Warning! Timeout T Cannot Protect You From Losing Coins

pipeSwap: Forcing the Early Release of a Secret for Atomic Swaps Across All Blockchains

Peifang Ni, Anqi Tian, Jing Xu

ABSTRACT

Atomic cross-chain swap, which allows users to exchange coins securely, is critical functionality to facilitate inter-currency exchange and trading. Although most classic atomic swap 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 path to scriptless cross-chain swap, and it ingeniously delegates scripting functionality to cryptographic lock mechanisms, particularly the adaptor signature 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* is directly broken. Moreover, this attack is easy to carry out and can be naturally generalized to other cross-chain swap protocols as well as the payment channel networks, highlighting a general difficulty in designing universal atomic swap.

We present pipeSwap, a cross-chain swap protocol that satisfies both security and practical universality. To avoid transactions of the same frozen coins being double-claimed to violate the *atomicity* property, pipeSwap proposes a novelly designed paradigm of pipelined coins flow by using *two-hop swap* and *two-hop refund* techniques. pipeSwap achieves *universality* by not relying on any specific script language, aside from the basic ability to verify signatures. Furthermore, we analyze why existing ideal functionality falls short in capturing the *atomicity* property of Universal Atomic Swaps, and define for the first time ideal functionality to guarantee *atomicity*. 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, which demonstrates its high efficiency.

KEYWORDS

Atomic Swaps, Strong Atomicity, Universality, Pipelined Coins Flow, Two-Hop Swap/Refund.

1 INTRODUCTION

With numerous and diverse blockchain systems coexisting today, it is impossible to envision each one evolving in isolation, especially given the explosive development of cryptocurrencies such as Bitcoin [29], Ethereum[46], and Ripple[39]. This proposes an extremely urgent demand of deploying robust currency payments across any blockchain based cryptocurrency. The atomic cross-chain swap protocol [32] is built on top of the underlying blockchains and introduced for securely exchanging coins between two distrusting users, who respectively hold coins in two distinct blockchains. The fundamental security property *atomicity* guarantees that the honest user cannot lose coins. Conventionally, the timeout parameter T, which is predefined specifically for each user, serves as the crux of describing *atomicity.* In slightly more detail, either the honest user obtains its desired coins before its timeout T, or its frozen coins are refunded after timeout T.

A Desideratum for Achieving Atomic Cross-Chain Swap in the Absence of Custom Scripts. Most classic efforts focus on studying the Hash Timelock Contracts (HTLC)-style solutions [10, 11, 19], which use the rich scripting languages supported by underlying blockchains to describe when and how the user can claim the counterparty's locked coins or refund its own locked coins. Unsurprisingly, this design form is incompatible with most existing cryptocurrencies (e.g., Bitcoin [29], Monero [25], Mimblewimble [34], Ripple [39] and Zcash [38]) and far from the universal solution. In addition, it raises a privacy concern due to the using of timelock functionality, which makes the transactions easier to be distinguished from the general ones [1], especially for blockchains having already achieved privacy [38]. Indeed, when introducing a new cryptocurrency or even among the existing ones, relying on specific scripting functionality would lead to fatal obstacles for communications.

Therefore, it is not only of practically relevant, but also theoretically interesting to investigate what are the minimum scripting functionalities necessary to design secure cross-chain swaps. Noticeably, instead of relying on traditional on-chain scripting to denote locked coins and their corresponding unlock conditions, Universal Atomic Swaps [42] make the best use of adaptor signature scheme [3] and verifiable timed discrete logarithm (VTD) [43] to become the closest solution for a universal proposal. Specifically, the witness extractability of adaptor signatures 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 its 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 scriptless cross-chain swap and thus become arguably the best candidate for implementing cryptocurrency exchange.

Is Timeout T Really Secure for Honest User? Nevertheless, the timeout T can potentially violate the *atomicity* property. Herein we present a new attack termed the *double-claiming* attack. It is noteworthy that in the context of Universal Atomic Swaps [42], the predefined timeout T_1

is intentionally designed for user P_1 to securely refund its frozen coins (i.e., user P_1 obtains the full secret key of its frozen address at timeout T_1), but timeout T_1 cannot deprive the right of user P_0 to post its swap transaction (i.e., the witness y is still valid). As a result, the timeout T_1 will become a flashpoint for security issues. For instance, when user P_1 posts a refund transaction after timeout T_1, the malicious P_0 can still release its swap transaction to make P_1's frozen coins double-claiming. Unfortunately, if user P_0's swap transaction is accepted and finally confirmed by the underlying blockchain, honest user P_1 will neither successfully enter into Swap Complete Phase nor prevail in its Swap Timeout Phase, ultimately resulting in the loss of coins. Even worse, the *double-claiming* attack is extremely easy to carry out in all the network models (cf. Section 3 for detailed discussions).

The above issues bring the fundamental challenge in designing atomic cross-chain swaps: the HTLC-style proposals put severe obstacles to achieving universality, whereas Universal Atomic Swaps protocol is susceptible to the *double-claiming* attack mentioned above. This naturally raises a question:

> *"Can we design a cross-chain swap protocol to achieve the best of both security and universality?"*

1.1 Our Contributions

In this work, we contribute to the rigorous understanding of atomic cross-chain swaps and answer the aforementioned question affirmatively by presenting a new protocol called pipeSwap. The contributions are outlined below:

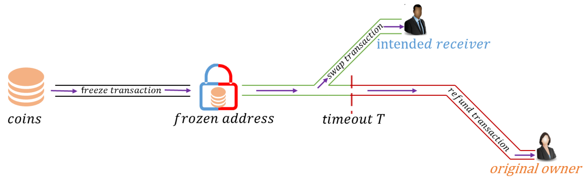


Figure 1: The pipelined life-cycle of swapped coins

- *Double-claiming attack.* We analyze the security of Universal Atomic Swaps [42], and discover a new form of attack called *double-claiming* attack, which can directly break *atomicity* with overwhelming probability, i.e., the protocol will end with the adversarial user both obtaining its counterparty's frozen coins and refunding its own frozen coins. We argue that this attack is easy to carry out and naturally generalizes to other cross-chain swap protocols as well as the payment channel networks [4, 20, 26, 44], since the attack exploits the fact that the frozen coin can be claimed by both its original owner and the intended receiver after its timeout T.
- *The stronger atomicity.* We analyze why the existing security model falls short in capturing the security of Universal Atomic Swaps, and observe that the ideal functionality $\mathcal{F}^{\mathbb{B}}_{swap}$ (Fig.3 in [42]) cannot cope with the condition that both users simultaneously initiate their respective *buy* and *abort* requirements. This motivates the definition of ideal functionality to guarantee stronger *atomicity*. Informally, such an ideal functionality states that each frozen coin can only be claimed by a swap transaction if the valid swap transaction can be generated before the timeout; otherwise, it can only be refunded.
- *pipeSwap.* Inspired by a novel paradigm of pipelined coins flow, we present a new atomic cross-chain swap protocol called pipeSwap. As depicted in Fig.1, each frozen coin is viewed as a drop of water and flows along the one-way arrows. Informally, for each frozen coin, it can only flow to its intended receiver via the green pipe if the swap completes successfully, otherwise, once the procedure enters into Timeout Phase, it definitely flows to its original owner via the red pipe. By this way, if the frozen coin has been claimed by a valid swap transaction, it will no longer continue to flow forward. Instead, after timeout T, the frozen coin will never rewind to its intended receiver. Thus pipeSwap satisfies the definition of stronger *atomicity*. Moreover, pipeSwap is universal, that is, the protocol does not require any specific script language, apart from the basic capability to verify signatures. Additionally, pipeSwap preserves *fungibility*, which means that an observer cannot distinguish a swap transaction from a standard one, owning to the fact that the on-chain transactions in pipeSwap are identical to standard one-to-one transactions. As a byproduct of our approach, the core idea of pipelined coins flow can be leveraged to design secure multi-hop swaps (including the multi-hop payments). The comparisons with prior approaches are shown in TABLE 1.
- *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 the best suitability of our design. In particular, pipeSwap has the running time of less than 1.7 seconds and the communication costs of less than 7 kb. Remarkably, even though pipeSwap provides stronger security protection against *double-claiming* attack, compared to Universal Atomic Swaps [42], it only takes a few more milliseconds for completing the second hop of swap/refund operation.

Table 1: The comparisons with prior approaches

Protocol	**Universality**	**Fungibility**	**DoC† attack resilience**
HTLC	✗	✗	✓
UAS*	✓	✓	✗
pipeSwap	✓	✓	✓

*Universal Atomic Swaps [42], †*double-claiming*.

1.2 Technique Overview

Recall that in a classic α-to-β swaps protocol [42], the user P_0 is given priority to post its swap transaction of coins β before timeout T_1, simultaneously releases a witness y w.r.t. hard relation $\mathcal{R}$ to P_1 for completing its swap operation of coins α. To avoid double-claiming of the same frozen coins to violate the *atomicity*, we resort to different techniques of forcing the earlier release of witness y, i.e., releasing witness y is viewed as a prerequisite for posting swap transaction of 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 the frozen coin before its timeout. Different from Universal Atomic Swaps, 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. Notably, a pre-transaction (i.e., pre-swap or pre-refund transaction) is designed for spending frozen coins ε, and the real spending transaction (i.e., swap or refund transaction) of coins β takes the corresponding pre-transaction as one of its inputs. Such a method forces the users to actively post a pre-transaction instead of waiting until the timeout.

two-hop swap. More importantly, the new freezing structure inspires us to novelly design a *two-hop swap* method of claiming frozen coins β, while frozen coins α can be directly unlocked with witness y. Informally, the puzzle Y ($(Y, y) \in \mathcal{R}$) is inserted in the signature of pre-swap transaction of coins ε instead of the final swap transaction of frozen coins β. *two-hop swap* can prevent user P_1 from losing coins α even if the adversary P_0 posts its swap transaction after timeout T_1, because the witness y has been released before timeout T_1.

two-hop refund. Obviously, it is not enough to guarantee the pipelined coins flow solely with the *two-hop swap* design, if the frozen coins β can be directly refunded by user P_1 after timeout T_1. We further propose the corresponding *two-hop refund* of frozen coins β, where frozen coins ε are firstly refunded by a pre-refund transaction after time $\overline{T}_1$ ($T_1 - \overline{T}_1 > \varphi$, where φ is the confirmation latency of underlying blockchain) and then frozen coins $\beta - \varepsilon$ are spent by the final refund transaction after timeout T_1. Notice that the design of *two-hop refund* can effectively force user P_0 to post its pre-swap transaction before time $\overline{T}_1$, otherwise the malicious delay would lead to frozen coins ε being claimed by a pre-refund transaction and finally coins β being refunded by user P_1 after timeout T_1.

2 BLOCKCHAIN AND CROSS-CHAIN ATOMIC SWAP

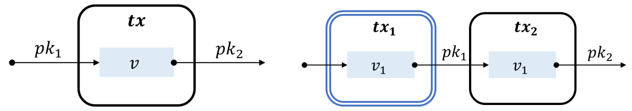
We first recall the formal definition of Unspent Transaction Output (UTXO) model [3], which is adopted by the majority of current blockchains (e.g., Bitcoin [29]), and then take a brief overview of Universal Atomic Swaps [42].

