Warning! The Timeout T Cannot Protect You From Losing Coins
PipeSwap: Forcing the Timely Release of a Secret for Atomic Cross-Chain Swaps

Peifang Ni*‡¶, Anqi Tian*†¶ and Jing Xu*†‡§
*Institute of Software, Chinese Academy of Sciences
†School of Computer Science and Technology,
University of Chinese Academy of Sciences
‡Zhongguancun Laboratory, Beijing, P.R.China
Email: {peifang2020, anqi2021, xujing}@iscas.ac.cn

***Abstract*—Atomic cross-chain swaps mitigate the interoperability challenges faced by current cryptocurrencies, thereby facilitating inter-currency exchange and trading between the distrusting users. Although numerous atomic swaps protocols utilizing Hash Timelock Contracts have been deployed and put into practice, they are substantially far from universality due to their inherent dependence of rich scripting language supported by the underlying blockchains. The recently proposed Universal Atomic Swaps protocol [IEEE S&P'22] represents a significant advancement in the field of scriptless cross-chain swaps by ingeniously delegating scripting functionalities to cryptographic locking mechanisms, particularly the adaptor signatures and timed commitment schemes. However, we identify a new form of attack termed the *double-claiming* attack that leverages these scriptless functionalities to undermine atomicity with a high probability. This attack is inherent to the designs adopted by the existing scriptless cross-chain swaps protocols as well as the payment channel networks. We further quantify the severity of this attack based on real-word swap transactions processed by the most widely deployed decentralized exchange platforms, highlighting the critical challenges in designing universal atomic swaps.**

To address the *double-claiming* attack while ensuring both security and practical universality, we also present a cross-chain swaps protocol called PipeSwap. Specifically, PipeSwap protects the frozen coins from being double-claimed by a novelly designed paradigm of pipelined coins flow that utilizes the techniques of *two-hop swap* and *two-hop refund*. In addition to a comprehensive 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 while maintaining less than 7 kb of communication overhead on commodity machines.

***Index Terms*—Atomic Swaps, Universality, Pipelined Coins Flow, Two-Hop Swap, Two-Hop Refund.**

¶ Peifang Ni & Anqi Tian are co-first authors.
§ Corresponding author.

1. Introduction

With a multitude of diverse blockchain systems coexisting today, it is unrealistic to envision each one evolving in isolation, especially given the explosive development of cryptocurrencies such as Bitcoin [1], Ethereum [2], Ripple [3] and Monero [4]. This highlights an extremely urgent demand for deploying secure mechanisms that facilitate cryptocurrency exchanges. The atomic cross-chain swaps protocol [5] has been introduced as a method to enable secure coin exchanges between two mutually distrusting users, each possessing some coins on distinct blockchains. The fundamental security property of *atomicity* guarantees that the honest user cannot lose coins. More specifically, *atomicity* refers to the entire execution process of swaps protocol being atomic that it either successfully facilitates the exchange of coins or fails completely, resulting in a refund of the coins to their original owner. One key component of this process is the *refund* mechanism, which relies on a predefined timeout parameter T associated with each frozen coin. This parameter plays a pivotal role in guaranteeing the frozen coins of an honest user can be reclaimed in case the counterpart stops collaborating. Importantly, the designed *refund* mechanism should not only prevent coins from being indefinitely locked in a shared account but also ensure their definitive and timely unlocking after timeout T. Such assurance is especially critical as fluctuations in the values of digital assets over time could otherwise result in losses for the honest user.

Most classic efforts focus on studying the Hash Timelock Contracts (HTLC) style protocols [6], [7], [8], [9], [10], which rely on the rich scripting languages supported by the underlying blockchains to enforce some specific spending behaviors, namely, when and how the locked coins can be claimed by the intended receiver or reclaimed by the original owner. At a high level, the user Alice can use an HTLC script with the hash function H to temporarily lock some coins until a specified timeout T as follows: The HTLC specifies a hash value $h := H(x)$ such that if Bob presents x before the timeout T, he is entitled to claim the locked coins; otherwise, these coins will be refunded to Alice definitively after the timeout T.

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

A Desideratum for Achieving Atomic Cross-Chain Swaps in the Absence of Custom Scripts. Universal Atomic Swaps protocol [13] marks a significant step forward in the universal proposal designed to ensure *atomicity*, akin to the HTLC-style protocols, even in the presence of malicious users. Noticeably, instead of relying on on-chain scripts to define the locked coins and their corresponding unlocking conditions, Universal Atomic Swaps require only the scripts to verify digital signatures from the underlying blockchains. This is achieved by leveraging the adaptor signature scheme [14] and verifiable timed discrete logarithm (VTD) scheme [15]. Specifically, the witness extractability property of adaptor signature facilitates successful swaps: when user $\mathcal{P}_0$ holding witness y posts a valid swap transaction, the witness y is subsequently released to user $\mathcal{P}_1$ to complete the swap operation. The VTD ensures that coins remaining locked after a predefined timeout T (i.e., in cases where the swap operation fails) are refunded to their original owners. Consequently, Universal Atomic Swaps present an innovative approach to scriptless cross-chain swaps and arguably emerge as the most promising candidate for securely exchanging digital assets.

Is Timeout T Really Secure for the Honest User? The scriptless implementation of timeout T may potentially be used to violate *atomicity*. Herein we introduce a new attack termed *double-claiming* attack. It is noteworthy that in the context of Universal Atomic Swaps [13], the predefined timeout T_1 is intentionally designed for user $\mathcal{P}_1$ to securely reclaim the frozen coins β (i.e., user $\mathcal{P}_1$ can obtain the full secret key of frozen account after the timeout T_1). However, this timeout T_1 does not preclude user $\mathcal{P}_0$ from claiming the frozen coins β. In other words, even after the timeout T_1, user $\mathcal{P}_0$ possessing with witness y continues to retain the ability of generating a valid swap transaction. Consequently, the timeout T_1 becomes a focal point for security vulnerabilities. For example, when the honest user $\mathcal{P}_1$ posts a refund transaction after the timeout T_1, the malicious user $\mathcal{P}_0$ can still release a valid swap transaction to make the frozen coins β double-claimed. Unfortunately, if this swap transaction is finally confirmed by the underlying blockchain, the honest user $\mathcal{P}_1$ will neither successfully enter the Swap Complete Phase nor prevail in the Swap Timeout Phase, ultimately leading to a loss of coins. Similarly, the recent effort Sweep-UC [16] is also susceptible to this *double-claiming* attack, causing honest users to lose coins just like in Universal Atomic Swaps.

Delving into the essence of *double-claiming* attack, we are surprised to observe the fatal imperfection in the scriptless implementation of timeout T designed for each frozen coin. Specifically, this scriptless implementation (e.g., the timed commitment) simply serves as a safeguard for the intended receiver of frozen coins. In other words, only this intended receiver can claim these frozen coins before the timeout T, while this privilege is not deprived after the timeout T. This inherent design flaw directly facilitates the occurrence of *double-claiming* attack. Even worse, this attack is general and extremely easy to carry out in all existing scriptless atomic swaps [13], [16] and multi-hop payment protocols [17], [18], [19], [20] (cf. Section 3 for detailed discussions).

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

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

1.1. Our Contributions

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

- Double-claiming attack. We investigate the security of scriptless atomic cross-chain swaps protocols [13], [16], identifying a new form of attack termed the *double-claiming* attack. This attack can directly break *atomicity* with a high probability, enabling a malicious user to simultaneously obtain the counterparty's frozen coins while also refunding his own coins. Furthermore, we conduct simulations of this attack to quantify its severity based on the real-word swap transactions [21]. What's worse, this attack is easy to carry out and can be naturally generalized to other scriptless cross-chain swaps protocols as well as payment channel networks [17], [18], [19], [20].
- Pipelined coins flow: a comprehensive solution to double-claiming attack. We introduce a novel paradigm of pipelined coins flow. As depicted in Fig.1, each frozen coin is analogous to a drop of water flowing along the unidirectional arrows. Informally, each frozen coin can only reach its intended receiver via the green pipeline for a successful swap. Otherwise, it will definitely return to its original owner through the red pipeline. This mechanism ensures that once a frozen coin has been claimed by a valid swap transaction, it will cease to

flow forward. However, after the timeout T, the frozen coin cannot revert back to the intended receiver at any point. Therefore, the concept of pipelined coins flow fundamentally resolves the issues associated with double-claimed frozen coins. As a byproduct, the framework of pipelined coins flow can serve as a foundation for designing secure scriptless multi-hop swaps (including the multi-hop payments).

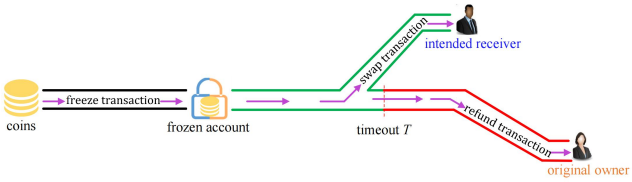


Figure 1: The pipelined life-cycle of swapped coins.

- PipeSwap: the realization of pipelined coins flow. Our central idea revolves around the timely release of a swap transaction for frozen coins, aimed at ensuring successful completion of swaps or, in the event of failure, facilitating the definitive refund of these frozen coins. By introducing the "two-hop completion" method, specifically the *two-hop swap* and *two-hop refund* techniques, we present an atomic cross-chain swaps protocol called PipeSwap that realizes the pipelined flow for frozen coins. Moreover, we refine the definition of *atomicity* by modeling it within the Global Universal Composability framework and provide a formal security proof of PipeSwap. Notably, PipeSwap is universal as it relies solely on the minimal scripting required for signature verification, while also preserving the property of *fungibility*, ensuring that an observer cannot distinguish a swap transaction from a standard one. The comparisons with prior approaches are shown in TABLE 1.
- Implementation. We develop a proof-of-concept implementation of PipeSwap for both Schnorr and ECDSA, and conduct extensive experiments to evaluate the overhead. The results demonstrate that our design exhibits high efficiency and best suitability. Specifically, PipeSwap achieves the running time of less than 1.7 seconds and communication costs of less than 7 kb. Remarkably, while providing enhanced security against *double-claiming* attack, compared to Universal Atomic Swaps [13], PipeSwap only takes a few milliseconds more for completing the second hop of swap/refund operations.

