USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Parallelizing Universal Atomic Swaps for Multi-Chain Cryptocurrency Exchanges

Danlei Xiao, Chuan Zhang, and Haotian Deng, *Beijing Institute of Technology;*
Jinwen Liang, *The Hong Kong Polytechnic University;* Licheng Wang and
Liehuang Zhu, *Beijing Institute of Technology*

https://www.usenix.org/conference/usenixsecurity25/presentation/xiao-danlei

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the
34th USENIX Security Symposium is sponsored by USENIX.

Parallelizing Universal Atomic Swaps for Multi-Chain Cryptocurrency Exchanges

Danlei Xiao[1], Chuan Zhang[1,✉], Haotian Deng[1], Jinwen Liang[2], Licheng Wang[1], and Liehuang Zhu[1]

[1]Beijing Institute of Technology, {danleix, chuanz, hdeng, lcwang, liehuangz}@bit.edu.cn
[2]The Hong Kong Polytechnic University, jinwen.liang@polyu.edu.hk

Abstract

Universal atomic swap is an emerging technique for secure cryptocurrency exchanges across diverse blockchains, eliminating the need for custom scripting language support from blockchains. While existing schemes primarily focus on exchanges among two users, extending these to multiple users across multiple blockchains incurs significant overheads due to the need for performing multiple two-party swaps *serially*. An intuitive insight is to *parallelize* the universal processes, but this idea still faces two technical challenges: (i) avoid asset theft during parallel asset locking; (ii) ensure atomicity by preventing partial execution of transactions with a uniform refund time used to avoid asset deadlock in parallel.

In this paper, we present `ParaSwap`, the first framework to parallelize universal atomic swaps for cryptocurrency exchanges among multiple users across multiple blockchains. We replace the serial multiple two-party swaps with a concurrent mechanism, where each participant concurrently locks and withdraws coins, achieving parallel execution. To prevent asset theft, the necessary witness for swaps is collaboratively determined by all participants. Then we introduce a novel re-lock approach to ensure atomicity with a uniform refund time, allowing participants to re-lock their assets to new addresses when the remaining time is insufficient to complete their withdrawal. Notably, `ParaSwap` employs adaptor signatures and verifiable timed discrete logarithm (VTD) technology, relying only on the bare minimum ability of blockchain to verify transaction signatures. We implement `ParaSwap` on four public blockchain test networks: Bitcoin, Ethereum, Avalanche, and Binance Smart Chain. Our evaluation demonstrates that `ParaSwap` reduces the exchange time complexity from $O(n)$ to $O(1)$, where n is the number of participants, and lowers gas costs by 26.2× to 46.8×, compared to existing methods.

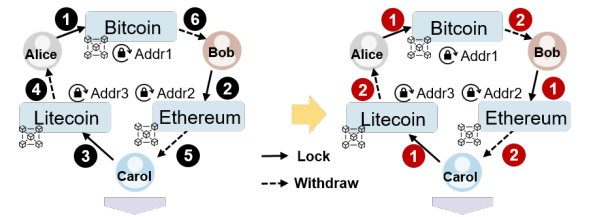
✉Chuan Zhang is the corresponding author.

1 Introduction

Cryptocurrency exchange is a critical issue in the current cryptocurrency and distributed ledger technology landscape [49]. According to GoinGecko [10] and DeFiLlama [7], the 24-hour trading volume of all cryptocurrency exchanges and the total value locked in liquidity pools have both reached tens of billions of dollars. Atomic swap plays a vital role in secure cryptocurrency exchanges across diverse blockchains. Its atomicity enables each cross-chain transaction to be executed entirely (i.e., asset swap) or not executed at all (i.e., asset refund), thereby preserving users' asset security [7,10,32,36,42,44,49]. For example, consider a swap between two users across two blockchains: Alice holds assets α on blockchain $\mathbb{C}_A$, and Bob holds assets β on blockchain $\mathbb{C}_B$. Atomic swaps ensure that the transfer of α from Alice to Bob occurs if and only if Bob transfers β from his holdings to Alice.

Universal atomic swap is viewed as a most promising technique of existing atomic swap schemes, as it only uses cryptographic technologies to realize atomic swaps. It avoids the dependence on the trusted third-party [45], [3] and trust assumptions [19], reduces on-chain costs [33,38,46], and does not require any custom scripting language support from the corresponding blockchains [41], [40].

Most recently, a new universal atomic swap construction supporting exchanges among two users [44], was proposed. However, when extended to accommodate atomic swaps among multiple users across multiple blockchains, they face significant linear time costs because the multiple two-party swaps must be executed *serially* among each pair of participants. Here we give an example of cryptocurrency exchanges across multiple blockchains. Consider a case in which Alice wants to exchange her BTC for LTC, Bob wants to exchange his ETH for BTC, and Carol wants to exchange her LTC for ETH. As shown in the left part of Figure 1, each participant sequentially locks their cryptocurrency in an address on their respective blockchain. Then, through serial withdrawals, Alice, Carol, and Bob each obtain their intended assets. However, we can observe that to finish the exchanges, each transfer must



Step	Lock	Step	Withdraw
1	Alice $\xrightarrow{BTC}$ Addr1	4	Addr3 $\xrightarrow{LTH}$ Alice
2	Bob $\xrightarrow{ETH}$ Addr2	5	Addr2 $\xrightarrow{ETH}$ Carol
3	Carol $\xrightarrow{LTH}$ Addr3	6	Addr1 $\xrightarrow{BTC}$ Bob

Step	Lock	Step	Withdraw
1	Alice $\xrightarrow{BTC}$ Addr1 Bob $\xrightarrow{ETH}$ Addr2 Carol $\xrightarrow{LTH}$ Addr3	2	Addr3 $\xrightarrow{LTH}$ Alice Addr2 $\xrightarrow{ETH}$ Carol Addr1 $\xrightarrow{BTC}$ Bob

Figure 1: An example to illustrate serial vs. parallel atomic swap process among 3 participants across 3 blockchains.

wait for the previous one to be completed, involving 6 steps, which incurs a linear time cost. An intuitive insight to improve efficiency is to *parallelize* the universal atomic swap process. That is, as shown in the right part of Figure 1, if we can make the lock and withdrawal transactions between each pair of participants happen at the same time, only 2 steps are needed to realize multi-party exchanges. In other words, the parallel process can reduce the exchange time complexity from $O(n)$ to $O(1)$, where n is the number of participants. However, this idea still faces two significant technical challenges.

The first one is asset theft. In the serial process, Alice, as the initiator, samples a random value (referred to as the witness) and generates its corresponding statement to lock assets she uses to swap. Others lock their assets only after confirming that the assets they intend to have been locked by others. This sequential locking avoids the risk of asset theft. In the parallel process, all participants lock their assets concurrently and do not have a reliable way to confirm that others have locked their assets as promised. *Challenge 1: How to avoid asset theft when participants lock their assets in parallel?*

Additionally, it's crucial to ensure that the assets won't be indefinitely locked if any participant wants to exit the exchange or goes offline. To address this problem, the serial process employs a time-out mechanism, which blocks specific values timed to refund the locked assets. Participants provide the witness to withdraw their intended assets. Due to the constraint that participants can only withdraw others' assets after their own assets have been successfully withdrawn, the time-out settings increase sequentially based on the withdrawal order. In contrast, the settings must be uniform in the parallel process, since all participants withdraw assets concurrently. However, this uniformity may leave some participants to withdraw assets successfully while others fail, suffering a loss. *Challenge 2: How to ensure atomicity by preventing partial execution of swaps with a uniform refund time?*

To deal with the above challenges, we propose `ParaSwap`, the first framework to parallelize universal atomic swaps for cryptocurrency exchanges among multiple parties across multiple blockchains. For the parallel plane, we replace the single initiator by setting all participants to act as initiators. All participants collaboratively determine a global statement and witness for concurrently locking and withdrawing to prevent asset theft. If a participant finds that his/her intended assets are not locked, they will not participate in generating the global witness for withdrawal operations, thereby safeguarding themselves against asset theft. Additionally, we introduce a novel re-lock approach to ensure atomicity with a uniform refund time. Participants can re-lock their assets to new addresses when the remaining time is insufficient for them to complete their withdrawal. Note that, besides the witness, participants need to reveal a secret value to withdraw their intended assets. The number of secret values required to withdraw assets from these new addresses increases incrementally. This approach prevents malicious participants from obtaining their intended assets while blocking others from accessing theirs within the uniform refund time.

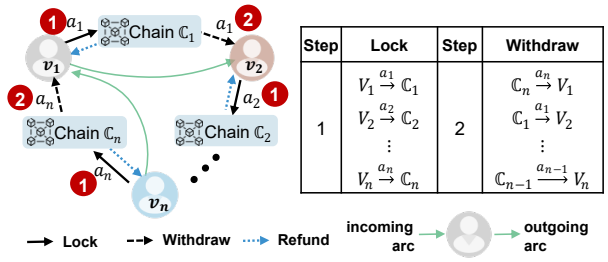
In summary, the main **contributions** of this paper are:

- We propose the first framework to parallelize universal atomic swaps for cryptocurrency exchanges among multiple parties across multiple blockchains, named `ParaSwap`, minimizing the significant time cost caused by the serial process.
- `ParaSwap` is built upon adaptor signatures and verifiable timed discrete logarithm (VTD) technology. It does not require any custom scripting language support from blockchains and only relies on the bare minimum ability of blockchains to verify transaction signatures.
- We conduct a rigorous security analysis of `ParaSwap`. `ParaSwap` guarantees atomicity in the parallel swaps through key assurances: (i) full adherence ensures parallel cryptocurrency exchange among all participants across multiple chains; (ii) participants adhering to `ParaSwap` do not suffer a loss, even in cases of participants or coalitions deviate from it.
- We implement `ParaSwap` on four public blockchain test networks: Bitcoin, Ethereum, Avalanche, and Binance Smart Chain, and evaluate its on-chain and off-chain performance. The results indicate that `ParaSwap` reduces the exchange time complexity from $O(n)$ to $O(1)$, where n represents the number of participants. Furthermore, `ParaSwap` achieves significant optimizations, lowering gas costs by 26.2× to 46.8× and reducing transaction size compared to existing multi-party methods.

2 Solution Overview

2.1 System Model

As shown in Figure 2, assume a setting where participant v_{i+1} owns assets a_{i+1} on ledger $\mathbb{C}_{i+1}$ and intends to securely



Step	Lock	Step	Withdraw
1	$V_1 \xrightarrow{a_1} \mathbb{C}_1$ $V_2 \xrightarrow{a_2} \mathbb{C}_2$ $\vdots$ $V_n \xrightarrow{a_n} \mathbb{C}_n$	2	$\mathbb{C}_n \xrightarrow{a_n} V_1$ $\mathbb{C}_1 \xrightarrow{a_1} V_2$ $\vdots$ $\mathbb{C}_{n-1} \xrightarrow{a_{n-1}} V_n$

Figure 2: System model.

exchange the assets a_{i+1} for v_i's assets a_i on ledger $\mathbb{C}_i$. Here, $i = 1,2,\ldots,n$, with $v_0 = v_n, v_1 = v_{n+1}, a_0 = a_n, a_1 = a_{n+1}$, where n is the number of participants. We assume participants possess a bootstrapping mechanism, such as a forum, to match demands and identify exchange partners, the same as assumed in [44]. For the sake of demonstration, the table of used notations can be found in Appendix A.

We formulate multi-party swaps as a directed cycle, integrating the intended asset transfers denoted as the directed arcs into an exchange. Each vertex denotes a participant. We simplify this complex multi-arc interaction scenario into a 2-arc computation paradigm involving *incoming and outgoing arcs*. Specifically, a participant's swap demand is represented as an *incoming arc* to a vertex, indicating the assets they intend to. The *outgoing arc* from the vertex denotes what the participant offers for swap. When the ownership of assets a_i is transferred from v_i to v_{i+1}, the assets swap occurs, we say the arc is executed or the assets are withdrawn.

2.2 Threat Model

Synchronous Communication Model. We assume a synchronous communication network, where communication proceeds in rounds, same as [27,42,44]. We formalize this using an ideal clock functionality $\mathcal{F}_{\texttt{clock}}$, based on [26], [31], with a full description provided in [23]. It requires all honest participants to confirm their readiness before the clock progresses to the next round. This functionality ensures awareness of the given round among participants. Additionally, we assume secure message transmission formalized by the ideal functionality $\mathcal{F}_{\texttt{smt}}$, same as [27,42,44].