2.1 The UTXO-based Blockchain

Transactions. Under the UTXO model, a transaction is a tuple of the form $(input, output, V, \Omega)$, which transfers coins from $m \geq 1$ inputs $input := \{in_1, \ldots, in_m\}$ to $l \geq 1$ outputs $output := \{op_1, \ldots, op_l\}$. In particular, $V := \{v_1, \cdots, v_m\}$ denotes the value of each *input* and $\Omega := \{\sigma_1, \cdots, \sigma_m\}$ is the witness of spending each *input*. Usually, we use public key pk to denote the input/output address, for example, $tx := (pk_1, pk_2, v, \sigma)$ means transferring coins v in address pk_1 to address pk_2, and σ is the signature of tx that verifies w.r.t. pk_1 and the coins in address pk_2 can only be further spent with signature under pk_2. Additionally, the conditions of spending coins can be some scripts supported by scripting language of the underlying blockchain (i.e., $TimeLock$ and HTLC), but in this paper we focus on the scriptless ones.

We use 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 $\mathcal{T}$ to record the posted transactions and also be viewed as a trusted ledger $\mathcal{L}$ to store all the unspent coins associated with each address pk. In essence, a blockchain is built and maintained by the parties who compete to be elected as the next leader to propose a candidate block, which contains a sequence of transactions. It is extremely important to notice that, in a secure real-world blockchain execution, the leaders prioritize packaging the transactions received first into blocks and, for two conflicting transactions (i.e., spending the same coins) received simultaneously, they randomly select one of the two transactions as a valid transaction. What's more, we strictly separate the *parties* who are responsible for the secure execution of the underlying blockchain from the *users* who only participate in the cross-chain swap protocol supported by the underlying blockchains. Therefore, the (honest) leaders do not care about the story behind each transaction and, in their views, all the valid transactions are treated equally. Unsurprisingly, it is reasonable that, the valid transaction tx' in a pair of conflicting transactions $\{tx, tx'\}$ is finally confirmed even if it is maliciously generated.



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

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

Figure 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) determines that we must clearly distinguish between the notions of *valid transaction* and *confirmed transaction*. Traditionally, bulletin board $\mathcal{T}$ records the set of valid transactions that could be finally confirmed but they are not yet (i.e., the input addresses have enough balance and they are signed correctly), while the ledger $\mathcal{L}$ stores all the finally confirmed transactions which can be further spent by a new transaction. Without loss of generality, we let bulletin board $\mathcal{T}$ sequentially record all the valid transactions, especially for the conflicting transactions, it only accepts the one that arrives at earlier or randomly selects one in the case of simultaneous arrival. Obviously, a valid transaction $tx \in \mathcal{T}$ can be confirmed finally (i.e., $tx \in \mathcal{L}$) if it has been in $\mathcal{T}$ for time φ.

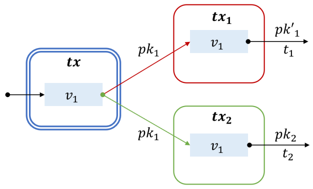


Figure 3: The green transaction tx_2 is honestly signed w.r.t. pk_1 and received by $\mathcal{T}$ at time t_2, while the red transaction tx_1, which tries to double-claim transaction tx, is maliciously signed w.r.t. pk_1 and received by $\mathcal{T}$ at time t_1.

For presentation simplicity, we use doubled blue edge rectangle and single edge rectangle to present the confirmed transaction and valid transaction respectively (see Fig.2(b)). After a clear understanding of transaction processing mechanism of the underlying blockchain, we have to pay more attention to the following simple but practical scenario (see Fig.3). Since the one who holds secret key sk_1 w.r.t. public key pk_1 can sign any transactions at its will, thus we cannot prevent the adversarial payer U_1 from generating two valid transactions tx_1, tx_2, where tx_2 pays for the intended receiver U_2 and tx_1 pays back to itself (i.e., users U_1 and U_2 own addresses pk_1, pk'_1 and pk_2, respectively). Fortunately, if $t_2 < t_1$ then tx_2 defeating tx_1 can be accepted by $\mathcal{T}$ and as a result the receiver U_2 obtains its deserved coins after time φ. However, if $t_1 = t_2$ then, with probability 50%, the receiver U_2 will lose coins as tx_1 being accepted by $\mathcal{T}$. Even worse, when the network delay is under adversarial control (e.g., the Δ-synchronous communication network [22]), the malicious transaction tx_1 would compete against tx_2 with a landslide.

Leveraging the above observation, we should be wary of some time points that can cause the same coins to be doubly claimed by different users. Especially for realizing atomic coins transfer/swap in a decentralized manner, only relying on a valid transaction rather than a confirmed one would lead to fatal vulnerabilities (see Section 3).

2.2 Cross-Chain Atomic Swap

Generally, a cross-chain swap protocol enables two distrusting users P_0 and P_1, who respectively own coins α and β in two distinct blockchains $\mathbb{B}_0$ and $\mathbb{B}_1$, to exchange coins securely. The fundamental security property of cross-chain swap protocol is *atomicity*: the honest users cannot lose coins. Specifically, honest user P_0 definitely gets coins β in $\mathbb{B}_1$ if swap succeeds. Otherwise, P_0 enters into its Timeout Phase and successfully unlocks frozen coins α.

Now we recall the design of Universal Atomic Swaps [42] (see Fig.4). We use hereunder notations: (1) item with superscript $\{(i\ 1-i) | i \in \{0,1\}\}$ is involved in the payment from user P_i to P_{1-i}; (2) item with subscript $\in \{frz, swp, rfd\}$ respectively refers to the freeze, swap and refund operation. It consists of four phases described as follows:

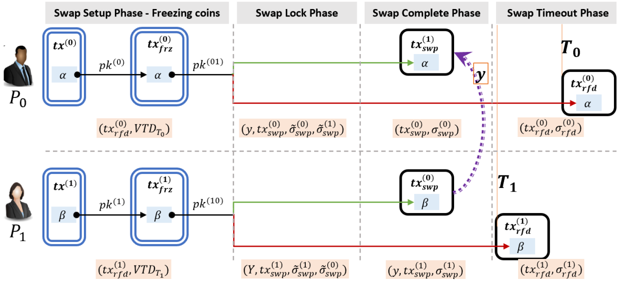


Figure 4: Universal Atomic Swaps: the execution flow of users P_0 and P_1 in an α-to-β swap. The orange rectangle stores the messages received by each user in the phase.

<u>*Swap Setup Phase-Freezing coins*</u>: Users P_0 and P_1 jointly generate the frozen addresses $pk^{(01)}$ and $pk^{(10)}$, where the corresponding secret keys $sk^{(01)} := sk_0^{(01)} \oplus sk_1^{(01)}$ and $sk^{(10)} := sk_0^{(10)} \oplus sk_1^{(10)}$ are shared between them, and respectively compute the timed commitments (Def.4) $VTD_{T_1} := (C^{(1)}, \pi^{(1)})$ and $VTD_{T_0} := (C^{(0)}, \pi^{(0)})$ of shares $sk_0^{(10)}$ and $sk_1^{(01)}$ (Note that, after timeout T_i, user P_i can get secret key $sk^{(i1-i)}$). After the above is successful, user P_i transfers coins from address $pk^{(i)}$ to $pk^{(i1-i)}$ via frozen transaction $tx^{(i)}_{frz}$;

<u>*Swap Lock Phase*</u>: Using adaptor signature w.r.t. hard relation $(Y, y) \in \mathcal{R}$ (Def.1) selected by user P_0, users P_0 and P_1 jointly generate pre-signatures $\tilde{\sigma}^{(1)}_{swp}$ of swap transaction $tx^{(1)}_{swp}$ and $\tilde{\sigma}^{(0)}_{swp}$ of swap transaction $tx^{(0)}_{swp}$ in sequence. Notice that, from now on, user P_0 with witness y can generate valid swap transaction $tx^{(0)}_{swp}$ at any time;

<u>*Swap Complete Phase*</u>: If user P_0 actively posts swap transaction $tx^{(0)}_{swp}$ before timeout T_1, P_1 can extract y (i.e., the purple dotted arrow) to generate signature $\sigma^{(1)}_{swp}$ of swap transaction $tx^{(1)}_{swp}$. Thus the users successfully swap coins;

<u>*Swap Timeout Phase*</u>: If Swap Complete Phase fails, user P_1 enters into its Swap Timeout Phase after timeout T_1 and posts refund transaction $tx^{(1)}_{rfd}$ with secret key $sk^{(10)}$. Correspondingly, after timeout T_0, user P_0 posts refund transaction $tx^{(0)}_{rfd}$. Therefore, the frozen coins are respectively refunded to their original owners.

<u>Security Analysis</u>. We summarize security analysis of Universal Atomic Swaps and defer detailed proofs to [42]:

- *Successful Swap*: If user P_0 honestly posts swap transaction $tx^{(0)}_{swp}$ before timeout T_1, user P_1 can extract y to generate its swap transaction $tx^{(1)}_{swp}$ successfully;
- *Failed Swap*: If user P_0 fails to post swap transaction $tx^{(0)}_{swp}$ before timeout T_1, user P_1 fails in Swap Complete Phase and enters into Swap Timeout Phase to refund its frozen coins via posting transaction $tx^{(1)}_{rfd}$. Similarly, user P_0 can refund its frozen coins after timeout T_0.

3 THE DOUBLE-CLAIMING ATTACK

What *Double-Claiming* Attack Is. In a cross-chain swap protocol, the malicious user (it is user P_0 or P_1) can deviate arbitrarily from the swap protocol. In addition, the underlying blockchain network may be under the adversarial control to delay or reorder messages, and it prioritizes transactions that arrive first (cf. Fig.7). The *double-claiming* attack refers to a situation where a malicious user attempts to create a double-claimed state for the frozen coins (i.e., the frozen coins are spent simultaneously by their original owner and the intended receiver), thereby obtaining the offered coins from its counterparty while refusing to transfer its own coins. This directly violates the *atomicity* property of the cross-chain swap protocol.

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

Reason 1: The balance security of transaction. Independent from the inner workings in cross-chain swap protocols, total balances of all the addresses in underlying blockchain are unchanged. Specifically, no new coins are generated causelessly and any coins can only be equivalently transferred to some new addresses. Furthermore, the scriptless nature of cross-chain swap determines that the verification of a transaction relies only on the balance and signature. Thus, in the view of 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.

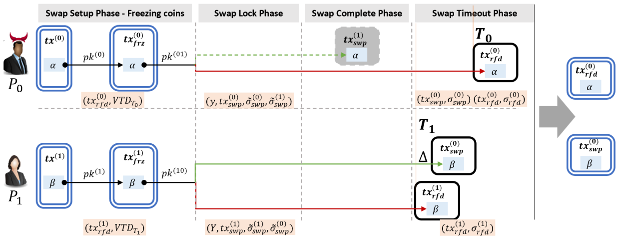
Reason 2: After timeout T, the frozen coins can be spent by both its original owner and intended receiver. We recall that the frozen coins can only be refunded after the predefined timeout T, while the intended receiver is able to claim these coins both before and after timeout T. At first glance, this seems reasonable that the timeout T provides enough time for the intended receiver to swap and guarantees the coins can be refunded in the case of failure. However, there is no mechanism to cancel the intended receiver's ability to claim frozen coins after timeout T (i.e., signing its swap transaction), which is the source of *double-claiming* attack.

How Easy It Is to Conduct This Attack. Note that the *double-claiming* attack is completely different from the *double-spending* attack [21], where the former enables the malicious user to prevent an honest transaction from being confirmed, while the latter enables the malicious miner who is involved in the maintenance of underlying blockchain to confirm the both spending transactions of one coin. Additionally, the *double-spending* attack requires the attacker (i.e., the malicious miner) to hold and consume enough resources (e.g., computational power [29] or stakes [6]), while the *double-claiming* attack is crazy-cheap and only requires the attacker (i.e., the malicious user) to have the ability of signing transactions (i.e., holding the signing secret key). In particular, the *double-claiming* attack can work in all the network models of the underlying blockchains (e.g., the strong synchrony [17], Δ-synchrony [33], partial synchrony [15] and asynchrony [2]). Even worse, with the weakening of the underlying blockchain network (i.e., from strong synchrony to asynchrony), the *double-claiming* attack can succeed with a higher probability, which can be detailed as follows.