TABLE 1: The comparisons among related works*

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

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

1.2. Technique Overview

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

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

Two-hop swap. The newly introduced freezing structure inspires us to develop a *two-hop swap* method for claiming frozen coins β. A pre-swap transaction is designed for claiming frozen coins ε, while the actual swap transaction of frozen coins β takes the corresponding pre-transaction as one of its inputs. Essentially, puzzle Y $((\mathrm{Y}, y) \in \mathcal{R})$ is inserted in the signature of pre-swap transaction, rather than in the swap transaction of frozen coins β. This *two-hop swap* method effectively compels user $\mathcal{P}_0$ to timely post the requisite pre-swap transaction, which not only facilitates user $\mathcal{P}_0$ to generate a swap transaction of frozen coins β but also enables user $\mathcal{P}_1$ to generate a swap transaction of frozen coins α upon the release of witness y.

Two-hop refund. Relying solely on the *two-hop swap* design is insufficient to guarantee the pipelined coins flow. Therefore, we further propose the corresponding *two-hop refund* method for reclaiming frozen coins β. Specifically, the frozen coins ε are firstly refunded by a pre-refund transaction after time $\overline{T}_1 < T_1 - 2\delta - 2\varphi$, where parameters δ and φ are respectively the upper bound of network delivery delay and confirmation latency in the underlying blockchain $\mathbb{B}_1$. Then, upon timeout T_1, the remaining frozen coins $\beta - \varepsilon$ are refunded by a final refund transaction. This *two-hop refund* method effectively compels user $\mathcal{P}_0$ to timely post the pre-swap transaction before time $\overline{T}_1$, as any malicious delays during this process would result in the frozen coins ε being reclaimed by a pre-refund transaction, ultimately leading to user $\mathcal{P}_1$ refunding coins β after timeout T_1.

2. Preliminaries and Background

We begin with revisiting the blockchain transaction processing mechanisms that are widely adopted by the majority of current blockchains (e.g., Bitcoin [1], Zcash [22], Monero [23] and Ethereum [2]). Following this, we provide a concise overview of Universal Atomic Swaps [13].

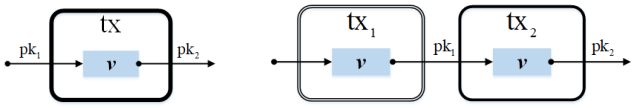
2.1. The Blockchain Transaction

Transactions.[1] A transaction tx that transfers coins of value v from account pk_1 to pk_2 can be formally expressed as $\mathrm{tx} := \mathrm{tx}(\mathrm{pk}_1, \mathrm{pk}_2, v)$. The validity of a transaction tx is determined as follows: (1) there are sufficient coins available in account pk_1, (2) it is signed with a signature σ that verifies w.r.t. pk_1, and (3) there has not been any conflicting transaction tx' (i.e., transaction tx cannot spend the same coins as those spent by tx'). The coins in account pk_2 can only be further spent with a signature w.r.t. pk_2. Additionally, the conditions of spending coins can be some scripts supported by the underlying blockchains, such as the timelock script and HTLC, but in this paper we focus on the scriptless ones. We utilize a transaction chart to visualize the coins flow within a transaction tx (cf. Fig.2(a)). The rounded rectangle represents transaction tx with the incoming arrow denoting input and the blue box labeled with value v representing the amount of coins involved, whose spending condition is written above the outgoing arrow.

Blockchain (cf. Fig.8). A blockchain can be used as an append-only bulletin board $\mathcal{T}$ (i.e., the transaction mempool) for storing the published transactions, while also serving as a trusted ledger $\mathcal{L}$ for recording all the unspent coins associated with each account. Essentially, the blockchain is built and maintained by parties (e.g., the miners or stakeholders) who compete to propose a candidate block containing a sequence of transactions. Particularly, when confronted with a transaction that conflicts with one already stored in $\mathcal{T}$ (i.e., both transactions claiming the same coins) and has the same transaction fees, parties will automatically deem the later-arriving transaction as invalid. In cases where both conflicting transactions are received simultaneously, a random selection is made to determine which of the two will be considered valid. Furthermore, we strictly differentiate the *parties* responsible for securing the underlying blockchains from the *users* participating in the cross-chain swaps protocols supported by these blockchains. Therefore, the (honest) parties do not concern themselves with the stories behind each individual transaction, and from their perspective, all valid transactions are treated equally. Consequently, it is reasonable that among conflicting transactions $\{\mathrm{tx}, \mathrm{tx}'\}$, the transaction tx' is finally confirmed even if it is generated by a malicious user.

Valid transaction vs Confirmed transaction. The transaction confirmation latency $\varphi > 0$ of the underlying blockchain (e.g., it is about one hour in Bitcoin) necessitates a clear distinction between the notions of *valid transaction* and *confirmed transaction*. Without loss of generality, a valid transaction $\mathrm{tx} \in \mathcal{T}$ can be finally confirmed (i.e., $\mathrm{tx} \in \mathcal{L}$) if it has been in $\mathcal{T}$ for time φ. For the clear presentation, we use double-bordered rectangles and single-bordered rectangles to represent the confirmed transactions and valid transactions, respectively (cf. Fig.2(b)).

1. In a manner akin to Universal Atomic Swaps [13], we assume that an agreement regarding the transaction fees for all swap-related transactions has been reached among all involved users.



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

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

Figure 2: The transaction flow.

Once the transaction processing mechanism of the underlying blockchain is understood, it becomes imperative to focus on a straightforward yet practical scenario illustrated in Fig.3. Since a user holding secret key sk_1 can sign any transactions to spend the coins in account pk_1 at his will, there is no way to prevent the malicious payer from generating two conflicting transactions tx_1 and tx_2. Here, transaction tx_2 would be used to pay the intended receiver, while transaction tx_1 would be used to transfer these coins back to the payer. In the fortunate scenario where $t_2 < t_1$, transaction tx_2 defeating tx_1 can be included by bulletin board $\mathcal{T}$, ultimately enabling the receiver to obtain the desired coins. However, when $t_1 = t_2$, the bulletin board $\mathcal{T}$ has an approximately 50% probability of prioritizing transaction tx_1, which leads to coin loss for the receiver. Even worse, when the network delivery delay is under adversarial control (e.g., the δ-synchronous network [24]), the malicious transaction tx_1 would outrun tx_2 with a landslide.

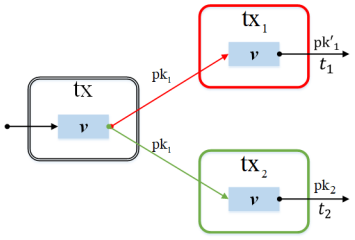


Figure 3: Transaction tx is double-claimed. The green transaction tx_2 is honestly signed w.r.t. pk_1 and received by $\mathcal{T}$ at time t_2, while the red transaction tx_1 is maliciously signed w.r.t. pk_1 and received by $\mathcal{T}$ at time t_1.

By leveraging the above observation, we need to be cautious about certain time points that could result in the same coins being doubly claimed by two conflicting transactions. This is especially critical when it comes to realizing atomic cross-chain swaps in a scriptless manner (see Section 3).

2.2. Atomic Cross-Chain Swaps

An atomic cross-chain swaps protocol facilitates two mutually distrusting users $\mathcal{P}_0$ and $\mathcal{P}_1$, who respectively own coins α and β in two distinct blockchains $\mathbb{B}_0$ and $\mathbb{B}_1$, to securely exchange coins. We now review the design of Universal Atomic Swaps [13] (cf. Fig.4). Similar to other works

in this area [16], [14], [25], [26], the users are connected with authenticated communication channels with guaranteed delivery of exactly one round and the malicious user (e.g., user $\mathcal{P}_0$ or $\mathcal{P}_1$) can deviate arbitrarily from the protocol. It takes the underlying blockchain as a global ledger functionality, where the communication network is under adversarial control to delay or reorder messages within δ rounds (cf. Fig.8 for a formal definition).

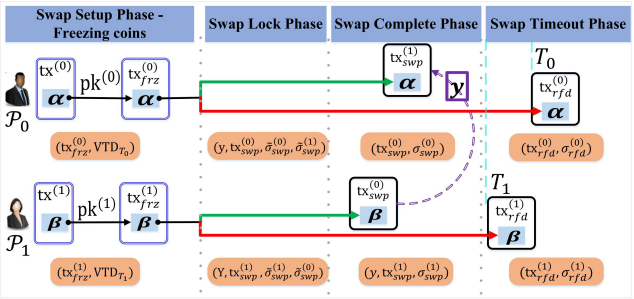


Figure 4: Universal Atomic Swaps: execution flow of users $\mathcal{P}_0$ and $\mathcal{P}_1$ in an α-to-β swap. The orange rectangle stores messages received by each user in the corresponding phase.

We use the hereunder notations: (1) an item with superscript $\{(i\ 1-i)|i \in \{0,1\}\}$ is involved in the payment from user $\mathcal{P}_i$ to $\mathcal{P}_{1-i}$; (2) an item with the subscript $\in \{frz, swp, rfd\}$ respectively refers to the freeze, swap and refund operations. Informally, it consists of four phases described as follows:

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

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

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

Swap Timeout Phase: After the timeout T_1, if coins β are still locked in the frozen account $\mathrm{pk}^{(10)}$, user $\mathcal{P}_1$ enters the Swap Timeout Phase to post a refund transaction $\mathrm{tx}_{rfd}^{(1)}$ to reclaim coins β. Similarly, after the timeout T_0, user $\mathcal{P}_0$ can post a refund transaction $\mathrm{tx}_{rfd}^{(0)}$. Therefore, the frozen coins are respectively refunded to their original owners (as indicated by the red arrows).

Security analysis. We summarize the security analysis of Universal Atomic Swaps and refer readers to [13] for the detailed proofs:

Successful Swap: If user $\mathcal{P}_0$ honestly posts a swap transaction $\mathrm{tx}_{swp}^{(0)}$ to claim coins β before the timeout T_1, the released witness y enables user $\mathcal{P}_1$ to successfully claim coins α via a swap transaction $\mathrm{tx}_{swp}^{(1)}$.