Ideal Blockchain Model. We assume an ideal ledger functionality $\mathcal{L}$, same as [17,27,34,44]. For simplicity of notation, we make use of $\mathcal{L}$ for managing the chains of all cryptocurrencies involved. In addition, it requires only the bare minimum ability to verify transaction signatures and does not necessitate support for any custom scripting language. The corresponding ideal functionality $\mathcal{F}_{\mathcal{L}}$ maintains $\mathcal{L}$ locally and ensures it is updated based on transactions among participants. Specifically, it provides the interface $\texttt{Publish}(\texttt{name}, Addr_1, Addr_2, a)$ to transfer asset a in a session with identifier name, from address $Addr_1$ to address $Addr_2$, if provided with the secret key of $Addr_1$. Participants leverage the Publish interface to execute transactions among themselves. It provides an additional interface Initialize to initialize their address along with the public key and an initial value, which are stored in the ledger. At any execution point, any participant can send a distinguished message Read to $\mathcal{F}_{\mathcal{L}}$, which responds by transmitting the full transcript of $\mathcal{L}$ to the participant. We refer the reader to [17] for a formal definition of this functionality.

There are two types of participants in ParaSwap: honest and malicious. Honest participants adhere to ParaSwap, while malicious participants may deviate from it in arbitrary manners to gain larger benefits, i.e., withdrawing the assets on their incoming arc while preventing the withdrawal of assets on their outgoing arc by others, such as sending messages with incorrect values or not sending messages at all to other participants (e.g., message manipulation attacks), or exploiting critical moments near the refund time to operate (e.g., boundary attacks). Additionally, malicious participants may form coalitions and share information to collaborate and gain larger overall benefits (e.g., collusion attacks).

Remark. Note that there is the so-called miner/maximal extractable value (MEV) extraction issue in blockchain systems [24], [25]. That is, malicious miners may manipulate transaction ordering over normal users to engage in illicit activities. MEV extraction is an inherent challenge in blockchain technology and has been widely acknowledged. Existing atomic swap schemes [29,30,42,44], including ours, focus on atomic swaps between participants who exchange their assets, and the MEV extraction issues are out of the study scope, thus, they all have risks associated with MEV extraction. Similar to the existing schemes, our work does not solve the MEV extraction issue, but also doesn't make it worse. Moreover, we notice that there have been some well-developed MEV countermeasures, such as Flashbots [8], SUAVE [9], MEV-Boost [12], and Skip Protocol [14]. Further ethical discussion on MEV is given in Appendix C.

2.3 Security Goal

We now describe the security goal of atomic swap construction. The formal definition is given in Section 5.1.

Atomicity. Atomicity requires that any honest participant either obtain his/her intended assets or refund his/her locked assets that they used for swap, even if other participants act maliciously or collude. This ensures balance security for honest participants.

2.4 Challenge and Method in Parallelizing the Universal Atomic Swap

As described in Section 1, the serial process in universal atomic swap protocols incurs significant linear time costs. Specifically, a single initiator samples a random value (referred to as the witness) and uses its corresponding statement to lock assets that he/she uses to swap. Other participants then

sequentially lock their assets only after confirming that the assets they intend to have been locked by others. The initiator provides the witness to withdraw his/her intended assets. Similarly, other participants can sequentially withdraw their intended assets upon learning the witness.

Parallelizing the lock and withdrawal operations. We intend to *parallelize* the serial universal atomic swap process, where all participants concurrently perform the lock and withdrawal operations. However, this parallelization has the following significant technical challenges.

- *Challenge 1: How to avoid asset theft when participants lock their assets in parallel?* In the serial process, the initiator begins by locking the assets he/she uses to swap. Then, other participants sequentially lock their assets to avoid the risk of asset theft. In contrast, the parallel process requires all participants to lock their assets concurrently, preventing them from confirming whether others have fulfilled their asset-locking commitments.

- *Challenge 2: How to ensure atomicity by preventing partial execution of swaps with a uniform refund time?* The lock operations not only need to prevent asset theft but also avoid deadlocks where assets are indefinitely locked, if any participant exits the exchange or goes offline. The serial process uses a time-out mechanism, which blocks specific values timed to refund the locked assets. Since withdrawals occur sequentially, and participants learn the witness in sequence, the time-out settings are incrementally increased based on the withdrawal order. In contrast, the parallel process requires a uniform time-out setting for concurrent withdrawals. However, this uniformity may leave some participants unsuccessful in withdrawing assets while their locked assets are taken, leaving partial execution of swaps and a loss.

As described above, we must guarantee the following three properties: (i) no participant can initiate the withdraw without knowing the global witness; (ii) participants can refund the locked assets if they are not withdrawn after the refund time; (iii) if a participant's locked assets are withdrawn (the outgoing arc is executed), he/she can withdraw the intended assets (executes the incoming arc). Based on the above properties, any participants will either withdraw the intended assets or refund the locked assets after the exchange.

Collaborative determination of a global statement and witness for asset theft. We replace the single initiator by setting all participants to act as initiators. Specifically, each participant samples a random value and generates its corresponding statement. These individual statements are then combined to collaboratively determine a global statement. Then, all participants concurrently perform the lock and withdrawal operations using the global statement and the corresponding global witness, respectively. Importantly, participants only proceed with generating the global witness if they confirm that the assets they aim to acquire have been successfully locked, safeguarding against asset theft.

Re-locking assets to new addresses for atomicity. Further, to ensure atomicity under a uniform refund time, we introduce a novel re-lock approach designed to prevent only the execution of the outgoing arc. Participants use the global witness and their secret values to withdraw their intended assets. Additionally, participants can re-lock their assets to new addresses when the remaining time is insufficient for them to complete their withdrawal, i.e., participants re-lock their assets to $(x+1)$-th address after time $x\Delta$, for $1 \leqslant x < n$. The re-lock mechanism ensures that the assets locked in the x-th address can only be withdrawn if secret values from x participants are provided before time $x\Delta$, where Δ is the time required for one participant to withdraw assets. A participant who fails to generate the correct witness, despite others doing so, must wait until his/her locked assets are withdrawn. In the case where his/her locked assets are withdrawn before time $x\Delta$ with the witness and x secret values, the participant combines them with his/her own secret value ($x+1$ secret values in total) to withdraw the intended assets from the $(x+1)$-th address. This approach prevents malicious participants from obtaining their intended assets while blocking others from accessing theirs within the uniform refund time.

3 Preliminaries

In this section, we review some background knowledge used in `ParaSwap` and present the fundamental building blocks. The security parameter is denoted by $\lambda \in \mathbb{N}$. We write $x := z$ to denote variable assignment, and $x \leftarrow A(y)$ to denote the execution of a probabilistic polynomial time (PPT) algorithm A on input y.

3.1 Hard Relations

Let R be a NP relation with (statement, witness) pairs (Y, y), and L be a set of positive instances with respect to R defined as $L := \{Y | \exists y \; s.t. (Y, y) \in R\}$. R is referred to a hard relation if (i) there exists a PPT sampling function $\mathsf{GenR}(1^\lambda)$ and outputs a (statement, witness) pair, (ii) R is decidable in polynomial time, and (iii) for all PPT adversaries $\mathcal{A}$, and the probability that $\mathcal{A}$ produces the witness given the statement is negligible.

3.2 Adaptor Signature

Adaptor signatures [17] enhance traditional digital signatures by adding a condition that must be fulfilled for the signature to be completed. Specifically, the process involves generating a pre-signature for a message using a statement first. The pre-signature can then be adapted into a valid on-chain signature if a user possesses the witness corresponding to the statement.

An adaptor signature scheme, denoted as Σ_{AS}, is defined with respect to a hard relation R, (statement, witness)

pair $(Y,y) \in R$, and a standard digital signature scheme $\Sigma_{\mathsf{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$. It consists of four algorithms: $(\mathsf{PreSig}, \mathsf{Adapt}, \mathsf{PreVf}, \mathsf{Ext})$, as outlined below:
$\hat{\sigma} \leftarrow \mathsf{PreSig}(sk, msg, Y)$: Generates a pre-signature $\hat{\sigma}$ on message msg using the secret key sk and the statement Y.
$\sigma \leftarrow \mathsf{Adapt}(\hat{\sigma}, y)$: Adapts the pre-signature $\hat{\sigma}$ into the valid signature σ utilizing the witness y.
$\{0,1\} \leftarrow \mathsf{PreVf}(pk, msg, Y, \hat{\sigma})$: Verifies whether the pre-signature $\hat{\sigma}$ is generated on message msg utilizing the statement Y and the public key pk.
$y \leftarrow \mathsf{Ext}(\hat{\sigma}, \sigma, Y)$: Extracts the witness y such that $(Y,y) \in R$ or $\perp$, with the pre-signature $\hat{\sigma}$ and the valid signature σ pair.

In this paper, we adopt the adaptor signature construction described in [17], which includes provably secure instantiations where the underlying signature schemes are the Schnorr and ECDSA signature schemes. The hard relation R utilized is the discrete logarithm (DLOG) relation, defined as $Y = G^y$, where G is a generator in a group. This construction ensures unforgeability under the chosen message attack, similar to the unforgeability of digital signature schemes, as well as pre-signature correctness, adaptability, and witness extractability. Specifically, (i) pre-signature correctness: if a pre-signature $\hat{\sigma}$ is honestly generated, it can be adapted into a valid signature σ; (ii) pre-signature adaptability: ensures that the pre-signature $\hat{\sigma}$ can be adapted into a valid signature σ given the correct witness y; (iii) witness extractability: ensures that from any pre-signature and valid signature pair, the correct witness y can be reliably extracted.

3.3 Verifiable Timed Discrete Logarithm

In a verifiable timed discrete logarithm (VTD) scheme [43], a committer generates a commitment for a secret x, where $X = G^x$ (with G being a generator in a group). The secret value x, known as the discrete logarithm (dlog) of X, is private, while X is public. The commitment time-locks the secret value x for a time parameter T. By performing sequential computations on the commitment for time T, anyone can force open the commitment, i.e., learn the secret x, providing a mechanism for delayed information release.

A VTD scheme Σ_{VTD} consists of four algorithms $(\mathsf{ComPro}, \mathsf{Vrfy}, \mathsf{Open}, \mathsf{ForceOp})$, defined as follows:
$(C, \pi_x) \leftarrow \mathsf{ComPro}(x, T)$: Generates a commitment C and a proof π_x on a secret dlog value x and a time parameter T.
$\{0,1\} \leftarrow \mathsf{Vrfy}(X, C, \pi_x)$: Verifies whether the value x such that $X = G^x$ embedded in the commitment C.
$(x, r) \leftarrow \mathsf{Open}(C)$: The creator of the commitment C can easily open it to know the value x and the randomness r used to generate the commitment.
$x \leftarrow \mathsf{ForceOp}(C)$: Anyone can force open the commitment C to learn the value x within time T.

The security properties of a VTD scheme are as follows: (i) soundness: ensures w.r.t. the commitment C, the ForceOp algorithm can successfully output the secret dlog value x within time T, and (ii) privacy: ensures that any PRAM algorithms (polynomial time algorithms with polynomially bounded amount of parallel computing power) with execution time at most T_0 (where $T_0 < T$) have only a negligible probability of successfully learning x from C and π_x.

Our work considers the VTD construction proposed in [43], leveraging the cryptographic primitive time-lock puzzle [39], which allows a user to conceal a secret within a puzzle that can only be solved after a pre-defined time T. In the VTD scheme, once a user conceals the dlog into the time-lock puzzle, they need to utilize a non-interactive zero-knowledge (NIZK) proof. Specifically, this proof proves that the time-lock puzzle can be solved after time T, and the concealed dlog x satisfies the equation $X = G^x$. Similar to existing universal atomic swaps [44], we propose that each participant conservatively estimates the computational power of other participants to prevent any participant with significant computational resources from opening the puzzle before the pre-defined time.

3.4 Two-Party Computation

The two-party computation (TPC)[1] protocol enables two parties to jointly compute a function based on their respective inputs while ensuring that both inputs remain private. Beyond ensuring output correctness, the protocol also guarantees input privacy, meaning that neither party gains any information beyond their own input and the computed output.