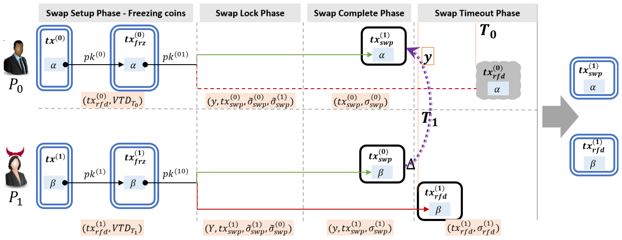
As depicted in Fig.5, *double-claiming* attack can work in Universal Atomic Swaps [42], where the underlying blockchain network is strong synchrony (i.e., there is no message propagation delay and $\Delta = 0$), in two manners.

User P_0 is malicious (Fig.5(a)). First, users P_0 and P_1 initiate an α-to-β swap via successfully freezing their respective coins, and then the adversary P_0 gets off-line until timeout T_1. In the Swap Timeout Phase, user P_1 posts refund transaction $tx^{(1)}_{rfd}$, and simultaneously the adversary P_0 goes re-online and posts swap transaction $tx^{(0)}_{swp}$. At timeout T_0, the adversary P_0 enters into Swap Timeout Phase to post refund transaction $tx^{(0)}_{rfd}$. Due to the fact that only one of transactions $tx^{(0)}_{swp}$ and $tx^{(1)}_{rfd}$ can be finally confirmed by the corresponding underlying blockchain. Accordingly, with probability close to 1/2, the malicious transactions $tx^{(0)}_{swp}$ and $tx^{(0)}_{rfd}$ are both confirmed. As a result, the adversary P_0 harvests double assets and the *atomicity* property is broken.

User P_1 is malicious (Fig.5(b)). Let us continue to recall some details of Universal Atomic Swaps. If user P_0 honestly posts swap transaction $tx^{(0)}_{swp}$ before timeout T_1, this swap protocol should be successful such that swap transactions $tx^{(0)}_{swp}$ and $tx^{(1)}_{swp}$ are finally confirmed by the underlying



(a) When the malicious user P_0 successfully initiates *double-claiming* attack, user P_1 fails to enter into its Swap Complete Phase. And the finally confirmed transactions are $tx^{(0)}_{rfd}$ and $tx^{(0)}_{swp}$.



(b) When the malicious user P_1 successfully initiates *double-claiming* attack, user P_0 fails to enter into its Swap Timeout Phase. And the finally confirmed transactions are $tx^{(1)}_{rfd}$ and $tx^{(1)}_{swp}$.

Figure 5: The *double-claiming* attack works in Universal Atomic Swaps [42].

blockchain. However, if user P_0 honestly posts swap transaction $tx_{swp}^{(0)}$ almost near timeout T_1, the adversary P_1 can issue its swap transaction $tx_{swp}^{(1)}$ immediately and try to enter into its Swap Timeout Phase by posting refund transaction $tx_{rfd}^{(1)}$. Similarly, due to the fact that only one of transactions $tx_{swp}^{(0)}$ and $tx_{rfd}^{(1)}$ can be finally confirmed by the corresponding underlying blockchain. Accordingly, with probability close to 1/2, the malicious transactions $tx_{swp}^{(1)}$ and $tx_{rfd}^{(1)}$ are both confirmed. As a result, the malicious user P_1 harvests double assets and the *atomicity* property is broken.

Furthermore, the strong network synchrony assumption is impractical for a large-scale distributed system such as Bitcoin [37]. When considering the Δ-synchronous blockchain network [33] that the network delay is under adversarial control and message is delivered within a known delay Δ, we note that the *double-claiming* attack can work easier. Generally, the attack strategies are the same as above. For malicious user P_0 (Fig.5(a)), as long as it releases swap transaction $tx_{swp}^{(0)}$ within time Δ after refund transaction $tx_{rfd}^{(1)}$ being posted, then transaction $tx_{swp}^{(0)}$ can defeat $tx_{rfd}^{(1)}$ and user P_0 harvests double assets (i.e., by confirming transactions $tx_{swp}^{(0)}$ and $tx_{rfd}^{(0)}$) with an absolute advantage. Correspondingly, for malicious user P_1 (Fig.5(b)), once user P_0 honestly posts swap transaction $tx_{swp}^{(0)}$ between time $T_1 - \Delta$ and T_1, the adversary P_1 can post its swap transaction $tx_{swp}^{(1)}$ immediately and then posts a competitive refund transaction $tx_{rfd}^{(1)}$ after timeout T_1 via delaying the transaction $tx_{swp}^{(0)}$. As a result, the malicious user P_1 harvests double assets (i.e., by confirming transactions $tx_{swp}^{(1)}$ and $tx_{rfd}^{(1)}$) with an absolute advantage and the *atomicity* property is broken. Therefore, the weaker but realistic blockchain network can be exploited to improve the successful probability of *double-claiming* attack.

The Generality of This Attack. It should be emphasized that *double-claiming* attack is not specifically tailored to Universal Atomic Swaps, but generally applies to the scriptless (or timelock scripts only) cross-chain swaps and multi-hop payments [4, 20, 26, 44] that predefine a limited lifespan T to realize the payment expire. In these schemes, the main vulnerability leading to *double-claiming* attack is that, after the timeout T, the two competing users simultaneously hold the ability to spend the same frozen coins.

4 OUR SOLUTION IN A NUTSHELL

Recall the core root of *double-claiming* attack is that the frozen coins β can be claimed by both users after timeout T_1. Therefore, our straightforward solution is to realize the *pipelined coins flow* for frozen coins β, that is, the corresponding swap transaction $tx_{swp}^{(0)}$ and refund transaction $tx_{rfd}^{(1)}$ cannot be both valid. As depicted in Fig.6, the key idea is how to force the early release of witness y, such that a valid transaction (i.e., either $tx_{swp}^{(0)}$ or $tx_{rfd}^{(1)}$) can be pre-determined at timeout T_1. Specifically, we introduce a "two-hop completion" method, called *two-hop swap* and *two-hop refund*, where *two-hop swap* forces user P_0 to post the pre-swap transaction $\overline{tx}_{swp}^{(0)}$ φ time before posting a valid swap transaction $tx_{swp}^{(0)}$, and the corresponding *two-hop refund* forces user P_1 to post the pre-refund transaction $\overline{tx}_{rfd}^{(1)}$ (it is locked until time $\overline{T}_1 \leq T_1 - \varphi$) before refunding its frozen coins by transaction $tx_{rfd}^{(1)}$, which further forces user P_0 actively posting $\overline{tx}_{swp}^{(0)}$ before time $\overline{T}_1$, otherwise it cannot generate a valid swap transaction $tx_{swp}^{(0)}$ before timeout T_1. As a result, if pre-swap transaction $\overline{tx}_{swp}^{(0)}$ is finally confirmed before timeout T_1, coins β can only be swapped by user P_0; otherwise, coins β can only be refunded by user P_1 after timeout T_1. Now we walk through how we realize the "two-hop completion".

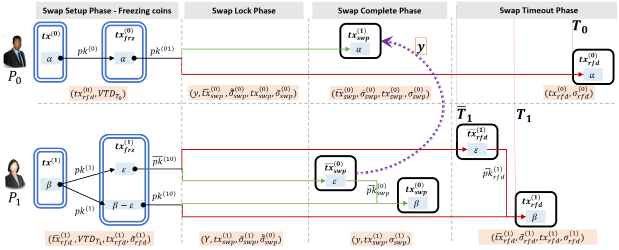


Figure 6: pipeSwap: the execution flow of users P_0 and P_1 in an α-to-β swap. The orange rectangle stores messages received by each user in this phase and the pre-transaction items are denoted with overline. Time parameters are set such that: $T_1 \geq \overline{T}_1 + \varphi, T_0 > T_1$ (the parameter φ is confirmation delay of the underlying blockchain). The gap φ between time $\overline{T}_1$ and T_1 ensures that one transaction in $\{\overline{tx}_{swp}^{(0)}, \overline{tx}_{rfd}^{(0)}\}$ will be finally confirmed before timeout T_1 and thus determines the final flow of coins β.

<ins>*First ingredient: splitting frozen coins β into two parts.*</ins> To ensure the flow direction of frozen coins β and prevent potential risk of malicious user P_0 or P_1, the frozen coins β are stored in two distinct outputs with values ε ($\varepsilon > 0$ is arbitrarily small, i.e., $\varepsilon \to 0$) and $\beta - \varepsilon$ respectively. Specifically, coins ε, $\beta - \varepsilon$ can only be further spent with respective secret keys $\overline{sk}^{(10)}$ and $sk^{(10)}$, which are shared between users P_0 and P_1.

<ins>*Second ingredient: two-hop swap.*</ins> To prevent the adversary P_0 from suddenly releasing its swap transaction $tx_{swp}^{(0)}$ near timeout T_1, a new swap method called *two-hop swap* is proposed. Specifically, user P_0 can generate a valid swap transaction $tx_{swp}^{(0)}$ only when its pre-swap transaction $\overline{tx}_{swp}^{(0)}$ has been finally confirmed by the underlying blockchain. In particular, the transaction $\overline{tx}_{swp}^{(0)}$ is jointly pre-signed by both users with respective key shares (i.e., $\overline{sk}_0^{(10)}$, $\overline{sk}_1^{(10)}$) and puzzle Y, while swap transaction $tx_{swp}^{(0)}$ takes $\overline{tx}_{swp}^{(0)}$ as one of its inputs is jointly signed by both users with respective key shares $sk_0^{(10)}$ and $sk_1^{(10)}$, where statement-witness pair $(Y, y) \in \mathcal{R}$ is selected by user P_0. That is, swap transaction $tx_{swp}^{(0)}$ is valid implying that user P_0 has posted the pre-swap

transaction $\overline{tx}_{swp}^{(0)}$ at least φ time ago, where φ is the confirmation latency of underlying blockchain. Since the witness y has been released by the posted pre-swap transaction $\overline{tx}_{swp}^{(0)}$, user P_1 can generate its valid swap transaction $tx_{swp}^{(1)}$ at least φ time earlier than user P_0.

Third ingredient: two-hop refund. To pre-determine the valid transaction among swap transaction $tx_{swp}^{(0)}$ and refund transaction $tx_{rfd}^{(1)}$ at timeout T_1, we require witness y to be released before time $\overline{T}_1 := T_1 - \varphi$, that is, user P_0 should post pre-swap transaction $\overline{tx}_{swp}^{(0)}$ before time $\overline{T}_1$. Accordingly, a new refund method called *two-hop refund* is proposed, where user P_1 can refund its frozen coins by transaction $tx_{rfd}^{(1)}$ after timeout T_1 only when its pre-refund transaction $\overline{tx}_{rfd}^{(1)}$ has been finally confirmed by the underlying blockchain, and the pre-refund transaction $\overline{tx}_{rfd}^{(1)}$ is locked until time $\overline{T}_1$. This means if pre-swap transaction $\overline{tx}_{swp}^{(0)}$ is confirmed before timeout T_1, it is impossible to generate a valid refund transaction $tx_{rfd}^{(1)}$ even after timeout T_1; otherwise, if pre-refund transaction $\overline{tx}_{rfd}^{(1)}$ is confirmed before timeout T_1, there is no valid swap transaction $tx_{swp}^{(0)}$ at any time. Nevertheless, there is a subtle issue remained. In particular, if user P_0 posts pre-swap transaction $\overline{tx}_{swp}^{(0)}$ almost near time $\overline{T}_1$, then the adversary P_1 can initiate *double-claiming* attack via posting pre-refund transaction $\overline{tx}_{rfd}^{(1)}$ immediately and making transactions $\overline{tx}_{swp}^{(1)}$ and $\overline{tx}_{rfd}^{(1)}$ both valid, due to the fact that the adversary can always delay the delivery of honest messages within time Δ. To solve the issue, we just let the pre-swap transaction $\overline{tx}_{swp}^{(0)}$ be posted at least Δ time before time $\overline{T}_1$.