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

3. The Double-Claiming Attack

3.1. Reasons that Lead to the Attack

Reason 1: After the timeout T, the frozen coins can be claimed by both the original owner and intended receiver. It should be noted that the frozen coins can only be reclaimed by a refund transaction after the predefined timeout T, while the intended receiver retains the ability to claim these coins both before and after the timeout T by generating a swap transaction. In other words, there exists no mechanism in place to revoke the intended receiver's ability of claiming the frozen coins after the timeout T, which is the source of *double-claiming* attack.

Reason 2: The transaction balance security. Independent of the inner workings in cross-chain swaps protocols, the total balances of all accounts in the underlying blockchain remain unchanged. This means that the coins can only be equivalently transferred from some accounts to new ones. Therefore, only one of the conflicting transactions (i.e., either the swap or refund transaction involving the same frozen coins) can be finally confirmed by the underlying blockchain.

3.2. Attack Description

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

Note that the *double-claiming* attack fundamentally differs from the *double-spending* attack [27]. The former is extremely cost-effective, allowing a malicious user with the ability of signing transactions to obstruct the confirmation of

an honest transaction. Whereas the latter enables a malicious miner, who possesses and expends sufficient resources (e.g., the computational power [1] or stakes [28]), to confirm both spending transactions of the same coins. Importantly, the *double-claiming* attack can work in all communication network models of the underlying blockchains (e.g., the δ-synchrony [29], partial synchrony [30] and asynchrony [31]). Moreover, as the synchronicity level of the underlying blockchain weakens (i.e., transitioning from synchrony to asynchrony), this attack has an increased probability of success, which can be detailed as follows.

How the *Double-Claiming* Attack Works. Fig.5 illustrates how the malicious user $\mathcal{P}_0$ initiates a *double-claiming* attack on Universal Atomic Swaps [13]. Initially, users $\mathcal{P}_0$ and $\mathcal{P}_1$ launch an α-to-β swap by successfully freezing their respective coins α and β, followed by completing the Swap Lock Phase. Subsequently, the malicious user $\mathcal{P}_0$ goes offline. After the timeout T_1, user $\mathcal{P}_1$ enters the Swap Timeout Phase with the intention of reclaiming coins β through a refund transaction $\mathrm{tx}_{rfd}^{(1)}$. Concurrently, the malicious user $\mathcal{P}_0$ comes back online and posts a swap transaction $\mathrm{tx}_{swp}^{(0)}$, aiming to invalidate the refund transaction $\mathrm{tx}_{rfd}^{(1)}$. If the swap transaction $\mathrm{tx}_{swp}^{(0)}$ is published, or even more concerning, the malicious user $\mathcal{P}_0$ colludes with the adversary $\mathcal{A}_{\mathbb{B}_1}$ of blockchain $\mathbb{B}_1$ to delay the publication of refund transaction $\mathrm{tx}_{rfd}^{(1)}$ up to δ rounds, then the swap transaction $\mathrm{tx}_{swp}^{(0)}$ will effectively conceal $\mathrm{tx}_{rfd}^{(1)}$, ultimately leading to its final confirmation with an absolute advantage. After the timeout T_0, the malicious user $\mathcal{P}_0$ can also enter the Swap Timeout Phase and post a refund transaction $\mathrm{tx}_{rfd}^{(0)}$ to successfully reclaim coins α[2]. This attack would result in the malicious user $\mathcal{P}_0$ illicitly acquiring double assets and the honest user $\mathcal{P}_1$ losing coins α, thereby violating the *atomicity*.[3]

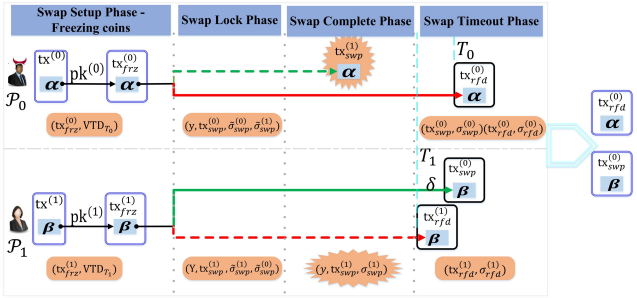


Figure 5: The *double-claiming* attack works successfully in Universal Atomic Swaps, resulting in the final confirmation of both transactions $\mathrm{tx}_{rfd}^{(0)}$ and $\mathrm{tx}_{swp}^{(0)}$.

Where the Timeouts T_0 and T_1 Fall Short. We present a detailed discussion of the timeouts T_0 and T_1 adopted in Universal Atomic Swaps [13] and related works [16], [17], [18], [19], [20], illustrating their insufficiency in protecting the honest user from losing coins in the double claiming scenario.

For the timing hardness parameters T_0 and T_1, Universal Atomic Swaps suggest that the parameter Δ (where $T_0 = T_1 + \Delta$) must be sufficiently large to tolerate the time differences in opening the VTD commitments. Essentially, the parameter Δ is designed to prevent the malicious user $\mathcal{P}_0$ from stealing coins β if he can open the VTD commitment earlier than expected. For example, if the malicious user $\mathcal{P}_0$ opens VTD commitment prior to the timeout T_1, he could first reclaim coins α with a refund transaction $\mathrm{tx}_{rfd}^{(0)}$, subsequently followed by posting a swap transaction $\mathrm{tx}_{swp}^{(0)}$ aimed at stealing coins β. Similarly, if the malicious user $\mathcal{P}_0$ opens VTD commitment after the timeout T_1, yet still notably before the timeout T_0, he could first claim coins β with a swap transaction $\mathrm{tx}_{swp}^{(0)}$, and then post a refund transaction $\mathrm{tx}_{rfd}^{(0)}$ to reclaim coins α [4]. Therefore, the parameter Δ is set to a sufficiently large value to ensure that there is enough time for confirming the swap transaction $\mathrm{tx}_{swp}^{(1)}$ before user $\mathcal{P}_0$ can generate a refund transaction $\mathrm{tx}_{rfd}^{(0)}$.

Nonetheless, the sufficiently large parameter Δ does little to mitigate the *double-claiming* attack, leaving the honest user $\mathcal{P}_1$ still susceptible to coin losses in the scenario illustrated in Fig.5. A strawman strategy may partially alleviate such an attack by allowing the honest user $\mathcal{P}_1$ to revert to the Swap Complete Phase if he fails to reclaim coins β after the timeout T_1 (i.e., once the swap transaction $\mathrm{tx}_{swp}^{(0)}$ has been confirmed). In this case, user $\mathcal{P}_1$ can still generate a swap transaction $\mathrm{tx}_{swp}^{(1)}$, and the parameter Δ guarantees its final confirmation before the time that malicious user $\mathcal{P}_0$ can reclaim coins α (i.e., the timeout T_0). Clearly, this result underscores insecurity of the refund mechanism inherent in this swaps protocol, and incorporating this reverting mechanism for honest users cannot fundamentally resolve the issue of *double-claiming* attack.

3.3. Analysis of the Attack

Generality of the Attack. It should be emphasized that the *double-claiming* attack is not exclusive to Universal Atomic Swaps, but can also be generally applied to other scriptless cross-chain swaps protocols (e.g., Sweep-UC [16]) and multi-hop payments [17], [18], [19], [20], all of which involve a predefined timeout T to realize the payment expire. In such frameworks, the primary vulnerability leading to the *double-claiming* attack is that the two competing users (e.g., $\mathcal{P}_0$ and $\mathcal{P}_1$) simultaneously hold the ability of claiming the same frozen coins immediately after the timeout T.

Quantitative Effect of the *double-claiming* Attack. We assume that users $\mathcal{P}_0$ and $\mathcal{P}_1$ are running Universal Atomic Swaps protocol [13], where both users have locked the same amount of coins. The malicious user $\mathcal{P}_0$ initiates

2. This holds true even if the swap transaction $\mathrm{tx}_{swp}^{(0)}$ has been confirmed by blockchain $\mathbb{B}_1$.

3. Even if this attack fails, the malicious user will never incur any loss of coins.

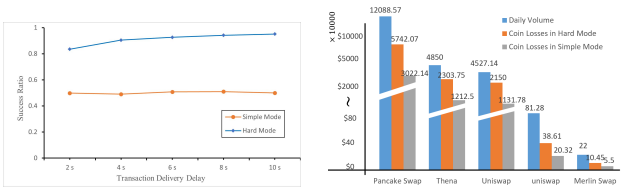
4. Provided that the swap transaction $\mathrm{tx}_{swp}^{(1)}$ is not published by the underlying blockchain $\mathbb{B}_0$, the malicious user $\mathcal{P}_0$ still has the chance to reclaim coins α by doubly claiming them.

the *double-claiming* attack by competing with honest user $\mathcal{P}_1$ for transaction confirmation priority. We perform the following simulation on the same device: The message delivery delay is set to $\delta = 2, 4, 6, 8, 10$. We simulate two nodes competing for message delivery in both the Simple and Hard modes.

Simple Mode: Both the conflicting transactions $\mathrm{tx}_{swp}^{(0)}$ and $\mathrm{tx}_{rfd}^{(1)}$ experience a deliberate delay within time δ before reaching the mempools of miners on blockchain $\mathbb{B}_1$.

Hard Mode: To make matters worse, the malicious user $\mathcal{P}_0$ may collude with the adversary $\mathcal{A}_{\mathbb{B}_1}$ of the underlying blockchain $\mathbb{B}_1$. This collusion leads to the instant delivery of the malicious swap transaction $\mathrm{tx}_{swp}^{(0)}$, while the honest refund transaction $\mathrm{tx}_{rfd}^{(1)}$ experiences a random delay ranging from 0 to δ.

For each δ, we run 1000 rounds of competition and repeat this process 10 times to obtain the averaged results (the source code is available on Github [32]). It is important to note that the honest refund transaction $\mathrm{tx}_{rfd}^{(1)}$ has a $\frac{1}{\delta+1}$ probability of being delivered without delay, and a $\frac{1}{2(\delta+1)}$ probability of causing the attack to fail.



(a) The success ratio of *double-claiming* attack.