Joint Address Management. The joint address management interactive protocol, denoted as Π_{JAMan}, enables two parties v_1 and v_2 to jointly manage a blockchain address. Specifically, the protocol utilizes the security parameter 1^λ and group parameters $(\mathbb{G}, G, q)$ provided by two parties as input, then outputs the public address $pk_{1,2}$ for both parties, along with the corresponding secret key shares $sk^1_{1,2}$ for v_1 and $sk^2_{1,2}$ for v_2. In this paper, we adopt the joint address management protocol described in [42] for Schnorr keys and the one in [28] for ECDSA keys. The keys satisfy $pk_{1,2} = G^{sk_{1,2}}$, where $sk_{1,2} = sk^1_{1,2} \oplus sk^2_{1,2}$. Here, the operation $\oplus$ corresponds to $+$ for Schnorr signatures and $\cdot$ for ECDSA signatures.

Joint Pre-Signing. The joint pre-signing interactive protocol, denoted as Π_{JPSig}, enables two parties v_1 and v_2 to jointly generate a pre-signature on a transaction concerning a statement Y of the hard relation R. Specifically, the protocol utilizes a transaction tx and the statement Y as public input, while v_1 and v_2 use their secret share $sk^1_{1,2}$ and $sk^2_{1,2}$ as private inputs. Here, $sk^1_{1,2}$ and $sk^2_{1,2}$ are shares of the secret key $sk_{1,2}$, which corresponds to the public key $pk_{1,2}$. Next, the protocol outputs the pre-signature $\hat{\sigma}_{tx}$ to both parties. In this paper, we adopt the joint pre-signature protocols described in [34].

[1] We denote as TPC to distinguish it from two-phase commit (2PC).

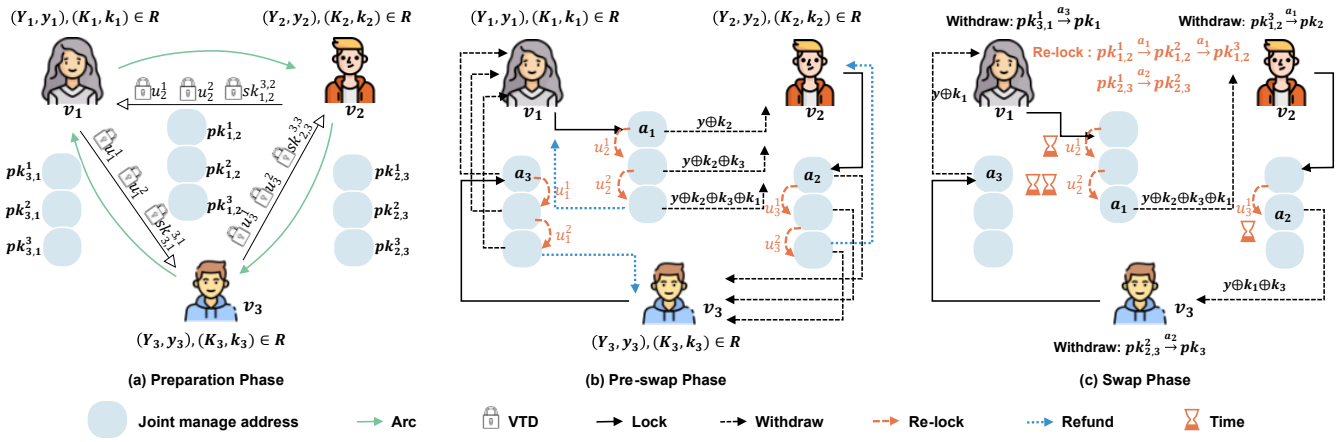


Figure 3: Workflow of ParaSwap.

4 ParaSwap Design

In this section, we first introduce the outline of ParaSwap, then provide a detailed description of it.

4.1 Outline of ParaSwap

We utilize the adaptor signature and verifiable timed discrete logarithm (VTD) technology. All participants act as initiators and collaboratively determine a global statement and witness. They are used to generate pre-signatures for lock operations and to adapt valid on-chain signatures from pre-signatures for withdrawal operations, respectively. When all participants have locked their assets, they will participate in generating the global witness. Note that, besides the witness, participants need to reveal a secret value to withdraw their intended assets. Participants can re-lock their assets to new addresses when the remaining time is insufficient for them to complete their withdrawal. The number of secret values required to withdraw assets from these new addresses increases incrementally.

ParaSwap consists of five phases. We assume that Δ is the maximum on-chain transaction confirmation time across the considered blockchains. For example, assuming assets are held on Ethereum, Binance Smart Chain, and Avalanche, the average confirmation delays are around 15 seconds on Ethereum [37], 3 seconds on Binance Smart Chain [4], and 0.3 seconds on Avalanche [2]. We thus set $\Delta = 15$ seconds to account for the slowest chain. We assume that completing off-chain operations takes at most ε time per phase. As the preparation phase has not locked assets, we define ε as the maximum time required across the pre-swap, witness sharing, and swap phases. According to our evaluation in Section 6.2, for 4 (resp. 10) participants, ε is around 2.77 (resp. 6.68) seconds when using ECDSA as the underlying signature scheme and about 0.12 (resp. 0.30) seconds when using Schnorr, respectively. The time parameter $T = n\Delta + t$ represents the refund time, where n is the number of participants, and the first three phases are completed within $t = \Delta + 3\varepsilon$. For example, with the Schnorr signature scheme, when $n = 4$ (resp. $n = 10$), t is around 15.36 (resp. 15.90) seconds and T is around 75.36 (resp. 165.90) seconds.

To exemplify the problem, we take the exchange among 3 participants across 3 blockchains as an example (as shown in Figure 3), where participant v_1 wants v_3's assets a_3, v_3 wants v_2's assets a_2, and v_2 wants v_1's assets a_1. Initially, each participant knows the directed cycle and has access to the public keys of the other participants.

Phase 1: preparation. Each participant v_i ($i = 1,2,3$) generates a (statement, witness) pair (Y_i, y_i) and a (document, identifier) pair (K_i, k_i), where $(Y_i, y_i), (K_i, k_i) \in R$, as shown in Figure 3(a). They then broadcast their statements and documents, while the witnesses and identifiers remain hidden. For each arc in the directed cycle, the two participants involved collaboratively generate three jointly managed addresses. For instance, v_1 and v_2 jointly generate the addresses $pk^i_{1,2}$, for $i = 1,2,3$, i.e., $pk^1_{1,2}, pk^2_{1,2}, pk^3_{1,2}$, where v_1 holds the secret key shares $sk^{i,1}_{1,2}$ and v_2 holds $sk^{i,2}_{1,2}$. To manage re-lock and refund operations, v_2 creates three VTDs and sends them to v_1. The first two VTDs blocks u^1_2 and u^2_2 at time $\Delta + t$ and $2\Delta + t$, with corresponding public values U^1_2 and U^2_2, where $(U^1_2, u^1_2), (U^2_2, u^2_2) \in R$. The third VTD blocks the secret share $sk^{3,2}_{1,2}$ associated with $pk^3_{1,2}$ at time $3\Delta + t$, ensuring the refund of the locked assets if necessary.

Phase 2: pre-swap. For each arc, each participant v_i locks the assets a_i into the first joint address, as shown in Figure 3(b). Then the two participants on each arc jointly generate the pre-signatures for the withdraw operations using the global statement $Y = Y_1 \cdot Y_2 \cdot Y_3$ and identity documents. For instance, after v_1 locks a_1 into $pk^1_{1,2}$, v_1 and v_2 jointly generate three pre-signatures based on $Y \cdot K_2, Y \cdot K_2 \cdot K_3, Y \cdot K_2 \cdot K_3 \cdot K_1$. These pre-signatures pre-define the withdrawal of a_1 from the first, second, and third joint addresses, where v_2 finalizes by adapting them into valid signatures using $y \oplus k_2, y \oplus k_2 \oplus k_3, y \oplus k_2 \oplus k_3 \oplus k_1$, respectively. They also jointly generate the pre-signatures for the re-lock operations using U^1_2 and U^2_2 to pre-

define the re-lock of a_1 from $pk^1_{1,2}$ to $pk^2_{1,2}$ after time $\Delta+t$, and from $pk^2_{1,2}$ to $pk^3_{1,2}$ after time $2\Delta+t$, respectively.

Phase 3: witness sharing. Each participant v_i sends the witness y_i to each other, while the identifier k_i, and the delayed values u^x_i for $x \in [1, n-1]$ remain hidden.

Phase 4: swap. Each participant v_i generates the global witness $y = y_1 \oplus y_2 \oplus y_3$, and uses y along with the identifier k_i to withdraw the intended assets a_{i-1} from the first joint address. However, if v_i fails to generate the correct y, he/she should wait for the execution of the outgoing arc. For instance, as illustrated in Figure 3(c), only v_1 successfully generates the correct y. At $\Delta+t$, v_3 and v_2 re-lock assets a_3 and a_2 into their respective second joint address. v_1 can only withdraw a_3 from the first address $pk^1_{3,1}$ using y and k_1 before $\Delta+t$, after which $y \oplus k_1$ is passed to v_3. So v_3 can withdraw a_2 from the second joint address using $y \oplus k_1 \oplus k_3$ before $2\Delta+t$. Note that if v_2 does not re-lock a_2 from the first address to the second, v_3 can perform the re-lock operation, since v_3 originally holds the delayed values.

Phase 5: refund. If the assets are not withdrawn before time $3\Delta+t$, the participant who locks the assets in the pre-swap phase can refund them by opening the final VTD.

In Figure 4 and 5, we detail the five phases of `ParaSwap`.

4.2 Detailed Construction

Based on the idea mentioned in Section 2.4 and aboved workflow, we describe the details of `ParaSwap` as follows:

Phase 1: preparation. To initiate the multi-party atomic swaps, each participant generates his/her own (statement, witness) and (document, identifier) pair. For each arc in the directed cycle, the two participants involved jointly generate n joint managed addresses. Subsequently, each participant generates n VTDs on their incoming arc to prevent asset deadlock if the exchange fails.

1. Each participant v_i generates a (statement, witness) pair (Y_i, y_i) and a random identifier rid k_i along with its corresponding identity document K_i, where $(Y_i, y_i), (K_i, k_i) \in R$. Here, $i = 1, 2, \ldots, n$, where n is the number of participants. The global statement Y and global witness y are computed as follows[2],
$$\begin{aligned} Y &= Y_1 \cdot Y_2 \cdots Y_n, \\ y &= y_1 \oplus y_2 \oplus \cdots \oplus y_n. \end{aligned} \tag{1}$$
2. After that, each participant v_i broadcasts his/her statement Y_i and document K_i to all other participants. However, the corresponding y_i and k_i remain hidden.
3. For each arc (v_i, v_{i+1}), the two involved participants jointly generate n addresses $pk^j_{i,i+1}$ using the TPC protocol Π_{JAMan}, where $j = 1, 2, \ldots, n$. The protocol outputs shares of the secret key $sk^j_{i,i+1}$, where $sk^{j,i}_{i,i+1}$ is held by v_i and $sk^{j,i+1}_{i,i+1}$ by v_{i+1}, respectively. These shares satisfy
$$sk^j_{i,i+1} = sk^{j,i}_{i,i+1} \oplus sk^{j,i+1}_{i,i+1}, \tag{2}$$
ensuring $sk^j_{i,i+1}$ can only be reconstructed if both participants contribute their respective shares.
4. v_{i+1} generates $n-1$ random delay values u^x_{i+1}, and the corresponding public values U^x_{i+1} $((U^x_{i+1}, u^x_{i+1}) \in R)$, where $x = 1, 2, \ldots, n-1$. Then v_{i+1} generates $n-1$ VTDs for delayed values u^x_{i+1} with time $x\Delta+t$, and a VTD for the last secret key share $sk^{n,i+1}_{i,i+1}$ with the refund time $T = n\Delta+t$. More concretely, we have that
$$\begin{aligned} (C^x_{i+1}, \pi^x_{i+1}) &\leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{ComPro}(u^x_{i+1}, x\Delta+t), \\ (C^n_{i+1}, \pi^n_{i+1}) &\leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{ComPro}(sk^{n,i+1}_{i,i+1}, n\Delta+t). \end{aligned} \tag{3}$$
v_{i+1} sends the n VTDs and U^x_{i+1} to v_i.
5. v_i verifies all VTDs using $\Sigma_{\mathsf{VTD}}.\mathsf{Vrfy}$ as follows,
$$0/1 \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{Vrfy}(C^x_{i+1}, \pi^x_{i+1}, U^x_{i+1}), \tag{4}$$
$$0/1 \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{Vrfy}(C^n_{i+1}, \pi^n_{i+1}, pk^n_{i,i+1}). \tag{5}$$

To avoid asset deadlocks where assets remain indefinitely locked, if any participant exits the exchange or goes offline, v_{i+1} generates the first $n-1$ VTDs for the re-lock operations and the last VTD for the refund operation. Specifically, v_i can re-lock the assets a_i from the x-th joint address $pk^x_{i,i+1}$ to $(x+1)$-th joint address $pk^{x+1}_{i,i+1}$ after time $x\Delta+t$ upon learning the delayed value u^x_{i+1}. If the exchange is not completed, v_i can refund a_i after time T. We assume the phase can be completed within ε time.