Clearly, the final flow direction of frozen coins β can be determined at timeout T_1. We now give an intuition that pipeSwap satisfies *atomicity*:

- *Successful Swap:* If user P_0 is honest, pre-swap transaction $\overline{tx}_{swp}^{(0)}$ posted before time $\overline{T}_1 - \Delta$ will be finally confirmed, and then swap transactions $tx_{swp}^{(0)}$ and $tx_{swp}^{(1)}$ could be validated before timeout T_1. Additionally, no valid refund transaction $tx_{rfd}^{(1)}$ exists even after timeout T_1. As a result, each user obtains its desired exchanged coins;
- *Failed Swap:* If user P_0 is malicious and user P_1 enters into its Swap Timeout Phase after timeout T_1, i.e., user P_1 cannot generate valid swap transaction $tx_{swp}^{(1)}$ before timeout T_1, then refund transaction $tx_{rfd}^{(1)}$ will be valid after timeout T_1 and no valid swap transaction $tx_{swp}^{(0)}$ exists at any time. As a result, user P_1 can successfully refund its frozen coins.

5 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 ideal functionality.

5.1 Modeling the System and Threats

We model security of cross-chain swaps in the Universal Composability (UC) model [12] and deploy the version with a global setup (GUC) [13]. We define the cross-chain swap model over users P_0 and P_1, and take the underlying blockchains $\mathbb{B} =: \{\mathbb{B}_0, \mathbb{B}_1\}$ as global ideal functionalities $\mathcal{F}_{\mathbb{B}} := \{\mathcal{F}_{\mathbb{B}_0}, \mathcal{F}_{\mathbb{B}_1}\}$ with the confirmation delay time φ (Fig.7).

The User. There are two designated users P_0 and P_1 to participate in the swap protocol. The malicious user (it is P_0 or P_1) is determined before the protocol starts, who can deviate arbitrarily from the swap protocol (e.g., delaying the posting of its transaction). There is a secure message transmission channel between users modeled by the ideal functionality $\mathcal{F}_{smt}$ [12].

Ideal Functionality $\mathcal{F}_{\mathbb{B}}(\Delta, \varphi)$

1. **Initialization**: upon receiving the address-balance pair $(pk, v) \hookleftarrow \mathcal{Z}$, set $\mathcal{L} := \{(pk_1, v_1), \cdots, (pk_\ell, v_\ell)\} \in \mathbb{R}_{\geq 0}^{2\ell}$, store and send $\mathcal{L} \hookrightarrow \mathcal{S}$.
2. **Posting transaction**: upon receiving $\mathrm{Post}(tx, t) \hookleftarrow \mathcal{Z}$, send $(post, tx, t) \hookrightarrow \mathcal{S}$ if $\mathrm{Valid}(tx) = 1$.
   - Upon receiving $(post, tx, t') \hookleftarrow \mathcal{S}$, if $t' - t \leq \Delta$, then set $t := t'$; otherwise, set $t := t + \Delta$. Update list $list := list \cup (tx, t)$;
   - For conflicting transactions (tx_0, t_0), $(tx_1, t_1) \in list$, if $t_0 < t_1$, then remove (tx_1, t_1) from $list$; else, if $t_0 = t_1$, then randomly select $b \in \{0,1\}$ and remove (tx_b, t_b) from $list$; otherwise, remove (tx_0, t_0) from $list$;
   - For $(tx, t) \in list$, update $\mathcal{T} := \mathcal{T} \cup (tx, t)$ at time t.
3. **Confirming transaction**: upon receiving $\mathrm{Confirm}(tx) \overset{t''}{\hookleftarrow} \mathcal{Z}$, if $\exists (tx, t) \in \mathcal{T}$ and $t'' - t \geq \varphi$, then update $\mathcal{L}$ as $tx.pk_{in}.balance := tx.pk_{in}.balance - v$ and $tx.pk_{op}.balance := tx.pk_{op}.balance + v$; otherwise, abort.

Figure 7: The blockchain ideal functionality $\mathcal{F}_{\mathbb{B}}(\Delta, \varphi)$

Δ-Synchronous Network. We assume the network of underlying blockchains is Δ-synchronous [33], i.e., the network delay is under adversarial control, up to a known delay upper bound Δ. The adversary $\mathcal{A}$ of the underlying blockchain can see the posted honest message but cannot modify or drop it.

The Blockchain Ideal Functionality. We take the underlying blockchain $\mathbb{B}$ ($\mathbb{B}_0$ or $\mathbb{B}_1$) involved in the cross-chain swap protocol as a global ideal functionality (just as in [3, 42]) with confirmation delay time φ that records the balance of each address (i.e., ledger $\mathcal{L}$) and maintains a trusted append-only bulletin board $\mathcal{T}$, denoted as $\mathcal{F}_{\mathbb{B}}(\Delta, \varphi)$. More precisely, functionality $\mathcal{F}_{\mathbb{B}}$ offers interface $\mathrm{Valid}(tx)$ to determine the validity of a transaction tx (i.e., checking $inputs \in \mathcal{L}$ have enough balance and they are signed correctly), uses interface $\mathrm{Post}(tx, t)$ to add valid transaction tx to bulletin board $\mathcal{T}$ at time $t' \leq t + \Delta$, where time t' is determined by the simulator $\mathcal{S}$. We emphasize that $\mathcal{T}$ always prefers to accept the earlier arrived transaction, i.e., upon receiving $\mathrm{Post}(tx_0, t_0)$ and

Post(tx_1, t_1) for posting conflicting transactions tx_0, tx_1, $\mathcal{T}$ will accept tx_0 if $t'_0 < t'_1$ or randomly select one of $\{tx_0, tx_1\}$ if $t'_0 = t'_1$. Additionally, functionality $\mathcal{F}_{\mathbb{B}}$ confirms transaction tx via interface Confirm(tx) (e.g., if tx has been recorded in $\mathcal{T}$ for time φ, it updates $\mathcal{L}$ via removing coins from input address pk_{in} to output address pk_{op}). See Fig.7 for the details.

5.2 Ideal Functionality $\mathcal{F}$ of Cross-Chain Swap

When taking a careful analysis of the ideal functionality $\mathcal{F}^{\mathbb{B}}_{swap}$ for fair swap of coins (Fig.3 in [42]), we have a fatal observation that $\mathcal{F}^{\mathbb{B}}_{swap}$ cannot cope with the condition of the users U_0 and U_1 simultaneously initiating their respective *buy* and *abort* requirements, which further confirms the *double-claiming* attack in Universal Atomic Swaps.

(A) Swap Setup Phase - Freezing Coins

1. Upon receiving $(frz, id, pk^{(0)}, sk^{(0)}) \overset{t}{\hookleftarrow} P_0$, invoke subroutine Freeze$(id, pk^{(0)}, sk^{(0)}, \alpha, pk^{(0)}_{\mathcal{F}}, T_0)$; upon receiving $(Confirmed, id, ok) \overset{t_1 \le t+\Delta+\varphi}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_0}$, send $(frz, id, ok) \overset{t_1}{\hookrightarrow} P_0$;

2. Upon receiving $(frz, id, pk^{(1)}, sk^{(1)}) \overset{t}{\hookleftarrow} P_1$, invoke subroutine Freeze$(id, pk^{(1)}, sk^{(1)}, \beta, pk^{(1)}_{\mathcal{F}}, T_1)$; upon receiving $(Confirmed, id, ok) \overset{t_1 \le t+\Delta+\varphi}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_1}$, send $(frz, id, ok) \overset{t_1}{\hookrightarrow} P_1$;

3. After the above steps are successful, send $(Setup, id, ok) \hookrightarrow P_i$ $(i \in \{0,1\})$ and proceed to procedure (B); otherwise, proceed to procedure (C).

(B) Swap Complete Phase

1. Upon receiving $(swp, id, pk^{(0)}_{swp}) \overset{t'}{\hookleftarrow} P_0$, do the following:

(1) if $t' < T_1$, set $b^{(0)} = 0$ and invoke subroutine Transfer$(id, pk^{(1)}_{\mathcal{F}}, sk^{(1)}_{\mathcal{F}}, \beta, pk^{(0)}_{swp}, t')$; upon receiving $(Post, id, ok) \overset{t'_1 \le t'+\Delta}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_1}$, send $(swp, id, ok) \overset{t'_1}{\hookrightarrow} P_0$;

(2) if $t' = T_1$, set $b^{(0)} \in_{\$} \{0,1\}$ and if $b^{(0)} = 0$, invoke subroutine Transfer$(id, pk^{(1)}_{\mathcal{F}}, sk^{(1)}_{\mathcal{F}}, \beta, pk^{(0)}_{swp}, t')$; upon receiving $(Post, id, ok) \overset{t'_1 \le t'+\Delta}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_1}$, send $(swp, id, ok) \overset{t'_1}{\hookrightarrow} P_0$;

(3) otherwise, abort.

2. Upon receiving $(swp, id, pk^{(1)}_{swp}) \overset{t'}{\hookleftarrow} P_1$, do the following:

(1) if $b^{(0)} \in \{0,1\}$, set $b^{(1)} = 1$ and invoke subroutine Transfer$(id, pk^{(0)}_{\mathcal{F}}, sk^{(0)}_{\mathcal{F}}, \alpha, pk^{(1)}_{swp}, t')$; upon $(Post, id, ok) \overset{t'_1 \le t'+\Delta}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_0}$, send $(swp, id, ok) \overset{t'_1}{\hookrightarrow} P_0$;

(2) otherwise, abort.

(C) Swap Timeout Phase

1. Upon receiving $(rfd, id, pk^{(i)}_{rfd}) \overset{t''}{\hookleftarrow} P_i$ $(i \in \{0,1\})$, do the following:

(1) if $(t'' \ge T_i) \wedge (b^{(1-i)} \neq 1 - i)$, invoke subroutine Unfreeze$(id, pk^{(i)}_{\mathcal{F}}, sk^{(i)}_{\mathcal{F}}, \alpha/\beta, pk^{(i)}_{rfd}, t'')$; upon receiving $(Post, id, ok) \overset{t''_1 \le t''+\Delta}{\hookleftarrow} \mathcal{F}_{\mathbb{B}_i}$, send $(rfd, id, ok) \overset{t''_1}{\hookrightarrow} P_i$;

(2) otherwise, abort.

Figure 8: The ideal functionality $\mathcal{F}$

To model security in our setting more comprehensively, we require that the ideal functionality $\mathcal{F}$ guarantees stronger *atomicity*: either both users interested in the swap successfully swap their coins, or the swap fails and honest user refunds its coins and the malicious user may lose coins for its malicious manners. Accordingly, we allow ideal functionality $\mathcal{F}$ to take actions according to the timeout T of each participating user.