(b) The coin losses resulted by *double-claiming* attack.

Figure 6: Simulated effect of the *double-claiming* attack.

We present our results in Fig.6 to illustrate the effect of the *double-claiming* attack. Specifically, the malicious swap transaction $\mathrm{tx}_{swp}^{(0)}$ can outrun the honest refund transaction $\mathrm{tx}_{rfd}^{(1)}$ with probabilities of approximately 50% and 95% respectively in the Simple and Hard modes (cf. Fig.6(a)). Furthermore, we take a weekly snapshot (March 9–15, 2025) of the top five real-world deployed exchange platforms supporting multi-cryptocurrency swaps, including Pancake Swap, Thena, Uniswap, uniswap, and Merlin Swap protocols [21]. Although these platforms rely on custom-designed contracts, their transaction types align closely with those of scriptless protocols, e.g., Universal Atomic Swaps. They provide publicly available, high-quality transaction data that accurately reflects real-world market demands, making them suitable for quantitative analysis and research. By leveraging the monetary value of historical swap transactions as a representative dataset (i.e., daily volume) along with our simulation results, we quantify the practical impact of attacks. Assuming that these transactions are executed by a scriptless decentralized exchange protocol such as Universal Atomic Swaps, the *double-claiming* attack could lead to coin losses amounting to $\$55,000 - \$30,221,428$ and $\$104,500 - \$57,420,714$ for honest users within 24 hours in the Simple and Hard modes respectively (cf. Fig.6(b)).

Responsible Disclosure. We have notified the *double-claiming* attack to the Merlin Swap's developers.

4. Our Solution in a Nutshell

As previously elaborated, the core challenge posed by the *double-claiming* attack lies in the fact that the frozen coins β can be simultaneously claimed by both users $\mathcal{P}_0$ and $\mathcal{P}_1$ after the timeout T_1. Therefore, our goal is to restrict user $\mathcal{P}_0$ from claiming coins β after the timeout T_1, thereby ensuring their definitive and timely refund to the original owner $\mathcal{P}_1$.

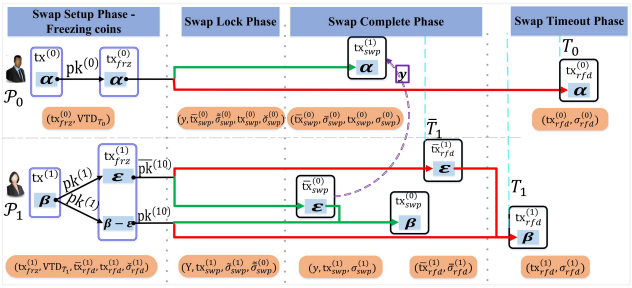


Figure 7: PipeSwap: the execution flow of users $\mathcal{P}_0$ and $\mathcal{P}_1$ in an α-to-β swap (cf. Remark 1).

Our straightforward solution is to realize the pipelined coins flow (cf. Fig.1) of frozen coins β, that is, the corresponding swap transaction $\mathrm{tx}_{swp}^{(0)}$ and refund transaction $\mathrm{tx}_{rfd}^{(1)}$ cannot both be valid. As depicted in Fig.7, the key idea is how to force the timely release of witness y (as indicated by the purple dotted arrow), such that a valid transaction of frozen coins β between the swap transaction $\mathrm{tx}_{swp}^{(0)}$ and refund transaction $\mathrm{tx}_{rfd}^{(1)}$ can be predetermined before the timeout T_1. For this purpose, we introduce a "two-hop completion" method, known as *two-hop swap* and *two-hop refund*. The *two-hop swap* compels user $\mathcal{P}_0$ to post the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ at least φ time before posting a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$. Similarly, the corresponding *two-hop refund* requires user $\mathcal{P}_1$ to post a pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ (it is locked until time $\overline{T}_1$) before refunding the frozen coins β by a refund transaction $\mathrm{tx}_{rfd}^{(1)}$. Thus, the "two-hop completion" method forces user $\mathcal{P}_0$ to actively post a pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ before the time $\overline{T}_1$, as otherwise the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ could be confirmed and user $\mathcal{P}_0$ cannot generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ at any time. As a result, before the timeout T_1, the final flow direction of frozen coins β can be determined in that if the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is finally confirmed, the frozen coins β can only be claimed by user $\mathcal{P}_0$; otherwise, they can only be reclaimed by user $\mathcal{P}_1$. Now we walk through how to realize the "two-hop completion" method.

First ingredient: splitting frozen coins β into two parts. To ensure the directional flow of frozen coins β, we divide them into two distinct frozen accounts with values ε (where

$\varepsilon > 0$ is an arbitrarily small amount, i.e., $\varepsilon \to 0$) and $\beta - \varepsilon$ respectively. Specifically, coins ε and $\beta - \varepsilon$ can only be further spent with the respective secret keys $\overline{\mathrm{sk}}^{(10)}$ and $\mathrm{sk}^{(10)}$, which are shared between users $\mathcal{P}_0$ and $\mathcal{P}_1$.

Second ingredient: two-hop swap. To prevent the malicious user $\mathcal{P}_0$ from suddenly releasing the swap transaction $\mathrm{tx}_{swp}^{(0)}$, we introduce a new swap method called *two-hop swap*. Specifically, user $\mathcal{P}_0$ can only generate a valid swap transaction $\mathrm{tx}_{swp}^{(0)}$ after the corresponding pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ has been confirmed by the underlying blockchain $\mathbb{B}_1$. The pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is jointly pre-signed by both users with the respective key shares (i.e., $\overline{\mathrm{sk}}_0^{(10)}$, $\overline{\mathrm{sk}}_1^{(10)}$) and puzzle Y, where the statement-witness pair $(\mathrm{Y}, y) \in \mathcal{R}$ is selected by user $\mathcal{P}_0$. Moreover, the swap transaction $\mathrm{tx}_{swp}^{(0)}$ takes $\overline{\mathrm{tx}}_{swp}^{(0)}$ as one of its inputs, and is jointly signed by both users with their respective key shares $\mathrm{sk}_0^{(10)}$ and $\mathrm{sk}_1^{(10)}$. Essentially, the validity of swap transaction $\mathrm{tx}_{swp}^{(0)}$ implies that the witness y has been released via the confirmed pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ and user $\mathcal{P}_1$ can generate a valid swap transaction $\mathrm{tx}_{swp}^{(1)}$ at least φ time before the timeout T_1.

Third ingredient: two-hop refund. To predetermine a valid transaction between the swap transaction $\mathrm{tx}_{swp}^{(0)}$ and refund transaction $\mathrm{tx}_{rfd}^{(1)}$ before the timeout T_1, we present the corresponding *two-hop refund* method. In this way, user $\mathcal{P}_1$ can only reclaim the coins β with a refund transaction $\mathrm{tx}_{rfd}^{(1)}$, contingent upon the confirmation of pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$, which is locked until time $\overline{T}_1$. This implies that once the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is confirmed, it would be impossible to generate a valid refund transaction $\mathrm{tx}_{rfd}^{(1)}$ even after the timeout T_1. Conversely, the refund transaction $\mathrm{tx}_{rfd}^{(1)}$ is always possible if no pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ has been confirmed.

So far, the final flow direction of frozen coins β can be determined before the timeout T_1, leaving only a minor issue to be addressed. Specifically, if user $\mathcal{P}_0$ posts the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ close to timepoint $\overline{T}_1$, the malicious user $\mathcal{P}_1$ can also initiate the *double-claiming* attack by posting the pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ immediately after the time $\overline{T}_1$ [5]. To mitigate this issue, we simply let the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ be posted at least $\delta + \varphi$ time units before the time $\overline{T}_1$.

Remark 1. *The time parameters are defined as follows: $\overline{T}_1 > t + 3\varphi$ is sufficiently large to confirm the frozen transactions $\mathrm{tx}_{frz}^{(0)}$ and $\mathrm{tx}_{frz}^{(1)}$, as well as the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$. Specifically, we have $T_1 > \overline{T}_1 + 2\delta + 2\varphi$ for confirming either a pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$, or a delayed posted pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ along with a swap transaction $\mathrm{tx}_{swp}^{(1)}$ before timeout T_1. Additionally, it holds that $T_0 > T_1 + \Delta$, where the parameter Δ is defined as in Universal Atomic Swaps. Herein, the parameters φ and δ represent the respective upper bounds of confirmation latency and a network delivery delay of blockchain $\mathbb{B}_1$, while t denotes the current time. For example, consider a real-world scenario in which blockchain $\mathbb{B}_1$ is Bitcoin, characterized by a confirmation latency φ of 60 minutes and the network delivery delay δ of 600 ms [33]. The timeout parameters are consequently set as follows: $\overline{T}_1 > t + 180$ minutes, $T_1 > t + 300.02$ minutes and $T_0 > t + 420.02$ minutes with respect to $\Delta = 120$ minutes.*

[5]. If the pre-swap transaction $\overline{\mathrm{tx}}_{swp}^{(0)}$ is not published by blockchain $\mathbb{B}_1$, the conflicting transaction $\overline{\mathrm{tx}}_{rfd}^{(1)}$ still has a chance to be published (cf. Fig.8).

5. Formal Definition of PipeSwap

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

5.1. Modeling the System and Threats

We now discuss the security model of generalized cross-chain swaps, which exactly follows the previous works in this area [16], [13], [34], [18], [19], [25]. In order to formally model the security of cross-chain swaps protocol, we adopt the Universal Composability (UC) model [35] framework and utilize the version with a global setup (GUC) [36]. We define a cross-chain swaps model involving two users $\{\mathcal{P}_0, \mathcal{P}_1\}$. The underlying blockchains $\mathbb{B} =: \{\mathbb{B}_0, \mathbb{B}_1\}$ serve as the global blockchain (ledger) functionalities $\mathcal{F}_{\mathbb{B}} := \{\mathcal{F}_{\mathbb{B}_0}, \mathcal{F}_{\mathbb{B}_1}\}$ (cf. Fig.8), each has a respective maximum confirmation delay of time φ_0 and φ_1 (i.e., $\varphi := \max\{\varphi_0, \varphi_1\}$). The GUC model specifies two worlds, a protocol Π is executed in the real world by the users, interacting with the adversary $\mathcal{A}$ and environment $\mathcal{Z}$. While in the ideal world, an ideal functionality $\mathcal{F}$ is executed by the users, interacting with the simulator $\mathcal{S}$ and environment $\mathcal{Z}$. We denote the ensemble corresponding to real world execution as $\mathrm{EXEC}(\lambda)^{\mathcal{F}_{\mathbb{B}}}_{\Pi,\mathcal{Z},\mathcal{A}}$, while that for ideal world execution is represented as $\mathrm{EXEC}(\lambda)^{\mathcal{F}_{\mathbb{B}}}_{\mathcal{F},\mathcal{Z},\mathcal{S}}$.

