Universal Atomic Swaps take a novel path to the scriptless cross-chain swaps and thus become arguably the best candidate for implementing cryptocurrency exchange.

Is Timeout T Really Secure for Honest User? Nevertheless, the scriptless implementation of timeout T can be potentially used to violate the *atomicity* property. Herein we introduce a new attack termed the *double-claiming* attack. It is noteworthy that in the context of Universal Atomic Swaps [13], the predefined timeout T_1 is intentionally designed for user $\mathcal{P}_1$ to securely refund his frozen coins β (i.e., user $\mathcal{P}_1$ can obtain the full secret key of frozen address after timeout T_1). However, the timeout T_1 cannot deprive the right of user $\mathcal{P}_0$ to claim the frozen coins β. In other words, the user $\mathcal{P}_0$ with witness y still retains the ability to generate a valid swap transaction after timeout T_1. As a result, the timeout T_1 becomes a focal point for security issues. For example, when the honest user $\mathcal{P}_1$ posts a refund transaction after timeout T_1, the malicious user $\mathcal{P}_0$ can still release a valid swap transaction to make the frozen coins β double-claimed. Unfortunately, if the swap transaction of user $\mathcal{P}_0$ is accepted and finally confirmed by the underlying blockchain, the honest user $\mathcal{P}_1$ will neither successfully enter into the Swap Complete Phase nor prevail in the Swap Timeout Phase, ultimately resulting in coins loss. Similarly, the more recent effort Sweep-UC [16] is also susceptible to this *double-claiming* attack, resulting in a similar coins loss as in Universal Atomic Swaps.

Delving into the essence of *double-claiming* attack, we are surprised to observe the fatal imperfection of the timeout T designed for each frozen coin. Specifically, the scriptless implementation of timeout T (e.g., the timed commitment) simply serves as an umbrella for the intended receiver of the frozen coins. In other words, only the intended receiver can claim these frozen coins before timeout T, while this privilege does not be deprived after timeout T. This inherent design flaw directly facilitates the occurrence of the *double-claiming* attack. Even worse, the *double-claiming* attack is general and extremely easy to carry out in all communication network models of the underlying blockchains (cf. Section III for detailed discussions).

The aforementioned issues bring a fundamental challenge in the design of atomic cross-chain swaps: the HTLC-style proposals pose significant obstacles to achieving universality, whereas the existing scriptless solutions are vulnerable to the *double-claiming* attack. This naturally raises a question:

"Can we design a cross-chain swaps protocol that achieves both security and universality?"

A. Our Contributions

In this work, we contribute to the rigorous understanding of atomic cross-chain swaps and answer the aforementioned question affirmatively by introducing a novel protocol named PipeSwap. The specific contributions are outlined as follows:

- Double-claiming attack. We analyze the security of scriptless atomic cross-chain swaps protocols [13, 16], and identify a new form of attack known as the *double-claiming* attack. This attack can directly break *atomicity* with overwhelming probability, specifically enabling a malicious user to simultaneously obtain the counterparty's frozen coins while also refunding his own frozen coins. Moreover, this attack is easy to carry out and can be naturally generalized to other scriptless cross-chain swaps protocols as well as the payment channel networks [17–20].
- Pipelined coins flow: a comprehensive solution to double-claiming attack. To resist the *double-claiming* attack, we introduce a novel paradigm of pipelined coins flow. As depicted in Fig.1, each frozen coin is viewed as a drop of water and flows along the one-way arrows. Informally, each frozen coin can only flow to its intended receiver via the green pipe for the successful swap. Otherwise, it definitely flows to its original owner via the red pipe. In this way, if the frozen coin has been claimed by a valid swap transaction, it will no longer continue to flow forward. After timeout T, however, the frozen coin will never rewind to the intended receiver. Therefore, the concept of pipelined coins flow fundamentally resolves the double-claimed frozen coins issue. As a byproduct of our approach, pipelined coins flow can be leveraged to design the secure scriptless multi-hop swaps (including the multi-hop payments).

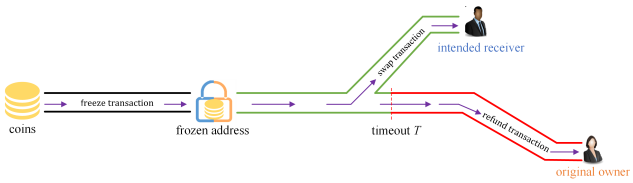


Fig. 1: The pipelined life-cycle of swapped coins

- PipeSwap: the realization of pipelined coins flow. Our key idea centers on the timely release of a swap transaction for the frozen coins in order to successfully complete swaps or, in the event of failure, refund the frozen coins. By introducing the "two-hop completion" method, specifically the *two-hop swap* and *two-hop refund* techniques, we present the atomic cross-chain swaps protocol known as PipeSwap that realizes the pipelined coins flow for frozen coins. Moreover, we refine the definition of *atomicity* by modeling it within the Global Universal Composability framework and provide a formal security proof of PipeSwap. Notably, PipeSwap is universal in that it only relies on the minimal scripting for verifying signatures, and it also preserves *fungibility*, ensuring that an observer cannot distinguish a swap transaction from a standard one. The comparisons with prior approaches are shown in TABLE I.

- Implementation. We develop a proof-of-concept implementation of PipeSwap for Schnorr and ECDSA, and conduct extensive experiments to evaluate the overhead. The results demonstrate the high efficiency and best suitability of our design. Specifically, PipeSwap has the running time of less than 1.7 seconds and communication costs of less than 7 kb. Remarkably, despite providing stronger security protection against *double-claiming* attack, compared to Universal Atomic Swaps [13], PipeSwap only takes a few milliseconds more for completing the second hop of swap/refund operation.

TABLE I: The comparisons among related works*

Protocol	HTLC	UAS	Sweep-UC	Our work
Universality	✗	✓	✓	✓
Fungibility	✗	✓	✓	✓
DoC‡ attack resilience	✓	✗	✗	✓

* HTLC [10], UAS (Universal Atomic Swaps) [13], Sweep-UC [16].
‡ double-claiming.

B. Technique Overview

Recall that, in a general α-to-β swaps protocol [13], user $\mathcal{P}_0$ is given priority to post a swap transaction of frozen coins β before timeout T_1, while simultaneously releasing the witness y w.r.t. hard relation $\mathcal{R}$ to user $\mathcal{P}_1$ to complete his swap operation of frozen coins α. To resist double-claiming of the same frozen coins, we resort to different techniques of forcing the timely release of witness y, i.e., releasing witness y is viewed as a prerequisite for posting a valid swap transaction of frozen coins β. We elaborate on technical contributions as detailed below.

A new freezing structure better prepared for atomicity. To instantiate the pipelined coins flow, our critical step is to correctly foresee the flow direction of each frozen coin before its timeout T. Different from Universal Atomic Swaps [13], we propose a new freezing structure in which the frozen coins β are stored in two distinct frozen addresses, and the smaller part with value $\varepsilon \to 0$ is used to compete for the final flow direction.

Two-hop swap. The new freezing structure inspires us to design a *two-hop swap* method for claiming frozen coins β, while frozen coins α can be directly unlocked with witness y. Notably, a pre-swap transaction is designed for claiming frozen coins ε, and the actual swap transaction of frozen coins β takes the corresponding pre-transaction as one of its inputs. In essence, the puzzle Y $((\mathrm{Y}, y) \in \mathcal{R})$ is inserted in the signature of pre-swap transaction of frozen coins ε instead of the swap transaction of frozen coins β. The *two-hop swap* method of frozen coins β can effectively compel user $\mathcal{P}_0$ to timely post the pre-swap transaction if he wants to generate a valid swap transaction, which further ensures user $\mathcal{P}_1$ to generate his swap transaction with the release of witness y.

Two-hop refund. Obviously, solely relying on the *two-hop swap* design is not sufficient to guarantee the pipelined coins flow, if user $\mathcal{P}_1$ can directly refund the frozen coins β after timeout T_1. We further propose the corresponding *two-hop refund* method for refunding frozen coins β. Specifically, the frozen coins ε are firstly refunded by a pre-refund transaction after time $\overline{T}_1$ ($\overline{T}_1 := T_1 - 2\delta - \varphi$, where parameter φ is the confirmation latency and parameter δ is the upper bound of network delivery delay in the underlying blockchain $\mathbb{B}_1$), and then upon timeout T_1, the remaining frozen coins $\beta - \varepsilon$ are refunded by the final refund transaction. Notice that the *two-hop refund* method of frozen coins β can effectively compel user $\mathcal{P}_0$ to timely post the pre-swap transaction before time $\overline{T}_1$, as any malicious delay would result in the frozen coins ε being claimed by a pre-refund transaction, ultimately leading to user $\mathcal{P}_1$ refunding frozen coins β after timeout T_1.

II. Preliminaries and Background

We first recall the formal definition of Unspent Transaction Output (UTXO) model [14], which is adopted by the majority of current blockchains (e.g., Bitcoin [1], Zcash [21], Monero [22] and Cardanos ADA [23]), and then take a brief overview of Universal Atomic Swaps [13].

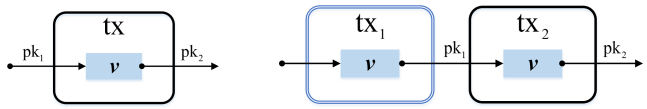
A. The UTXO-based Blockchain

Transactions. In the UTXO model, a transaction is represented as a tuple $(\mathrm{input}, \mathrm{output}, \mathrm{V}, \Omega)$ to transfer coins from $l \geq 1$ inputs $\mathrm{input} := \{\mathrm{in}_1, \ldots, \mathrm{in}_l\}$ to $o \geq 1$ outputs $\mathrm{output} := \{\mathrm{op}_1, \ldots, \mathrm{op}_o\}$. Here, $\mathrm{V} := \{\mathrm{v}_1, \cdots, \mathrm{v}_o\}$ denotes the value of coins transferred to each output and $\Omega := \{\sigma_1, \cdots, \sigma_l\}$ is the witness of spending each input. Usually, we use the public key pk to denote the input/output address, for example, the transaction $\mathrm{tx} := (\mathrm{pk}_1, \mathrm{pk}_2, v, \sigma)$ means transferring coins with value v from address pk_1 to pk_2, with σ serving as the signature of tx that verifies w.r.t. pk_1. The coins in address pk_2 can only be further spent with a valid signature w.r.t. pk_2. Additionally, the conditions of spending coins can be some scripts supported by the underlying blockchains, such as timelock script and HTLC, but in this paper we focus on the scriptless ones.

We use a transaction chart to visualize the coins flow between addresses. As depicted in Fig.2(a), the rounded rectangle represents transaction tx with the incoming arrow as input and the blue box with value v represents the amount of coins, whose spending condition is written above the outgoing arrow.

Blockchain. A blockchain can be used as an append-only bulletin board $\mathcal{T}$ for recording the published transactions, and also be viewed as a trusted ledger $\mathcal{L}$ for storing all the unspent coins associated with each address. Essentially, a blockchain is built and maintained by the parties who compete to be elected as the next leader to propose a candidate block containing a sequence of transactions. It is extremely important to notice that in a secure real-world blockchain execution, the leaders prioritize packaging the earliest received transactions into blocks and, for two conflicting transactions (i.e., claiming the same coins) received simultaneously, they randomly select one of the two transactions as valid. Furthermore, we strictly differentiate the *parties* responsible for securing the underlying blockchains from the *users* participating in the cross-chain

swaps protocol supported by these blockchains. Therefore, the (honest) leaders do not concern themselves with the stories behind each individual transaction, and in their views, all the valid transactions are treated equally. Consequently, it is reasonable that the valid transaction tx' in a pair of conflicting transactions $\{\mathrm{tx}, \mathrm{tx}'\}$ is finally confirmed even if tx' is generated by a malicious user.



(a) Transaction tx signed w.r.t. pk_1 transfers coins with value v from address pk_1 to pk_2, and it can be further spent by a transaction signed w.r.t. pk_2.

(b) Transaction tx_1 has been finally confirmed by the underlying blockchain, and is subsequently spent by a valid transaction tx_2, which has been recorded in $\mathcal{T}$ but has not yet been confirmed.

Fig. 2: The transaction flow

Valid transaction vs Confirmed transaction. The transaction confirmation latency $\varphi > 0$ of underlying blockchain (e.g., it is about one hour in Bitcoin) necessitates a clear distinction between the notions of *valid transaction* and *confirmed transaction*. Traditionally, the bulletin board $\mathcal{T}$ records the set of valid transactions that could be finally confirmed but are not yet (i.e., the input addresses have sufficient balance and are correctly signed), while the ledger $\mathcal{L}$ stores transactions that have been finally confirmed and can be further spent by new transactions. Without loss of generality, we assume that bulletin board $\mathcal{T}$ sequentially records all the valid transactions, especially for the conflicting transactions, it only accepts the one that arrives earlier or randomly selects one in case of simultaneous arrival. Obviously, a valid transaction $\mathrm{tx} \in \mathcal{T}$ can be finally confirmed (i.e., $\mathrm{tx} \in \mathcal{L}$) if it has been in $\mathcal{T}$ for time φ.

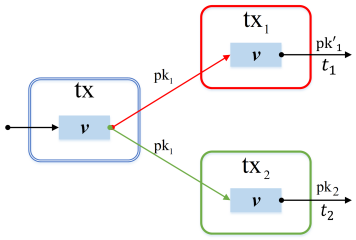


Fig. 3: Transaction tx is double-claimed. The green transaction tx_2 is honestly signed w.r.t. pk_1 and received by $\mathcal{T}$ at time t_2, while the red transaction tx_1 is maliciously signed w.r.t. pk_1 and received by $\mathcal{T}$ at time t_1.

For the clear presentation, we use double-bordered blue rectangles and single-bordered rectangles to represent confirmed transactions and valid transactions, respectively (see Fig.2(b)). Once the transaction processing mechanism of the underlying blockchain is understood, it becomes imperative to focus on a straightforward yet practical scenario illustrated in Fig.3. Since a user holding secret key sk_1 corresponding to public key pk_1 can sign any transactions at his will, there is no way to prevent the malicious payer from generating two valid transactions tx_1 and tx_2. Here, transaction tx_2 would be used to pay the intended receiver, while transaction tx_1 would be used to transfer coins back to the payer. In the fortunate scenario where $t_2 < t_1$, transaction tx_2 defeating tx_1 can be accepted by bulletin board $\mathcal{T}$, ultimately enabling the receiver to obtain the desired coins. However, if $t_1 = t_2$, there is a 50% probability the receiver will lose coins as tx_1 being accepted by bulletin board $\mathcal{T}$. Even worse, when the network delivery delay is under adversarial control (e.g., the δ-synchronous network [24]), the malicious transaction tx_1 would compete against tx_2 with a landslide.