Formally, ideal functionality $\mathcal{F}$ (see Fig.8 and Fig.9) communicates with users P_0, P_1, the environment $\mathcal{Z}$, the simulator $\mathcal{S}$, and the underlying blockchain functionalities $\mathcal{F}_{\mathbb{B}_0}$ and $\mathcal{F}_{\mathbb{B}_1}$. It consists of three procedures and each is triggered by one message sent by user P_i $(i \in \{0,1\})$, including its request and the session id.

(A) Swap Setup Phase-Freezing Coins: Users P_0 and P_1 initiate an α-to-β swap with their respective *freeze* messages $(frz, id, pk^{(0)}, sk^{(0)})$ and $(frz, id, pk^{(1)}, sk^{(1)})$, which specifies that the coins in addresses $pk^{(0)}$ (owned by user P_0 and can be spent with secret key $sk^{(0)}$) and $pk^{(1)}$ (owned by user P_1 and can be spent with secret key $sk^{(1)}$) are to be swapped. Ideal functionality $\mathcal{F}$ calls subroutine Freeze$(id, pk^{(i)}, sk^{(i)}, pk^{(i)}_{\mathcal{F}}, T_i)$ to transfer coins in address $pk^{(i)}$ to a specific address $pk^{(i)}_{\mathcal{F}}$ controlled by $\mathcal{F}$ until timeout T_i, where $T_0 > T_1$;

(B) Swap Complete Phase: User P_0 sends its *swap* message $(swp, id, pk^{(0)}_{swp})$ at time t'. If $t' < T_1$, $\mathcal{F}$ transfers coins from address $pk^{(1)}_{\mathcal{F}}$ to $pk^{(0)}_{swp}$ (controlled by user P_0) and sets $b^{(0)} = 0$ to indicate that user P_0 has successfully finished *swap* operation. If $t' = T_1$ (it implies that user P_0 is trying to initiate *double-claiming* attack), $\mathcal{F}$ randomly determines whether user P_0 completing *swap* operation (i.e., $b^{(0)} \in_{\$} \{0,1\}$). Otherwise, if $t' > T_1$, user P_0 fails in *swap* phase (i.e., $b^{(0)} = \perp$). While user P_1 can request *swap* only when user P_0 has initiated its *swap* operation (i.e., $b^{(0)} = 0/1$);

(C) Swap Timeout Phase: The frozen coins are released to the original owners after the respective timeout T_1 and T_0.

//Subroutine **Freeze**

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

//Subroutine **Transfer**

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

//Subroutine **Unfreeze**

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

Figure 9: The subroutines

Remark 1. *Careful readers might notice that the functionality $\mathcal{F}$ finishes* Swap Setup Phase *only when the frozen transactions have been finally confirmed by the underlying blockchains, while, in* Swap Complete Phase *and* Swap Timeout Phase*, $\mathcal{F}$ responds with ok as long as the corresponding swap transaction and refund transaction have been added in the respective bulletin board. The correctness lies in the facts that only the finally confirmed transaction can be further spent by a new transaction (where the frozen transaction will be spent by a swap or refund transaction) and, as defined in blockchain functionality $\mathcal{F}_{\mathbb{B}}$ (Fig.7), transactions in bulletin board can certainly be finally confirmed in time φ.*

Security Analysis. Now we analyze that the ideal functionality $\mathcal{F}$ satisfies the stronger *atomicity*:

•*Successful Swap:* If user P_0 honestly initiates its *swap* operation before timeout T_1, then $\mathcal{F}$ will enable both users P_0 and P_1 to complete the swap via respectively transferring the frozen coins (controlled by $\mathcal{F}$) to their corresponding addresses $pk_{swp}^{(0)}$ and $pk_{swp}^{(1)}$ (i.e., $b^{(0)} = 0$ and $b^{(1)} = 1$). Especially, if the adversary P_0 tries to delay its *swap* operation until timeout T_1, then $\mathcal{F}$ will enable P_0 to complete swap (i.e., $b^{(0)} = 0$) with probability $\frac{1}{2}$, instead, $\mathcal{F}$ is certainly to transfer user P_0's frozen coins to address $pk_{swp}^{(1)}$ (i.e., $b^{(1)} = 1$);

•*Failed Swap:* If the adversary P_0 fails to initiate its *swap* operation until timeout T_1, $\mathcal{F}$ will enable user P_1 to complete both *swap* and *refund* operations for $b^{(0)} = 1$, or just refund its own frozen coins for $b^{(0)} = \perp$.

Obviously, the ideal functionality $\mathcal{F}$ is secure against the *double-claiming* attack, ensuring that regardless of how the malicious user behaves, the honest user never loses coins.

6 PIPESWAP: PROTOCOL DESCRIPTION

6.1 Cryptographic Building Blocks

To guarantee universality, we insist on the fundamental building blocks from [42]: adaptor signature [3] and verifiable timed discrete logarithm (VTD) [43].

Adaptor Signature. The adaptor signature (cf. Def.1 in Appendix B) allows users to insert a puzzle Y (e.g., statement-witness pair $(Y, y) \in \mathcal{R}$, cf. Def.3 in Appendix B) into the generation of a signature on message $m \in \{0,1\}^{\lambda}$. The user with secret key first computes a pre-signature $\tilde{\sigma}$ of message m which by itself is not a valid signature, but can later be adapted into a valid signature σ (i.e., $SIG.Vf(pk, m, \sigma) = 1$, cf. Def.2 in Appendix B) with witness y. In addition, witness y can be further extracted by $\tilde{\sigma}$ and σ.

Here, the digital signature scheme satisfies the standard notion of unforgeability [5] and, to show the universality of our construction, we assume $SIG \in \{Schnorr, ECDSA\}$ to capture most existing cryptocurrencies, e.g., Bitcoin, Ethereum and Ripple. Adaptor signature is required to satisfy security properties of unforgeability, witness extractability and pre-signature adaptability (cf. [42] for the 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). VTD is required to satisfy the security properties of soundness and privacy (cf. [42] for the detailed definitions).

In this work, we use adaptor signature scheme and VTD in a black-box manner and refer the readers to [3, 36, 42, 43] for efficient constructions. In slightly more detail, as it is in [42], we adopt the construction of adaptor signature in [3], where the underlying signature scheme is Schnorr or ECDSA, and the hard relation $\mathcal{R}$ is the discrete log (dlog) relation (i.e., the language is defined as $\mathcal{L}_{\mathcal{R}_{dlog}} := \{Y | \exists y \in \mathbb{Z}_q^*, s.t.\ Y = g^y \in \mathbb{G}\}$). For the construction of VTD, the committer embeds the dlog.value x inside a time-lock puzzle H, uses a non-interactive zero-knowledge proof (NIZK) to prove that H can be solved in time T and the value x satisfies equation $H = g^x$, where such an efficient construction of NIZK [43] can be from the cut-and-choose techniques, Shamir secret sharing [40] and homomorphic time-lock puzzles [27].

Additionally, since the frozen address pk is jointly controlled by users P_0 and P_1 (i.e., the corresponding secret key sk is shared between them), it is inevitable that we rely on the interactive protocols (denoted as Γ_{AdpSig}^{SIG} and Γ_{Sig}^{SIG}) to realize the jointly (pre-)signing of a message m under public key pk, which can be efficiently instantiated w.r.t. $SIG \in \{Shnorr, ECDSA\}$ with the protocols in [26].

6.2 Procedures of pipeSwap

Recall that in the classic setting, users P_0 (owns coins α on blockchain $\mathbb{B}_0$) and P_1 (owns coins β on blockchain $\mathbb{B}_1$) wish to complete the α-to-β cross-chain swap. As we have briefly mentioned before (cf. Section 4), the *pipelined coins flow* of the frozen coins definitely guarantees *atomicity*. Forcing the earlier release of witness y is the crux of making pipeSwap secure, and this is achieved by three critical ingredients, i.e., *splitting frozen coins β into $(\varepsilon, \beta - \varepsilon)$*, *two-hop swap* and *two-hop refund*.

Protocol details. For ease of understanding, we illustrate the coin flow of pipeSwap in Fig.10 and describe pipeSwap in Fig.11, where the key point of each phase is presented below:

(A) Swap Setup Phase-Freezing Coins: This phase allows users P_0 and P_1 to transfer their swapped coins to the corresponding frozen addresses, which are jointly controlled by both users. To be better prepared for *two-hop swap* and *two-hop refund*, we let user P_1 split frozen coins β into two

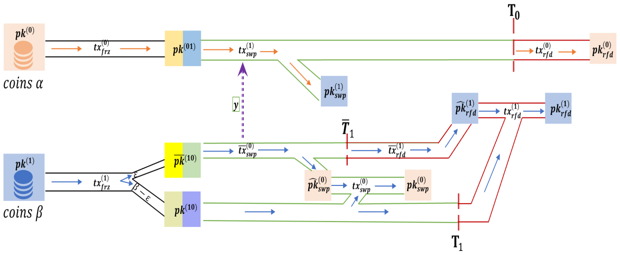


Figure 10: The coins flow of pipeSwap

addresses $\overline{pk}^{(10)}$ and $pk^{(10)}$ with respective values ε and $\beta-\varepsilon$. Importantly, to guarantee the frozen coins β are refunded as it is hoped, pre-refund transaction $\overline{tx}_{rfd}^{(1)} := (\overline{pk}^{(10)}, \widehat{pk}_{rfd}^{(1)}, \varepsilon)$ is locked until time $\overline{T}_1 := T_1 - \varphi$ (i.e., user P_0 makes a timed commitment of secret key share $\overline{sk}_0^{(10)}$ with timing hardness $\overline{T}_1$), while the refund transaction $tx_{rfd}^{(1)} := ((pk^{(10)}, \widehat{pk}_{rfd}^{(1)}), pk_{rfd}^{(1)}, \beta)$ is generated with secret keys $sk^{(10)}$ (jointly held by users P_0 and P_1) and $\widehat{sk}_{rfd}^{(1)}$ (held by user P_1), and will be valid at timeout T_1 (i.e., the time of confirming $\overline{tx}_{rfd}^{(1)}$). Meanwhile, user P_1 makes a timed commitment of secret key share $sk_1^{(01)}$ with timing hardness $T_0 > T_1$ for refunding frozen coins α (i.e., user P_0 can generate the refund transaction $tx_{rfd}^{(0)} := (pk^{(01)}, pk_{rfd}^{(0)}, \alpha)$ after timeout T_0);

(B1) Swap Lock Phase: This phase prepares for atomic swaps. Both users jointly pre-sign swap transactions $tx_{swp}^{(1)} := (pk^{(01)}, pk_{swp}^{(1)}, \alpha)$ and $\overline{tx}_{swp}^{(0)} := (\overline{pk}^{(10)}, \widehat{pk}_{swp}^{(0)}, \varepsilon)$ in sequence, where statement-witness pair $(Y, y) \in \mathcal{R}_{dlog}$ is selected by user P_0. Similarly, to guarantee the coins β are swapped as it is hoped, the swap transaction $tx_{swp}^{(0)} := ((pk^{(10)}, \widehat{pk}_{swp}^{(0)}), pk_{swp}^{(0)}, \beta)$ is well generated with secret keys $sk^{(10)}$ (jointly held by users P_0 and P_1) and $\widehat{sk}_{swp}^{(0)}$ (held by user P_0), and the final confirmation of transaction $\overline{tx}_{swp}^{(0)}$ is an essential prerequisite for the validity of $tx_{swp}^{(0)}$. Thus, in order to generate a valid swap transaction, user P_0 is forced to release witness y at least φ time earlier (i.e., posting pre-swap transaction $\overline{tx}_{swp}^{(0)}$);

(B2) Swap Complete Phase: If user P_0 can generate a valid swap transaction $tx_{swp}^{(0)}$ before timeout T_1 (i.e., it honestly posts pre-swap transaction $\overline{tx}_{swp}^{(0)}$ before time $\overline{T}_1$), then both swap transactions $tx_{swp}^{(0)}$ and $tx_{swp}^{(1)}$ must be finally confirmed by the underlying blockchains;

(C) Swap Timeout Phase: If user P_0 fails to generate a swap transaction $tx_{swp}^{(0)}$ before timeout T_1 (i.e., pre-swap transaction $\overline{tx}_{swp}^{(0)}$ is not confirmed before timeout T_1, and pre-refund transaction $\overline{tx}_{rfd}^{(1)}$ is finally confirmed), then user P_1 posts refund transaction $tx_{rfd}^{(1)}$ after timeout T_1. Similarly, if user P_1 fails to post swap transaction $tx_{swp}^{(1)}$ before timeout T_0, then user P_0 posts refund transaction $tx_{rfd}^{(0)}$ after timeout T_0.