UC security. The protocol Π UC-realizes ideal functionality $\mathcal{F}$ w.r.t. a global blockchain $\mathcal{F}_{\mathbb{B}}$ if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that $\mathrm{EXEC}^{\mathcal{F}_{\mathbb{B}}}_{\Pi,\mathcal{Z},\mathcal{A}} \approx \mathrm{EXEC}^{\mathcal{F}_{\mathbb{B}}}_{\mathcal{F},\mathcal{Z},\mathcal{S}}$, where "$\approx$" denotes the computational indistinguishability.

The adversary. In the cross-chain swaps protocol, two designated users $\mathcal{P}_0$ and $\mathcal{P}_1$ are involved. The static adversary $\mathcal{A}$ selects and fully controls one user, either user $\mathcal{P}_0$ or $\mathcal{P}_1$, at the beginning of the process. Moreover, the Byzantine adversary $\mathcal{A}$ can arbitrarily deviate from the protocol specification (e.g., carrying out the *double-claiming* attack).

The communication network. We consider the synchronous communication between the participating users $\mathcal{P}_0$ and $\mathcal{P}_1$, implying that we assume a global clock functionality $\mathcal{F}_{clock}$ [24]. The cross-chain swaps protocol executes in rounds, ensuring that each user is aware of the current round and can expect messages to be received at a certain time. There is also a secure message transmission channel between users $\mathcal{P}_0$ and $\mathcal{P}_1$ modeled by the ideal functionality $\mathcal{F}_{smt}$ [35], [13].

The atomicity. At the end of a cross-chain swaps protocol, both users will either successfully exchange their coins or, in the event of a failed swap, the frozen coins will be refunded to their original owners. To enhance *atomicity*, we specifically aim to achieve that: each frozen coin should only be claimed by the intended receiver before its timeout T; if not claimed by then, it will be definitely refunded to the original owner after its timeout T.

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

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

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

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

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

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

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

Figure 8: Global ideal functionality that models blockchains.

The blockchain functionality. Similarly, we also assume the δ-synchronous communication network of the underlying blockchains, i.e., the network delivery delay is under adversarial control up to a known upper bound δ. Furthermore, the adversary $\mathcal{A}_{\mathbb{B}}$ of the underlying blockchains can delay or reorder the delivery of messages (transactions) within δ rounds, but cannot modify or drop them. We take the underlying blockchains $\mathbb{B}$ (i.e., blockchains $\mathbb{B}_0$ and $\mathbb{B}_1$) involved in the cross-chain swaps as a global ledger functionality $\mathcal{F}_{\mathbb{B}}$ with maximum confirmation delay time φ, which records the balance pk.bal of each account pk (i.e., the ledger $\mathcal{L}$) and maintains a trusted append-only bulletin board $\mathcal{T}$. The functionality $\mathcal{F}_{\mathbb{B}}$ offers interface $\mathrm{Valid}(\mathrm{tx})$ to verify the validity of a transaction (e.g., checking that the $\mathrm{inputs} \in \mathcal{L}$ have sufficient balance, the signatures are correct and tx does not conflict with any existing transactions in $\mathcal{T}$), uses interface $\mathrm{Publish}(\mathrm{tx}, t)$ to add a valid transaction tx to bulletin board $\mathcal{T}$ at time $t' \leq t + \delta$, where time t' is determined by the ideal adversary $\mathcal{S}_{\mathbb{B}}$. Especially, when receiving messages $\mathrm{Publish}(\mathrm{tx}_0, t_0)$ and $\mathrm{Publish}(\mathrm{tx}_1, t_1)$, which are intended to publish the conflicting transactions tx_0 and tx_1 that have the same transaction fees, the bulletin board $\mathcal{T}$ will randomly select one from $\{\mathrm{tx}_0, \mathrm{tx}_1\}$ if $t'_0 = t'_1$. The functionality $\mathcal{F}_{\mathbb{B}}$ confirms a transaction tx via interface $\mathrm{Confirm}(\mathrm{tx})$ (e.g., if the transaction tx has been recorded in bulletin board $\mathcal{T}$ for time φ, the functionality $\mathcal{F}_{\mathbb{B}}$ updates ledger $\mathcal{L}$ via transferring coins from input account pk_{in} to output account pk_{op}). Please refer to Fig.8 for more details.

5.2. Ideal Functionality of Cross-Chain Swaps

(A) Swap Setup Phase - Freezing Coins

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

(B) Swap Complete Phase

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

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

(C) Swap Timeout Phase

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

Figure 9: Ideal functionality that models cross-chain swaps.

We formalize the security of a cross-chain swaps protocol as an ideal functionality $\mathcal{F}$ (cf. Fig.9 and Fig.10), which interacts with users $\mathcal{P}_0$ and $\mathcal{P}_1$, environment $\mathcal{Z}$, ideal adversary $\mathcal{S}$, and the underlying blockchain functionalities $\mathcal{F}_{\mathbb{B}_0}$ and $\mathcal{F}_{\mathbb{B}_1}$. It consists of three procedures, each triggered by a message sent by user $\mathcal{P}_i$ $(i \in \{0, 1\})$ including the respective request and session identifier id.

(A) Swap Setup Phase-Freezing Coins: Users $\mathcal{P}_0$ and $\mathcal{P}_1$

//Subroutine **Freeze**

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

//Subroutine **Transfer**

Transfer$(\mathrm{id}, \mathrm{pk}_{\mathcal{F}}, \mathrm{sk}_{\mathcal{F}}, v, \mathrm{pk}_{swp})$: transfer frozen coins v from account $\mathrm{pk}_{\mathcal{F}}$ to pk_{swp} via generating a swap transaction $\mathrm{tx}_{swp} := \mathrm{tx}(\mathrm{pk}_{\mathcal{F}}, \mathrm{pk}_{swp}, v, \sigma)$ with secret key $\mathrm{sk}_{\mathcal{F}}$ and invoking interface $\mathrm{Confirm}(\mathrm{tx}_{swp})$ of blockchain functionality $\mathcal{F}_{\mathbb{B}}$. If transaction tx_{swp} has been confirmed by $\mathcal{F}_{\mathbb{B}}$ (i.e., $\mathrm{tx}_{swp} \in \mathcal{L}$), then respond $(\mathrm{Confirmed}, \mathrm{id})$.

//Subroutine **Unfreeze**

Unfreeze$(\mathrm{id}, \mathrm{pk}_{\mathcal{F}}, \mathrm{sk}_{\mathcal{F}}, v, \mathrm{pk}_{rfd})$: transfer frozen coins v from account $\mathrm{pk}_{\mathcal{F}}$ to pk_{rfd} via generating a refund transaction $\mathrm{tx}_{rfd} := \mathrm{tx}(\mathrm{pk}_{\mathcal{F}}, \mathrm{pk}_{rfd}, v, \sigma)$ with secret key $\mathrm{sk}_{\mathcal{F}}$ and invoking interface $\mathrm{Confirm}(\mathrm{tx}_{rfd})$ of blockchain functionality $\mathcal{F}_{\mathbb{B}}$. If transaction tx_{rfd} has been confirmed by $\mathcal{F}_{\mathbb{B}}$ (i.e., $\mathrm{tx}_{rfd} \in \mathcal{L}$), then respond $(\mathrm{Confirmed}, \mathrm{id})$.

Figure 10: The subroutines of ideal functionality $\mathcal{F}$.

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

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

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

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

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

Failed Swap: After the timeout T_i $(i \in \{0, 1\})$, if $\mathrm{b}^{(1-\mathrm{i})} = \perp$, the functionality $\mathcal{F}$ will enable user $\mathcal{P}_i$ to refund his frozen coins.

6. PipeSwap: Protocol Description

6.1. Cryptographic Building Blocks

We insist on the fundamental building blocks from [13], namely, the adaptor signature scheme [14] and verifiable timed discrete logarithm (VTD) scheme [15]. For the formal definitions, we refer the readers to Appendix B.

Adaptor signature. The adaptor signature scheme allows one user with secret key sk and a puzzle Y to generate a pre-signature $\tilde{\sigma}$ on a message m which by itself is not a valid signature, but can later be adapted into a valid signature σ if the user knows witness y (e.g., $(\mathrm{Y}, y) \in \mathcal{R}$). Additionally, the witness y can be further extracted by the pre-signature $\tilde{\sigma}$ and signature σ. Adaptor signature scheme is required to satisfy security properties of unforgeability, witness extractability and pre-signature adaptability.

Verifiable timed dlog (VTD). The VTD enables a committer to generate a timed commitment C with timing hardness T of a value $x \in \mathbb{Z}^*_q$ such that $H = \mathrm{g}^x$, where H is public and x is referred to as the dlog. (discrete logarithm) of H (wrt. group $\mathbb{G}$). The verifier checks the well-formedness of commitment C and can learn the value x by forcibly opening commitment C in time T. VTD is required to satisfy security properties of soundness and privacy.

In this work, we also use the adaptor signature scheme and VTD in a black-box manner, and direct readers to efficient constructions in [13], [14], [15], [37]. To elaborate further, following the approach presented in [13], we adopt the construction of adaptor signature in [14] with the underlying signature scheme being Schnorr or ECDSA, and the hard relation $\mathcal{R}$ being the discrete log (dlog) relation (i.e., the language is defined as $L_{\mathcal{R}_{\mathrm{dlog}}} := \{\mathrm{Y} | \exists y \in \mathbb{Z}^*_q, s.t.\ \mathrm{Y} = \mathrm{g}^y \in \mathbb{G}\}$). For the construction of VTD, the committer embeds the dlog.value y inside a time-lock puzzle Y, uses a non-interactive zero-knowledge proof (NIZK) to prove that Y can be solved in time T and the value y satisfies $\mathrm{Y} = \mathrm{g}^y$. An efficient construction of the NIZK [15] can be derived from the cut-and-choose techniques, Shamir secret sharing [38] and homomorphic time-lock puzzles [39].