Phase 2: pre-swap. After the successful preparation phase, each participant locks the assets offering for swap to the first joint address. Both participants on each arc jointly generate the pre-signatures for the withdraw and re-lock operations.

1. Each participant v_i locks a_i into the first joint address $pk^1_{i,i+1}$ by signing the lock transaction of the form
$$tx^i_{lck} : pk_i \xrightarrow{a_i} pk^1_{i,i+1}. \tag{6}$$
2. Participant v_{i+1} defines the withdraw transaction $tx^{j,i+1}_{wtdw}$, which transfers the assets a_i from the j-th joint address $pk^j_{i,i+1}$ to his/her own address pk_{i+1} for $j \in [1, n]$. Meanwhile v_i defines the re-lock transaction $tx^{x,i}_{rlck}$, which transfers a_i from the x-th joint address $pk^x_{i,i+1}$ to the $(x+1)$-th joint address $pk^{x+1}_{i,i+1}$ for $x \in [1, n-1]$. More concretely, we have that
$$tx^{j,i+1}_{wtdw} : pk^j_{i,i+1} \xrightarrow{a_i} pk_{i+1}, \tag{7}$$
$$tx^{x,i}_{rlck} : pk^x_{i,i+1} \xrightarrow{a_i} pk^{x+1}_{i,i+1}. \tag{8}$$

[2] Here, the operation $\oplus$ corresponds to $+$ when using the Schnorr signature scheme and $\cdot$ for ECDSA signatures.

3. For each arc in the directed cycle, the two participants involved use the secret key shares to jointly generate $2n-1$ pre-signatures on the above transactions below.

4. **Withdraw operations.** Participant v_i and v_{i+1} jointly generate the pre-signatures on withdraw transactions using the TPC protocol Π_{JPSig}. Π_{JPSig} takes the withdraw transaction $tx^{j,i+1}_{wtdw}$ and the statement S^j_{i+1} as public inputs, while the secret key shares $sk^{j,i}_{i,i+1}$ and $sk^{j,i+1}_{i,i+1}$ from v_i and v_{i+1}, respectively, serve as private inputs. Next, it outputs the pre-signature $\hat{\sigma}^{j,i+1}_{wtdw}$ to both participants for $j \in [1,n]$. More concretely, we have that

$$\hat{\sigma}^{j,i+1}_{wtdw} \leftarrow \Pi_{\mathsf{JPSig}}(tx^{j,i+1}_{wtdw}, sk^{j,i}_{i,i+1}, sk^{j,i+1}_{i,i+1}, S^j_{i+1}). \quad (9)$$

The statement S^j_{i+1} and its corresponding witnesses w^j_{i+1} are computed as follows,

$$S^j_{i+1} = Y \cdot K_{i+1} \cdot \cdots \cdot K_{i+j}, \quad (10)$$

$$w^j_{i+1} = y \oplus k_{i+1} \oplus \cdots \oplus k_{i+j}. \quad (11)$$

5. **Re-lock operations.** Similar to what is mentioned above, Π_{JPSig} takes the transaction $tx^{x,i}_{rlck}$ and the statement U^x_{i+1} corresponding to the delay value u^x_{i+1} as public inputs, and the secret key shares $sk^{x,i}_{i,i+1}$ and $sk^{x,i+1}_{i,i+1}$ from v_i and v_{i+1} as private inputs. The protocol outputs the pre-signature $\hat{\sigma}^{x,i}_{rlck}$ to both participants for $x \in [1,n-1]$.

$$\hat{\sigma}^{x,i}_{rlck} \leftarrow \Pi_{\mathsf{JPSig}}(tx^{x,i}_{rlck}, sk^{x,i}_{i,i+1}, sk^{x,i+1}_{i,i+1}, U^x_{i+1}). \quad (12)$$

6. Note that, both participants need to verify them using $\Sigma_{\mathsf{AS}}.\mathsf{PreVf}$ as follows:

$$0/1 \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{PreVf}(pk^j_{i,i+1}, tx^{j,i+1}_{wtdw}, S^j_{i+1}, \hat{\sigma}^{j,i+1}_{wtdw}), \quad (13)$$

$$0/1 \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{PreVf}(pk^x_{i,i+1}, tx^{x,i}_{rlck}, U^x_{i+1}, \hat{\sigma}^{x,i}_{rlck}). \quad (14)$$

The pre-swap phase can be completed within at most $\Delta+\varepsilon$ time, where Δ is spent on the lock operations (on-chain).

Phase 3: witness sharing. After the successful pre-swap phase, each participant v_i reveals his/her witness y_i to others while keeping the rid k_i and all delayed values hidden. Any participant can exit the exchange without disclosing their witness. We assume that the phase can be done within ε time.

Phase 4: swap. Each participant uses the global witness or the delayed value to adapt the pre-signatures generated in the pre-swap phase into valid signatures, enabling them to execute withdraw or re-lock operations.

- **Normal swap:** Each participant starts the assets swap upon acquiring all witnesses before time $t = \Delta + 3\varepsilon$, which is enough to complete all prior phases. Then they compute the global witness y and use it, along with their identifiers, to withdraw their intended assets from the first joint address.

  For instance, v_{i+1} generates the global witness y as Eq. 1, and adapts the pre-signature $\hat{\sigma}^{1,i+1}_{wtdw}$ into the valid signature $\sigma^{1,i+1}_{wtdw}$ by utilizing $w^1_{i+1} = y \oplus k_{i+1}$ as follows,

$$\sigma^{1,i+1}_{wtdw} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Adapt}(\hat{\sigma}^{1,i+1}_{wtdw}, w^1_{i+1}) \quad (15)$$

  Then, v_{i+1} posts the (transaction, signature) pair $(tx^{1,i+1}_{wtdw}, \sigma^{1,i+1}_{wtdw})$ on chain to withdraw a_i.

- **Partial sequential swap (malicious case):** If, for any reason, v_i fails to generate the correct witness y, he/she should wait for his/her outgoing arc to be executed, as shown in Figure 3(c). According to the pre-signatures on the re-lock transactions, for $x \in [1,n-1]$, v_i will re-lock a_i from the x-th joint address $pk^x_{i,i+1}$ to the $(x+1)$-th joint address $pk^{x+1}_{i,i+1}$ at time $x\Delta+t$, i.e., learning the delayed value u^x_{i+1}. If v_{i+1} withdraws a_i from the x-th joint address, v_i can extract w^x_{i+1} by applying $\Sigma_{\mathsf{AS}}.\mathsf{Ext}$ on the pre-signature and on-chain signature pair as follows,

$$w^x_{i+1} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Ext}(\sigma^{x,i+1}_{wtdw}, \hat{\sigma}^{x,i+1}_{wtdw}, S^x_{i+1}). \quad (16)$$

  Then, v_i generates w^{x+1}_i combining with his/her own rid k_i (thus $x+1$ rids in total), such that $w^{x+1}_i := w^x_{i+1} \oplus k_i$, to compute $\sigma^{x+1,i}_{wtdw}$ as follow,

$$\sigma^{x+1,i}_{wtdw} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Adapt}(\hat{\sigma}^{x+1,i}_{wtdw}, w^{x+1}_i). \quad (17)$$

  v_i withdraws the assets a_{i-1} before $(x+1)\Delta+t$ from the $(x+1)$-th joint address $pk^{x+1}_{i-1,i}$ of v_{i-1} and v_i.

Note that, if v_{i-1} does not perform re-lock operations to prevent v_i from withdrawing the assets a_{i-1} from the $(x+1)$-th joint address (as v_i only possesses w^{x+1}_i), v_i can perform re-lock operations, since v_i originally holds the delayed values.

Phase 5: refund. If the assets a_i are not withdrawn before the time $(n-1)\Delta+t$, participant v_i has already re-locked the assets a_i into the last joint address $pk^n_{i,i+1}$ of v_i and v_{i+1}. If, in the next time Δ, a_i is still not withdrawn, v_i opens the last VTD of the secret key share $sk^{n,i+1}_{i,i+1}$ to refund a_i. v_i generates the secret key $sk^n_{i,i+1} = sk^{n,i}_{i,i+1} \oplus sk^{n,i+1}_{i,i+1}$ to sign the refund transaction of the form

$$tx^i_{rfnd} : pk^n_{i,i+1} \xrightarrow{a_i} pk_i. \quad (18)$$

Then, v_i posts the (transaction, signature) pair $(tx^i_{rfnd}, \sigma^i_{rfnd})$ on chain to refund a_i.

5 Security Analysis

We formalize the security of `ParaSwap` in the universal composability (UC) framework [22], [23]. The description of the ideal functionality $\mathcal{F}_{\mathsf{PS}}$ for `ParaSwap` and a formal proof of the following theorem is provided in Appendix B.

Global Input: $(\mathbb{C}_i, \mathbb{C}_{i-1}, a_i, pk_i, pk_{i+1})$, T, t, Δ, ε, for $j \in [1,n], x \in [1,n-1]$. Here, $t = \Delta + 3\varepsilon$ and $T = n\Delta + 3\varepsilon = (n+1)\Delta + 3\varepsilon$.
Participant v_i's **Input:** sk_i; **Participant** v_{i+1}'s **Input:** sk_{i+1}

Preparation Phase

Each participant v_i generates $(Y_i, y_i), (K_i, k_i) \in R$, and sends Y_i, K_i to all participants. Before participant v_i locks the assets a_i intended for the exchange, he/she must complete the following first:

1. Execute the TPC joint address management protocol Π_{JAMan} with the common input $(\mathbb{G}, G, q)$. Output $(pk^j_{i,i+1}, sk^{j,i}_{i,i+1})$ to participant v_i, $(pk^j_{i,i+1}, sk^{j,i+1}_{i,i+1})$ to participant v_{i+1}, for $j \in [1,n]$.
2. Participant v_{i+1} picks $(u^x_{i+1}, U^x_{i+1}) \in R$, for $x \in [1,n-1]$, and generates $(C^x_{i+1}, \pi^x_{i+1}) \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{ComPro}(u^x_{i+1}, x\Delta + t)$, for $x \in [1,n-1]$,
$(C^n_{i+1}, \pi^n_{i+1}) \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{ComPro}(sk^{n,i+1}_{i,i+1}, n\Delta + t)$, then sends $(C^j_{i+1}, \pi^j_{i+1}, U^x_{i+1})$ to v_i.
3. Participant v_i checks $\{0,1\} \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{Vrfy}(C^x_{i+1}, \pi^x_{i+1}, U^x_{i+1})$ and $\{0,1\} \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{Vrfy}(C^n_{i+1}, \pi^n_{i+1}, pk^n_{i,i+1})$, and aborts if outputs 0.