Security intuitions. We brief security intuitions in the following and defer detailed proofs to Appendix C.

Successful Swap. This directly stems from the security of underlying blockchains and adaptor signature. If user P_0 can generate valid swap transaction $tx_{swp}^{(0)}$ before timeout T_1 (i.e., its pre-swap transaction $\overline{tx}_{swp}^{(0)}$ has been finally confirmed), then user P_1 can post its swap transaction $tx_{swp}^{(1)}$ with witness y extracted from the signature of $\overline{tx}_{swp}^{(0)}$ before timeout T_1. Thus, swap transactions of both users must be finally confirmed. Notice that the above analysis includes the case that the adversary P_0 successfully posts a swap transaction $tx_{swp}^{(0)}$ at timeout T_1 (i.e., it initiates a *double-claiming* attack), but this cannot prevent user P_1 from completing its Swap Complete Phase before timeout T_1.

Failed Swap. We consider the following possible cases.

- User P_0 does not initiate its swap operation before timeout T_1 (i.e., it does not post transaction $\overline{tx}_{swp}^{(0)}$ before timeout T_1), then both users can successfully refund their frozen coins;
- User P_0 fails to generate a valid swap transaction $tx_{swp}^{(0)}$ before/at timeout T_1 (i.e., pre-refund transaction $\overline{tx}_{rfd}^{(1)}$ is finally confirmed before timeout T_1), user P_1 can successfully refund its frozen coins. Moreover, since user P_0 has posted pre-swap transaction $\overline{tx}_{swp}^{(0)}$, user P_1 also can extract witness y to complete Swap Complete Phase by posting swap transaction $tx_{swp}^{(1)}$ before timeout T_1;
- The adversary P_0 can never generate a valid swap transaction $tx_{swp}^{(0)}$ after timeout T_1, thus user P_1 can successfully refund its frozen coins after timeout T_1 and even swap P_0's frozen coins.

Therefore, pipeSwap runs as expected, and satisfies *atomicity* in that the honest user never lose coins.

6.3 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 the 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, the signatures of Schnorr and ECDSA are instantiated over secp256k1 curve, and the transaction size is set to 250 bytes approximating the basic Bitcoin transaction. We implement the two-party computation protocol for digital signature Γ_{Sig}^{SIG}, and use the implementations of adaptor signatures Γ_{AdpSig}^{SIG} and VTD respectively in [41] and [43]. Our code used in evaluations is available at https://github.com/Anqi333/pipeSwap.

Computation Time. We first measure the time of basic operations required in pipeSwap, and the results are shown in Table 2. Then we measure the computation time required by both users together in Table 3. We observe 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%, because both users jointly complete two VTD computations.

Table 3: 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

*Universal Atomic Swaps [42].

Communication Overhead. We measure the communication overhead as the amount of messages that users exchange during the execution of interactive algorithms in the Swap Setup Phase and Swap Lock Phase (cf. Table 4).

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

(A) Swap Setup Phase - Freezing Coins

1. Users P_0 and P_1 respectively completes *Setup*:
   1) User P_0 runs Setup process (Fig.12) and sends $(pk_0^{(01)}, \overline{pk}_0^{(10)}, pk_0^{(10)}, (C^{(1)}, \pi^{(1)})) \hookrightarrow P_1$;
   2) User P_1 runs Setup process (Fig.12) and sends $(pk_1^{(01)}, \overline{pk}_1^{(10)}, pk_1^{(10)}, (C^{(0)}, \pi^{(0)})) \hookrightarrow P_0$.
2. Users P_0 and P_1 generate their frozen addresses:
   1) User P_0 does the following:
      - It checks if $VTD.Vf(pk_1^{(01)}, C^{(0)}, \pi^{(0)}) = 1$, and stops otherwise;
      - It generates frozen address $pk^{(01)} = pk_0^{(01)} \oplus pk_1^{(01)}$.
   2) User P_1 does the following:
      - It checks if $VTD.Vf(\overline{pk}_0^{(10)}, C^{(1)}, \pi^{(1)}) = 1$, and stops otherwise;
      - It generates frozen addresses $\overline{pk}^{(10)} = \overline{pk}_0^{(10)} \oplus \overline{pk}_1^{(10)}$ and $pk^{(10)} = pk_0^{(10)} \oplus pk_1^{(10)}$.
3. Users P_0 and P_1 transfer swapped coins to the corresponding frozen addresses.
   1) User P_0 does the following:
      - It generates frozen transaction $tx_{frz}^{(0)} := (pk^{(0)}, pk^{(01)}, \alpha)$ and signature $\sigma_{frz}^{(0)} \leftarrow \Sigma_{SIG}.Sig(sk^{(0)}, tx_{frz}^{(0)})$;
      - It posts $(tx_{frz}^{(0)}, \sigma_{frz}^{(0)})$ on blockchain $\mathbb{B}_0$ and starts solving $VTD.ForceOp(C^{(0)})$.
   2) Users P_0 and P_1 jointly do the following:
      - P_1 generates frozen transaction $tx_{rfz}^{(1)} := (pk^{(1)}, (\overline{pk}^{(10)}, pk^{(10)}), (\varepsilon, \beta - \varepsilon))$, pre-refund transaction $\overline{tx}_{rfd}^{(1)} := (\overline{pk}^{(10)}, \widehat{pk}_{rfd}^{(1)}, \varepsilon)$ and refund transaction $tx_{rfd}^{(1)} := ((pk^{(10)}, \widehat{pk}_{rfd}^{(1)}), pk_{rfd}^{(1)}, \beta)$. It sends $(tx_{rfz}^{(1)}, \overline{tx}_{rfd}^{(1)}, tx_{rfd}^{(1)}) \hookrightarrow P_0$;
      - P_0 checks that transactions $(tx_{rfz}^{(1)}, \overline{tx}_{rfd}^{(1)}, tx_{rfd}^{(1)})$ are well formed (i.e., satisfy *two-hop refund* framework), and stops otherwise;
      - P_0 and P_1 run a 2PC protocol Γ_{Sig}^{SIG} with input $(sk_0^{(10)}, sk_1^{(10)}, tx_{rfd}^{(1)})$ (Fig.12) and obtain signature $\check{\sigma}_{rfd}^{(1)}$;
      - P_1 computes signature $\sigma_{frz}^{(1)} \leftarrow \Sigma_{SIG}.Sig(sk^{(1)}, tx_{frz}^{(1)})$;
      - P_1 posts $(tx_{frz}^{(1)}, \sigma_{frz}^{(1)})$ on blockchain $\mathbb{B}_1$ and starts solving $VTD.ForceOp(C^{(1)})$.

(B1) Swap Lock Phase

1. User P_0 runs $(Y, y) \leftarrow \mathcal{R}Gen(1^\lambda)$ and sends $Y \hookrightarrow P_1$.
2. Users P_0 and P_1 generates their swap transactions:
   1) User P_0 does the following:
      - It generates pre-swap transaction $\overline{tx}_{swp}^{(0)} := (\overline{pk}^{(10)}, \widehat{pk}_{swp}^{(0)}, \varepsilon)$ and swap transaction $tx_{swp}^{(0)} := ((pk^{(10)}, \widehat{pk}_{swp}^{(0)}), pk_{swp}^{(0)}, \beta)$;
      - It sends $(\overline{tx}_{swp}^{(0)}, tx_{swp}^{(0)}) \hookrightarrow P_1$.
   2) User P_1 does the following:
      - It checks that transactions $(\overline{tx}_{swp}^{(0)}, tx_{swp}^{(0)})$ are well formed (i.e., satisfy *two-hop swap* framework), and stops otherwise;
      - It generates swap transaction $tx_{swp}^{(1)} := (pk^{(01)}, pk_{swp}^{(1)}, \alpha)$;
      - It sends $tx_{swp}^{(1)} \hookrightarrow P_0$.
3. Users P_0 and P_1 run a 2PC protocol Γ_{AdpSig}^{SIG} with input $(sk_0^{(01)}, sk_1^{(01)}, Y, tx_{swp}^{(1)})$ (Fig.12), and obtain pre-signature $\widetilde{\sigma}_{swp}^{(1)}$;
4. After step 3 is successful, P_0 and P_1 run a 2PC protocol Γ_{Sig}^{SIG} with input $(sk_0^{(10)}, sk_1^{(10)}, tx_{swp}^{(0)})$ (Fig.12) and obtain signature $\check{\sigma}_{swp}^{(0)}$, and then run 2PC protocol Γ_{AdpSig}^{SIG} with input $(\overline{sk}_0^{(10)}, \overline{sk}_1^{(10)}, Y, \overline{tx}_{swp}^{(0)})$ (Fig.12) and obtain pre-signature $\widetilde{\overline{\sigma}}_{swp}^{(0)}$.

(B2) Swap Complete Phase

5. User P_0 does the following:
   1) It computes $\overline{\sigma}_{swp}^{(0)} \leftarrow \Sigma_{AS}^{SIG}.Adapt(\widetilde{\overline{\sigma}}_{swp}^{(0)}, y)$ and posts $(\overline{tx}_{swp}^{(0)}, \overline{\sigma}_{swp}^{(0)})$ on blockchain $\mathbb{B}_1$ before time $\overline{T}_1 - \Delta$;
   2) It computes $\widehat{\sigma}_{swp}^{(0)} \leftarrow \Sigma_{SIG}.Sig(\widehat{sk}_{swp}^{(0)}, tx_{swp}^{(0)})$ and posts $(tx_{swp}^{(0)}, (\check{\sigma}_{swp}^{(0)}, \widehat{\sigma}_{swp}^{(0)}))$ on blockchain $\mathbb{B}_1$ if $\overline{tx}_{swp}^{(0)}$ is finally confirmed.
6. Upon receiving $(\overline{tx}_{swp}^{(0)}, \overline{\sigma}_{swp}^{(0)})$, user P_1 dose the following:
   1) It computes $y \leftarrow \Sigma_{AS}^{SIG}.Ext(\overline{\sigma}_{swp}^{(0)}, \widetilde{\overline{\sigma}}_{swp}^{(0)}, Y)$ and $\sigma_{swp}^{(1)} \leftarrow \Sigma_{AS}^{SIG}.Adap(\widetilde{\sigma}_{swp}^{(1)}, y)$;
   2) It posts $(tx_{swp}^{(1)}, \sigma_{swp}^{(1)})$ on blockchain $\mathbb{B}_0$.