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

6.2. Procedures of PipeSwap

In a general setting, users $\mathcal{P}_0$ (holding coins α on blockchain $\mathbb{B}_0$) and $\mathcal{P}_1$ (holding coins β on blockchain $\mathbb{B}_1$) aim to perform the α-to-β cross-chain swaps. As previously discussed (cf. Section 4), the *pipelined coins flow* of frozen

coins guarantees *atomicity*. The crux of securing PipeSwap lies in facilitating the timely release of witness y, which is achieved through three essential ingredients: *splitting frozen coins* β *into* $(\varepsilon, \beta-\varepsilon)$, a *two-hop swap* and a *two-hop refund*.

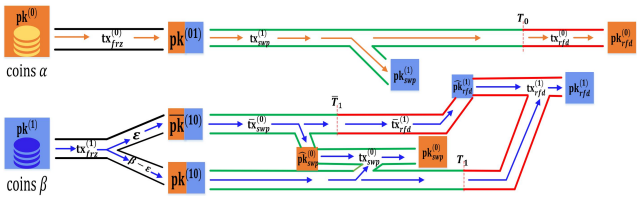


Figure 11: The pipelined coins flow of PipeSwap.

Protocol details. For ease of understanding, we depict the coins flow of PipeSwap in Fig.11 and provide a description of PipeSwap in Fig.12, summarizing the key points of each phase as follows:

(A) Swap Setup Phase-Freezing Coins: In this phase, users $\mathcal{P}_0$ and $\mathcal{P}_1$ transfer the swapped coins to the corresponding frozen accounts under their joint control. To be better prepared for the *two-hop swap* and *two-hop refund*, user $\mathcal{P}_1$ splits coins β into two frozen accounts $\overline{\mathrm{pk}}^{(10)}$ and $\mathrm{pk}^{(10)}$ with respective values of ε and $\beta-\varepsilon$. To ensure that the frozen coins β are refunded as expected, the pre-refund transaction $\overline{\mathrm{tx}}^{(1)}_{rfd} := \mathrm{tx}(\overline{\mathrm{pk}}^{(10)}, \widehat{\mathrm{pk}}^{(1)}_{rfd}, \varepsilon)$ is locked until time $\overline{T}_1 < T_1 - 2\delta - 2\varphi$ (meaning user $\mathcal{P}_0$ makes a timed commitment of secret key share $\overline{\mathrm{sk}}_0^{(10)}$ with timing hardness $\overline{T}_1$). Meanwhile, the refund transaction $\mathrm{tx}^{(1)}_{rfd} := \mathrm{tx}((\mathrm{pk}^{(10)}, \widehat{\mathrm{pk}}^{(1)}_{rfd}), \mathrm{pk}^{(1)}_{rfd}, \beta)$ is jointly signed with secret keys $\mathrm{sk}^{(10)}$ (held by users $\mathcal{P}_0$ and $\mathcal{P}_1$) and $\widehat{\mathrm{sk}}^{(1)}_{rfd}$ (held by user $\mathcal{P}_1$), and will become valid when the pre-refund transaction $\overline{\mathrm{tx}}^{(1)}_{rfd}$ has been confirmed. Additionally, user $\mathcal{P}_1$ commits to a timed refunding of frozen coins α by providing the secret key share $\mathrm{sk}_1^{(01)}$ with timing hardness T_0, which allows user $\mathcal{P}_0$ to generate a refund transaction $\mathrm{tx}^{(0)}_{rfd} := \mathrm{tx}(\mathrm{pk}^{(01)}, \mathrm{pk}^{(0)}_{rfd}, \alpha)$ after the timeout T_0.

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

(B2) Swap Complete Phase: User $\mathcal{P}_0$ posts a pre-swap transaction $\overline{\mathrm{tx}}^{(0)}_{swp}$ before the time $\overline{T}_1 - \delta - \varphi$. This confirmed pre-swap transaction $\overline{\mathrm{tx}}^{(0)}_{swp}$ ensures that user $\mathcal{P}_0$ can generate a valid swap transaction $\mathrm{tx}^{(0)}_{swp}$, also enables user $\mathcal{P}_1$ to generate a valid swap transaction $\mathrm{tx}^{(1)}_{swp}$ upon the release of witness y.

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

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

Successful Swap. The timely posted pre-swap transaction $\overline{\mathrm{tx}}^{(0)}_{swp}$ (i.e., before the time $\overline{T}_1 - \delta - \varphi$) ensures user $\mathcal{P}_0$ to generate a valid swap transaction $\mathrm{tx}^{(0)}_{swp}$, which will be definitely confirmed before the timeout T_1. Furthermore, the confirmation of pre-swap transaction $\overline{\mathrm{tx}}^{(0)}_{swp}$ facilitates user $\mathcal{P}_1$ in generating a swap transaction $\mathrm{tx}^{(1)}_{swp}$, which will also be definitely confirmed before the timeout T_1. As a result, both users can timely obtain their desired coins.

Failed Swap. We consider the following possible cases:

- The malicious user $\mathcal{P}_0$ does not initiate the swap operation (i.e., the pre-swap transaction $\overline{\mathrm{tx}}^{(0)}_{swp}$ is not posted). As a result, both users can timely refund their frozen coins after the respective timeouts T_1 and T_0.
- The malicious user $\mathcal{P}_0$ deliberately delays the posting of pre-swap transaction $\overline{\mathrm{tx}}^{(0)}_{swp}$, causing that this transaction cannot be confirmed before the time $\overline{T}_1$. This delay compels honest user $\mathcal{P}_1$ to post a pre-refund transaction $\overline{\mathrm{tx}}^{(1)}_{rfd}$ after the time $\overline{T}_1$. Specifically, if the pre-swap transaction $\overline{\mathrm{tx}}^{(0)}_{swp}$ is confirmed (potentially through collusion with the adversary of blockchain $\mathbb{B}_1$), honest user P_1's swap transaction $\mathrm{tx}^{(1)}_{swp}$ could be definitely confirmed before the timeout T_1. Conversely, if the pre-refund transaction $\overline{\mathrm{tx}}^{(1)}_{rfd}$ is confirmed, honest user $\mathcal{P}_1$ would be able to timely refund frozen coins β after the timeout T_1.

Overall, if the malicious user $\mathcal{P}_0$ succeeds in generating a valid swap transaction, it ensures that the honest user P_1's swap transaction will be confirmed before the timeout T_1. Conversely, should this not occur, the honest user P_1 is assured of refunding frozen coins after the timeout T_1. Furthermore, it is always guaranteed that the honest user $\mathcal{P}_0$'s swap transaction can be confirmed before the timeout T_1, thereby preventing the malicious user $\mathcal{P}_1$ from generating a valid refund transaction. Therefore, it can be confidently asserted that PipeSwap enhances the *atomicity* against the *double-claiming* attack, regardless of whether user $\mathcal{P}_0$ or $\mathcal{P}_1$ is malicious.

6.3. Evaluation and Comparison

We develop a prototypical C implementation to demonstrate the feasibility of our construction and evaluate its per-

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

(A) Swap Setup Phase - Freezing Coins

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

1) $\mathcal{P}_0$ runs Setup process (cf. Fig.13) and sends $(\mathrm{pk}_0^{(01)}, \overline{\mathrm{pk}}_0^{(10)}, \mathrm{pk}_0^{(10)}, (\mathrm{C}^{(1)}, \pi^{(1)})) \hookrightarrow \mathcal{P}_1$.
2) $\mathcal{P}_1$ runs Setup process (cf. Fig.13) and sends $(\mathrm{pk}_1^{(01)}, \overline{\mathrm{pk}}_1^{(10)}, \mathrm{pk}_1^{(10)}, (\mathrm{C}^{(0)}, \pi^{(0)})) \hookrightarrow \mathcal{P}_0$.

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

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

where secret keys corresponding to pk^{i1-i} and $\overline{\mathrm{pk}}^{(10)}$ are respectively $\mathrm{sk}^{i1-i} = \mathrm{sk}_i^{i1-i} \oplus \mathrm{sk}_{1-i}^{i1-i}$ and $\overline{\mathrm{sk}}^{(10)} = \overline{\mathrm{sk}}_1^{(10)} \oplus \overline{\mathrm{sk}}_0^{(10)}$.

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

1) $\mathcal{P}_0$ generates freeze transaction $\mathrm{tx}_{frz}^{(0)} := \mathrm{tx}(\mathrm{pk}^{(0)}, \mathrm{pk}^{(01)}, \alpha)$ and signature $\sigma_{frz}^{(0)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\mathrm{sk}^{(0)}, \mathrm{tx}_{frz}^{(0)})$, and then posts $(\mathrm{tx}_{frz}^{(0)}, \sigma_{frz}^{(0)})$ on blockchain $\mathbb{B}_0$ and starts solving $\mathrm{VTD.ForceOp}(\mathrm{C}^{(0)})$.
2) Users $\mathcal{P}_0$ and $\mathcal{P}_1$ jointly do the following:
   - $\mathcal{P}_1$ generates freeze transaction $\mathrm{tx}_{frz}^{(1)} := \mathrm{tx}(\mathrm{pk}^{(1)}, (\overline{\mathrm{pk}}^{(10)}, \mathrm{pk}^{(10)}), (\varepsilon, \beta - \varepsilon))$, pre-refund transaction $\overline{\mathrm{tx}}_{rfd}^{(1)} := \mathrm{tx}(\overline{\mathrm{pk}}^{(10)}, \widehat{\mathrm{pk}}_{rfd}^{(1)}, \varepsilon)$, refund transaction $\mathrm{tx}_{rfd}^{(1)} := \mathrm{tx}((\mathrm{pk}^{(10)}, \widehat{\mathrm{pk}}_{rfd}^{(1)}), \mathrm{pk}_{rfd}^{(1)}, \beta)$, and sends $(\mathrm{tx}_{frz}^{(1)}, \overline{\mathrm{tx}}_{rfd}^{(1)}, \mathrm{tx}_{rfd}^{(1)}) \hookrightarrow \mathcal{P}_0$;
   - $\mathcal{P}_0$ checks if $\mathrm{tx}_{frz}^{(1)}, \overline{\mathrm{tx}}_{rfd}^{(1)}, \mathrm{tx}_{rfd}^{(1)}$ are well formed (i.e., satisfying *two-hop refund* framework), and stops otherwise;
   - $\mathcal{P}_0$ and $\mathcal{P}_1$ run protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$ with input $(\mathrm{sk}_0^{(10)}, \mathrm{sk}_1^{(10)}, \mathrm{tx}_{rfd}^{(1)})$ (cf. Fig.13) and obtain signature $\check{\sigma}_{rfd}^{(1)}$;
   - $\mathcal{P}_1$ computes signature $\sigma_{frz}^{(1)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\mathrm{sk}^{(1)}, \mathrm{tx}_{frz}^{(1)})$ and posts $(\mathrm{tx}_{frz}^{(1)}, \sigma_{frz}^{(1)})$ on blockchain $\mathbb{B}_1$, and then starts solving $\mathrm{VTD.ForceOp}(\mathrm{C}^{(1)})$.