By leveraging the above observation, we need to be cautious about certain time points that can result in the same coins being doubly claimed by different transactions. This is especially important when it comes to realizing atomic cross-chain swaps in a decentralized manner (see Section III).

B. Atomic Cross-Chain Swaps

Generally, a cross-chain swaps protocol facilitates two mutually distrusting users $\mathcal{P}_0$ and $\mathcal{P}_1$, who respectively own coins α and β in two distinct blockchains $\mathbb{B}_0$ and $\mathbb{B}_1$, to securely exchange coins. The fundamental security property *atomicity* states that the honest users cannot lose coins. Specifically, an honest user will successfully receive the counterparty's coins if the swap is successful. Otherwise, after timeout T, the honest user will definitely refund his frozen coins.

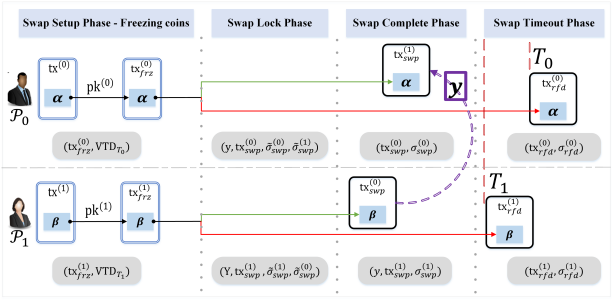


Fig. 4: Universal Atomic Swaps: the execution flow of users $\mathcal{P}_0$ and $\mathcal{P}_1$ in an α-to-β swap. The gray rectangle stores the messages received by each user in the corresponding phase.

Now we review the design of Universal Atomic Swaps [13] (see Fig.4). Similar to other works in this area [14, 16, 25, 26], in Universal Atomic Swaps protocol, the users are connected with authenticated communication channels with guaranteed delivery of exactly one round and the malicious user (e.g., user $\mathcal{P}_0$ or $\mathcal{P}_1$) can deviate arbitrarily from the protocol. It takes the underlying blockchain as a global ledger functionality, where the communication network is under adversarial control to delay or reorder messages within δ rounds, and prioritizes valid transactions based on their arrival time (cf. Fig.7 for a formal definition).

We use hereunder notations: (1) an item with superscript $\{(i\ 1-i)|i \in \{0,1\}\}$ is involved in the payment from user

$\mathcal{P}_i$ to $\mathcal{P}_{1-i}$; (2) an item with subscript $\in \{frz, swp, rfd\}$ respectively refers to the freeze, swap and refund operations. Informally, it consists of four phases described as follows:

Swap Setup Phase-Freezing coins: Users $\mathcal{P}_0$ and $\mathcal{P}_1$ jointly generate the frozen addresses $\mathrm{pk}^{(01)}$ and $\mathrm{pk}^{(10)}$, where the corresponding secret keys $\mathrm{sk}^{(01)} := \mathrm{sk}_0^{(01)} \oplus \mathrm{sk}_1^{(01)}$ and $\mathrm{sk}^{(10)} := \mathrm{sk}_0^{(10)} \oplus \mathrm{sk}_1^{(10)}$ are shared between them. They also compute the timed commitments (Def.4) $\mathrm{VTD}_{T_1} := (\mathrm{C}^{(1)}, \pi^{(1)})$ and $\mathrm{VTD}_{T_0} := (\mathrm{C}^{(0)}, \pi^{(0)})$ of shares $\mathrm{sk}_0^{(10)}$ and $\mathrm{sk}_1^{(01)}$ (Note that, after timeout T_i, user $\mathcal{P}_i$ can get secret key $\mathrm{sk}^{(i1-i)}$). After the both VTD are verified, user $\mathcal{P}_i$ transfers coins from address $\mathrm{pk}^{(i)}$ to the frozen address $\mathrm{pk}^{(i1-i)}$ via a freeze transaction $\mathrm{tx}_{frz}^{(i)}$.

Swap Lock Phase: Using adaptor signature w.r.t. hard relation $(\mathrm{Y}, y) \in \mathcal{R}$ (Def.1) selected by user $\mathcal{P}_0$, users $\mathcal{P}_0$ and $\mathcal{P}_1$ jointly generate the pre-signatures $\tilde{\sigma}_{swp}^{(1)}$ of swap transaction $\mathrm{tx}_{swp}^{(1)}$ and $\tilde{\sigma}_{swp}^{(0)}$ of swap transaction $\mathrm{tx}_{swp}^{(0)}$ in sequence. Notice that, from now on, user $\mathcal{P}_0$ with witness y can generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ at any time.

Swap Complete Phase: If user $\mathcal{P}_0$ actively posts the swap transaction $\mathrm{tx}_{swp}^{(0)}$ before timeout T_1, user $\mathcal{P}_1$ can extract witness y (as indicated by the purple dotted arrow) to successfully complete the Swap Complete Phase via positing a valid swap transaction $\mathrm{tx}_{swp}^{(1)}$. As a result, users $\mathcal{P}_0$ and $\mathcal{P}_1$ successfully exchange their coins (as indicated by the green arrows).

Swap Timeout Phase: After timeout T_1, if coins β are still locked in the frozen address $\mathrm{pk}^{(10)}$, user $\mathcal{P}_1$ will enter into the Swap Timeout Phase to abort the swap protocol by posting a refund transaction $\mathrm{tx}_{rfd}^{(1)}$ with secret key $\mathrm{sk}^{(10)}$. Similarly, after timeout T_0, user $\mathcal{P}_0$ can abort the swap protocol by posting a refund transaction $\mathrm{tx}_{rfd}^{(0)}$ to unlock coins α. Therefore, the frozen coins are respectively refunded to their original owners (as indicated by the red arrows).

Security analysis. We summarize security analysis of Universal Atomic Swaps and defer detailed proofs to [13]:

Successful Swap: If user $\mathcal{P}_0$ honestly posts the swap transaction $\mathrm{tx}_{swp}^{(0)}$ before timeout T_1, user $\mathcal{P}_1$ can extract witness y to generate a swap transaction $\mathrm{tx}_{swp}^{(1)}$ successfully.

Failed Swap: If user $\mathcal{P}_0$ fails to post the swap transaction $\mathrm{tx}_{swp}^{(0)}$ before timeout T_1, user $\mathcal{P}_1$ will enter into the Swap Timeout Phase to refund the frozen coins β via posting a refund transaction $\mathrm{tx}_{rfd}^{(1)}$ and abort the swap protocol. Similarly, user $\mathcal{P}_0$ can refund the frozen coins α after timeout T_0.

III. The Double-Claiming Attack

What *Double-Claiming* Attack Is. In a cross-chain swaps protocol, the *double-claiming* attack refers to a situation where a malicious user utilizes timeout T to create a double-claimed state for the counterparty's frozen coins (i.e., after timeout T, the frozen coins are claimed simultaneously by the refund transaction and swap transaction), thereby obtaining the offered coins from the honest user while refusing to transfer his own coins. This attack directly violates the *atomicity* property.

Note that the *double-claiming* attack is fundamentally different from the *double-spending* attack [27]. The former attack enables a malicious user to prevent an honest transaction from being confirmed, whereas the latter attack enables a malicious miner who is involved in the maintenance of the underlying blockchain to confirm both spending transactions of the same coins. Additionally, the *double-spending* attack requires the attacker (i.e., the malicious miner) to posses and expend enough resources (e.g., computational power [1] or stakes [28]). Conversely, the *double-claiming* attack is crazy-cheap and only requires the attacker (i.e., the malicious user) to have the ability of signing transactions. Notably, the *double-claiming* attack can work in all communication network models of the underlying blockchains (e.g., the δ-synchrony [29], partial synchrony [30] and asynchrony [31]). Moreover, as the synchronicity level of the underlying blockchain weakens (i.e., transitioning from synchrony to asynchrony), the *double-claiming* attack has a higher probability of success, which can be detailed as follows.

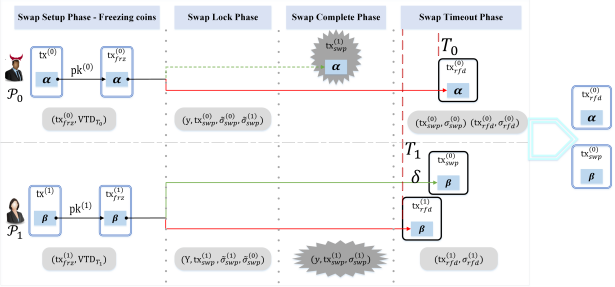
Fig.5 illustrates two manners of *double-claiming* attacks on Universal Atomic Swaps [13], where the underlying blockchains take at most δ rounds to publish a transaction.

User $\mathcal{P}_0$ is malicious (Fig.5(a)). Initially, users $\mathcal{P}_0$ and $\mathcal{P}_1$ launch an α-to-β swap by successfully freezing their respective coins α and β, and then completing the Swap Lock Phase. Subsequently, the malicious user $\mathcal{P}_0$ goes offline. After timeout T_1, user $\mathcal{P}_1$ aborts the swap protocol and enters into the Swap Timeout Phase. During this phase, user $\mathcal{P}_1$ posts a refund transaction $\mathrm{tx}_{rfd}^{(1)}$ to unfreeze the frozen coins β. Concurrently, the malicious user $\mathcal{P}_0$ comes back online and posts a swap transaction $\mathrm{tx}_{swp}^{(0)}$. After timeout T_0, the malicious user $\mathcal{P}_0$ can also enter into his Swap Timeout Phase and post a refund transaction $\mathrm{tx}_{rfd}^{(0)}$ to refund the frozen coins α. Notice that, after timeout T_1, even if the malicious user $\mathcal{P}_0$ posts a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$, the Universal Atomic Swaps protocol is designed in such a way that user $\mathcal{P}_1$ is not permitted to revert back to the previous Swap Complete Phase[1]. Additionally, only one of transactions, $\mathrm{tx}_{swp}^{(0)}$ and $\mathrm{tx}_{rfd}^{(1)}$, can be finally confirmed by the underlying blockchain $\mathbb{B}_1$. Accordingly, if there is a delay in publishing the refund transaction $\mathrm{tx}_{rfd}^{(1)}$ while the malicious swap transaction $\mathrm{tx}_{swp}^{(0)}$ arrives earlier or even worse, if the malicious user $\mathcal{P}_0$ colludes with the adversary $\mathcal{A}_{\mathbb{B}_1}$ of underlying blockchain $\mathbb{B}_1$ to delay publishing the refund transaction $\mathrm{tx}_{rfd}^{(1)}$ up to δ rounds, the malicious swap transaction $\mathrm{tx}_{swp}^{(0)}$ will be finally confirmed with an absolute advantage. This would result in the malicious user $\mathcal{P}_0$ illicitly acquiring double assets, thereby violating the *atomicity* property (i.e., the honest user $\mathcal{P}_1$ loses coins β).

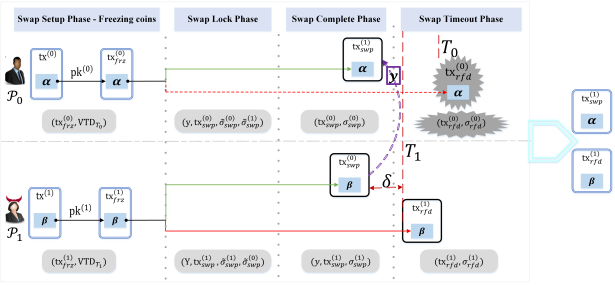
User $\mathcal{P}_1$ is malicious (Fig.5(b)). If honest user $\mathcal{P}_0$ promptly posts a swap transaction $\mathrm{tx}_{swp}^{(0)}$ before timeout T_1, this swap protocol should be successful such that both swap transactions, $\mathrm{tx}_{swp}^{(0)}$ and $\mathrm{tx}_{swp}^{(1)}$, are finally confirmed by their respective

[1]If the user $\mathcal{P}_1$, who has initiated the abort operation, is allowed to perform the aforementioned actions (e.g., the swap operation), the final result of Universal Atomic Swaps protocol cannot be guaranteed.

blockchains. However, once the honest user $\mathcal{P}_0$ posts a swap transaction $\mathrm{tx}_{swp}^{(0)}$ close to the timeout T_1 (e.g., within δ rounds), the malicious user $\mathcal{P}_1$ can immediately post his swap transaction $\mathrm{tx}_{swp}^{(1)}$ and then enter into his Swap Timeout Phase by posting a valid refund transaction $\mathrm{tx}_{rfd}^{(1)}$ after timeout T_1[2]. At this time, a race condition occurs between transactions $\mathrm{tx}_{swp}^{(0)}$ and $\mathrm{tx}_{rfd}^{(1)}$. According to the same attack strategies as above, the malicious refund transaction $\mathrm{tx}_{rfd}^{(1)}$ can be finally confirmed by the underlying blockchain $\mathbb{B}_1$. As a result, the malicious user $\mathcal{P}_1$ harvests double assets, thereby violating the *atomicity* property (i.e., the honest user $\mathcal{P}_0$ loses coins α).[3]



(a) When the malicious user $\mathcal{P}_0$ successfully executes a *double-claiming* attack, the honest user $\mathcal{P}_1$ fails in the Swap Complete Phase and Swap Timeout Phase. The finally confirmed transactions are $\mathrm{tx}_{rfd}^{(0)}$ and $\mathrm{tx}_{swp}^{(0)}$.



(b) When the malicious user $\mathcal{P}_1$ successfully executes a *double-claiming* attack, the honest user $\mathcal{P}_0$ fails in the Swap Complete Phase and cannot enter into the Swap Timeout Phase. The finally confirmed transactions are $\mathrm{tx}_{rfd}^{(1)}$ and $\mathrm{tx}_{swp}^{(1)}$.

Fig. 5: The *double-claiming* attack works in Universal Atomic Swaps [13]

Why There Exists This Attack. Essentially, the core reasons that lead to the *double-claiming* attack are:

Reason 1: The transaction balance security. Independent of the inner workings in cross-chain swaps protocols, the total balances of all addresses in the underlying blockchain remain unchanged. This means that no new coins are generated causelessly, and the coins can only be equivalently transferred from some addresses to new addresses. Furthermore, the scriptless nature of cross-chain swaps determines that the transaction verification relies solely on the balance and signature. Thus, in the view of the underlying blockchain, the conflicting transactions (e.g., the swap and refund transactions of the same frozen coins) are separately valid and the balance security determines that only one of these two transactions can be finally confirmed.