Pre-swap Phase

1. Participant v_i generates $tx^i_{lck} := (pk_i, pk^1_{i,i+1}, a_i)$ and the corresponding signature $\sigma^i_{lck} \leftarrow \Sigma_{\mathsf{DS}}.\mathsf{Sign}(sk_i, tx^i_{lck})$. Then v_i posts $(tx^i_{lck}, \sigma^i_{lck})$ on $\mathbb{C}_i$.
2. Participant v_i starts solving $\Sigma_{\mathsf{VTD}}.\mathsf{ForceOp}(C^j_{i+1})$, for $j \in [1,n]$.
3. Participant v_{i+1} defines the withdraw transactions $tx^{j,i+1}_{wtdw} := tx(pk^j_{i,i+1}, pk_{i+1}, a_i)$, for $j \in [1,n]$, and sends them to v_i.
4. Participant v_i defines the re-lock transactions $tx^{x,i}_{rlck} := tx(pk^x_{i,i+1}, pk^{x+1}_{i,i+1}, a_i)$, for $x \in [1,n-1]$, and send them to v_{i+1}.
5. For $j \in [1,n]$, participants v_i and v_{i+1} generate $S^j_{i+1} = Y \cdot K_{i+1} \cdot K_{i+2} \cdot \cdots \cdot K_{i+j}$, and run the TPC joint pre-signing protocol Π_{JPSig} for withdraw transactions that does the following:
   (a) The protocol Π_{JPSig} takes $(tx^{j,i+1}_{wtdw}, S^j_{i+1})$ as public input from both participants, takes $sk^{j,i}_{i,i+1}$ from participant v_i and $sk^{j,i+1}_{i,i+1}$ from v_{i+1} as private input, and computes $sk^j_{i,i+1} = sk^{j,i}_{i,i+1} \oplus sk^{j,i+1}_{i,i+1}$.
   (b) The protocol Π_{JPSig} computes $\hat{\sigma}^{j,i+1}_{wtdw} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{PreSig}(tx^{j,i+1}_{wtdw}, sk^j_{i,i+1}, S^j_{i+1})$ and outputs $\hat{\sigma}^{j,i+1}_{wtdw}$ to participant v_i and v_{i+1}.
   (c) Participant v_i and v_{i+1} check $\{0,1\} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{PreVf}(tx^{j,i+1}_{wtdw}, pk^j_{i,i+1}, S^j_{i+1}, \hat{\sigma}^{j,i+1}_{wtdw})$, and abort if outputs 0.
6. For $x \in [1,n-1]$, participants v_i and v_{i+1} run the TPC protocol Π_{JPSig} for re-lock transactions that does the following:
   (a) The protocol Π_{JPSig} takes $(tx^{x,i}_{rlck}, U^x_{i+1})$ as public input from both participants, takes $sk^{x,i}_{i,i+1}$ from participant v_i and $sk^{x,i+1}_{i,i+1}$ from v_{i+1} as private input, and computes $sk^x_{i,i+1} = sk^{x,i}_{i,i+1} \oplus sk^{x,i+1}_{i,i+1}$.
   (b) The protocol Π_{JPSig} computes $\hat{\sigma}^{x,i}_{rlck} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{PreSig}(tx^{x,i}_{rlck}, sk^x_{i,i+1}, U^x_{i+1})$, and outputs $\hat{\sigma}^{x,i}_{rlck}$ to participant v_i and v_{i+1}.
   (c) Participant v_i and v_{i+1} check $\{0,1\} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{PreVf}(tx^{x,i}_{rlck}, pk^x_{i,i+1}, U^x_{i+1}, \hat{\sigma}^{x,i}_{rlck})$, and abort if outputs 0.

Witness Sharing Phase

Each Participant v_i sends the witness y_i to all participants, for $i \in [1,n]$.

Figure 4: Preparation, pre-swap, and witness sharing phase of `ParaSwap`.

Swap Phase

1. If the received $y_1, y_2 \cdots, y_n$ are correct, participants v_i and v_{i+1} compute $y = y_1 \oplus y_2 \cdots \oplus y_n$ and do the following:
   (a) Participant v_{i+1} computes $\sigma_{wtdw}^{1,i+1} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Adapt}(\hat{\sigma}_{wtdw}^{1,i+1}, y \oplus k_{i+1})$, and posts $(tx_{wtdw}^{1,i+1}, \sigma_{wtdw}^{1,i+1})$ on $\mathbb{C}_i$.
   (b) Participant v_i computes $\sigma_{wtdw}^{1,i} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Adapt}(\hat{\sigma}_{wtdw}^{1,i}, y \oplus k_i)$, and posts $(tx_{wtdw}^{1,i}, \sigma_{wtdw}^{1,i})$ on $\mathbb{C}_{i-1}$.
2. Otherwise, participant v_{i+1} waits for the execution of his/her outgoing arc and computes $\Sigma_{\mathsf{AS}}.\mathsf{Ext}$, $\sigma_{wtdw}^{j,i+1} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Adapt}(\hat{\sigma}_{wtdw}^{j,i+1}, y \oplus k_{i+1} \oplus \cdots k_{i+j})$, and posts $(tx_{wtdw}^{j,i+1}, \sigma_{wtdw}^{j,i+1})$ on $\mathbb{C}_i$, for $j \in [1,n]$. Then participant v_i can compute $y \oplus k_{i+1} \oplus \cdots k_{i+j} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Ext}(\sigma_{wtdw}^{j,i+1}, \hat{\sigma}_{wtdw}^{j,i+1})$.
3. For $x \in [1, n-1]$, if participant v_{i+1} fails to post $(tx_{wtdw}^{x,i+1}, \sigma_{wtdw}^{x,i+1})$ for any reason before time $x\Delta + t$ on $\mathbb{C}_{i+1}$, participant v_i does the following:
   (a) Finish computing $u_{i+1}^x \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{ForceOp}(C_{i+1}^x)$.
   (b) Compute $\sigma_{rlck}^{x,i} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Adapt}(\hat{\sigma}_{rlck}^{x,i}, u_{i+1}^x)$, and post $(tx_{rlck}^{x,i}, \sigma_{rlck}^{x,i})$ on $\mathbb{C}_i$ to re-lock a_i from $pk_{i,i+1}^x$ to $pk_{i,i+1}^{x+1}$.

Refund Phase

Similarly, if participant v_{i+1} fails to post $(tx_{wtdw}^{n,i+1}, \sigma_{wtdw}^{n,i+1})$ before time $T = n\Delta + t$ on $\mathbb{C}_i$, participant v_i does the follows:

1. Finish computing $sk_{i,i+1}^{n,i+1} \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{ForceOp}(C_{i+1}^n)$.
2. Compute $\sigma_{rfnd}^i \leftarrow \Sigma_{\mathsf{DS}}.\mathsf{Sign}(tx_{rfnd}^i, sk_{i,i+1}^{n,i} \oplus sk_{i,i+1}^{n,i+1})$, and refunds a_i by posting $(tx_{rfnd}^i, \sigma_{rfnd}^i)$ on $\mathbb{C}_i$.

Figure 5: Swap and refund phase of ParaSwap

Theorem 1. *Assume the underlying signature scheme is the Schnorr or ECDSA signature scheme. Let* $\Sigma_{\mathsf{AS}} := (\mathsf{PreSig}, \mathsf{Adapt}, \mathsf{PreVf}, \mathsf{Ext})$ *denote a secure adaptor signature scheme based on a strongly unforgeable signature scheme* $\Sigma_{\mathsf{DS}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$*, along with a hard relation R. Let* Π_{JAMan} *and* Π_{JPSig} *denote UC secure TPC protocols for joint address management and joint pre-signing, respectively. Let* Σ_{VTD} *denote a timed sound and private verifiable timed discrete logarithm scheme. Consequently,* ParaSwap*, as outlined in Figure 4 and 5, UC-realizes the ideal functionality* $\mathcal{F}_{\mathsf{PS}}$ *with access to the functionalities* ($\mathcal{F}_{\mathtt{clock}}, \mathcal{F}_{\mathtt{smt}}, \mathcal{F}_{\mathcal{L}}$)*.*

5.1 Property Analysis of ParaSwap

We prove that ParaSwap performs in parallel and ensures atomicity.

Theorem 2. ParaSwap *performs in parallel if all participants adhere to it.*

Proof. If all participants adhere to ParaSwap, two involved participants generate n necessary joint addresses for each arc in the directed cycle. Each participant generates n VTDs of $n-1$ delayed values and the last secret key share on their incoming arc in the preparation phase. In the pre-swap phase, participants lock their assets on chain after verifying the VTDs, then the two involved participants jointly generate the pre-signatures for withdraw and re-lock operations. In the witness sharing phase, participants send their witnesses to each other. The first three phases are completed by time $t = \Delta + 3\varepsilon$, and all participants can generate the correct global witness using all witnesses. Thus, each participant concurrently withdraws the assets on the incoming arc using the global witness and his/her identifier rid, taking at most Δ. In conclusion, the exchange is complete within $2\Delta + 3\varepsilon$.

Therefore, as all participants adhere to ParaSwap, all operations are performed in parallel, and Theorem 2 is proven. □

Definition 1. *(Atomicity). An atomic swap construction satisfies atomicity if, for each PPT participant* v_i*, the following conditions hold simultaneously:*

1) If v_i*'s locked assets are withdrawn by others,* v_i *can withdraw the assets he/she intends.*

2) If v_i*'s locked assets are not withdrawn by others after the refund time,* v_i *can refund the locked assets.*

Theorem 3. ParaSwap *ensures atomicity for the security of the adaptor signature scheme* Σ_{AS} *and the security of the VTD scheme* Σ_{VTD}*.*

Proof. If the atomicity of ParaSwap is compromised, the following cases can occur: (i) the assets a_i are withdrawn by v_{i+1} from the x-th joint address, but v_i cannot withdraw the assets a_{i-1}, or (ii) v_i cannot refund the locked assets if they are not withdrawn after the refund time. More precisely, we discuss the two cases as follows:

- This first case can only happen if: i) v_i cannot re-lock a_{i-1} to the $(x+1)$-th joint address of v_{i-1} and v_i, which implies the pre-signature adaptability of Σ_{AS} [17], or ii) v_i cannot generate the valid signature for the withdraw transaction of $(x+1)$-th joint address. The latter indicates that v_{i+1} provides a valid signature on his/her withdraw transaction which either does not reveal w^x_{i+1}, the sum of the global witness and the rids of x participants, which implies the witness indistinguishability of Σ_{AS}, or reveals an invalid value that is useless to v_i, which implies the pre-signature adaptability of Σ_{AS}.
- The second case can only happen if the assets a_i that are not withdrawn, cannot be re-locked into the last joint address $pk^n_{i,i+1}$, and then refunded. This entails the following sub-events: i) v_i cannot learn the delayed values or the secret key share from VTDs, which implies the soundness of Σ_{VTD} [43], or ii) the VTDs reveal invalid values that are useless to v_i, which implies the pre-signature adaptability of Σ_{AS}.

□

5.2 Tackling Attacks

In the following, we analyze how ParaSwap tackles the security threats mentioned in Section 2.2.

Threat-1: message manipulation attacks. A malicious participant v_{i+1} attempts to disrupt ParaSwap by sending incorrect or no VTDs in the preparation phase, or providing incorrect or no secret key shares in the pre-swap phase. However, the verification mechanisms of Σ_{VTD} and Σ_{AS} ensure that any verification failure prompts v_i to halt the current phase and not share his/her own witness. This prevents v_{i+1} from generating y, thereby protecting a_i from loss.

Threat-2: boundary attacks. A malicious participant v_{i+1} acts maliciously by sending incorrect or no witness to other participants during the witness sharing phase, and then attempts to withdraw a_i near the assets refund time. However, after $\Delta+t$, v_i learns u^1_{i+1} from the first VTD and re-locks a_i from $pk^1_{i,i+1}$ to $pk^2_{i,i+1}$. If v_{i+1} attempts to withdraw a_i from $pk^2_{i,i+1}$, he/she will require two rids, which can only be revealed by other participants when withdrawing assets on their incoming arc. Therefore, v_{i+1} can only withdraw a_i before $\Delta+t$ from the first joint address. As described in Theorem 3, if v_{i+1} withdraws a_i, v_i can then withdraw a_{i-1}. Thus, other participants can still avoid any losses.

Threat-3: collusion attacks. Assume $n-1$ participants, excluding the target participant v_i, form a coalition by sharing their witnesses and rids to gain larger overall benefits. While this could prevent v_i from withdrawing a_{i-1} under the normal swap scenario, the coalition is limited by the time point $(n-1)\Delta+t$ from the first $n-1$ joint address, since they possess only $n-1$ rids. The rid k_i of v_i is revealed only when v_i withdraws a_{i-1}. Therefore, if the coalition withdraws a_i, v_i still has a Δ-time window to withdraw a_{i-1} from the next address. Notably, even if v_{i-1} does not re-lock a_{i-1} to block v_i from withdrawing, v_i can perform the re-lock operations. This is because v_i originally holds the delayed values required for re-locking, allowing v_i to complete the swap without losses.

6 Performance Analysis

In this section, we analyze the performance of ParaSwap theoretically and experimentally.