(B1) Swap Lock Phase

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

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

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

03 Users $\mathcal{P}_0$ and $\mathcal{P}_1$ run protocol $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$ with input $(\mathrm{sk}_0^{(01)}, \mathrm{sk}_1^{(01)}, \mathrm{Y}, \mathrm{tx}_{swp}^{(1)})$ (cf. Fig.13), and obtain pre-signature $\widetilde{\sigma}_{swp}^{(1)}$.

04 After step 03 is successful, $\mathcal{P}_0$ and $\mathcal{P}_1$ run protocol $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$ with input $(\mathrm{sk}_0^{(10)}, \mathrm{sk}_1^{(10)}, \mathrm{tx}_{swp}^{(0)})$ (cf. Fig.13) and obtain signature $\check{\sigma}_{swp}^{(0)}$, and then run protocol $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$ with input $(\overline{\mathrm{sk}}_0^{(10)}, \overline{\mathrm{sk}}_1^{(10)}, \mathrm{Y}, \overline{\mathrm{tx}}_{swp}^{(0)})$ (cf. Fig.13) and obtain pre-signature $\widetilde{\overline{\sigma}}_{swp}^{(0)}$.

(B2) Swap Complete Phase

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

06 If $\overline{\mathrm{tx}}_{swp}^{(0)}$ is not confirmed before time $\overline{T}_1$, $\mathcal{P}_1$ computes $\overline{\mathrm{sk}}_0^{(10)} \leftarrow \mathrm{VTD.ForceOp}(\mathrm{C}^{(1)})$, $\overline{\mathrm{sk}}^{(10)} := \overline{\mathrm{sk}}_0^{(10)} \oplus \overline{\mathrm{sk}}_1^{(10)}$, $\overline{\sigma}_{rfd}^{(1)} \leftarrow \Sigma_{\mathrm{SIG}}.\mathrm{Sig}(\overline{\mathrm{sk}}^{(10)}, \overline{\mathrm{tx}}_{rfd}^{(1)})$, and posts $(\overline{\mathrm{tx}}_{rfd}^{(1)}, \overline{\sigma}_{rfd}^{(1)})$ on blockchain $\mathbb{B}_1$.

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

(C) Swap Refund Phase

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

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

Figure 12: PipeSwap: atomic cross-chain swaps between users $\mathcal{P}_0$ and $\mathcal{P}_1$.

//The Setup process

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 13: The subroutines of protocol PipeSwap.

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

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

formance. We conduct experiments on a PC with the following configuration: CPU(Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz with 4 cores), RAM(16.0 GB) and OS(x64-based Windows). Basically, we instantiate the signatures of Schnorr and ECDSA over the secp256k1 curve, and set the transaction size to 250 bytes to approximate a basic Bitcoin transaction. We implement the two-party computation protocol for digital signature $\Gamma_{\mathrm{Sig}}^{\mathrm{SIG}}$, and use the implementations of adaptor signature $\Gamma_{\mathrm{AdpSig}}^{\mathrm{SIG}}$ and VTD respectively in [25] and [15]. The source code is available on Github [32].

Computation time. We measure the time of basic operations required in PipeSwap, with results shown in TABLE 2. Subsequently, the computation time required by both users together is recorded in TABLE 3. It is observed that (1) each instance of PipeSwap requires only 1.605 seconds for Schnorr and 1.624 seconds for ECDSA; (2) the computation time of Swap Setup-Freezing Phase accounts for more than 99%, as both users collaboratively complete two VTD computations with a statistical parameter of $n = 64$.

Communication overhead. We measure the communication overhead based on the amount of messages exchanged between users during the interactive algorithms in the Swap Setup Phase and Swap Lock Phase (cf. TABLE 4). Specifically, PipeSwap requires 6.4 kb for Schnorr and 7 kb for ECDSA, primarily dominated by that of respectively exchanging the VTD.Commit-proof pairs of secret key shares.

Efficiency comparison. To compare PipeSwap (cf. Fig.12) and Universal Atomic Swaps (cf. Fig.5 in [13]) w.r.t. the operations required by both users together, we conduct an evaluation of PipeSwap and Universal Atomic Swaps using the same settings, libraries, and security parameters for all cryptographic implementations. Before delving into details, let us clarify that Universal Atomic Swaps repeat the same operations for each swapped coin. Therefore, we limit our examination to the one-to-one atomic swap in [13]. The completion time for Universal Atomic Swaps is 1.6 seconds for Schnorr and 1.617 seconds for ECDSA, and the communication overhead is 5.1 kb for Schnorr and

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

*UAS (Universal Atomic Swaps) [13].

TABLE 4: The communication overheads (bytes)*

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

*UAS (Universal Atomic Swaps) [13].

5.4 kb for ECDSA. Based on this comparison, we find that PipeSwap is only $\leq$ 7 ms slower and incurs extra ≤ 1.6 kb communication overhead for preparing and signing two extra transactions (i.e., the pre-swap and pre-refund transactions), which remains within acceptable limits.

On-chain costs comparison. Compared to Universal Atomic Swaps, PipeSwap requires only one additional on-chain transaction to confirm either $\overline{\mathrm{tx}}^{(0)}_{swp}$ or $\overline{\mathrm{tx}}^{(1)}_{rfd}$. Based on the exchange rates as of February 16, 2025 (i.e., $97,173$ USD per BTC or $2,686$ USD per ETH), PipeSwap incurs an extra cost of approximately 0.0000125 BTC (≈ 1.20 USD) or 0.00021 ETH (≈ 0.56 USD), assuming a base fee of 10 Gwei.

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

7. Conclusions and Future Works

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

[14] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized channels from limited blockchain scripts and adaptor signatures," in *Advances in Cryptology–ASIACRYPT 2021*. Springer, 2021, pp. 635–664.

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

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

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

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

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

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

[21] "Footprint analytics," 2024, https://www.footprint.network/research/defi/dex-overview/dex-metrics?series_date=past90days~.

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

[23] 2023, https://thecharlatan.ch/Monero-Unlock-Time-Privacy.

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

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

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

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

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

[29] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2017, pp. 643–673.

[30] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[31] H. Attiya and J. Welch, *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004, vol. 19.

[32] "Implementation of pipeswap," 2024, https://github.com/Anqi333/pipeSwap.

[33] https://clm.kentik.com/?q=56cdefa10x.

[34] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 967–984.

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

[36] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*. Springer, 2007, pp. 61–85.

[37] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.

[38] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[39] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *Annual International Cryptology Conference*. Springer, 2019, pp. 620–649.

[40] R. Han, H. Lin, and J. Yu, "On the optionality and fairness of atomic swaps," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 62–75.

[41] V. Zakhary, D. Agrawal, and A. E. Abbadi, "Atomic commitment across blockchains," *arXiv preprint arXiv:1905.02847*, 2019.

[42] K. Narayanam, V. Ramakrishna, D. Vinayagamurthy, and S. Nishad, "Atomic cross-chain exchanges of shared assets," in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 2022, pp. 148–160.

[43] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 193–210.

[44] M. Borkowski, M. Sigwart, P. Frauenthaler, T. Hukkinen, and S. Schulte, "Dextt: Deterministic cross-blockchain token transfers," *IEEE Access*, vol. 7, pp. 111 030–111 042, 2019.

[45] K. Narayanam, V. Ramakrishna, D. Vinayagamurthy, and S. Nishad, "Generalized htlc for cross-chain swapping of multiple assets with co-ownerships," *arXiv preprint arXiv:2202.12855*, 2022.

[46] A. Kiayias and D. Zindros, "Proof-of-work sidechains," in *Financial Cryptography and Data Security: FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*. Springer, 2020, pp. 21–34.

[47] G. Verdian, P. Tasca, C. Paterson, and G. Mondelli, "Quant overledger whitepaper," *Release V0*, vol. 1, p. 31, 2018.

[48] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *the 2019 ACM SIGSAC Conference*, 2019.

[49] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[50] J. V. Bulck, D. Oswald, E. Marin, A. Aldoseri, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in *the 2019 ACM SIGSAC Conference*, 2019.

[51] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

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

[53] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," *CoRR, abs/1702.05812*, 2017.

[54] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," *ACM*, 2017.

[55] M. Backes and D. Hofheinz, "How to break and repair a universally composable signature functionality," in *International Conference on Information Security*. Springer, 2004, pp. 61–72.