(C) Swap Refund Phase

1. If user P_0 fails to post $(\overline{tx}_{swp}^{(0)}, \overline{\sigma}_{swp}^{(0)})$ on blockchain $\mathbb{B}_1$ before time $\overline{T}_1$, then user P_1 does the following:
   1) It finishes computing $\overline{sk}_0^{(10)} \leftarrow VTD.ForceOp(C^{(1)})$ and computes $\overline{sk}^{(10)} := \overline{sk}_0^{(10)} \oplus \overline{sk}_1^{(10)}$;
   2) It computes $\overline{\sigma}_{rfd}^{(1)} \leftarrow \Sigma_{SIG}.Sig(\overline{sk}^{(10)}, \overline{tx}_{rfd}^{(1)})$ and posts $(\overline{tx}_{rfd}^{(1)}, \overline{\sigma}_{rfd}^{(1)})$ on blockchain $\mathbb{B}_1$;
   3) If $\overline{tx}_{rfd}^{(1)}$ is finally confirmed, it computes $\widehat{\sigma}_{rfd}^{(1)} \leftarrow \Sigma_{SIG}.Sig(\widehat{sk}_{rfd}^{(1)}, tx_{rfd}^{(1)})$ and posts $(tx_{rfd}^{(1)}, (\check{\sigma}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}))$ on blockchain $\mathbb{B}_1$.
2. Similarly, if user P_1 fails to post $(tx_{swp}^{(1)}, \sigma_{swp}^{(1)})$ on blockchain $\mathbb{B}_0$ before time T_0, then user P_0 does the following:
   1) It finishes computing $sk_1^{(01)} \leftarrow VTD.ForceOp(C^{(0)})$ and computes $sk^{(01)} := sk_0^{(01)} \oplus sk_1^{(01)}$;
   2) It computes $\sigma_{rfd}^{(0)} \leftarrow \Sigma_{SIG}.Sig(sk^{(01)}, tx_{rfd}^{(0)})$ and posts $(tx_{rfd}^{(0)}, \sigma_{rfd}^{(0)})$ on blockchain $\mathbb{B}_0$.

Figure 11: pipeSwap: a secure cross-chain swap between users P_0 and P_1

//The Setup process

User P_0 does the following:

(1) It runs $\Sigma_{SIG}.KGen(1^\lambda) \rightarrow \{(sk_0^{(01)}, pk_0^{(01)}), (\overline{sk}_0^{(10)}, \overline{pk}_0^{(10)}), (sk_0^{(10)}, pk_0^{(10)}), (sk_{rfd}^{(0)}, pk_{rfd}^{(0)}), (\widehat{sk}_{swp}^{(0)}, \widehat{pk}_{swp}^{(0)}), (sk_{swp}^{(0)}, pk_{swp}^{(0)})\}$;

(2) It computes commitment $VTD.Commit(\overline{sk}_0^{(10)}, \overline{T}_1) \rightarrow (C^{(1)}, \pi^{(1)})$.

User P_1 does the following:

(1) It runs $\Sigma_{SIG}.KGen(1^\lambda) \rightarrow \{(sk_1^{(01)}, pk_1^{(01)}), (\overline{sk}_1^{(10)}, \overline{pk}_1^{(10)}), (sk_1^{(10)}, pk_1^{(10)}), (\widehat{sk}_{rfd}^{(1)}, \widehat{pk}_{rfd}^{(1)}), (sk_{rfd}^{(1)}, pk_{rfd}^{(1)}), (sk_{swp}^{(1)}, pk_{swp}^{(1)})\}$;

(2) It computes commitment $VTD.Commit(sk_1^{(01)}, T_0) \rightarrow (C^{(0)}, \pi^{(0)})$.

//The 2PC protocol Γ_{Sig}^{SIG}

It takes private inputs sk_0 and sk_1 held by users P_0 and P_1 respectively, and public message m:

(1) It sets secret key as $sk := sk_0 \oplus sk_1$;

(2) It computes $\breve{\sigma} \leftarrow \Sigma_{SIG}.Sig(sk, m)$ and sends $\breve{\sigma}$ to both users P_0 and P_1;

(3) Users P_0 and P_1 respectively check if $\Sigma_{SIG}.Vf(pk, m, \sigma) = 1$, and stops otherwise.

//The 2PC protocol Γ_{AdpSig}^{SIG}

It takes private inputs sk_0 and sk_1 held by users P_0 and P_1 respectively, and public messages (m, Y):

(1) It sets secret key as $sk := sk_0 \oplus sk_1$;

(2) It computes $\widetilde{\sigma} \leftarrow \Sigma_{AS}^{SIG}.pSig(sk, m, Y)$ and sends $\widetilde{\sigma}$ to both users P_0 and P_1;

(3) Users P_0 and P_1 respectively check if $\Sigma_{AS}^{SIG}.pVf(pk, m, Y, \widetilde{\sigma}) = 1$, and stops otherwise.

Figure 12: The subroutines

Table 2: The computation time of basic operations (ms)

	Σ_{SIG}			Γ_{Sig}^{SIG}		Γ_{AdpSig}^{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		

In particular, 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 4: The communication overheads (bytes)

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

*Universal Atomic Swaps [42].

Efficiency Comparison. To compare pipeSwap (Fig.6) with Universal Atomic Swaps (its Fig.5 in [42]) w.r.t. the operations required of both users together, we evaluate pipeSwap and Universal Atomic Swaps with the same setting and libraries and security parameters for all cryptographic implementations. Before delving into details, let us be clear first that Universal Atomic Swaps repeat the same operations for each swapped coin, thus here we only consider the one-to-one atomic swap in [42]. Universal Atomic Swaps complete in 1.6 seconds for Schnorr and 1.617 seconds for ECDSA, and requires 5.1 kb for Schnorr and 5.4 kb for ECDSA. Therefore, pipeSwap is only ≤ 7 ms slower and incurs extra ≤ 1.6 kb communication overhead, which is acceptable, even though we need to prepare and sign two extra transactions (the pre-swap and pre-refund transactions). 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 P_1 is $\overline{T}_1 < T_1$, which means that pipeSwap takes less computational costs.

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

7 CONCLUSIONS AND FUTURE WORKS

In this paper, we identify a new form of attack, called *double-claiming* attack, againsts Universal Atomic Swaps [IEEE S&P'22]. This attack can lead to honest user losing coins with overwhelming probability and *atomicity* property is directly broken. We introduce a novel approach of utilizing *two-hop swap* and *two-hop refund* techniques to secure coin flows, and design pipeSwap, a universal atomic cross-chain swap protocol.

Several interesting questions can be considered in future work. pipeSwap is efficiently instantiated with standard signature schemes Schnorr and ECDSA, and direct building on other signature schemes may need further care. It is interesting to explore extensions to some more complex but practical scenarios, e.g., multi-hop swaps $P_1 \rightarrow P_2 \rightarrow \cdots \rightarrow P_n \rightarrow P_1$, 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] 2023. https://thecharlatan.ch/Monero-Unlock-Time-Privacy.

[2] Hagit Attiya and Jennifer Welch. 2004. *Distributed computing: fundamentals, simulations, and advanced topics*. Vol. 19. John Wiley & Sons.

[3] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2021. Generalized channels from limited blockchain scripts and adaptor signatures. In *Advances in Cryptology–ASIACRYPT 2021*. Springer, 635–664.

[4] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2021. Breaking and Fixing Virtual Channels: Domino Attack and Donner. *Cryptology ePrint Archive* (2021).

[5] Michael Backes and Dennis Hofheinz. 2004. How to break and repair a universally composable signature functionality. In *International Conference on Information Security*. Springer, 61–72.

[6] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. 2016. Cryptocurrencies without proof of work. In *Financial Cryptography and Data Security*. Springer, 142–157.

[7] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. 2019. Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware. In *the 2019 ACM SIGSAC Conference*.

[8] Michael Borkowski, Marten Sigwart, Philipp Frauenthaler, Taneli Hukkinen, and Stefan Schulte. 2019. Dextt: Deterministic Cross-Blockchain Token Transfers. *IEEE Access* 7 (2019), 111030–111042. https://doi.org/10.1109/ACCESS.2019.2934707

[9] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, and Frank Piessens. 2019. A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes. In *the 2019 ACM SIGSAC Conference*.

[10] Sergiu Bursuc and Steve Kremer. 2019. 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, 361–382.

[11] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. 2017. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 229–243.

[12] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 136–145.

[13] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. 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, 61–85.

[14] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2019. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 142–157.

[15] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)* 35, 2 (1988), 288–323.

[16] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 949–966.

[17] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 281–310.

[18] Runchao Han, Haoyu Lin, and Jiangshan Yu. 2019. On the optionality and fairness of atomic swaps. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 62–75.

[19] Maurice Herlihy. 2018. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*. 245–254.

[20] Philipp Hoenisch, Subhra Mazumdar, Pedro Moreno-Sanchez, and Sushmita Ruj. 2022. LightSwap: An Atomic Swap Does Not Require Timeouts at both Blockchains. In *International Workshop on Data Privacy Management*. Springer, 219–235.

[21] Ghassan O Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. 2015. Misbehavior in bitcoin: A study of double-spending and accountability. *ACM Transactions on Information and System Security (TISSEC)* 18, 1 (2015), 1–32.

[22] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. 2013. Universally composable synchronous computation. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*. Springer, 477–498.

[23] Rami Khalil and Arthur Gervais. 2017. Revive: Rebalancing Off-Blockchain Payment Networks. *ACM* (2017).

[24] Aggelos Kiayias and Dionysis Zindros. 2020. 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, 21–34.

[25] Russell WF Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. 2019. Omniring: Scaling private payments without trusted setup. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 31–48.

[26] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2018. Anonymous multi-hop locks for blockchain scalability and interoperability. *Cryptology ePrint Archive* (2018).

[27] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. 2019. Homomorphic time-lock puzzles and applications. In *Annual International Cryptology Conference*. Springer, 620–649.

[28] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. 2017. Sprites: Payment channels that go faster than lightning. *CoRR, abs/1702.05812* (2017).

[29] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[30] Krishnasuri Narayanam, Venkatraman Ramakrishna, Dhinakaran Vinayagamurthy, and Sandeep Nishad. 2022. Atomic cross-chain exchanges of shared assets. arXiv:2202.12855 [cs.CR]

[31] Krishnasuri Narayanam, Venkatraman Ramakrishna, Dhinakaran Vinayagamurthy, and Sandeep Nishad. 2022. Generalized HTLC for cross-chain swapping of multiple assets with co-ownerships. *arXiv preprint arXiv:2202.12855* (2022).

[32] Tier Nolan. 2013. Alt chains and atomic transfers. Bitcoin Forum. https://bitcointalk.org/index.php?topic=193281.0.

[33] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 643–673.

[34] Andrew Poelstra. 2016. Mimblewimble. (2016).

[35] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments. (2016).

[36] Ronald L Rivest, Adi Shamir, and David A Wagner. 1996. Time-lock puzzles and timed-release crypto. (1996).

[37] Muhammad Saad, Afsah Anwar, Srivatsan Ravi, and David Mohaisen. 2021. Revisiting nakamoto consensus in asynchronous networks: A comprehensive analysis of bitcoin safety and chain-quality. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 988–1005.

[38] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*. IEEE, 459–474.

[39] David Schwartz, Noah Youngs, Arthur Britto, et al. 2014. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper* 5, 8 (2014), 151.

[40] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[41] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. 2021. A2L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs. In *2021 IEEE Symposium on Security and Privacy (SP)*. 1834–1851. https://doi.org/10.1109/SP40001.2021.00111

[42] Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. 2022. Universal Atomic Swaps: Secure Exchange of Coins Across All Blockchains. In *2022 IEEE Symposium on Security and Privacy (SP)*. 1299–1316. https://doi.org/10.1109/SP46214.2022.9833731

[43] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. 2020.