Reason 2: After timeout T, the frozen coins can be claimed by both the original owner and intended receiver. It should be noted that the frozen coins can only be refunded after the predefined timeout T, while the intended receiver is able to claim these coins both before and after timeout T (i.e., by generating a valid swap transaction). This may initially seem reasonable as the timeout T provides sufficient time for the intended receiver to complete the swaps and guarantees that the frozen coins can be refunded in case of failure. However, there is no mechanism in place to revoke the intended receiver's ability to claim these frozen coins after timeout T, which is the source of *double-claiming* attack.

Where the Timeouts T_0 and T_1 Fall Short. we start by offering a detailed discussions of the timeouts T_0 and T_1 adopted in Universal Atomic Swaps to show their insufficiency in protecting the honest user from losing coins in the double claiming scenario.

For the timing hardness parameters T_0 and T_1, Universal Atomic Swaps suggest that the parameter Δ (such that $T_0 = T_1 + \Delta$) is large enough to tolerate the time differences in opening the VTD commitments. In essence, parameter Δ is designed to prevent the malicious user $\mathcal{P}_0$ from stealing coins if he can open his VTD commitment much earlier than expected. For example, if malicious user $\mathcal{P}_0$ opens the VTD commitment much earlier than timeout T_1, he can first post a refund transaction $\mathrm{tx}_{rfd}^{(0)}$ to refund frozen coins α, followed by posting a swap transaction $\mathrm{tx}_{swp}^{(0)}$ to steal the frozen coins β of user $\mathcal{P}_1$. Similarly, if malicious user $\mathcal{P}_0$ opens the VTD commitment after timeout T_1 but much earlier than timeout T_0, he can first post a swap transaction $\mathrm{tx}_{swp}^{(0)}$ to obtain the frozen coins β of user $\mathcal{P}_1$, followed by posting a refund transaction $\mathrm{tx}_{rfd}^{(0)}$ to refund frozen coins α [4]. Therefore, the parameter Δ is set sufficiently large such that provides enough time to finally confirm the swap transaction $\mathrm{tx}_{swp}^{(1)}$ of user $\mathcal{P}_1$ before user $\mathcal{P}_0$ can generate a valid refund transaction $\mathrm{tx}_{rfd}^{(0)}$.

However, a sufficiently large parameter Δ still cannot protect honest user $\mathcal{P}_1$ from losing coins in the following malicious scenario (as illustrated in Fig.5(a)). Informally, the malicious user $\mathcal{P}_0$ deliberately delays posting a swap transaction $\mathrm{tx}_{swp}^{(0)}$ until timeout T_1, which is exactly the time that user $\mathcal{P}_1$ enters into his Swap Timeout Phase to abort the swap protocol by posting a refund transaction $\mathrm{tx}_{rfd}^{(1)}$. Consequently, the frozen coins β of honest user $\mathcal{P}_1$ are subjected to double-claim. Since the user $\mathcal{P}_1$ is no longer permitted to submit a swap transaction $\mathrm{tx}_{swp}^{(1)}$ after timeout T_1, there is no measure in place to prevent malicious user $\mathcal{P}_0$ from entering into the

[2]As long as the swap transaction $\mathrm{tx}_{swp}^{(0)}$ of user $\mathcal{P}_0$ is not published by the underlying blockchain $\mathbb{B}_1$, the malicious user $\mathcal{P}_1$ still has the chance to publish his refund transaction $\mathrm{tx}_{rfd}^{(1)}$ (cf. Fig.7).

[3]Even if the above attacks are failed, the malicious user will never lose coins.

[4]As long as the swap transaction $\mathrm{tx}_{swp}^{(1)}$ of user $\mathcal{P}_1$ is not published by the underlying blockchain $\mathbb{B}_0$, the malicious user $\mathcal{P}_0$ still has the chance to refund his frozen coins α by doubly claiming them (cf. Fig.7).

Swap Timeout Phase to refund frozen coins α after timeout T_0, even if Δ is large enough. For the conflicting transactions $\mathrm{tx}_{swp}^{(0)}$ and $\mathrm{tx}_{rfd}^{(1)}$, once the swap transaction $\mathrm{tx}_{swp}^{(0)}$ is finally confirmed by the underlying blockchain $\mathbb{B}_1$, the malicious user $\mathcal{P}_0$ will harvest double assets.

The Generality of This Attack. It should be emphasized that the *double-claiming* attack is not exclusive to Universal Atomic Swaps, but can also be generally applied to other scriptless cross-chain swaps protocols (e.g., Sweep-UC [16]) and multi-hop payments [17–20], all of which involve a predefined timeout T to realize the payment expire. In these protocols, the primary vulnerability leading to the *double-claiming* attack is that the two competing users simultaneously hold the ability to claim the same frozen coins immediately after the timeout T.

IV. Our Solution in a Nutshell

As previously elaborated, the core challenge posed by the *double-claiming* attack lies in the fact that the frozen coins β can be claimed by both users $\mathcal{P}_0$ and $\mathcal{P}_1$ after timeout T_1. Therefore, our goal is to protect the frozen coins β from being double-claimed after timeout T and ensure their refund to the original owner $\mathcal{P}_1$ with certainty as the HTLC dose.

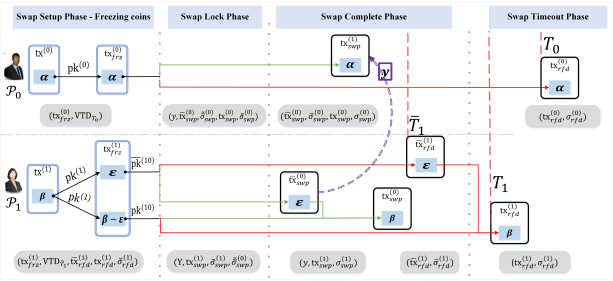


Fig. 6: PipeSwap: the execution flow of users $\mathcal{P}_0$ and $\mathcal{P}_1$ in an α-to-β swap. The gray rectangle stores messages received by each user in this phase and the pre-transaction items are denoted with overline. Time parameters are set such that: $T_1 := \overline{T}_1 + 2\delta + \varphi, T_0 = T_1 + \Delta$ (where φ is confirmation latency and δ is the upper bound of network delivery delay in the underlying blockchain $\mathbb{B}_1$). The gap $2\delta+\varphi$ between $\overline{T}_1$ and T_1 ensures that, even under the maximum adversarial delay δ, the frozen coins β can only be swaped before timeout T_1; otherwise, they are definitely refunded to their original owner $\mathcal{P}_1$ after timeout T_1.

Our straightforward solution is to realize the pipelined coins flow (Fig.1) of frozen coins β, that is, the corresponding swap transaction $\mathrm{tx}_{swp}^{(0)}$ and refund transaction $\mathrm{tx}_{rfd}^{(1)}$ cannot be both valid. As depicted in Fig.6, the key idea is how to force the timely release of witness y (as indicated by the purple dotted arrow), such that a valid transaction of frozen coins β (i.e., either the swap transaction $\mathrm{tx}_{swp}^{(0)}$ or refund $\mathrm{tx}_{rfd}^{(1)}$) can be pre-determined before timeout T_1. For this purpose, we introduce a "two-hop completion" method, known as *two-hop swap* and *two-hop refund*. The *two-hop swap* compels user $\mathcal{P}_0$ to post the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ at least φ time before posting a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$. Similarly, the corresponding *two-hop refund* requires user $\mathcal{P}_1$ to post a pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ (it is locked until time $\overline{T}_1 := T_1 - 2\delta - \varphi$) before refunding his frozen coins β by the refund transaction $\mathrm{tx}_{rfd}^{(1)}$. Thus, the "two-hop completion" method forces user $\mathcal{P}_0$ to actively post a pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ before time $\overline{T}_1$, as otherwise the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ could be confirmed and he cannot generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ at any time. As a result, before timeout T_1, the final flow direction of frozen coins β can be determined in that if the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is finally confirmed, the frozen coins β can only be swapped by user $\mathcal{P}_0$; otherwise, the frozen coins β can only be refunded by user $\mathcal{P}_1$. Now we walk through how to realize the "two-hop completion".

First ingredient: splitting frozen coins β into two parts. To ensure the flow direction of frozen coins β, the coins β are frozen in two distinct addresses with values ε (where $\varepsilon > 0$ is arbitrarily small, i.e., $\varepsilon \to 0$) and $\beta - \varepsilon$ respectively. Specifically, coins ε and $\beta - \varepsilon$ can only be further spent with the respective secret keys $\overline{\mathrm{sk}}^{(10)}$ and $\mathrm{sk}^{(10)}$, which are shared between users $\mathcal{P}_0$ and $\mathcal{P}_1$.

Second ingredient: two-hop swap. To prevent the malicious user $\mathcal{P}_0$ from suddenly releasing the swap transaction $\mathrm{tx}_{swp}^{(0)}$, a new swap method called *two-hop swap* is proposed. Specifically, user $\mathcal{P}_0$ can only generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ when the corresponding pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ has been confirmed by the underlying blockchain $\mathbb{B}_1$. In particular, the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is jointly pre-signed by both users with the respective key shares (i.e., $\overline{\mathrm{sk}}_0^{(10)}$, $\overline{\mathrm{sk}}_1^{(10)}$) and puzzle Y, where the statement-witness pair $(\mathrm{Y}, y) \in \mathcal{R}$ is selected by user $\mathcal{P}_0$. While the swap transaction $\mathrm{tx}_{swp}^{(0)}$ takes $\overline{\mathrm{tx}}_{swp}^{(0)}$ as one of its inputs, and is jointly signed by both users with the respective key shares $\mathrm{sk}_0^{(10)}$ and $\mathrm{sk}_1^{(10)}$. Essentially, the validity of swap transaction $\mathrm{tx}_{swp}^{(0)}$ implies that user $\mathcal{P}_0$ has posted the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ at least φ time ago, where φ is the confirmation latency of underlying blockchain $\mathbb{B}_1$. Furthermore, since the witness y has been released by the posted pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$, user $\mathcal{P}_1$ can generate a valid swap transaction $\mathrm{tx}_{swp}^{(1)}$ at least φ time earlier than user $\mathcal{P}_0$ does.

Third ingredient: two-hop refund. To pre-determine a valid transaction between the swap transaction $\mathrm{tx}_{swp}^{(0)}$ and refund transaction $\mathrm{tx}_{rfd}^{(1)}$ before timeout T_1, we present the corresponding *two-hop refund* method. In this way, user $\mathcal{P}_1$ can only refund the frozen coins β with a refund transaction $\mathrm{tx}_{rfd}^{(1)}$ after timeout T_1 when the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ has been finally confirmed by the underlying blockchain $\mathbb{B}_1$, and the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ is locked until time $\overline{T}_1$. This implies that once the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is finally confirmed, it would be impossible to generate a valid refund transaction $\mathrm{tx}_{rfd}^{(1)}$ even after timeout T_1. Conversely, if the pre-

refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ is finally confirmed, there will never exist a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$.

So far, the final flow direction of frozen coins β can be determined before timeout T_1, leaving only a subtle issue to be addressed. In particular, if user $\mathcal{P}_0$ posts the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ almost near time $\overline{T}_1$, the malicious user $\mathcal{P}_1$ can also initiate the *double-claiming* attack by posting the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ immediately after time $\overline{T}_1$ and making both transactions $\overline{\mathrm{tx}}_{swp}^{(0)}$ and $\overline{\mathrm{tx}}_{rfd}^{(1)}$ valid. To address this issue, we simply let the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ be posted at least δ time units before time $\overline{T}_1$.

We now give an intuition that PipeSwap satisfies *atomicity*:
Successful Swap: If user $\mathcal{P}_0$ is honest, the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ posted before time $\overline{T}_1 - \delta$ will be finally confirmed before timeout T_1, and then both swap transactions $\mathrm{tx}_{swp}^{(0)}$ and $\mathrm{tx}_{swp}^{(1)}$ could be valid before timeout T_1. Additionally, no valid refund transaction $\mathrm{tx}_{rfd}^{(1)}$ exists even after timeout T_1. As a result, both users $\mathcal{P}_0$ and $\mathcal{P}_1$ obtain their desired coins.
Failed Swap: If user $\mathcal{P}_0$ is malicious and user $\mathcal{P}_1$ has entered into his Swap Timeout Phase after timeout T_1, i.e., the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ has been finally confirmed by the underlying blockchain $\mathbb{B}_1$, user $\mathcal{P}_1$ can generate a valid refund transaction $\mathrm{tx}_{rfd}^{(1)}$ after timeout T_1 and there will never be a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ at any time. As a result, user $\mathcal{P}_1$ can definitely refund the frozen coins β after timeout T_1.

V. Formal Definition of PipeSwap

Notations. We denote by λ the security parameter and by $A(x;r) \rightarrow z$ or $z \leftarrow A(x;r)$ the output z of algorithm A with inputs x and randomness $r \in_{\$} \{0,1\}^{\lambda}$ (it is only mentioned explicitly when required). We write the events that "send message m to P at time t" as "$m \overset{t}{\hookrightarrow} P$" and "receive message m from P at time t" as "$m \overset{t}{\hookleftarrow} P$", where P could be a user or an ideal functionality.