6.1 Theoretical Analysis

In Table 1 and 2, we compare the computational and communication complexity of ParaSwap and TUSwap (the two-party universal atomic swap protocol) [44], involving n participants. As shown in Table 1, in the preparation phase, both schemes have a computational complexity of $O(n)\varepsilon$. This is because each participant concurrently generates and verifies n VTDs in ParaSwap, and all n participants generate and verify one VTD sequentially in TUSwap. In the pre-swap and swap phases, the on-chain lock and withdraw operations in ParaSwap are performed in parallel, whereas TUSwap requires all participants to perform these operations sequentially, leading to a time complexity of $O(1)\Delta$ and $O(n)\Delta$, respectively. Additionally, the off-chain pre-signature generation and verification, similar to the generation of VTDs in the preparation phase, have the same complexity of $O(n)\varepsilon$ in both schemes. In the witness sharing phase of ParaSwap, each participant concurrently broadcasts his/her witness to others, leading to a time complexity of $O(1)\varepsilon$, while TUSwap does not need this.

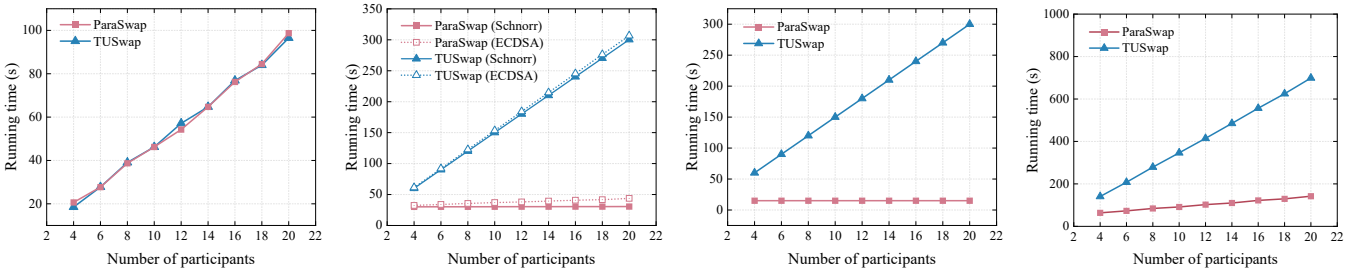
Table 1: Comparison of computational complexity.

Scheme	Preparation	Pre-swap	Witness sharing	Swap
ParaSwap	$O(n)\varepsilon$	$O(1)\Delta+O(n)\varepsilon$	$O(1)\varepsilon$	$O(1)\Delta$
TUSwap	$O(n)\varepsilon$	$O(n)\Delta+O(n)\varepsilon$	0	$O(n)\Delta$

Table 2: Comparison of communication complexity.

Scheme	Preparation	Pre-swap	Witness sharing
ParaSwap	$O(n^2)$	$O(n^2)$	$O(n^2)$
TUSwap	$O(n)$	$O(n)$	0

As shown in Table 2, in the preparation, each participant in ParaSwap sends n VTDs to the participant who locks the assets that the former wants, resulting in a communication complexity of $O(n^2)$ for n participants. While TUSwap requires only one VTD per participant, leading to a complexity of $O(n)$. Similarly, in the pre-swap phase, ParaSwap involves the joint generation of $2n-1$ adaptor signatures per pair of participants, maintaining $O(n^2)$ complexity for n participants, whereas TUSwap requires two adaptor signatures, achieving $O(n)$. In the witness sharing phase of ParaSwap, each participant broadcasts his/her witness to others, leading to a



(a) Preparation phase (b) Pre-swap phase (c) Swap phase (d) Overall running time for swap

Figure 6: Running time of ParaSwap and TUSwap [44].

complexity of $O(n^2)$. In contrast, TUSwap does not need to share witnesses. Although ParaSwap incurs higher communication overhead, the additional cost is acceptable in practice. This is further discussed in Section 6.2, where we evaluate the performance of ParaSwap in real-world multi-party multi-chain exchanges.

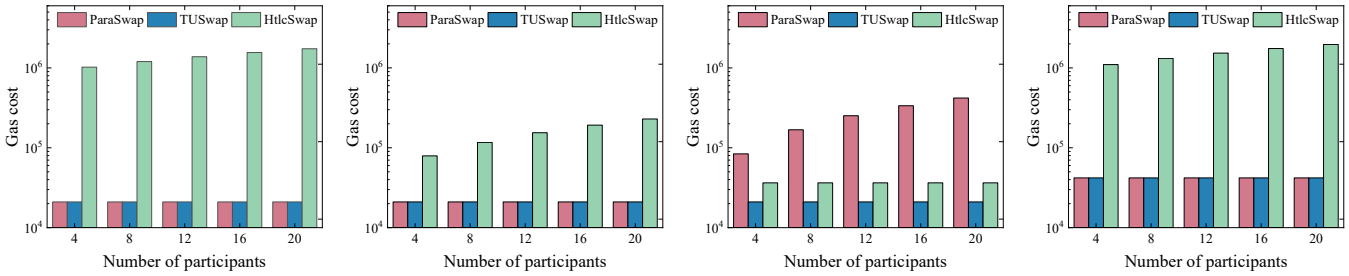
6.2 Experimental Evaluations

Setup and Implementation. We implement ParaSwap in C, where the VTD scheme Σ_{VTD} relies on OpenSSL, Crypto, and SSS [13] programming libraries, and the TPC protocol Π_{JPSig} for the adaptor signature scheme Σ_{AS} is implemented with reference to [42]. The security parameter λ is set to 256. To simulate a real-world multi-chain environment, we evaluate ParaSwap with n participants distributed across four public blockchain test networks: Ethereum, Bitcoin (Rootstock, a Bitcoin sidechain supporting Ethereum-compatible functionality), Binance Smart Chain, and Avalanche. Each blockchain is hosted on a separate Aliyun server. The number of participants n varies from 4 to 20, and each participant is configured on a machine with an Intel Core i5 CPU and 8GB of RAM. Each experiment result is the average running time of 100 tests, all conducted on Ubuntu 22.04 LTS. For comparison, we also implement TUSwap (the two-party universal atomic swap protocol) proposed in [44] and HtlcSwap (the multi-party atomic swap protocol across multiple chains using hash-time lock contract) proposed in [30].

Computation Overhead. Here we evaluate the running time of ParaSwap and TUSwap with varying numbers of participants. Note that it is challenging to evaluate the time of on-chain operations in practice accurately. Thus, we reference Ethereum, where the average confirmation delay for one transaction is around 15s, as discussed in Section 4.1. In Figure 6a, we plot the running time of the preparation phase, where both schemes exhibit a similar linear increase as the number of participants grows. This is because, in ParaSwap, each participant concurrently generates and verifies n VTDs, whereas, in TUSwap, all n participants generate and verify one VTD sequentially. Figure 6b shows the running time in the pre-swap phase. Both schemes are implemented using ECDSA and Schnorr signature schemes, which are commonly adopted in blockchain systems to sign transactions. We can see that TUSwap consistently takes longer than ParaSwap, regardless of the signature scheme used. This arises from TUSwap's serial on-chain lock operations. Furthermore, schemes using ECDSA as the underlying signature are slower compared to those using Schnorr. Specifically, generating pre-signatures in ECDSA requires significantly more time—roughly 339.69ms per instance of Π_{JPSig} compared to 14.72ms for Schnorr. From Figure 6c, ParaSwap maintains a constant running time of 15s in the swap phase, regardless of the number of participants, as all participants concurrently withdraw assets. In contrast, the running time of TUSwap increases linearly, since the serial withdraw operations.

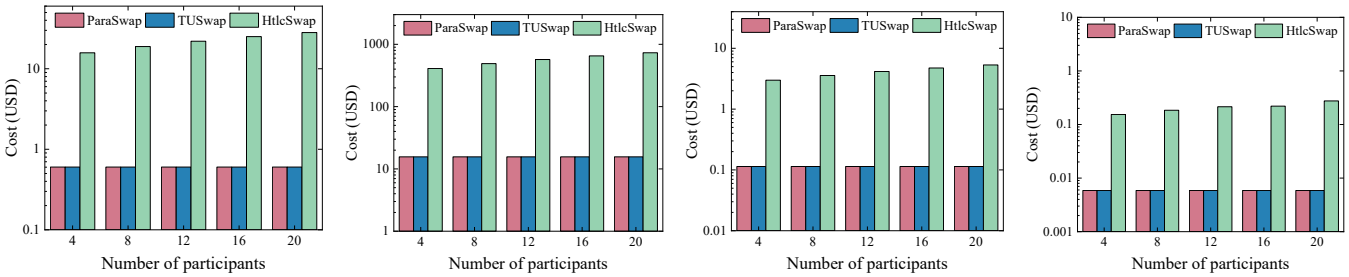
We omit the witness sharing and refund phases in evaluation since the former depends on network latency, and the latter only occurs when assets remain unswapped, with a predetermined refund time. As shown in Figure 6d, ParaSwap significantly outperforms TUSwap, reducing the total runtime by at least 79.7% for an exchange involving 20 participants, where TUSwap requires approximately 699s and ParaSwap only takes around 142s. Moreover, the performance advantage of ParaSwap grows with the number of participants.

Communication Overhead. Here we compare the communication overhead in ParaSwap and TUSwap for n participants, and the results are shown in Table 3. In the preparation phase, each VTD incurs a communication cost of 1564 KB, based on the statistical parameter $n_0 = 64$ and the settings described in [43] (RSA 1024-bit module). In the pre-swap phase, each pair of participants requires 416 bytes to generate an ECDSA-based adaptor signature, while 256 bytes per Schnorr-based adaptor signature. As expected, these findings are the same as those represented in [34] and [44]. While our work achieves a significant improvement in time consumption from $O(n)$ in TUSwap to $O(1)$ through concurrency, this improvement inevitably introduces additional communication overhead to ensure atomicity under concurrent execution. However, the additional communication overhead is acceptable in practice. For example, for 10 participants, our work saves 258s compared to TUSwap, while increasing about 138 MB of *off-chain* communication, which is acceptable in a LAN or WLAN. Specifically, under a 1 Gbps network band-



(a) Create/Lock operation (b) Withdraw operation (c) Refund operation (d) Overall gas cost for swaps

Figure 7: Gas cost per participant of ParaSwap, TUSwap [44], and HtlcSwap [30].



(a) Cost on Ethereum (b) Cost on Bitcoin (c) Cost on Binance Smart Chain (d) Cost on Avalanche

Figure 8: On-chain costs per participant when asset swap, where participants are evenly developed on Ethereum, Bitcoin, Binance Smart Chain, and Avalanche. The gas price is set at 4 Gwei, with an exchange rate of 3579 USD per Ether (ETH), 93003 USD per Bitcoin (BTC), 677 USD per Binance Coin (BNB), and 35 USD per Avalanche Coin (AVAX) (as of Dec. 2024).

width [16], the extra 138 MB can be transmitted in around 0.2s, which is negligible compared to the time savings. Given that the saved latency directly improves user experience, especially in high-frequency or multi-party swap scenarios, we believe this trade-off is well justified.

Table 3: Comparison of communication overhead.

Phase	TUSwap	ParaSwap
Preparation (KB)	$1564 \times n$	$1564 \times n^2$
Pre-swap based on Schnorr (B)	$256 \times 2n$	$256 \times n(2n-1)$
Pre-swap based on ECDSA (B)	$416 \times 2n$	$416 \times n(2n-1)$

On-Chain Cost. Here we evaluate the on-chain cost of ParaSwap, TUSwap, and HtlcSwap among n participants. We calculate the on-chain cost for three main operations in HtlcSwap: (i) contract creation, (ii) assets withdrawal using the correct hash preimage, and (iii) assets refund after the specified time has expired. In ParaSwap and TUSwap, the corresponding operations involve submitting lock, withdraw (potentially including re-lock), and refund transactions on chain. We measure each transaction with a standard cost of 21000 gas (the normal Ethereum transaction cost).

In Figure 7a, the gas cost per participant of ParaSwap is 21000 for only one lock transaction, which is 48.7× and 82.7× lower than those of HtlcSwap and equal to TUSwap. In Figure 7b, ParaSwap incurs a consistent gas cost of 21,000 per participant, cause it only involves one withdraw transaction, the same as TUSwap. However, as shown in Figure 7c, ParaSwap's gas cost per participant increases with the number of participants during the assets refund. This is because, apart from one refund transaction, ParaSwap also requires $n-1$ re-lock transactions, resulting in a total gas cost of $21,000n$. Overall, as shown in Figure 7d, ParaSwap reduces gas costs by 26.2× to 46.8× compared to HtlcSwap during normal asset swaps. Additionally, Figure 8 shows that deployments of ParaSwap on Ethereum, Bitcoin, Binance Smart Chain, and Avalanche reduce on-chain costs by 749.3 USD compared to HtlcSwap when there are 20 participants.