Verifiable timed signatures made practical. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1733–1750.

[44] Sri Aravinda Krishnan Thyagarajan and Giulio Malavolta. 2021. Lockable signatures for blockchains: Scriptless scripts for all signatures. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 937–954.

[45] Gilbert Verdian, Paolo Tasca, Colin Paterson, and Gaetano Mondelli. 2018. Quant overledger whitepaper. *Release V0* 1 (2018), 31.

[46] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[47] Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. 2019. Atomic commitment across blockchains. *arXiv preprint arXiv:1905.02847* (2019).

[48] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. 2019. X-claim: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 193–210.

A RELATED WORKS

Tier Nolan first introduced the conceptualisation of "atomic swap" [32]. Its fundamental security *atomicity* states that the swap either ends with success (i.e., the owners of involved coins are exchanged) or failure (i.e., the involved coins are refunded to their original owners) [18, 47].

Essentially, a secure cross-chain swap between users P_0 and P_1 should fulfill two fundamental functionalities to guarantee the honest user P_0 cannot lose coins (1) if user P_1 has claimed P_0's frozen coins, P_0 is able claim P_1's frozen coins before P_1 can refund them; and (2) if user P_1 is malicious, P_0 can refund its frozen coins. While *atomicity* is easily realized by the trusted third party, the blockchain community has made significant efforts to achieve (fully) decentralized cross-chain swaps [30, 47, 48]. HTLC-based protocols use the rich scripting languages supported by the underlying blockchains to describe when and how the frozen coins can be unlocked [8, 31]. Subsequently, HTLC-style solutions have been widely applied and deployed in practice [10, 11, 19]. However, these protocols are far from the universal solution and suffer from the inherent drawbacks of HTLC, including 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 at odds with the blockchains that have already achieved privacy [38].

Recently, LightSwap [20] studies the swap that enables the user to run an instance of swap on a mobile phone and is committed to proposing the first secure atomic swap protocol that does not require the timeout functionality by one of the two participating users. However, it still requires one of the two involved blockchains supporting timelock functionality and thus cannot achieve universality. Universal Atomic Swaps [42] use cryptographic building blocks adaptor signature and timed commitment to present the first fully universal solution.

Besides, some literatures [24, 45] use a third blockchain as the coordinator, but it requires the involved users with the capability of transferring to/from coins from this blockchain. Also, the cross-chain swaps functionality is inserted into a trusted hardware [7], which is not only unrealistic but also exists serious vulnerabilities [9, 14].

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 [35]). Similarly, most of the existing PCN proposals are restricted to the Turing complete scripting language [16, 23, 28] thus suffering from the inherent drawbacks of HTLC. Anonymous Multi-Hop Locks [26], lockable signatures [44] and A^{2}L [41] are recently proposed to construct scriptless PNC. However, these studies only defer successful atomic payment to specific signature schemes but still rely on on-chain time-lock functionality to ensure payment expiry, and thus is not universal.

B DEFINITIONS OF CRYPTOGRAPHIC BUILDING BLOCKS

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

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

Definition 3. *(Hard Relation) A hard relation* $\mathcal{R}$ *is described as* $\mathcal{L}_\mathcal{R} := \{Y | \exists y, s.t. (Y, y) \in \mathcal{R}\}$ *and satisfies:*
1) $\mathcal{R}Gen(1^\lambda) \rightarrow (Y, y)$*: The sampling algorithm takes as input security parameter* λ *and outputs statement-witness pair* $(Y, y) \in \mathcal{R}$*;*
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 $(Commit, Vf, Open, ForceOp)$ defined as:*
1) $Commit(x, r, T) \to (C, \pi)$: 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) $Vf(H, C, \pi) \to b$: The verification algorithm inputs group element $H := g^x$, C and π, outputs $b = 1$ if C is a valid commitment of x with hardness T and $b = 0$ otherwise;
3) $Open(C) \to (x, r)$: The open algorithm inputs commitment C, outputs the committed value x and randomness r;
4) $ForceOp(C) \to x$: The force open algorithm inputs commitment C and outputs the committed value x.

C SECURITY ANALYSIS

Theorem 1. *(Atomicity) Assume Σ_{AS}^{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 Γ_{Sig}^{SIG} and Γ_{AdpSig}^{SIG} are UC-secure 2PC protocols for jointly computing $\Sigma_{SIG}.Sig$ and $\Sigma_{AS}^{SIG}.pSig$; VTD is a secure timed commitment of dlog. Then protocol* pipeSwap *running in the $(\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{smt})$-hybrid world UC-realizes ideal functionality $\mathcal{F}$.*

Proof. We now prove that protocol pipeSwap (Fig.6) UC-realizes the cross-chain swap 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 $\{P_0, P_1\}$ 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 Γ_{Sig}^{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 Γ_{Sig}^{SIG});
Hybrid $\mathcal{H}_2$: It is the same as the above execution except that the 2PC protocol Γ_{AdpSg}^{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 P_1 and outputs a valid swap transaction $(tx_{swp}^{(1)}, \sigma_{swp}^{(1)})$ before the simulator initiates swap operation on behalf of P_0, the simulator aborts;
Hybrid $\mathcal{H}_4$: It is the same as the above execution except that the adversary corrupts user P_0 and outputs a valid swap transaction $(tx_{swp}^{(0)}, \sigma_{swp}^{(0)})$ before timeout T_1. The simulator outputs $(tx_{swp}^{(1)}, \sigma_{swp}^{(1)})$ and if $\Sigma_{SIG}.Vf(pk^{(01)}, tx_{swp}^{(1)}, \sigma_{swp}^{(1)}) \neq 1$, the simulator aborts;
Hybrid $\mathcal{H}_5$: It is the same as the above execution except that the adversary corrupts user P_0 and outputs a valid swap transaction $(tx_{swp}^{(0)}, \sigma_{swp}^{(0)})$ at timeout T_1. The simulator outputs $(tx_{swp}^{(1)}, \sigma_{swp}^{(1)})$ and if $\Sigma_{SIG}.Vf(pk^{(01)}, tx_{swp}^{(1)}, \sigma_{swp}^{(1)}) \neq 1$, the simulator aborts;
Hybrid $\mathcal{H}_6$: It is the same as the above execution except that the adversary corrupts user P_0 and initiates the swap operation $(tx_{swp}^{(0)}, \sigma_{swp}^{*(0)})$ at timeout T_1. The simulator outputs $(tx_{swp}^{(1)}, \sigma_{swp}^{(1)})$ and $(tx_{rfd}^{(1)}, (\check{\sigma}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}))$ if $\Sigma_{SIG}.Vf(pk^{(01)}, tx_{swp}^{(1)}, \sigma_{swp}^{(1)}) \neq 1$ or $\Sigma_{SIG}.Vf(\widehat{pk}_{rfd}^{(1)}, tx_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}) \neq 1$ or $\Sigma_{SIG}.Vf(pk^{(10)}, tx_{rfd}^{(1)}, \check{\sigma}_{rfd}^{(1)}) \neq 1$, the simulator aborts;
Hybrid $\mathcal{H}_7$: It is the same as the above execution except that the adversary corrupts user P_0 and initiates the swap operation $(tx_{swp}^{(0)}, \sigma_{swp}^{*(0)})$ after timeout T_1. The simulator outputs $(tx_{rfd}^{(1)}, (\check{\sigma}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}))$ if $\Sigma_{SIG}.Vf(\widehat{pk}_{rfd}^{(1)}, tx_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}) \neq 1$ or $\Sigma_{SIG}.Vf(pk^{(10)}, tx_{rfd}^{(1)}, \check{\sigma}_{rfd}^{(1)}) \neq 1$, the simulator aborts;
Hybrid $\mathcal{H}_8$: It is the same as the above execution except that the adversarial P_0 outputs a valid refund transaction $(tx_{rfd}^{(0)}, \sigma_{rfd}^{(0)})$ before timeout T_0, the simulator aborts;
Hybrid $\mathcal{H}_9$: It is the same as the above execution except that the adversarial P_1 outputs a valid refund transaction $(tx_{rfd}^{(1)}, \sigma_{rfd}^{(1)})$ before timeout T_1, the simulator aborts;
Hybrid $\mathcal{H}_{10}$: It is the same as the above execution except that the adversary corrupts user P_0 and the simulator obtains $(tx_{rfd}^{(1)}, \sigma_{rfd}^{(1)})$ after timeout T_1, if $\Sigma_{SIG}.Vf(\widehat{pk}_{rfd}^{(1)}, tx_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}) \neq 1$ or $\Sigma_{SIG}.Vf(pk^{(10)}, tx_{rfd}^{(1)}, \check{\sigma}_{rfd}^{(1)}) \neq 1$, the simulator aborts;
Hybrid $\mathcal{H}_{11}$: It is the same as the above execution except that the adversary corrupts user P_1 and the simulator obtains $(tx_{rfd}^{(0)}, \sigma_{rfd}^{(0)})$ after timeout T_0, if $\Sigma_{SIG}.Vf(pk^{(01)}, tx_{rfd}^{(0)}, \sigma_{rfd}^{(0)}) \neq 1$, the simulator aborts;

Simulator $\mathcal{S}$: The simulator $\mathcal{S}$ is defined as the execution in $\mathcal{H}_{11}$ 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 $\mathcal{H}_{11}$. 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 Γ_{Sig}^{SIG}. The security of 2PC protocol Γ_{Sig}^{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 Γ_{AdpSig}^{SIG}. The security of 2PC protocol Γ_{AdpSig}^{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 P_1 and outputs a valid swap transaction $(tx_{swp}^{(1)}, \sigma_{swp}^{(1)})$ before the simulator initiate a swap on behalf of user 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 P_0 and outputs a valid swap transaction $(tx_{swp}^{(0)}, \sigma_{swp}^{(0)})$ 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;

$\underline{\mathcal{H}_4 \approx_c \mathcal{H}_5 \approx_c \mathcal{H}_6}$: The only difference between the hybrids is that in $\mathcal{H}_5$ and $\mathcal{H}_6$ the simulator aborts, if the adversary initiates a (valid) swap operation at timeout T_1, the simulator cannot post its valid swap transaction or and refund transaction. With the security of underlying blockchain, adaptor signature and VTD, the probability of the events triggered in $\mathcal{H}_5$ and $\mathcal{H}_6$ is negligible;

$\underline{\mathcal{H}_6 \approx_c \mathcal{H}_7}$: The only difference between the hybrids is that in $\mathcal{H}_7$ the simulator aborts, if the adversary 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 $\mathcal{H}_7$ is negligible;

$\underline{\mathcal{H}_7 \approx_c \mathcal{H}_8}$: The only difference between the hybrids is that in $\mathcal{H}_8$ the simulator aborts, if the adversary P_0 outputs a valid refund transaction before timeout T_0. With the security of VTD, the probability of the event triggered in $\mathcal{H}_8$ is negligible;

$\underline{\mathcal{H}_8 \approx_c \mathcal{H}_9}$: The only difference between the hybrids is that in $\mathcal{H}_9$ the simulator aborts, if the adversary 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 $\mathcal{H}_9$ is negligible;

$\underline{\mathcal{H}_9 \approx_c \mathcal{H}_{10}}$: The only difference between the hybrids is that in $\mathcal{H}_{10}$ 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 $\mathcal{H}_{10}$ is negligible;

$\underline{\mathcal{H}_{10} \approx_c \mathcal{H}_{11}}$: The only difference between the hybrids is that in $\mathcal{H}_{11}$ 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 $\mathcal{H}_{11}$ is negligible.

□