A. Modeling the System and Threats

We now discuss the security model of generalized cross-chain swaps, which exactly follows the previous works in this area [13, 16, 18, 19, 25, 32]. In order to formally model the security of cross-chain swaps protocol, we adopt the Universal Composability (UC) model [33] framework and deploy the version with a global setup (GUC) [34]. We define the cross-chain swaps model over two users $\{\mathcal{P}_0, \mathcal{P}_1\}$ and take the underlying blockchains $\mathbb{B} =: \{\mathbb{B}_0, \mathbb{B}_1\}$ as the global blockchain (ledger) functionalities $\mathcal{F}_{\mathbb{B}} := \{\mathcal{F}_{\mathbb{B}_0}, \mathcal{F}_{\mathbb{B}_1}\}$ with a maximum confirmation delay of φ (see Fig.7). The GUC model specifies two worlds, a protocol Π is executed in the real world by the users, interacting with the adversary $\mathcal{A}$ and environment $\mathcal{Z}$. While in the ideal world, an ideal functionality $\mathcal{F}$ is executed by the users, interacting with the simulator $\mathcal{S}$ and environment $\mathcal{Z}$. We denote the ensemble corresponding to real world protocol execution as $\mathrm{EXEC}(\lambda)^{\mathcal{F}_{\mathbb{B}}}_{\Pi,\mathcal{Z},\mathcal{A}}$ and the ensemble corresponding to the ideal world execution as $\mathrm{EXEC}(\lambda)^{\mathcal{F}_{\mathbb{B}}}_{\mathcal{F},\mathcal{Z},\mathcal{S}}$.
UC security. Protocol Π UC-realizes the ideal functionality $\mathcal{F}$ w.r.t. a global blockchain $\mathcal{F}_{\mathbb{B}}$ if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that $\mathrm{EXEC}^{\mathcal{F}_{\mathbb{B}}}_{\Pi,\mathcal{Z},\mathcal{A}} \approx \mathrm{EXEC}^{\mathcal{F}_{\mathbb{B}}}_{\mathcal{F},\mathcal{Z},\mathcal{S}}$, where "$\approx$" denotes the computational indistinguishability.
The adversary. In the cross-chain swaps protocol, two designated users $\mathcal{P}_0$ and $\mathcal{P}_1$ are involved. The static adversary $\mathcal{A}$ selects and fully controls one user, either user $\mathcal{P}_0$ or $\mathcal{P}_1$, at the beginning of the process. Moreover, the adversary $\mathcal{A}$ is fully malicious and can deviate arbitrarily from the protocol specification (e.g., by carrying out the *double-claiming* attack).
The communication network. We consider the synchronous communication between the participating users $\mathcal{P}_0$ and $\mathcal{P}_1$, implying that we assume a global clock functionality $\mathcal{F}_{clock}$ [24]. The cross-chain swaps protocol executes in rounds, ensuring that each user is aware of the current round and can expect messages to be received at a certain time. There is also a secure message transmission channel between users $\mathcal{P}_0$ and $\mathcal{P}_1$ modeled by the ideal functionality $\mathcal{F}_{smt}$ [13, 33].
The atomicity. At the end of a cross-chain swaps protocol, both users will either successfully exchange their coins or, in the event of a failed swap, the frozen coins will be refunded to their original owners. In accordance with the HTLC, we specifically aim to achieve that: each frozen coin should only be claimed by the intended receiver before its timeout T; if not claimed by then, it will be definitely refunded to its original owner after its timeout T.
The blockchain functionality. Similarly, we also assume the δ-synchronous communication network of the underlying blockchains, i.e., the network delivery delay is under adversarial control up to a known upper bound δ. Furthermore, the adversary $\mathcal{A}_{\mathbb{B}}$ of the underlying blockchains can delay or reorder the delivery of messages (transactions) within δ rounds, but cannot modify or drop them. We take the underlying blockchains $\mathbb{B}$ (i.e., blockchains $\mathbb{B}_0$ and $\mathbb{B}_1$) involved in the cross-chain swaps as a global ledger functionality $\mathcal{F}_{\mathbb{B}}$ with maximum confirmation delay time φ, which records the balance pk.bal of each address pk (i.e., the ledger $\mathcal{L}$) and maintains a trusted append-only bulletin board $\mathcal{T}$. The functionality $\mathcal{F}_{\mathbb{B}}$ offers interface $\mathrm{Valid}(\mathrm{tx})$ to determine the validity of a transaction (e.g., checking whether $\mathrm{inputs} \in \mathcal{L}$ have sufficient balance and are signed correctly), uses interface $\mathrm{Publish}(\mathrm{tx}, t)$ to add a valid transaction tx to bulletin board $\mathcal{T}$ at time $t' \leq t + \delta$, where time t' is determined by the ideal adversary $\mathcal{S}_{\mathbb{B}}$. We emphasize that the bulletin board $\mathcal{T}$ always gives priority to accepting the transaction that arrives first. For example, when receiving messages $\mathrm{Publish}(\mathrm{tx}_0, t_0)$ and $\mathrm{Publish}(\mathrm{tx}_1, t_1)$ for respectively publishing the conflicting transactions tx_0 and tx_1, the bulletin board $\mathcal{T}$ will accept tx_0 if $t'_0 < t'_1$, or randomly select one from $\{\mathrm{tx}_0, \mathrm{tx}_1\}$ if $t'_0 = t'_1$. The functionality $\mathcal{F}_{\mathbb{B}}$ confirms a transaction tx via interface $\mathrm{Confirm}(\mathrm{tx})$ (e.g., if the transaction tx has been recorded in bulletin board $\mathcal{T}$ for time φ, the functionality $\mathcal{F}_{\mathbb{B}}$ updates ledger $\mathcal{L}$ via transferring coins from input address pk_{in} to output address pk_{op}). Please refer to Fig.7 for more details.

The blockchain functionality $\mathcal{F}_{\mathbb{B}}$ interacts with users $\mathcal{P}_0$ and $\mathcal{P}_1$, ideal adversary $\mathcal{S}_{\mathbb{B}}$ and environment $\mathcal{Z}$. It is parameterized by a digital signature scheme $\mathrm{SIG} = (\mathrm{KGen}, \mathrm{Sig}, \mathrm{Vf})$.

Interface $\mathrm{Initiate}(\mathrm{pk}, v)$, called by $\mathcal{Z}$:

01 Set ledger $\mathcal{L} := \{(\mathrm{pk}_1, v_1), \cdots, (\mathrm{pk}_\ell, v_\ell)\} \in \mathbb{R}^{2\ell}_{\geq 0}$.
02 Store and send $\mathcal{L}$ to every entity.

Interface $\mathrm{Publish}(\mathrm{tx}, t)$, called by $\mathcal{P}_i$:

01 If $\mathrm{Valid}(\mathrm{tx}) = 1$, send $(\mathrm{publish}, \mathrm{tx}, t) \hookrightarrow \mathcal{S}_{\mathbb{B}}$.
02 Upon receiving $(\mathrm{publish}, \mathrm{tx}, t') \hookleftarrow \mathcal{S}_{\mathbb{B}}$, if $t' - t \leq \delta$, set $t := t'$; otherwise, set $t := t + \delta$. Update list $list := list \cup (\mathrm{tx}, t)$.
03 For the conflicting transactions (tx_0, t_0), $(\mathrm{tx}_1, t_1) \in list$, if $t_0 < t_1$, remove (tx_1, t_1) from $list$; else, if $t_0 = t_1$, randomly select $b \in \{0, 1\}$ and remove (tx_b, t_b) from $list$; otherwise, remove (tx_0, t_0) from $list$.
04 For $(\mathrm{tx}, t) \in list$, update $\mathcal{T} := \mathcal{T} \cup (\mathrm{tx}, t)$ at time t.

Interface $\mathrm{Confirm}(\mathrm{tx})$, called by $\mathcal{P}_i$ at time t'':

01 If $\exists (\mathrm{tx}, t) \in \mathcal{T}$ and $t'' - t \geq \varphi$, update $\mathcal{L}$ as $\mathrm{tx.pk}_{in}.\mathrm{bal} := \mathrm{tx.pk}_{in}.\mathrm{bal} - v$ and $\mathrm{tx.pk}_{op}.\mathrm{bal} := \mathrm{tx.pk}_{op}.\mathrm{bal} + v$.
02 Otherwise, abort.

Fig. 7: Global ideal functionality $\mathcal{F}_{\mathbb{B}}$ that models the blockchain

B. Ideal Functionality of Cross-Chain Swaps

We formalize the security properties of a cross-chain swaps protocol as an ideal functionality $\mathcal{F}$. Informally, the functionality $\mathcal{F}$ (Fig.8 and Fig.9) interacts with users $\mathcal{P}_0$ and $\mathcal{P}_1$, environment $\mathcal{Z}$, ideal adversary $\mathcal{S}$, and the underlying blockchain functionalities $\mathcal{F}_{\mathbb{B}_0}$ and $\mathcal{F}_{\mathbb{B}_1}$. It consists of three procedures, each triggered by a message sent by user $\mathcal{P}_i$ $(i \in \{0, 1\})$ including the respective request and session identifier id.

(A) Swap Setup Phase-Freezing Coins: Users $\mathcal{P}_0$ and $\mathcal{P}_1$ initiate an α-to-β swap with their respective freeze messages, denoted as $(\mathrm{frz}, \mathrm{id}, \mathrm{pk}^{(0)}, \mathrm{sk}^{(0)}, \alpha)$ and $(\mathrm{frz}, \mathrm{id}, \mathrm{pk}^{(1)}, \mathrm{sk}^{(1)}, \beta)$. These messages specify that the coins α in addresses $\mathrm{pk}^{(0)}$ (owned by user $\mathcal{P}_0$ and can be spent with secret key $\mathrm{sk}^{(0)}$) and the coins β in address $\mathrm{pk}^{(1)}$ (owned by user $\mathcal{P}_1$ and can be spent with secret key $\mathrm{sk}^{(1)}$) are to be exchanged. The functionality $\mathcal{F}$ calls subroutine $\mathrm{Freeze}(\mathrm{id}, \mathrm{pk}^{(i)}, \mathrm{sk}^{(i)}, \alpha/\beta, \mathrm{pk}^{(i)}_{\mathcal{F}}, T_i)$ to transfer coins α/β from address $\mathrm{pk}^{(i)}$ to a specific address $\mathrm{pk}^{(i)}_{\mathcal{F}}$ controlled by $\mathcal{F}$ for a specified period of time T_i, where $T_0 = T_1 + \Delta$.

(B) Swap Complete Phase: User $\mathcal{P}_0$ sends the swap message $(\mathrm{swp}, \mathrm{id}, \mathrm{pk}^{(0)}_{swp})$ at time t'. If $t' < T_1$, the functionality $\mathcal{F}$ transfers coins β from address $\mathrm{pk}^{(1)}_{\mathcal{F}}$ to $\mathrm{pk}^{(0)}_{swp}$ and sets $\mathrm{b}^{(0)} = 0$ to indicate that user $\mathcal{P}_0$ has successfully completed the swap operation. Otherwise, user $\mathcal{P}_0$ fails in the swap phase (i.e., $\mathrm{b}^{(0)} = \perp$). Meanwhile, upon receiving swap message $(\mathrm{swp}, \mathrm{id}, \mathrm{pk}^{(1)}_{swp})$ from user $\mathcal{P}_1$, the functionality $\mathcal{F}$ transfers coins α from address $\mathrm{pk}^{(0)}_{\mathcal{F}}$ to $\mathrm{pk}^{(1)}_{swp}$ and sets $\mathrm{b}^{(1)} = 1$ if $\mathrm{b}^{(0)} = 0$; otherwise, sets $\mathrm{b}^{(1)} = \perp$.

(C) Swap Timeout Phase: After timeout T_i, if the coins are still stored in address $\mathrm{pk}^{(i)}_{\mathcal{F}}$, the functionality $\mathcal{F}$ transfers these coins to their original owner $\mathcal{P}_i$.

(A) Swap Setup Phase - Freezing Coins

01 Upon receiving $(\mathrm{frz}, \mathrm{id}, \mathrm{pk}^{(0)}, \mathrm{sk}^{(0)}, \alpha) \overset{t}{\hookleftarrow} \mathcal{P}_0$, invoke subroutine $\mathrm{Freeze}(\mathrm{id}, \mathrm{pk}^{(0)}, \mathrm{sk}^{(0)}, \alpha, \mathrm{pk}^{(0)}_{\mathcal{F}}, T_0)$; upon receiving $(\mathrm{Confirmed}, \mathrm{id}, \mathrm{ok}) \overset{t_1 \leq t+\delta+\varphi}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_0}$, send $(\mathrm{frz}, \mathrm{id}, \mathrm{ok}) \overset{t_1}{\hookrightarrow} \mathcal{P}_0$.
02 Upon receiving $(\mathrm{frz}, \mathrm{id}, \mathrm{pk}^{(1)}, \mathrm{sk}^{(1)}, \beta) \overset{t}{\hookleftarrow} \mathcal{P}_1$, invoke subroutine $\mathrm{Freeze}(\mathrm{id}, \mathrm{pk}^{(1)}, \mathrm{sk}^{(1)}, \beta, \mathrm{pk}^{(1)}_{\mathcal{F}}, T_1)$; upon receiving $(\mathrm{Confirmed}, \mathrm{id}, \mathrm{ok}) \overset{t_1 \leq t+\delta+\varphi}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_1}$, send $(\mathrm{frz}, \mathrm{id}, \mathrm{ok}) \overset{t_1}{\hookrightarrow} \mathcal{P}_1$.
03 After the above steps are successful, send $(\mathrm{Setup}, \mathrm{id}, \mathrm{ok}) \hookrightarrow \mathcal{P}_i$ $(i \in \{0, 1\})$ and proceed to procedure (B); otherwise, proceed to procedure (C).

(B) Swap Complete Phase

01 Upon receiving $(\mathrm{swp}, \mathrm{id}, \mathrm{pk}^{(0)}_{swp}) \overset{t'}{\hookleftarrow} \mathcal{P}_0$, do the following:
- If $t' < T_1$, set $\mathrm{b}^{(0)} = 0$ and invoke subroutine $\mathrm{Transfer}(\mathrm{id}, \mathrm{pk}^{(1)}_{\mathcal{F}}, \mathrm{sk}^{(1)}_{\mathcal{F}}, \beta, \mathrm{pk}^{(0)}_{swp}, t')$; upon receiving $(\mathrm{Publish}, \mathrm{id}, \mathrm{ok}) \overset{t'_1 \leq t'+\delta}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_1}$, send $(\mathrm{swp}, \mathrm{id}, \mathrm{ok}) \overset{t'_1}{\hookrightarrow} \mathcal{P}_0$.
- Otherwise, set $\mathrm{b}^{(0)} = \perp$ and abort.

02 Upon receiving $(\mathrm{swp}, \mathrm{id}, \mathrm{pk}^{(1)}_{swp}) \overset{t'}{\hookleftarrow} \mathcal{P}_1$, do the following:
- If $\mathrm{b}^{(0)} = 0$, set $\mathrm{b}^{(1)} = 1$ and invoke subroutine $\mathrm{Transfer}(\mathrm{id}, \mathrm{pk}^{(0)}_{\mathcal{F}}, \mathrm{sk}^{(0)}_{\mathcal{F}}, \alpha, \mathrm{pk}^{(1)}_{swp}, t')$; upon receiving $(\mathrm{Publish}, \mathrm{id}, \mathrm{ok}) \overset{t'_1 \leq t'+\delta}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_0}$, send $(\mathrm{swp}, \mathrm{id}, \mathrm{ok}) \overset{t'_1}{\hookrightarrow} \mathcal{P}_1$.
- Otherwise, set $\mathrm{b}^{(1)} = \perp$ and abort.