Appendix A. Other Related Works

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

Essentially, a secure cross-chain swaps protocol between users $\mathcal{P}_0$ and $\mathcal{P}_1$ should fulfill two fundamental functionalities to guarantee the honest user $\mathcal{P}_0$ cannot lose coins (1) if user $\mathcal{P}_1$ has claimed the frozen coins of $\mathcal{P}_0$, $\mathcal{P}_0$ is able to claim the frozen coins of $\mathcal{P}_1$ before $\mathcal{P}_1$ can refund them; and (2) if user $\mathcal{P}_1$ is malicious, $\mathcal{P}_0$ can definitely refund his frozen coins. While *atomicity* is easily realized by the trusted third party, the blockchain community has made significant efforts to achieve the (fully) decentralized cross-chain swaps [42], [41], [43]. HTLC-based protocols use the rich scripting languages supported by the underlying blockchains to describe when and how the frozen coins can be unlocked [44], [45]. Subsequently, these protocols have been widely applied and deployed in practice [7], [6], [8]. However, they are far from the universal solution and suffer from the inherent drawbacks of HTLC, including the high execution costs and large transaction sizes. Additionally, since these transactions are easier to distinguish from the standard one that does not include any custom scripts, thus these protocols are in conflict with the blockchains that have already achieved privacy [12].

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

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

The studies of payment channel networks (PCN) enable any two users to complete payments even if they do not have a direct payment channel, and have become the most widely deployed solution for realizing blockchain scalability (e.g., lightning network [51]). Similarly, most existing PCNs proposals are restricted to the Turing complete scripting language [52], [53], [54] thus suffering from the inherent drawbacks of HTLC. Anonymous Multi-Hop Locks [18], lockable signatures [19] and $\mathrm{A^2L}$ [25] are recently proposed to construct the scriptless PCNs.

Appendix B. Definitions of Cryptographic Building Blocks

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

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

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

The security properties: (i) *unforgeability* is similar to the unforgeability of standard signature schemes [55], except that producing a forgery σ for some message m is hard even given a pre-signature $\tilde{\sigma}$ on m, with respect to some uniformly sampled instance $\mathrm{Y} \in \mathcal{L}_{\mathcal{R}}$; (ii) *witness extractability* guarantees that given a valid signature σ and a pre-signature $\tilde{\sigma}$ for m and an instance Y, one can efficiently extract a witness y for Y; and (iii) *pre-signature adaptability* guarantees that any valid pre-signature $\tilde{\sigma}$ generated with respect to an instance Y, can be adapted into a valid signature σ if given the witness y for the instance Y.

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

The security properties: (i) *soundness* guarantees that the verifier can be convinced that, given commitment C, the ForceOp algorithm will produce the committed dlog. value x in time T; and (ii) *privacy* guarantees that all polynomial time algorithms whose running time is at most $t < T$ succeed in extracting x from the commitment C and proof π with at most negligible probability.

Appendix C. Security Analysis

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

Proof. We now prove that protocol PipeSwap (Fig.7) UC-realizes the cross-chain swaps ideal functionality $\mathcal{F}$ (Fig.9).

To show the indistinguishability between the ideal world and the real world, we construct a simulator $\mathcal{S}$ to simulate the protocol PipeSwap in the real world while interacting with the ideal functionality $\mathcal{F}$. At the beginning, $\mathcal{S}$ corrupts one user of $\{\mathcal{P}_0, \mathcal{P}_1\}$ as $\mathcal{A}$ does. We begin with the real world protocol execution, gradually change the simulation in these hybrids and then we argue about the proximity of neighbouring experiments.

Hybrid $\mathcal{H}_0$: It is the same as the real world protocol execution (Fig.12);

Hybrid $\mathcal{H}_1$: It is the same as the above execution except that the 2PC protocol $\Gamma_{\text{Sig}}^{\text{SIG}}$ in the Swap Setup Phase and Swap Lock Phase to generate signatures is simulated using the 2PC simulators $\mathcal{S}_{2pc,1}$ for the corrupted user (notice that such a simulator exists for a secure 2PC protocol $\Gamma_{\text{Sig}}^{\text{SIG}}$);

Hybrid $\mathcal{H}_2$: It is the same as the above execution except that the 2PC protocol $\Gamma_{\text{AdpSg}}^{\text{SIG}}$ in the Swap Lock Phase to generate pre-signatures is simulated using the 2PC simulators $\mathcal{S}_{2pc,2}$ for the corrupted user;

Hybrid $\mathcal{H}_3$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_1$ and outputs a valid swap transaction $(\text{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)})$ before the simulator initiates swap operation on behalf of $\mathcal{P}_0$, the simulator aborts;

Hybrid $\mathcal{H}_4$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_0$ and outputs a valid swap transaction $(\text{tx}_{swp}^{(0)}, \sigma_{swp}^{(0)})$ before timeout T_1. The simulator outputs $(\text{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)})$ and if $\Sigma_{\text{SIG}}.\text{Vf}(\text{pk}^{(01)}, \text{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)}) \neq 1$, the simulator aborts;

Hybrid $\mathcal{H}_5$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_0$ and initiates the swap operation $(\text{tx}_{swp}^{(0)}, \sigma_{swp}^{*(0)})$ after timeout T_1. The simulator outputs $(\text{tx}_{rfd}^{(1)}, (\check{\sigma}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}))$ if $\Sigma_{\text{SIG}}.\text{Vf}(\widehat{\text{pk}}_{rfd}^{(1)}, \text{tx}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}) \neq 1$ or $\Sigma_{\text{SIG}}.\text{Vf}(\text{pk}^{(10)}, \text{tx}_{rfd}^{(1)}, \check{\sigma}_{rfd}^{(1)}) \neq 1$, the simulator aborts;

Hybrid $\mathcal{H}_6$: It is the same as the above execution except that the adversarial $\mathcal{P}_0$ outputs a valid refund transaction $(\text{tx}_{rfd}^{(0)}, \sigma_{rfd}^{(0)})$ before timeout T_0, the simulator aborts;

Hybrid $\mathcal{H}_7$: It is the same as the above execution except that the adversarial $\mathcal{P}_1$ outputs a valid refund transaction $(\text{tx}_{rfd}^{(1)}, \sigma_{rfd}^{(1)})$ before timeout T_1, the simulator aborts;

Hybrid $\mathcal{H}_8$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_0$ and the simulator obtains $(\text{tx}_{rfd}^{(1)}, \sigma_{rfd}^{(1)})$ after timeout T_1, if $\Sigma_{\text{SIG}}.\text{Vf}(\widehat{\text{pk}}_{rfd}^{(1)}, \text{tx}_{rfd}^{(1)}, \widehat{\sigma}_{rfd}^{(1)}) \neq 1$ or $\Sigma_{\text{SIG}}.\text{Vf}(\text{pk}^{(10)}, \text{tx}_{rfd}^{(1)}, \check{\sigma}_{rfd}^{(1)}) \neq 1$, the simulator aborts;

Hybrid $\mathcal{H}_9$: It is the same as the above execution except that the adversary corrupts user $\mathcal{P}_1$ and the simulator obtains $(\text{tx}_{rfd}^{(0)}, \sigma_{rfd}^{(0)})$ after timeout T_0, if $\Sigma_{\text{SIG}}.\text{Vf}(\text{pk}^{(01)}, \text{tx}_{rfd}^{(0)}, \sigma_{rfd}^{(0)}) \neq 1$, the simulator aborts.

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

Below we show the indistinguishability between $\mathcal{H}_0$ and $\mathcal{H}_9$. In addition, we use $\approx_c$ to denote computational indistinguishability for a PPT algorithm.

$\mathcal{H}_0 \approx_c \mathcal{H}_1$: The indistinguishability directly follows from the security of 2PC protocol $\Gamma_{\text{Sig}}^{\text{SIG}}$. The security of 2PC protocol $\Gamma_{\text{Sig}}^{\text{SIG}}$ for signature generation guarantees the existence of $\mathcal{S}_{2pc,1}$;

$\mathcal{H}_1 \approx_c \mathcal{H}_2$: The indistinguishability directly follows from the security of 2PC protocol $\Gamma_{\text{AdpSig}}^{\text{SIG}}$. The security of 2PC protocol $\Gamma_{\text{AdpSig}}^{\text{SIG}}$ for pre-signature generation guarantees the existence of $\mathcal{S}_{2pc,2}$;

$\mathcal{H}_2 \approx_c \mathcal{H}_3$: The only difference between the hybrids is that in $\mathcal{H}_3$ the simulator aborts, if the adversary corrupts user $\mathcal{P}_1$ and outputs a valid swap transaction $(\text{tx}_{swp}^{(1)}, \sigma_{swp}^{(1)})$ before the simulator initiate a swap on behalf of user $\mathcal{P}_0$;

$\mathcal{H}_3 \approx_c \mathcal{H}_4$: The only difference between the hybrids is that in $\mathcal{H}_4$ the simulator aborts, if the adversary corrupts user $\mathcal{P}_0$ and outputs a valid swap transaction $(\text{tx}_{swp}^{(0)}, \sigma_{swp}^{(0)})$ before timeout T_1, while the simulator cannot obtains its valid swap transaction. The probability of the event triggered in $\mathcal{H}_4$ is negligible;

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

$\mathcal{H}_5 \approx_c \mathcal{H}_6$: The only difference between the hybrids is that in $\mathcal{H}_6$ the simulator aborts, if the adversary $\mathcal{P}_0$ outputs a valid refund transaction before timeout T_0. With the security of VTD, the probability of the event triggered in $\mathcal{H}_6$ is negligible;

$\mathcal{H}_6 \approx_c \mathcal{H}_7$: The only difference between the hybrids is that in $\mathcal{H}_7$ the simulator aborts, if the adversary $\mathcal{P}_1$ outputs a valid refund transaction before timeout T_1. With the security of underlying blockchain for confirming the conflicting transactions, the probability of the event triggered in $\mathcal{H}_7$ is negligible;

$\mathcal{H}_7 \approx_c \mathcal{H}_8$: The only difference between the hybrids is that in $\mathcal{H}_8$ the simulator aborts, if it cannot post a valid refund transaction after timeout T_1. With the security of underlying blockchain and VTD, the probability of the event triggered in $\mathcal{H}_8$ is negligible;

$\mathcal{H}_8 \approx_c \mathcal{H}_9$: The only difference between the hybrids is that in $\mathcal{H}_9$ the simulator aborts, if it cannot post a valid refund transaction after timeout T_0. With the security of VTD, the probability of the event triggered in $\mathcal{H}_9$ is negligible.

□