HtlcSwap requires not only higher fee costs but also larger transaction sizes due to the need to store contract parameters, such as hash values, exchange partner, and time parameters. The additional overhead, given the limited on-chain storage space, further restricts the scalability of the scheme.

7 Related Work

The increasingly growing cryptocurrency exchanges among users sparked cross-chain atomic swap scheme research.

Trusted Third Party. As a simple approach to achieving cross-chain atomic swaps, the notary scheme [49], [48], introduces a trusted third party or a group of parties to act as the verifier(s) and coordinator(s) of cross-chain transactions. When an initiator performs a transaction on the source chain, the notary witnesses events on that chain to verify the transac-

tion's validity, and notifies another chain of the corresponding operation after the verification is passed [3,6,15,45]. However, notary schemes have the following drawbacks: (i) whether a trusted third party or a group of parties, the approach is against the decentralization inherent to blockchains, and (ii) vulnerable to single-point failures and malicious behaviors. For example, on May 7, 2019, hackers stole 40 million worth of Bitcoin in a major security breach at Binance [11].

Sidechain/Relay. A sidechain/relay enables the transfer of assets between different blockchains through a technique known as two-way pegging. For instance, XCLAIM [51] employs the BTCRelay [5], an Ethereum smart contract that stores Bitcoin block headers, for trustless atomic swaps between Bitcoin and Ethereum. However, sidechains/relays are typically ecosystem-specific, lack broad compatibility, and involve high implementation complexity and technical costs. An alternative approach for implementing atomic swap functionality is by leveraging a trusted execution environment (TEE) [19]. Bentov et al. [18] constructs a secure real-time cryptocurrency exchange platform that utilizes a TEE as a trusted relay. However, it assumes all participants have a TEE, which is impractical; and the recent research has identified significant security vulnerabilities in TEEs [47].

Hash-Time Lock Contract. In recent studies, the hash-time lock contract (HTLC) has emerged as the primary technology for addressing atomic cross-chain swap issues in the cryptocurrency domain [20, 21, 29, 33, 38, 46]. Originating from [1], HTLCs leverage hashlocks and timelocks to ensure transaction atomicity. Hashlocks require the user to know the preimage of a hash to unlock assets, while timelocks ensure assets are refunded to the original account if not unlocked within a specified time. Most recently, a new atomic cross-chain swap protocol supporting multiple parties across multiple chains was proposed [30]. However, HTLCs involve complex implementation details and on-chain costs, including high execution costs and large transaction sizes, due to large scripts. Furthermore, such an approach requires that all cryptocurrencies involved must be compatible and support the specific hash function, posing a considerable barrier in practice. Another major limitation is that some cryptocurrencies do not support the computing of HTLCs, such as Ripple [41], Zcash [40], and Stellar [35].

It is worth noting that some cross-chain atomic swap schemes, such as [1,29,30,42,44,51], follow a two-phase commit (2PC) design. The 2PC mechanism is a line of work from the databases that has existed for decades. It typically involves a pre-commit phase followed by a commit phase. According to [50], the 2PC mechanism was first ported to blockchain research in [1], which proposes the original idea of HTLC. In this scheme, parties first pre-commit to the exchange and can explicitly abort the protocol in case of disagreement or failure during the commit phase. We explain this mechanism more specifically as follows. Suppose Alice holds assets a_1 on $\mathbb{C}_1$, Bob holds assets a_2 on $\mathbb{C}_2$, and Alice wants to exchange a_1 for a_2 and acts as an initiator. She chooses a secret value y and computes a hash lock $Y = H(y)$ using a cryptographic hash function H. Then Alice chooses a time lock t_1 which specifies that if Bob fails to withdraw a_1 before t_1 elapses, a_1 will be refunded to Alice. For the **pre-commit phase**, Alice locks a_1 into an HTLC in $\mathbb{C}_1$, and sends Y and t_1 to Bob. Afterwards, Bob chooses a time $t_2 < t_1$ and locks a_2 into an HTLC in $\mathbb{C}_2$ using the same hash lock h. At this point, Bob cannot withdraw a_1 because only Alice knows y. Both parties have locked their assets under the same release conditions. For the **commit phase**, there are two possible cases: (Case i) Alice withdraws a_2 from the HTLC in $\mathbb{C}_2$, revealing the secret y to Bob before t_2, then Bob can use y to withdraw a_1 from the HTLC in $\mathbb{C}_1$ before t_1; (Case ii) Alice does not withdraw a_2 before t_2, ensuring that y is not revealed, thereby Bob cannot withdraw a_1, which eventually is refunded to Alice after t_1.

In our work, we extend the two-party atomic swaps to support atomic swaps among multiple parties across multiple blockchains. Specifically, instead of the single initiator, we set all participants to act as initiators. During the pre-commit phase, each participant locks their assets using the global statement Y, which is generated from the secret values y_i chosen by all participants. All assets are locked with a uniform refund time. During the commit phase, each participant generates the global secret y from the individual y_i to withdraw their intended assets. Notably, both asset locking and withdrawal are performed in parallel by all participants, without the sequential dependency described above, where Bob waits for Alice to lock and withdraw assets. Additionally, we introduce a novel re-lock approach to ensure atomicity by preventing partial execution of swaps with a uniform refund time.

8 Conclusion

In this paper, we investigate the problem of atomic swaps among multiple parties across multiple blockchains. We propose `ParaSwap`, the first framework to parallelize universal atomic swaps for cryptocurrency exchanges across multiple chains. We replace the serial multiple two-party swaps with a new concurrent mechanism and introduce a novel re-lock approach to ensure atomicity. We conduct a rigorous security analysis and implement `ParaSwap`. Large-scale experiment evaluations show that `ParaSwap` significantly outperforms existing multi-party schemes in both cost and efficiency.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 62232002) and the Young Elite Scientists Sponsorship Program by CAST (Grant No. 2023QNRC001).

9 Ethics Considerations

In developing and evaluating ParaSwap, we carefully considered the ethical implications for all stakeholders in the blockchain ecosystem, including blockchain users, developers, researchers, and the broader community. Our analysis follows ethical principles outlined in The Menlo Report and USENIX Security '25 Ethics guidelines, ensuring that our work benefits users while minimizing potential risks.

Beneficence (Benefits vs. Harms)

- Benefits: ParaSwap parallelizes universal atomic swaps for cryptocurrency exchanges among multiple parties across multiple blockchains and minimizes the significant time cost from $O(n)$ caused by the serial process to $O(1)$. The framework avoids the risk of asset theft and ensures atomicity by preventing partial execution of transactions with a uniform refund time used to avoid asset deadlock in parallel, offering significant benefits to legitimate users, especially in decentralized finance (DeFi) environments.
- Potential Harms: In our work, malicious miners may manipulate transaction ordering over normal users to engage in illicit activities (i.e., MEV extraction). However, this is an inherent challenge in blockchain technology, and existing atomic swap schemes, such as [29, 30, 42, 44], including ours, have risks associated with MEV extraction. Similar to previous related works, our work focuses on atomic swaps between participants who exchange their assets, and the MEV extraction issues are out of the study scope. Moreover, there are substantial and well-developed MEV countermeasures, such as Flashbots [8], SUAVE [9], MEV-Boost [12], and Skip Protocol [14]. Further ethical discussion on MEV is given in Appendix C.

Respect for Persons. We ensured that the research respects the autonomy of all individuals involved by: (i) All protocol details of ParaSwap are transparent and publicly documented. (ii) We have not collected or analyzed any private user data during the development and evaluation of ParaSwap. Users maintain full control over their assets and transaction privacy.

Justice. ParaSwap design ensures fairness by: (i) ParaSwap is designed to be accessible to all blockchain users, regardless of their technical sophistication. (ii) The protocol distributes computational costs fairly between parties, ensuring that no single participant bears an undue burden.

Respect for Law and Public Interest. In developing ParaSwap, we took into account legal considerations and public interest: (i) The protocol design fully complies with existing blockchain and cryptocurrency regulations, ensuring it adheres to legal frameworks and user protection laws. (ii) The implementation uses well-established cryptographic primitives and follows security best practices, ensuring the protocol's robustness against attacks.

Implementation and Testing Ethics. We conducted all experimental evaluations ethically by: (i) Our experimental evaluation was conducted using only test networks and simulated environments. We did not interact with or impact any live blockchain systems or real user assets. (ii) Ensuring that the results are reproducible and verifiable by making the full implementation and evaluation scripts publicly available. (iii) Gas cost measurements were conducted on controlled test networks to avoid impacting real-world networks.

Disclosure and Documentation. We adhere to responsible disclosure by: (i) Our work specification and security proofs are fully documented. (ii) All code will be released as open source upon publication. (iii) Any discovered vulnerabilities will follow standard disclosure procedures.

Terms of Service. Our experiments do not violate any terms of service.

Wellbeing for Team Members. Our research activities do not have the potential to negatively impact team members. And we do not crawl morally questionable websites from a network.

Deception. Participants are fully informed of the purposes and risks (among other things) of participating in experiments.

Experiments with Live Systems Without Informed Consent. Our work does not involve experiments with live systems.

We believe that ParaSwap makes a positive contribution to blockchain technology by improving the efficiency of cross-chain cryptocurrency exchanges while taking into account potential risks. We encourage further community engagement to explore challenges in efficiency, overhead, and security in the domain of cross-chain cryptocurrency exchanges with ethical considerations.

10 Open Science

Data and Code Availability. We are committed to the principles of open science and reproducible research. To support this, the full implementation of ParaSwap, including cryptographic tools and off-chain components (e.g., the two-party computation protocol of adaptor signature and the VTD scheme), is open-source and permanent on Zenodo[3]. All experimental parameters, including configurations and settings used for testing gas costs and system performance, are documented and shared. We believe that our artifacts are available.

References

[1] Alt chains and atomic transfers. bitcointalk.org.

[2] Avalanche whitepaper. https://www.avalabs.org/whitepapers.

[3] https://doi.org/10.5281/zenodo.15594022

[3] Binance. https://www.binance.com. Accessed 23 Sept 2023.

[4] Bnb smart chain white paper. https://github.com/bnb-chain/whitepaper/tree/master.

[5] Btcrelay: Ethereum contract for bitcoin spv. https://github.com/ethereum/btcrelay. Accessed 23 Sept 2024.

[6] Coinbase. https://www.coinbase.com. Accessed 23 Sept 2023.

[7] Defillama. https://defillama.com/. Accessed 4 Oct 2023.

[8] Flashbots auction. https://docs.flashbots.net/flashbots-auction/overview.

[9] The future of mev is suave. https://writings.flashbots.net/the-future-of-mev-is-suave.

[10] Goingecko. https://www.coingecko.com/en/exchanges. Accessed 4 Oct 2023.

[11] Hackers steal $40 million worth of bitcoin in massive security breach. https://www.cnn.com/2019/05/08/tech/bitcoin-binance-hack/index.html. Accessed 23 Sept 2024.

[12] Mev-boost. https://github.com/flashbots/mev-boost.

[13] Shamir secret sharing library. https://github.com/dsprenkels/sss.git.

[14] Skip protocol. https://www.skip.build/.

[15] Uniswap v3 core. https://app.uniswap.org/swap. Accessed 23 Sept 2023.

[16] Naser Al-Falahy and Omar Y Alani. Technologies for 5g networks: Challenges and opportunities. *It Professional*, 19(1):12–20, 2017.

[17] Lukas Aumayr, Oğuzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized bitcoin-compatible channels. 2020.

[18] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1521–1538, 2019.

[19] Iddo Bentov, Ari Juels, Fan Zhang, Philip Daian, and Lorenz Breidenbach. Real-time cryptocurrency exchange using trusted hardware, February 8 2022. US Patent 11,244,309.

[20] BitcoinWiki. Hash time locked contracts. https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts. Accessed 4 Oct 2023.

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

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

[23] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi 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*, pages 61–85. Springer, 2007.

[24] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 154–167, 2016.

[25] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*, pages 910–927. IEEE, 2020.

[26] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 625–656. Springer, 2019.

[27] Zhonghui Ge, Jiayuan Gu, Chenke Wang, Yu Long, Xian Xu, and Dawu Gu. Accio: Variable-amount, optimized-unlinkable and nizk-free off-chain payments via hubs. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1541–1555, 2023.

[28] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, pages 295–310. Springer, 1999.