(C) Swap Timeout Phase

01 Upon receiving $(\mathrm{rfd}, \mathrm{id}, \mathrm{pk}^{(i)}_{rfd}) \overset{t''}{\hookleftarrow} \mathcal{P}_i$, do the following:
- If $(t'' > T_i) \wedge (\mathrm{b}^{(1-i)} = \perp)$, invoke subroutine $\mathrm{Unfreeze}(\mathrm{id}, \mathrm{pk}^{(i)}_{\mathcal{F}}, \mathrm{sk}^{(i)}_{\mathcal{F}}, \alpha/\beta, \mathrm{pk}^{(i)}_{rfd}, t'')$; upon receiving $(\mathrm{Publish}, \mathrm{id}, \mathrm{ok}) \overset{t''_1 \leq t''+\delta}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_i}$, send $(\mathrm{rfd}, \mathrm{id}, \mathrm{ok}) \overset{t''_1}{\hookrightarrow} \mathcal{P}_i$.
- Otherwise, abort.

Fig. 8: Ideal functionality $\mathcal{F}$ that models the cross-chain swaps

Remark 1: Careful readers may notice that the functionality $\mathcal{F}$ completes the Swap Setup Phase only when the freeze transactions have been finally confirmed by the underlying blockchains. In contrast, during the Swap Complete Phase and Swap Timeout Phase, the functionality $\mathcal{F}$ responds with ok when the corresponding swap transaction and refund transaction have been added to their respective bulletin boards. The correctness lies in the facts that only a finally confirmed transaction can be further spent by a new transaction (wherein the freeze transaction will be spent by a swap or refund transaction), and as defined in blockchain functionality $\mathcal{F}_{\mathbb{B}}$ (Fig.7), transactions in the bulletin board can indeed be finally confirmed within time φ.

Security analysis. We now analyze that the ideal functionality $\mathcal{F}$ satisfies *atomicity*:

Successful Swap: If user $\mathcal{P}_0$ initiates the swap operation honestly before timeout T_1, the functionality $\mathcal{F}$ will enable both users $\mathcal{P}_0$ and $\mathcal{P}_1$ to complete the swap via transferring the frozen coins (controlled by $\mathcal{F}$) to their respective addresses $\mathrm{pk}^{(0)}_{swp}$ and $\mathrm{pk}^{(1)}_{swp}$ (i.e., $\mathrm{b}^{(0)} = 0$ and $\mathrm{b}^{(1)} = 1$).

Failed Swap: After the timeout T_i $(i \in \{0, 1\})$, if $\mathrm{b}^{(1-\mathrm{i})} = \perp$,

//Subroutine **Freeze**

Freeze$(\mathrm{id}, \mathrm{pk}, \mathrm{sk}, v, \mathrm{pk}_{\mathcal{F}}, T)$: set timeout T and transfer coins v from address pk to $\mathrm{pk}_{\mathcal{F}}$ (controlled by $\mathcal{F}$) via generating a freeze transaction $\mathrm{tx}_{frz} := (\mathrm{pk}, \mathrm{pk}_{\mathcal{F}}, v, \sigma)$ with secret key sk and invoking interface $\mathrm{Confirm}(\mathrm{tx}_{frz})$ of blockchain functionality $\mathcal{F}_{\mathbb{B}}$. If transaction tx_{frz} has been finally confirmed by $\mathcal{F}_{\mathbb{B}}$ (i.e., $\mathrm{tx}_{frz} \in \mathcal{L}$), then respond $(\mathrm{frz}, \mathrm{id}, \mathrm{ok})$.

//Subroutine **Transfer**

Transfer$(\mathrm{id}, \mathrm{pk}_{\mathcal{F}}, \mathrm{sk}_{\mathcal{F}}, v, \mathrm{pk}_{swp}, t)$: transfer frozen coins v from address $\mathrm{pk}_{\mathcal{F}}$ to pk_{swp} via generating a swap transaction $\mathrm{tx}_{swp} := (\mathrm{pk}_{\mathcal{F}}, \mathrm{pk}_{swp}, v, \sigma)$ with secret key $\mathrm{sk}_{\mathcal{F}}$ and invoking interface $\mathrm{Publish}(\mathrm{tx}_{swp}, t)$ of blockchain functionality $\mathcal{F}_{\mathbb{B}}$. If transaction tx_{swp} has been added to bulletin board $\mathcal{T}$ (i.e., $\mathrm{tx}_{swp} \in \mathcal{T}$), then respond $(\mathrm{swp}, \mathrm{id}, \mathrm{ok})$.

//Subroutine **Unfreeze**

Unfreeze$(\mathrm{id}, \mathrm{pk}_{\mathcal{F}}, \mathrm{sk}_{\mathcal{F}}, v, \mathrm{pk}_{rfd}, t)$: transfer frozen coins v from address $\mathrm{pk}_{\mathcal{F}}$ to pk_{rfd} via generating a refund transaction $\mathrm{tx}_{rfd} := (\mathrm{pk}_{\mathcal{F}}, \mathrm{pk}_{rfd}, v, \sigma)$ with secret key $\mathrm{sk}_{\mathcal{F}}$. If transaction tx_{rfd} has been added to bulletin board $\mathcal{T}$ (i.e., $\mathrm{tx}_{rfd} \in \mathcal{T}$), then respond $(\mathrm{rfd}, \mathrm{id}, \mathrm{ok})$.

Fig. 9: The subroutines of ideal functionality $\mathcal{F}$

the functionality $\mathcal{F}$ will enable user $\mathcal{P}_i$ to refund his frozen coins.

VI. PipeSwap: Protocol Description

A. Cryptographic Building Blocks

To guarantee universality, we insist on the fundamental building blocks from [13], namely, the adaptor signature scheme [14] and verifiable timed discrete logarithm (VTD) scheme [15].

Adaptor signatures. The adaptor signature scheme (cf. Def.1 in Appendix B) is defined w.r.t. a digital signature scheme $\mathrm{SIG} = (\mathrm{KGen}, \mathrm{Sig}, \mathrm{Vf})$ (cf. Def.2 in Appendix B) and a hard relation $\mathcal{R}$ (cf. Def.3 in Appendix B). It allows users to insert a puzzle Y (e.g., statement-witness pair $(\mathrm{Y}, y) \in \mathcal{R}$) into the generation of a signature on message $\mathrm{m} \in \{0,1\}^{\lambda}$. The users with secret key first compute a pre-signature $\tilde{\sigma}$ of message m which by itself is not a valid digital signature but can later be adapted into a valid signature σ with witness y (i.e., $\mathrm{SIG.Vf}(\mathrm{pk}, \mathrm{m}, \sigma) = 1$). Additionally, witness y can be further extracted by $\tilde{\sigma}$ and σ.

The digital signature scheme satisfies the standard notion of unforgeability [35]. To showcase the universality of our construction, we assume $\mathrm{SIG} \in \{\mathrm{Schnorr}, \mathrm{ECDSA}\}$ to encompass most existing cryptocurrencies such as Bitcoin, Ethereum and Ripple. Adaptor signature scheme is required to satisfy security properties of unforgeability, witness extractability and pre-signature adaptability (cf. [13] for detailed definitions).

Verifiable timed dlog (VTD). The VTD enables the committer to generate a timed commitment C of value x with timing hardness T, which can be verified publicly and forcibly opened in time T (cf. Def.4 in Appendix B). The VTD is required to satisfy security properties of soundness and privacy (cf. [13] for detailed definitions).

In this work, we also use adaptor signature scheme and VTD in a black-box manner, and direct readers to efficient constructions in [13–15, 36]. To elaborate further, following the approach presented in [13], we adopt the construction of adaptor signature in [14] with the underlying signature scheme being Schnorr or ECDSA, and the hard relation $\mathcal{R}$ being the discrete log (dlog) relation (i.e., the language is defined as $\mathcal{L}_{\mathcal{R}_{\mathrm{dlog}}} := \{\mathrm{Y} | \exists y \in \mathbb{Z}_q^*, s.t.\ \mathrm{Y} = \mathrm{g}^y \in \mathbb{G}\}$). For the construction of VTD, the committer embeds the dlog.value y inside a time-lock puzzle Y, uses a non-interactive zero-knowledge proof (NIZK) to prove that Y can be solved in time T and the value y satisfies equation $\mathrm{Y} = \mathrm{g}^y$. An efficient construction of the NIZK [15] can be derived from the cut-and-choose techniques, Shamir secret sharing [37] and homomorphic time-lock puzzles [38].

Additionally, as the frozen address pk is under joint control of users $\mathcal{P}_0$ and $\mathcal{P}_1$ (i.e., the corresponding secret key sk is shared between them), it is inevitable that we also rely on interactive protocols (represented as $\Gamma^{\mathrm{SIG}}_{\mathrm{AdpSig}}$ and $\Gamma^{\mathrm{SIG}}_{\mathrm{Sig}}$) to realize jointly (pre-)signing a message m under public key pk. This can be efficiently instantiated w.r.t. $\mathrm{SIG} \in \{\mathrm{Shnorr}, \mathrm{ECDSA}\}$ utilizing protocols in [18].

B. Procedures of PipeSwap

In the general setting, users $\mathcal{P}_0$ (who owns coins α on blockchain $\mathbb{B}_0$) and $\mathcal{P}_1$ (who owns coins β on blockchain $\mathbb{B}_1$) aim to complete the α-to-β cross-chain swaps. As mentioned previously (cf. Section IV), *pipelined coins flow* of the frozen coins guarantees *atomicity*. The crux of securing PipeSwap lies in ensuring the timely release of witness y, which is achieved by three critical ingredients: *splitting frozen coins β into $(\varepsilon, \beta - \varepsilon)$*, a *two-hop swap* and a *two-hop refund*.

Protocol details. For ease of understanding, we depict the coins flow of PipeSwap in Fig.10 and provide a description of PipeSwap in Fig.11, including the key points of each phase presented below:

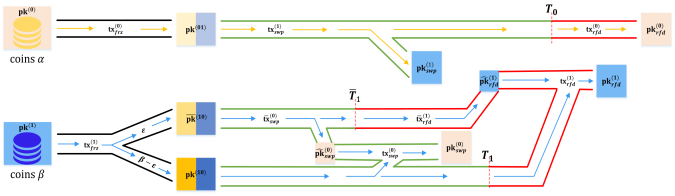


Fig. 10: The pipelined coins flow of PipeSwap

(A) Swap Setup Phase-Freezing Coins: In this phase, users $\mathcal{P}_0$ and $\mathcal{P}_1$ transfer their swapped coins to the corresponding frozen addresses, which are under joint control of both users. To be better prepared for the *two-hop swap* and *two-hop refund*, we let user $\mathcal{P}_1$ split coins β into two frozen addresses $\overline{\mathrm{pk}}^{(10)}$ and $\mathrm{pk}^{(10)}$ with respective values ε and $\beta - \varepsilon$. Importantly, to guarantee that the frozen coins β are refunded as expected, the pre-refund transaction $\overline{\mathrm{tx}}^{(1)}_{rfd} := (\overline{\mathrm{pk}}^{(10)}, \widehat{\mathrm{pk}}^{(1)}_{rfd}, \varepsilon)$ is locked until time $\overline{T}_1 := T_1 - 2\delta - \varphi$ (meaning user $\mathcal{P}_0$ makes a timed commitment of secret key share $\overline{\mathrm{sk}}^{(10)}_0$ with timing hardness $\overline{T}_1$). Meanwhile, the refund transaction

$\mathrm{tx}_{rfd}^{(1)} := ((\mathrm{pk}^{(10)}, \widehat{\mathrm{pk}}_{rfd}^{(1)}), \mathrm{pk}_{rfd}^{(1)}, \beta)$ is jointly signed with secret keys $\mathrm{sk}^{(10)}$ (held by users $\mathcal{P}_0$ and $\mathcal{P}_1$) and $\widehat{\mathrm{sk}}_{rfd}^{(1)}$ (held by user $\mathcal{P}_1$), and will become valid when the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ has been confirmed. Additionally, user $\mathcal{P}_1$ commits to a timed refunding of frozen coins α by providing the secret key share $\mathrm{sk}_1^{(01)}$ with timing hardness T_0, which allows user $\mathcal{P}_0$ to generate a refund transaction $\mathrm{tx}_{rfd}^{(0)} := (\mathrm{pk}^{(01)}, \mathrm{pk}_{rfd}^{(0)}, \alpha)$ after timeout T_0.

(B1) Swap Lock Phase: This phase is the preparation for atomic swaps. Both users jointly pre-sign swap transactions $\mathrm{tx}_{swp}^{(1)} := (\mathrm{pk}^{(01)}, \mathrm{pk}_{swp}^{(1)}, \alpha)$ and $\overline{\mathrm{tx}}_{swp}^{(0)} := (\overline{\mathrm{pk}}^{(10)}, \widehat{\mathrm{pk}}_{swp}^{(0)}, \varepsilon)$ in sequence, where the statement-witness pair $(\mathrm{Y}, y) \in \mathcal{R}_{\mathrm{dlog}}$ is selected by user $\mathcal{P}_0$. Similarly, to guarantee that the frozen coins β are swapped as expected, the swap transaction $\mathrm{tx}_{swp}^{(0)} := ((\mathrm{pk}^{(10)}, \widehat{\mathrm{pk}}_{swp}^{(0)}), \mathrm{pk}_{swp}^{(0)}, \beta)$ is jointly signed with secret keys $\mathrm{sk}^{(10)}$ (held by users $\mathcal{P}_0$ and $\mathcal{P}_1$) and $\widehat{\mathrm{sk}}_{swp}^{(0)}$ (held by user $\mathcal{P}_0$), and the final confirmation of pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is an essential prerequisite for generating a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$. Therefore, in order to generate a valid swap transaction, user $\mathcal{P}_0$ is forced to timely release witness y at least φ time before (i.e., by posting the pre-swap transaction $\overline{tx}_{swp}^{(0)}$).

(B2) Swap Complete Phase: User $\mathcal{P}_0$ posts a pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ before time $\overline{T}_1 - \delta$, and its final confirmation ensures user $\mathcal{P}_0$ to generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ before timeout T_1. Additionally, the posted pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ enables user $\mathcal{P}_1$ to generate a valid swap transaction $\mathrm{tx}_{swp}^{(1)}$ with the release of witness y.

(C) Swap Timeout Phase: After timeout T_1, if user $\mathcal{P}_0$ fails to unfreeze the frozen coins β (i.e., the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is not finally confirmed), user $\mathcal{P}_1$, who has the finally confirmed pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$, posts a refund transaction $\mathrm{tx}_{rfd}^{(1)}$. Similarly, after timeout T_0, if user $\mathcal{P}_1$ fails to post a swap transaction $\mathrm{tx}_{swp}^{(1)}$, user $\mathcal{P}_0$ posts a refund transaction $\mathrm{tx}_{rfd}^{(0)}$.

Security intuitions. With the rigorous construction, PipeSwap achieves the same level of security as HTLC. We now provide brief security intuitions below and defer the detailed proofs to Appendix C.

Successful Swap. Before timeout T_1, if user $\mathcal{P}_0$ can generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ (i.e., the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ has been finally confirmed), user $\mathcal{P}_1$ can post a swap transaction $\mathrm{tx}_{swp}^{(1)}$ with witness y extracted from the signature of $\overline{\mathrm{tx}}_{swp}^{(0)}$. Therefore, both swap transactions $\mathrm{tx}_{swp}^{(0)}$ and $\mathrm{tx}_{swp}^{(1)}$ must be finally confirmed.

Failed Swap. We consider the following possible cases:

- User $\mathcal{P}_0$ does not initiate his swap operation before timeout T_1 (i.e., the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is not confirmed before timeout T_1), then both users can successfully refund their frozen coins.
- Due to delay of posting the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$, malicious user $\mathcal{P}_0$ fails to generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ before timeout T_1 (i.e., the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ is finally confirmed). As a result, user $\mathcal{P}_1$ can successfully refund the frozen coins β after timeout T_1.
- The malicious $\mathcal{P}_0$ never can generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$, unless the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ has been finally confirmed before timeout T_1, which further enables user $\mathcal{P}_1$ to complete his swap operation by generating a valid swap transaction $\mathrm{tx}_{swp}^{(1)}$ before timeout T_1. Otherwise, after timeout T_1, user $\mathcal{P}_1$ can generate a valid refund transaction $\mathrm{tx}_{rfd}^{(1)}$ with the finally confirmed pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ to successfully refund the frozen coins β.

C. Evaluation and Comparison

Implementation details. We develop a prototypical C implementation to demonstrate the feasibility of our construction and evaluate its performance. We conduct experiments on a PC with the following configuration: CPU(Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz with 4 cores), RAM(16.0 GB) and OS(x64-based Windows). Basically, we instantiate the signatures of Schnorr and ECDSA over the secp256k1 curve, and set the transaction size to 250 bytes to approximate a basic Bitcoin transaction. We implement the two-party computation protocol for digital signature $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$, and use the implementations of adaptor signature $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$ and VTD respectively in [25] and [15]. The source code is available on Github [39].

Computation time. We first measure the time of basic operations required in PipeSwap, and the results are shown in TABLE II. Subsequently, the computation time required by both users together is recorded in TABLE III. It is observed that (1) each instance of PipeSwap requires only 1.605 seconds for Schnorr and 1.624 seconds for ECDSA; (2) the computation time of Swap Setup-Freezing Phase accounts for more than 99%, as both users jointly complete two VTD computations.

Communication overhead. We measure the communication overhead as the amount of exchanged messages between users during the execution of interactive algorithms in the Swap Setup Phase and Swap Lock Phase (cf. TABLE IV). Specifically, PipeSwap requires 6.4 kb for Schnorr and 7 kb for ECDSA, which is dominated by that of respectively exchanging the VTD.Commit-proof pair of secret key share.

TABLE III: The computation time (ms)*

		Setup Phase	Lock Phase	Complete Phase
Schnorr	**PipeSwap**	1593.752	9.508	1.353
	UAS	1590.05	8.786	0.706
ECDSA	**PipeSwap**	1594.513	27.431	2.09
	UAS	1590.384	26.05	1.044

*UAS (Universal Atomic Swaps) [13].

Efficiency comparison. To compare PipeSwap (Fig.11) and Universal Atomic Swaps (its Fig.5 in [13]) w.r.t. the operations required by both users together, we conduct an

Assume the swapped coins α and β are respectively stored in addresses $\mathrm{pk}^{(0)}$ and $\mathrm{pk}^{(1)}$ on blockchains $\mathbb{B}_0$ and $\mathbb{B}_1$. Global parameters are $(\mathbb{G}, \mathrm{q}, \mathrm{g})$, δ, φ, $T_1 := \overline{T}_1 + 2\delta + \varphi$ and $T_0 := T_1 + \Delta$; $\oplus := +$ if SIG = Schnorr and $\oplus := *$ if SIG = ECDSA.

(A) Swap Setup Phase - Freezing Coins

01 Users $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively complete Setups:

1) $\mathcal{P}_0$ runs Setup process (Fig.12) and sends $(\mathrm{pk}_0^{(01)}, \overline{\mathrm{pk}}_0^{(10)}, \mathrm{pk}_0^{(10)}, (\mathrm{C}^{(1)}, \pi^{(1)})) \hookrightarrow \mathcal{P}_1$.
2) $\mathcal{P}_1$ runs Setup process (Fig.12) and sends $(\mathrm{pk}_1^{(01)}, \overline{\mathrm{pk}}_1^{(10)}, \mathrm{pk}_1^{(10)}, (\mathrm{C}^{(0)}, \pi^{(0)})) \hookrightarrow \mathcal{P}_0$.

02 Users $\mathcal{P}_0$ and $\mathcal{P}_1$ generate their frozen addresses:

1) $\mathcal{P}_0$ checks if $\mathrm{VTD.Vf}(\mathrm{pk}_1^{(01)}, \mathrm{C}^{(0)}, \pi^{(0)}) = 1$ and generates frozen address $\mathrm{pk}^{(01)} = (\mathrm{pk}_1^{(01)})^{\oplus \mathrm{sk}_0^{(01)}}$; otherwise, stops.
2) $\mathcal{P}_1$ checks if $\mathrm{VTD.Vf}(\overline{\mathrm{pk}}_0^{(10)}, \mathrm{C}^{(1)}, \pi^{(1)}) = 1$, and generates frozen addresses $\overline{\mathrm{pk}}^{(10)} = (\overline{\mathrm{pk}}_0^{(10)})^{\oplus \overline{\mathrm{sk}}_1^{(10)}}$ and $\mathrm{pk}^{(10)} = (\mathrm{pk}_0^{(10)})^{\oplus \mathrm{sk}_1^{(10)}}$; otherwise, stops.

03 Users $\mathcal{P}_0$ and $\mathcal{P}_1$ transfer the swapped coins to the corresponding frozen addresses.

1) $\mathcal{P}_0$ generates the freeze transaction $\mathrm{tx}_{frz}^{(0)} := (\mathrm{pk}^{(0)}, \mathrm{pk}^{(01)}, \alpha)$ and signature $\sigma_{frz}^{(0)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\mathrm{sk}^{(0)}, \mathrm{tx}_{frz}^{(0)})$, and then posts $(\mathrm{tx}_{frz}^{(0)}, \sigma_{frz}^{(0)})$ on blockchain $\mathbb{B}_0$ and starts solving $\mathrm{VTD.ForceOp}(\mathrm{C}^{(0)})$.
2) Users $\mathcal{P}_0$ and $\mathcal{P}_1$ jointly do the following:
   - $\mathcal{P}_1$ generates the freeze transaction $\mathrm{tx}_{frz}^{(1)} := (\mathrm{pk}^{(1)}, (\overline{\mathrm{pk}}^{(10)}, \mathrm{pk}^{(10)}), (\varepsilon, \beta - \varepsilon))$, pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)} := (\overline{\mathrm{pk}}^{(10)}, \widehat{\mathrm{pk}}_{rfd}^{(1)}, \varepsilon)$, refund transaction $\mathrm{tx}_{rfd}^{(1)} := ((\mathrm{pk}^{(10)}, \widehat{\mathrm{pk}}_{rfd}^{(1)}), \mathrm{pk}_{rfd}^{(1)}, \beta)$, and sends $(\mathrm{tx}_{frz}^{(1)}, \overline{\mathrm{tx}}_{rfd}^{(1)}, \mathrm{tx}_{rfd}^{(1)}) \hookrightarrow \mathcal{P}_0$;
   - $\mathcal{P}_0$ checks if $\mathrm{tx}_{frz}^{(1)}, \overline{\mathrm{tx}}_{rfd}^{(1)}, \mathrm{tx}_{rfd}^{(1)}$ are well formed (i.e., satisfying *two-hop refund* framework), and stops otherwise;
   - $\mathcal{P}_0$ and $\mathcal{P}_1$ run protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$ with input $(\mathrm{sk}_0^{(10)}, \mathrm{sk}_1^{(10)}, \mathrm{tx}_{rfd}^{(1)})$ (Fig.12) and obtain signature $\check{\sigma}_{rfd}^{(1)}$;
   - $\mathcal{P}_1$ computes signature $\sigma_{frz}^{(1)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\mathrm{sk}^{(1)}, \mathrm{tx}_{frz}^{(1)})$ and posts $(\mathrm{tx}_{frz}^{(1)}, \sigma_{frz}^{(1)})$ on blockchain $\mathbb{B}_1$, and then starts solving $\mathrm{VTD.ForceOp}(\mathrm{C}^{(1)})$.

(B1) Swap Lock Phase

01 User $\mathcal{P}_0$ runs $(\mathrm{Y}, y) \leftarrow \mathcal{R}\mathrm{Gen}(1^\lambda)$ and sends $\mathrm{Y} \hookrightarrow \mathcal{P}_1$.

02 Users $\mathcal{P}_0$ and $\mathcal{P}_1$ generates their swap transactions:

1) $\mathcal{P}_0$ generates pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)} := (\overline{\mathrm{pk}}^{(10)}, \widehat{\mathrm{pk}}_{swp}^{(0)}, \varepsilon)$ and swap transaction $\mathrm{tx}_{swp}^{(0)} := ((\mathrm{pk}^{(10)}, \widehat{\mathrm{pk}}_{swp}^{(0)}), \mathrm{pk}_{swp}^{(0)}, \beta)$, and sends $(\overline{\mathrm{tx}}_{swp}^{(0)}, \mathrm{tx}_{swp}^{(0)}) \hookrightarrow \mathcal{P}_1$.
2) $\mathcal{P}_1$ checks if $\overline{\mathrm{tx}}_{swp}^{(0)}, \mathrm{tx}_{swp}^{(0)}$ are well formed (i.e., satisfying *two-hop swap* framework), and generates swap transaction $\mathrm{tx}_{swp}^{(1)} := (\mathrm{pk}^{(01)}, \mathrm{pk}_{swp}^{(1)}, \alpha)$ and sends $\mathrm{tx}_{swp}^{(1)} \hookrightarrow \mathcal{P}_0$; otherwise, stops.

03 Users $\mathcal{P}_0$ and $\mathcal{P}_1$ run protocol $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$ with input $(\mathrm{sk}_0^{(01)}, \mathrm{sk}_1^{(01)}, \mathrm{Y}, \mathrm{tx}_{swp}^{(1)})$ (Fig.12), and obtain pre-signature $\widetilde{\sigma}_{swp}^{(1)}$.

04 After step 03 is successful, $\mathcal{P}_0$ and $\mathcal{P}_1$ run protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$ with input $(\mathrm{sk}_0^{(10)}, \mathrm{sk}_1^{(10)}, \mathrm{tx}_{swp}^{(0)})$ (Fig.12) and obtain signature $\check{\sigma}_{swp}^{(0)}$, and then run protocol $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$ with input $(\overline{\mathrm{sk}}_0^{(10)}, \overline{\mathrm{sk}}_1^{(10)}, \mathrm{Y}, \overline{\mathrm{tx}}_{swp}^{(0)})$ (Fig.12) and obtain pre-signature $\widetilde{\overline{\sigma}}_{swp}^{(0)}$.

(B2) Swap Complete Phase

05 User $\mathcal{P}_0$ computes $\overline{\sigma}_{swp}^{(0)} \leftarrow \Sigma_{\mathrm{AS}}^{\mathrm{SIG}}.\mathrm{Adapt}(\widetilde{\overline{\sigma}}_{swp}^{(0)}, y)$ and posts $(\overline{\mathrm{tx}}_{swp}^{(0)}, \overline{\sigma}_{swp}^{(0)})$ on blockchain $\mathbb{B}_1$ before time $\overline{T}_1 - \delta$. If $\overline{\mathrm{tx}}_{swp}^{(0)}$ has been confirmed, he computes $\widehat{\sigma}_{swp}^{(0)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\widehat{\mathrm{sk}}_{swp}^{(0)}, \mathrm{tx}_{swp}^{(0)})$ and posts $(\mathrm{tx}_{swp}^{(0)}, (\check{\sigma}_{swp}^{(0)}, \widehat{\sigma}_{swp}^{(0)}))$ on blockchain $\mathbb{B}_1$.

06 If user $\mathcal{P}_0$ fails to post $(\overline{\mathrm{tx}}_{swp}^{(0)}, \overline{\sigma}_{swp}^{(0)})$ before time $\overline{T}_1$, $\mathcal{P}_1$ finishes computing $\overline{\mathrm{sk}}_0^{(10)} \leftarrow \mathrm{VTD.ForceOp}(\mathrm{C}^{(1)})$ and computes $\overline{\mathrm{sk}}^{(10)} := \overline{\mathrm{sk}}_0^{(10)} \oplus \overline{\mathrm{sk}}_1^{(10)}$. He then computes $\overline{\sigma}_{rfd}^{(1)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\overline{\mathrm{sk}}^{(10)}, \overline{\mathrm{tx}}_{rfd}^{(1)})$ and posts $(\overline{\mathrm{tx}}_{rfd}^{(1)}, \overline{\sigma}_{rfd}^{(1)})$ on blockchain $\mathbb{B}_1$.

07 If $\overline{\mathrm{tx}}_{swp}^{(0)}$ has been finally confirmed, user $\mathcal{P}_1$ computes $y \leftarrow \Sigma_{\mathrm{AS}}^{\mathrm{SIG}}.\mathrm{Ext}(\overline{\sigma}_{swp}^{(0)}, \widetilde{\overline{\sigma}}_{swp}^{(0)}, \mathrm{Y})$ and $\sigma_{swp}^{(1)} \leftarrow \Sigma_{\mathrm{AS}}^{\mathrm{SIG}}.\mathrm{Adap}(\widetilde{\sigma}_{swp}^{(1)}, y)$, and posts $(\mathrm{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)})$ on blockchain $\mathbb{B}_0$.

(C) Swap Refund Phase

01 After timeout T_1, if $\overline{\mathrm{tx}}_{rfd}^{(1)}$ has been confirmed, user $\mathcal{P}_1$ computes $\widehat{\sigma}_{rfd}^{(1)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\widehat{\mathrm{sk}}_{rfd}^{(1)}, \mathrm{tx}_{rfd}^{(1)})$ and posts $(\mathrm{tx}_{rfd}^{(1)}, (\check{\sigma}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}))$ on blockchain $\mathbb{B}_1$.

02 Similarly, after timeout T_0, if user $\mathcal{P}_1$ fails to post $(\mathrm{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)})$, $\mathcal{P}_0$ finishes computing $\mathrm{sk}_1^{(01)} \leftarrow \mathrm{VTD.ForceOp}(\mathrm{C}^{(0)})$ and computes $\mathrm{sk}^{(01)} := \mathrm{sk}_0^{(01)} \oplus \mathrm{sk}_1^{(01)}$. He then computes $\sigma_{rfd}^{(0)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\mathrm{sk}^{(01)}, \mathrm{tx}_{rfd}^{(0)})$ and posts $(\mathrm{tx}_{rfd}^{(0)}, \sigma_{rfd}^{(0)})$ on blockchain $\mathbb{B}_0$.

Fig. 11: PipeSwap: a secure cross-chain swaps protocol between users $\mathcal{P}_0$ and $\mathcal{P}_1$

//The Setup process

User $\mathcal{P}_0$ does the following:

(1) Run $\Sigma_{\mathrm{SIG}}.\mathrm{KGen}(1^\lambda) \to \{(\mathrm{sk}_0^{(01)}, \mathrm{pk}_0^{(01)}), (\overline{\mathrm{sk}}_0^{(10)}, \overline{\mathrm{pk}}_0^{(10)}), (\mathrm{sk}_0^{(10)}, \mathrm{pk}_0^{(10)}), (\mathrm{sk}_{rfd}^{(0)}, \mathrm{pk}_{rfd}^{(0)}), (\widehat{\mathrm{sk}}_{swp}^{(0)}, \widehat{\mathrm{pk}}_{swp}^{(0)}), (\mathrm{sk}_{swp}^{(0)}, \mathrm{pk}_{swp}^{(0)})\}$.

(2) Compute commitment $\mathrm{VTD.Commit}(\overline{\mathrm{sk}}_0^{(10)}, \overline{T}_1) \to (\mathrm{C}^{(1)}, \pi^{(1)})$.

User $\mathcal{P}_1$ does the following:

(1) Run $\Sigma_{\mathrm{SIG}}.\mathrm{KGen}(1^\lambda) \to \{(\mathrm{sk}_1^{(01)}, \mathrm{pk}_1^{(01)}), (\overline{\mathrm{sk}}_1^{(10)}, \overline{\mathrm{pk}}_1^{(10)}), (\mathrm{sk}_1^{(10)}, \mathrm{pk}_1^{(10)}), (\widehat{\mathrm{sk}}_{rfd}^{(1)}, \widehat{\mathrm{pk}}_{rfd}^{(1)}), (\mathrm{sk}_{rfd}^{(1)}, \mathrm{pk}_{rfd}^{(1)}), (\mathrm{sk}_{swp}^{(1)}, \mathrm{pk}_{swp}^{(1)})\}$.

(2) Compute commitment $\mathrm{VTD.Commit}(\mathrm{sk}_1^{(01)}, T_0) \to (\mathrm{C}^{(0)}, \pi^{(0)})$.

//The 2PC protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$

It takes private inputs sk_0 and sk_1 held by users $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively, and public message m:

(1) Set secret key as $\mathrm{sk} := \mathrm{sk}_0 \oplus \mathrm{sk}_1$.

(2) Compute signature $\check{\sigma} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\mathrm{sk}, \mathrm{m})$ and sends $\check{\sigma}$ to both users $\mathcal{P}_0$ and $\mathcal{P}_1$.

(3) Users $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively check if $\Sigma_{\mathrm{SIG}}.\mathrm{Vf}(\mathrm{pk}, \mathrm{m}, \check{\sigma}) = 1$, and stop otherwise.

//The 2PC protocol $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$

It takes private inputs sk_0 and sk_1 held by users $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively, and public messages (m, Y):

(1) Set secret key as $\mathrm{sk} := \mathrm{sk}_0 \oplus \mathrm{sk}_1$.

(2) Compute pre-signature $\widetilde{\sigma} \leftarrow \Sigma_{\mathrm{AS}}^{\mathrm{SIG}}.\mathrm{pSig}(\mathrm{sk}, \mathrm{m}, \mathrm{Y})$ and sends $\widetilde{\sigma}$ to both users $\mathcal{P}_0$ and $\mathcal{P}_1$.

(3) Users $\mathcal{P}_0$ and $\mathcal{P}_1$ respectively check if $\Sigma_{\mathrm{AS}}^{\mathrm{SIG}}.\mathrm{pVf}(\mathrm{pk}, \mathrm{m}, \mathrm{Y}, \widetilde{\sigma}) = 1$, and stop otherwise.

Fig. 12: The subroutines of protocol PipeSwap

TABLE II: The computation time of basic operations (ms)

	Σ_{SIG}			$\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$		$\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$				VTD (n=64)	
	KGen	Sig	Vf	Sig	Vf	pSig	pVf	Ext	Adapt	Commit	Vf
Schnorr	0.745	0.647	1.142	0.722	1.332	4.393	2.837	0.7	0.003	413.057	378.341
ECDSA	0.687	1.046	1.461	1.381	1.479	13.025	7.156	0.936	0.054		

TABLE IV: The communication overheads (bytes)*

		Setup Phase	Lock Phase
Schnorr	**PipeSwap**	5060	1518
	UAS	4240	1012
ECDSA	**PipeSwap**	5220	1998
	UAS	4240	1332

*UAS (Universal Atomic Swaps) [13].

evaluation of PipeSwap and Universal Atomic Swaps using the same settings, libraries, and security parameters for all cryptographic implementations. Before delving into details, let us clarify that Universal Atomic Swaps repeat the same operations for each swapped coin. Therefore, we only consider the one-to-one atomic swap in [13]. The completion time for Universal Atomic Swaps is 1.6 seconds for Schnorr and 1.617 seconds for ECDSA, and the communication overhead is 5.1 kb for Schnorr and 5.4 kb for ECDSA. Based on this comparison, we find that PipeSwap is only $\leq$ 7 ms slower and incurs extra $\leq$ 1.6 kb communication overhead for preparing and signing two extra transactions (i.e., the pre-swap and pre-refund transactions), which remains within acceptable limits. Additionally, we stress that PipeSwap sets the same timeout parameters T_0 and T_1 as Universal Atomic Swaps, while the actual hardness parameter for user $\mathcal{P}_1$ is lower than T_1, represented by $\overline{T}_1 < T_1$, resulting in the reduced computational costs incurred by PipeSwap.

Therefore, PipeSwap not only achieves *atomicity* and *universality*, but is also efficient with low overhead.

VII. Conclusions and Future Works

In this paper, we identify a new form of attack known as *double-claiming* attack, which specifically targets the scriptless realization of atomic cross-chain swaps such as Universal Atomic Swaps [IEEE S&P'22] and Sweep-UC [IEEE S&P'24]. This attack can lead to honest user losing coins with overwhelming probability and the *atomicity* property is directly broken. We introduce a novel approach to secure coins flow by utilizing *two-hop swap* and *two-hop refund* techniques, and design PipeSwap, a universal atomic cross-chain swaps protocol that satisfies the *atomicity* property.

Several thought-provoking questions can be considered in future work. The instantiation of PipeSwap with standard signatures Schnorr and ECDSA is efficient, but further caution may be required when extending to other signature schemes. An interesting area for exploration is the extension to more complex but practical scenarios, e.g., multi-hop swaps $\mathcal{P}_1 \to \mathcal{P}_2 \to \cdots \to \mathcal{P}_n \to \mathcal{P}_1$, where each intermediate user only holds the desired coins of its right neighbor. Also, we leave to further work how to apply the pipelined coins flow paradigm to scriptless payment channel networks protocols providing stronger security.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[3] D. Schwartz, N. Youngs, A. Britto *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, no. 8, p. 151, 2014.

[4] R. W. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang, "Omniring: Scaling private payments without trusted setup," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 31–48.

[5] T. Nolan, "Alt chains and atomic transfers," Bitcoin Forum, 2013, https://bitcointalk.org/index.php?topic=193281.0.

[6] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 229–243.

[7] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM symposium on principles of distributed computing*, 2018, pp. 245–254.

[8] S. Bursuc and S. Kremer, "Contingent payments on a public ledger: models and reductions for automated verification," in *Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24*. Springer, 2019, pp. 361–382.

[9] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 455–471.

[10] "What is atomic swap and how to implement it," 2019, https://www.axiomadev.com/blog/what-is-atomic-swap-and-how-to-implement-it/.

[11] A. Poelstra, "Mimblewimble," 2016.

[12] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 459–474.

[13] S. A. Thyagarajan, G. Malavolta, and P. Moreno-Sanchez, "Universal atomic swaps: Secure exchange of coins across all blockchains," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1299–1316.

[14] 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 *Advances in Cryptology–ASIACRYPT 2021*. Springer, 2021, pp. 635–664.

[15] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder, "Verifiable timed signatures made practical," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1733–1750.

[16] L. Hanzlik, J. Loss, S. A. Thyagarajan, and B. Wagner, "Sweep-uc: Swapping coins privately," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 81–81.

[17] P. Hoenisch, S. Mazumdar, P. Moreno-Sanchez, and S. Ruj, "Lightswap: An atomic swap does not require timeouts at both blockchains," in *International Workshop on Data Privacy Management*. Springer, 2022, pp. 219–235.

[18] 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.

[19] S. A. K. Thyagarajan and G. Malavolta, "Lockable signatures for blockchains: Scriptless scripts for all signatures," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 937–954.

[20] L. Aumayr, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Breaking and fixing virtual channels: Domino attack and donner," *Cryptology ePrint Archive*, 2021.

[21] D. Hopwood, S. Bowe, T. Hornby, N. Wilcox *et al.*, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, vol. 4, no. 220, p. 32, 2016.

[22] 2023, https://thecharlatan.ch/Monero-Unlock-Time-Privacy.

[23] M. M. Chakravarty, J. Chapman, K. MacKenzie, O. Melkonian, M. Peyton Jones, and P. Wadler, "The extended utxo model," in *Financial Cryptography and Data Security: FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers 24*. Springer, 2020, pp. 525–539.

[24] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*. Springer, 2013, pp. 477–498.

[25] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A2l: Anonymous atomic locks for scalability in payment channel hubs," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1834–1851.

[26] X. Qin, S. Pan, A. Mirzaei, Z. Sui, O. Ersoy, A. Sakzad, M. F. Esgin, J. K. Liu, J. Yu, and T. H. Yuen, "Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts," in *2023 IEEE symposium on security and privacy (SP)*. IEEE, 2023, pp. 2462–2480.

[27] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun, "Misbehavior in bitcoin: A study of double-spending and accountability," *ACM Transactions on Information and System Security (TISSEC)*, vol. 18, no. 1, pp. 1–32, 2015.

[28] I. Bentov, A. Gabizon, and A. Mizrahi, "Cryptocurrencies without proof of work," in *Financial Cryptography and Data Security*. Springer, 2016, pp. 142–157.

[29] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2017, pp.

643–673.
[30] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.
[31] H. Attiya and J. Welch, *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004, vol. 19.
[32] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 967–984.
[33] 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.
[34] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*. Springer, 2007, pp. 61–85.
[35] M. Backes and D. Hofheinz, "How to break and repair a universally composable signature functionality," in *International Conference on Information Security*. Springer, 2004, pp. 61–72.
[36] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.
[37] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[38] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *Annual International Cryptology Conference*. Springer, 2019, pp. 620–649.
[39] "Implementation of pipeswap," 2024, https://github.com/Anqi333/pipeSwap.
[40] R. Han, H. Lin, and J. Yu, "On the optionality and fairness of atomic swaps," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 62–75.
[41] V. Zakhary, D. Agrawal, and A. E. Abbadi, "Atomic commitment across blockchains," *arXiv preprint arXiv:1905.02847*, 2019.
[42] K. Narayanam, V. Ramakrishna, D. Vinayagamurthy, and S. Nishad, "Atomic cross-chain exchanges of shared assets," in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 2022, pp. 148–160.
[43] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 193–210.
[44] M. Borkowski, M. Sigwart, P. Frauenthaler, T. Hukkinen, and S. Schulte, "Dextt: Deterministic cross-blockchain token transfers," *IEEE Access*, vol. 7, pp. 111 030–111 042, 2019.
[45] K. Narayanam, V. Ramakrishna, D. Vinayagamurthy, and S. Nishad, "Generalized htlc for cross-chain swapping of multiple assets with co-ownerships," *arXiv preprint arXiv:2202.12855*, 2022.
[46] A. Kiayias and D. Zindros, "Proof-of-work sidechains," in *Financial Cryptography and Data Security: FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer, 2020, pp. 21–34.
[47] G. Verdian, P. Tasca, C. Paterson, and G. Mondelli, "Quant overledger whitepaper," *Release V0*, vol. 1, p. 31, 2018.
[48] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *the 2019 ACM SIGSAC Conference*, 2019.
[49] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.
[50] J. V. Bulck, D. Oswald, E. Marin, A. Aldoseri, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *the 2019 ACM SIGSAC Conference*, 2019.
[51] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.
[52] 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.
[53] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," *CoRR, abs/1702.05812*, 2017.
[54] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," *ACM*, 2017.

APPENDIX

A. Other related works

Tier Nolan first introduced the concept of "atomic swap" [5]. Its fundamental security *atomicity* states that the swap protocol either ends with success (i.e., the involved coins are exchanged) or failure (i.e., the involved coins are refunded to their original owners) [40, 41].

Essentially, a secure cross-chain swaps protocol between users $\mathcal{P}_0$ and $\mathcal{P}_1$ should fulfill two fundamental functionalities to guarantee the honest user $\mathcal{P}_0$ cannot lose coins (1) if user $\mathcal{P}_1$ has claimed the frozen coins of $\mathcal{P}_0$, $\mathcal{P}_0$ is able to claim the frozen coins of $\mathcal{P}_1$ before $\mathcal{P}_1$ can refund them; and (2) if user $\mathcal{P}_1$ is malicious, $\mathcal{P}_0$ can definitely refund his frozen coins. While *atomicity* is easily realized by the trusted third party, the blockchain community has made significant efforts to achieve the (fully) decentralized cross-chain swaps [41–43]. HTLC-based protocols use the rich scripting languages supported by the underlying blockchains to describe when and how the frozen coins can be unlocked [44, 45]. Subsequently, these protocols have been widely applied and deployed in practice [6–8]. However, they are far from the universal solution and

suffer from the inherent drawbacks of HTLC, including the high execution costs and large transaction sizes. Additionally, since these transactions are easier to distinguish from the standard one that does not include any custom scripts, thus these protocols are in conflict with the blockchains that have already achieved privacy [12].

Recently, the effort LightSwap [17], which is dedicated to introducing a secure atomic swaps protocol that eliminates the need for timeout functionality by one of the participating users, has made it possible for users to run a swaps protocol on a mobile phone. However, it still relies on one of the two involved blockchains supporting timelock functionality and therefore falls short of achieving universality. Universal Atomic Swaps [13] use cryptographic building blocks, specifically the adaptor signature and timed commitment, to present the first fully universal solution.

Furthermore, some literatures [46, 47] have employed a third blockchain as the coordinator, but this approach necessitates that the participating users possess the ability to transfer coins to and from this blockchain. Additionally, the functionality of cross-chain swaps is integrated into a trusted hardware [48], which not only seems unrealistic but also presents serious vulnerabilities [49, 50].

The studies of payment channel networks (PCNs) enable any two users to complete payments even if they do not have a direct payment channel, and have become the most widely deployed solution for realizing blockchain scalability (e.g., lightning network [51]). Similarly, most of the existing PCNs proposals are restricted to the Turing complete scripting language [52–54] thus suffering from the inherent drawbacks of HTLC. Anonymous Multi-Hop Locks [18], lockable signatures [19] and A^2L [25] are recently proposed to construct the scriptless PNCs.

B. Definitions of cryptographic building blocks

Definition 1: (Adaptor Signature) [14] An adaptor signature scheme $\Sigma_{\mathrm{AS}}^{\mathrm{SIG}}$ w.r.t. a hard relation $\mathcal{R}$ and a digital signature scheme $\Sigma_{\mathrm{SIG}} := (\mathrm{KGen}, \mathrm{Sig}, \mathrm{Vf})$ consists of algorithms $\{\mathrm{pSig}, \mathrm{Adapt}, \mathrm{pVf}, \mathrm{Ext}\}$ defined as:
1) $\mathrm{pSig}(\mathrm{sk}, \mathrm{m}, \mathrm{Y}) \to \tilde{\sigma}$: The pre-signing algorithm inputs secret key sk, message $\mathrm{m} \in \{0,1\}^\lambda$ and statement $\mathrm{Y} \in \mathcal{L}_\mathcal{R}$, outputs pre-signature $\tilde{\sigma}$;
2) $\mathrm{pVf}(\mathrm{pk}, \mathrm{m}, \mathrm{Y}, \tilde{\sigma}) \to \mathrm{b}$: The pre-verification algorithm inputs public key pk, message $\mathrm{m} \in \{0,1\}^\lambda$, statement $\mathrm{Y} \in \mathcal{L}_\mathcal{R}$ and pre-signature $\tilde{\sigma}$, outputs a bit $\mathrm{b} \in \{0,1\}$;
3) $\mathrm{Adapt}(\tilde{\sigma}, y) \to \sigma$: The adaptor algorithm inputs pre-signature $\tilde{\sigma}$ and witness y, outputs signature σ;
4) $\mathrm{Ext}(\sigma, \tilde{\sigma}, \mathrm{Y}) \to y$: The extraction algorithm inputs signature σ, pre-signature $\tilde{\sigma}$ and statement $\mathrm{Y} \in \mathcal{L}_\mathcal{R}$, outputs witness y such that $(\mathrm{Y}, y) \in \mathcal{R}$.

Definition 2: (Digital Signature) A digital signature scheme Σ_{SIG} consists of algorithms $(\mathrm{KGen}, \mathrm{Sig}, \mathrm{Vf})$ defined as:
1) $\mathrm{KGen}(1^\lambda) \to (\mathrm{pk}.\mathrm{sk})$: The key generation algorithm inputs security parameter λ and outputs a public-secret key pair $(\mathrm{pk}, \mathrm{sk})$;
2) $\mathrm{Sig}(\mathrm{sk}, \mathrm{m}) \to \sigma$: The signing algorithm inputs secret key sk and a message $\mathrm{m} \in \{0,1\}^\lambda$, outputs a signature σ;
3) $\mathrm{Vf}(\mathrm{pk}, \mathrm{m}, \sigma) \to \mathrm{b}$: The verification algorithm inputs the verification key pk, message m and signature σ, outputs $\mathrm{b} = 1$ if σ is a valid signature of m under public key pk and $\mathrm{b} = 0$ otherwise.

Definition 3: (Hard Relation) A hard relation $\mathcal{R}$ is described as $\mathcal{L}_\mathcal{R} := \{\mathrm{Y} | \exists y, s.t. (\mathrm{Y}, y) \in \mathcal{R}\}$ and satisfies:
1) $\mathcal{R}Gen(1^\lambda) \to (\mathrm{Y}, y)$: The sampling algorithm takes as input security parameter λ and outputs statement-witness pair $(\mathrm{Y}, y) \in \mathcal{R}$;
2) The relation is poly-time decidable;
3) There is no adversary $\mathcal{A}$ with statement Y can output witness y with non-negligible probability.

Definition 4: (Verifiable Timed Dlog) A VTD w.r.t. a group $\mathbb{G}$ with prime order q and generator g consists of four algorithms $(\mathrm{Commit}, \mathrm{Vf}, \mathrm{Open}, \mathrm{ForceOp})$ defined as:
1) $\mathrm{Commit}(x, r, T) \to (\mathrm{C}, \pi)$: The commitment algorithm inputs discrete log $x \in \mathbb{Z}_q^*$, randomness $r \in_\$ \{0,1\}^\lambda$ and timing hardness T, outputs commitment C and proof π;
2) $\mathrm{Vf}(H, \mathrm{C}, \pi) \to \mathrm{b}$: The verification algorithm inputs group element $H := \mathrm{g}^x$, C and π, outputs $\mathrm{b} = 1$ if C is a valid commitment of x with hardness T and $\mathrm{b} = 0$ otherwise;
3) $\mathrm{Open}(\mathrm{C}) \to (x, r)$: The open algorithm inputs commitment C, outputs the committed value x and randomness r;
4) $\mathrm{ForceOp}(\mathrm{C}) \to x$: The force open algorithm inputs commitment C and outputs the committed value x.

C. Security analysis

Theorem 1: (*Atomicity*) Assume $\Sigma_{\mathrm{AS}}^{\mathrm{SIG}}$ is a secure adaptor signature scheme w.r.t. a secure digital signature scheme Σ_{SIG} and a hard dlog relation $\mathcal{R}$; protocols $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$ and $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$ are UC-secure 2PC protocols for jointly computing $\Sigma_{\mathrm{SIG}}.\mathrm{Sig}$ and $\Sigma_{\mathrm{AS}}^{\mathrm{SIG}}.\mathrm{pSig}$; VTD is a secure timed commitment of dlog. Then protocol PipeSwap running in the $(\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{smt}, \mathcal{F}_{clock})$-hybrid world UC-realizes ideal functionality $\mathcal{F}$.

Proof: We now prove that protocol PipeSwap (Fig.6) UC-realizes the cross-chain swaps ideal functionality $\mathcal{F}$ (Fig.8).

To show the indistinguishability between the ideal world and the real world, we construct a simulator $\mathcal{S}$ to simulate the protocol PipeSwap in the real world while interacting with the ideal functionality $\mathcal{F}$. At the beginning, $\mathcal{S}$ corrupts one user of $\{\mathcal{P}_0, \mathcal{P}_1\}$ as $\mathcal{A}$ does. We begin with the real world protocol execution, gradually change the simulation in these hybrids and then we argue about the proximity of neighbouring experiments.
Hybrid $\mathcal{H}_0$: It is the same as the real world protocol execution (Fig.11);
Hybrid $\mathcal{H}_1$: It is the same as the above execution except that the 2PC protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$ in the Swap Setup Phase and Swap Lock Phase to generate signatures is simulated using the 2PC simulators $\mathcal{S}_{2pc,1}$ for the corrupted user (notice that such a simulator exists for a secure 2PC protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$);
Hybrid $\mathcal{H}_2$: It is the same as the above execution except that the 2PC protocol $\Gamma_{\mathrm{AdpSg}}^{\mathrm{SIG}}$ in the Swap Lock Phase to generate pre-signatures is simulated using the 2PC simulators $\mathcal{S}_{2pc,2}$

for the corrupted user;
Hybrid $\mathcal{H}_3$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_1$ and outputs a valid swap transaction $(\mathrm{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)})$ before the simulator initiates swap operation on behalf of $\mathcal{P}_0$, the simulator aborts;
Hybrid $\mathcal{H}_4$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_0$ and outputs a valid swap transaction $(\mathrm{tx}_{swp}^{(0)}, \sigma_{swp}^{(0)})$ before timeout T_1. The simulator outputs $(\mathrm{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)})$ and if $\Sigma_{\mathrm{SIG}}.\mathrm{Vf}(\mathrm{pk}^{(01)}, \mathrm{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)}) \neq 1$, the simulator aborts;
Hybrid $\mathcal{H}_5$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_0$ and initiates the swap operation $(\mathrm{tx}_{swp}^{(0)}, \sigma_{swp}^{*(0)})$ after timeout T_1. The simulator outputs $(\mathrm{tx}_{rfd}^{(1)}, (\check{\sigma}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}))$ if $\Sigma_{\mathrm{SIG}}.\mathrm{Vf}(\widehat{\mathrm{pk}}_{rfd}^{(1)}, \mathrm{tx}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}) \neq 1$ or $\Sigma_{\mathrm{SIG}}.\mathrm{Vf}(\mathrm{pk}^{(10)}, \mathrm{tx}_{rfd}^{(1)}, \check{\sigma}_{rfd}^{(1)}) \neq 1$, the simulator aborts;
Hybrid $\mathcal{H}_6$: It is the same as the above execution except that the adversarial $\mathcal{P}_0$ outputs a valid refund transaction $(\mathrm{tx}_{rfd}^{(0)}, \sigma_{rfd}^{(0)})$ before timeout T_0, the simulator aborts;
Hybrid $\mathcal{H}_7$: It is the same as the above execution except that the adversarial $\mathcal{P}_1$ outputs a valid refund transaction $(\mathrm{tx}_{rfd}^{(1)}, \sigma_{rfd}^{(1)})$ before timeout T_1, the simulator aborts;
Hybrid $\mathcal{H}_8$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_0$ and the simulator obtains $(\mathrm{tx}_{rfd}^{(1)}, \sigma_{rfd}^{(1)})$ after timeout T_1, if $\Sigma_{\mathrm{SIG}}.\mathrm{Vf}(\widehat{\mathrm{pk}}_{rfd}^{(1)}, \mathrm{tx}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}) \neq 1$ or $\Sigma_{\mathrm{SIG}}.\mathrm{Vf}(\mathrm{pk}^{(10)}, \mathrm{tx}_{rfd}^{(1)}, \check{\sigma}_{rfd}^{(1)}) \neq 1$, the simulator aborts;
Hybrid $\mathcal{H}_9$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_1$ and the simulator obtains $(\mathrm{tx}_{rfd}^{(0)}, \sigma_{rfd}^{(0)})$ after timeout T_0, if $\Sigma_{\mathrm{SIG}}.\mathrm{Vf}(\mathrm{pk}^{(01)}, \mathrm{tx}_{rfd}^{(0)}, \sigma_{rfd}^{(0)}) \neq 1$, the simulator aborts.

Simulator $\mathcal{S}$: The simulator $\mathcal{S}$ is defined as the execution in $\mathcal{H}_9$ while interacting with the ideal functionality $\mathcal{F}$. It simulates the view of the adversary and receives messages from the ideal functionality $\mathcal{F}$.

Below we show the indistinguishability between $\mathcal{H}_0$ and $\mathcal{H}_9$. In addition, we use $\approx_c$ to denote computational indistinguishability for a PPT algorithm.
$\mathcal{H}_0 \approx_c \mathcal{H}_1$: The indistinguishability directly follows from the security of 2PC protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$. The security of 2PC protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$ for signature generation guarantees the existence of $\mathcal{S}_{2pc,1}$;
$\mathcal{H}_1 \approx_c \mathcal{H}_2$: The indistinguishability directly follows from the security of 2PC protocol $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$. The security of 2PC protocol $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$ for pre-signature generation guarantees the existence of $\mathcal{S}_{2pc,2}$;
$\mathcal{H}_2 \approx_c \mathcal{H}_3$: The only difference between the hybrids is that in $\mathcal{H}_3$ the simulator aborts, if the adversary corrupts user $\mathcal{P}_1$ and outputs a valid swap transaction $(\mathrm{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)})$ before the simulator initiate a swap on behalf of user $\mathcal{P}_0$;
$\mathcal{H}_3 \approx_c \mathcal{H}_4$: The only difference between the hybrids is that in $\mathcal{H}_4$ the simulator aborts, if the adversary corrupts user $\mathcal{P}_0$ and outputs a valid swap transaction $(\mathrm{tx}_{swp}^{(0)}, \sigma_{swp}^{(0)})$ before timeout T_1, while the simulator cannot obtains its valid swap transaction. The probability of the event triggered in $\mathcal{H}_4$ is negligible;
$\mathcal{H}_4 \approx_c \mathcal{H}_5$: The only difference between the hybrids is that in $\mathcal{H}_5$ the simulator aborts, if the adversary initiates a swap operation after timeout T_1, the simulator cannot post its valid refund transaction. With the security of underlying blockchain and VTD, the probability of the event triggered in $\mathcal{H}_5$ is negligible;
$\mathcal{H}_5 \approx_c \mathcal{H}_6$: The only difference between the hybrids is that in $\mathcal{H}_6$ the simulator aborts, if the adversary $\mathcal{P}_0$ outputs a valid refund transaction before timeout T_0. With the security of VTD, the probability of the event triggered in $\mathcal{H}_6$ is negligible;
$\mathcal{H}_6 \approx_c \mathcal{H}_7$: The only difference between the hybrids is that in $\mathcal{H}_7$ the simulator aborts, if the adversary $\mathcal{P}_1$ outputs a valid refund transaction before timeout T_1. With the security of underlying blockchain and VTD, the probability of the event triggered in $\mathcal{H}_7$ is negligible;
$\mathcal{H}_7 \approx_c \mathcal{H}_8$: The only difference between the hybrids is that in $\mathcal{H}_8$ the simulator aborts, if it cannot post a valid refund transaction after timeout T_1. With the security of underlying blockchain and VTD, the probability of the event triggered in $\mathcal{H}_8$ is negligible;
$\mathcal{H}_8 \approx_c \mathcal{H}_9$: The only difference between the hybrids is that in $\mathcal{H}_9$ the simulator aborts, if it cannot post a valid refund transaction after timeout T_0. With the security of VTD, the probability of the event triggered in $\mathcal{H}_9$ is negligible.

■