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

[30] Soichiro Imoto, Yuichi Sudo, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Atomic cross-chain swaps with improved space, time and local time complexities. *Information and Computation*, 292:105039, 2023.

[31] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *Theory of Cryptography Conference*, pages 477–498. Springer, 2013.

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

[33] Léonard Lys, Arthur Micoulet, and Maria Potop-Butucaru. R-swap: Relay based atomic cross-chain swap protocol. In *Algorithmic Aspects of Cloud Computing: 6th International Symposium, ALGOCLOUD 2021, Lisbon, Portugal, September 6–7, 2021, Revised Selected Papers 6*, pages 18–37. Springer, 2021.

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

[35] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 32:1–45, 2015.

[36] Pedro Moreno-Sanchez, Arthur Blue, Duc V Le, Sarang Noether, Brandon Goodell, and Aniket Kate. Dlsag: non-interactive refund transactions for interoperable payment channels in monero. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, pages 325–345. Springer, 2020.

[37] Michael Pacheco, Gustavo Oliva, Gopi Krishnan Rajbahadur, and Ahmed Hassan. Is my transaction done yet? an empirical study of transaction processing times in the ethereum blockchain platform. *ACM Transactions on Software Engineering and Methodology*, 32(3):1–46, 2023.

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

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

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

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

[42] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A 2 l: Anonymous atomic locks for scalability in payment channel hubs. In *2021 IEEE symposium on security and privacy (SP)*, pages 1834–1851. IEEE, 2021.

[43] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. Verifiable timed signatures made practical. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1733–1750, 2020.

[44] Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps: Secure exchange of coins across all blockchains. In *2022 IEEE symposium on security and privacy (SP)*, pages 1299–1316. IEEE, 2022.

[45] Hangyu Tian, Kaiping Xue, Xinyi Luo, Shaohua Li, Jie Xu, Jianqing Liu, Jun Zhao, and David SL Wei. Enabling cross-chain transactions: A decentralized cryptocurrency exchange protocol. *IEEE Transactions on Information Forensics and Security*, 16:3928–3941, 2021.

[46] Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. Mad-htlc: because htlc is crazy-cheap to attack. In *2021 IEEE symposium on security and privacy (SP)*, pages 1230–1248. IEEE, 2021.

[47] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D Garcia, and Frank Piessens. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1741–1758, 2019.

[48] Gang Wang. Sok: Exploring blockchains interoperability. *Cryptology ePrint Archive*, 2021.

[49] Pengcheng Xia, Haoyu Wang, Bowen Zhang, Ru Ji, Bingyu Gao, Lei Wu, Xiapu Luo, and Guoai Xu. Characterizing cryptocurrency exchange scams. *Computers & Security*, 98:101993, 2020.

[50] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. Sok:

Communication across distributed ledgers. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 3–36. Springer, 2021.

[51] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE symposium on security and privacy (SP)*, pages 193–210. IEEE, 2019.

A Formalized notations

For the sake of demonstration, Table 4 provides commonly used notations.

Table 4: Formalized notations involved in ParaSwap

Notation	Definition
n	number of participants in an exchange
i, j, x	variant $i, j \in [1,n]$ and $x \in [1,n-1]$
v_i	participant v_i, where $v_0 = v_n$, $v_1 = v_{n+1}$
(v_i, v_{i+1})	arc from v_i to v_{i+1}
a_i	assets that v_i uses to swap and v_{i+1} wants
$\mathbb{C}_i$	ledger $\mathbb{C}_i$ or chain $\mathbb{C}_i$
Δ	time for one on-chain operation
ε	time for off-chain operations per phase
Y_i, y_i	statement and witness of v_i, $(Y_i, y_i) \in R$
K_i, k_i	document and identifier of v_i, $(K_i, k_i) \in R$
$pk^j_{i,i+1}$	j-th joint address managed by v_i and v_{i+1}
$sk^j_{i,i+1}$	corresponding secret key of $pk^j_{i,i+1}$
$sk^{j,i}_{i,i+1}, sk^{j,i+1}_{i,i+1}$	shares of $sk^j_{i,i+1}$ owned by v_i and v_{i+1}, respectively
u^x_{i+1}, U^x_{i+1}	x-th delayed value of v_{i+1} and its public value, where $(U^x_{i+1}, u^x_{i+1}) \in R$
tx^i_{lck}, tx^i_{rfnd}	lock and refund transaction for a_i of v_i
$tx^{j,i+1}_{wtdw}, \hat{\sigma}^{j,i+1}_{wtdw}, \sigma^{j,i+1}_{wtdw}$	j-th withdraw transaction, pre-signature and signature for a_i of v_{i+1}
$tx^{x,i}_{rlck}, \hat{\sigma}^{x,i}_{rlck}, \sigma^{x,i}_{rlck}$	x-th re-lock transaction, pre-signature and signature for a_i of v_i

B Security Analysis

Universal Composability. We model our security analysis in the universal composability (UC) framework introduced by [22], which is extended to include a global setup as described in [23]. In our work, we focus on *static corruptions*, where the adversary declares the participants they corrupt at the beginning. For our ideal functionality $\mathcal{F}_{\text{PS}}$ we rely on the clock functionality $\mathcal{F}_{\texttt{clock}}$, the secure message transmission functionality $\mathcal{F}_{\texttt{smt}}$, and the ledger functionality $\mathcal{F}_{\mathcal{L}}$, as discussed in Section 2.2. More precisely, Let Π be a protocol with access to $\mathcal{F}_{\texttt{clock}}$, $\mathcal{F}_{\texttt{smt}}$, and $\mathcal{F}_{\mathcal{L}}$. We use $\mathcal{E}$ to represent the environment. We use the notation $\text{EXEC}_{\Pi,\mathcal{A},\mathcal{E}}$ to represent the executions ensemble of the environment $\mathcal{E}$ when interacting with the adversary $\mathcal{A}$.

Definition 2. *(Universal Composability). A protocol Π UC-realizes an ideal functionality $\mathcal{F}$ w.r.t a clock functionality $\mathcal{F}_{\texttt{clock}}$, a secure message transmission functionality $\mathcal{F}_{\texttt{smt}}$, and a ledger functionality $\mathcal{F}_{\mathcal{L}}$, if for any PPT adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$, such that for any environment $\mathcal{E}$, the ensembles $EXEC^{\mathcal{F}_{\texttt{clock}},\mathcal{F}_{\texttt{smt}},\mathcal{F}_{\mathcal{L}}}_{\Pi,\mathcal{A},\mathcal{E}}$ and $EXEC^{\mathcal{F}_{\texttt{clock}},\mathcal{F}_{\texttt{smt}},\mathcal{F}_{\mathcal{L}}}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

B.1 Ideal Functionality

We present the ideal functionality $\mathcal{F}_{\text{PS}}$ for ParaSwap in Figure 9, where it implicitly utilizes $\mathcal{F}_{\texttt{clock}}$, $\mathcal{F}_{\texttt{smt}}$, and $\mathcal{F}_{\mathcal{L}}$. The ideal functionality $\mathcal{F}_{\text{PS}}$ interacts with n participants $v_1, v_2, \ldots, v_n$, and the simulator $\mathcal{S}$. In round 1, participant v_{i+1} sends a message $(\texttt{name}, a_{i+1}, a_i, pk_{i+1}, sk_{i+1}, pk_i)$ to initiate the swap. This message specifies the assets a_{i+1} locked by v_{i+1} and the assets a_i that v_{i+1} wants, where $i \in [1,n]$.

In round 2, upon receiving n initiation message, $\mathcal{F}_{\text{PS}}$ invokes the Lock subroutine to transfer the assets to the first name-specific address under its control. This action is possible because $\mathcal{F}_{\text{PS}}$ has access to the first secret keys of each asset and interacts with $\mathcal{F}_{\mathcal{L}}$ to finalize the transfers. If any transaction fails, $\mathcal{F}_{\text{PS}}$ will return the assets that have been locked to their respective original owner.

In round 3, if any participant v_{i+1} aborts, $\mathcal{F}_{\text{PS}}$ unlocks all the assets and returns them to their respective original owners.

In round $r = 4 \ldots n+2$, for some $i \in [1,n]$, if $\mathcal{F}_{\text{PS}}$ receives a trade message from participant v_{i+1}, it invokes the Transfer subroutine to transfer the assets a_i from its control to v_{i+1}. Then $\mathcal{F}_{\text{PS}}$ invokes the ReLock subroutine to transfer any remaining unlocked assets to the $(r-2)$-th name-specific address remaining under its control.

In round $r = n+3$, if $\mathcal{F}_{\text{PS}}$ receives a trade message from participant v_{i+1} for some $i \in [1,n]$, it invokes the Transfer subroutine to transfer the assets a_i from its control to v_{i+1}.

Finally, in round $r = n+4$, any assets still under the functionality's control are unlocked and returned to their respective initial owners.

Atomicity. The ideal functionality guarantees that if participant v_{i+1} in round 3 aborts the swap, for any $i \in [1,n]$, all the assets are returned to the respective participants. However, if v_{i+1} proceeds the swap for his/her intended assets a_i in round r ($4 \leqslant r \leqslant n+3$), participant v_i is ensured to complete the swap before round $r+1$ for the assets a_{i-1}. Since no participant can unilaterally transfer the assets while preventing others from transfers (since locked with functionality), the functionality ensures atomicity for the swap.

B.2 Security analysis of ParaSwap

Proof. (Proof of Theorem 1): We prove ParaSwap in Figure 4 and 5 UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{PS}}$ in Figure 9.

We outline a simulator $\mathcal{S}$ that controls x participants, where $x \in [1, n-1]$, that are corrupted by a PPT $\mathcal{A}$. The simulator $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\mathsf{PS}}$ to simulate the real-world execution of ParaSwap. In the static corruption model, the environment $\mathcal{E}$ specifies the corrupted and honest participants at the start of a session. The simulator $\mathcal{S}$ impersonates the honest participants, while interactions among corrupted participants do not involve $\mathcal{S}$, as expected by the environment. Additionally, communications between honest participants occur over secure channels, so the adversary only knows that communication occurred, but not its content. We omit these operations to simplify the description of $\mathcal{S}$.

To describe the operations involved in simulating the multi-party atomic swap, we outline a sequence of hybrid executions. We initiate with a real-world execution, then incrementally alter the simulation across hybrids, and subsequently analyze the proximity between adjacent experiments. The final hybrid represents the execution of $\mathcal{S}$ for the swap operation. Notably, hybrid executions are switched every session, one at a time, and we focus on a single time here for simplicity.

Hybrid $\mathcal{H}_0$: This hybrid is the execution of ParaSwap.

Hybrid $\mathcal{H}_1$: This is identical to Hybrid $\mathcal{H}_0$ with the exception the TPC protocol Π_{JAMan} when locking assets for key shares generation is replaced with a simulation by a TPC simulator $\mathcal{S}_{\mathsf{TPC,A}}$ for corrupted participants. Π_{JAMan} guarantees the existence of such a simulator. The indistinguishability between Hybrid $\mathcal{H}_0$ and Hybrid $\mathcal{H}_1$ relies on the security of Π_{JAMan}.

Hybrid $\mathcal{H}_2$: This is identical to Hybrid $\mathcal{H}_1$ with the exception the TPC protocol Π_{JPSig} for pre-signature genaration is simulated using $\mathcal{S}_{\mathsf{TPC,S}}$ for corrupted participants. The indistinguishability between Hybrid $\mathcal{H}_1$ and Hybrid $\mathcal{H}_2$ relies on the security of Π_{JPSig}.

Hybrid $\mathcal{H}_3$: This is identical to Hybrid $\mathcal{H}_2$ with the exception that if, the adversary corrupts participant v_{i+1} and produces σ^* on a withdraw transaction $tx^{j,i+1}_{wtdw}$ under the public key $pk^j_{i,i+1}$ such that $\Sigma_{\mathsf{DS}}.\mathsf{Vrfy}(pk^j_{i,i+1}, tx^{j,i+1}_{wtdw}, \sigma^*) = 1$, before the simulator initiates the swap, the simulator aborts. The indistinguishability between Hybrid $\mathcal{H}_2$ and Hybrid $\mathcal{H}_3$ relies on the security of signature scheme Σ_{DS}.

Hybrid $\mathcal{H}_4$: This is identical to Hybrid $\mathcal{H}_3$ with the exception that if, the adversary corrupts participant v_{i+1} and produces σ^* on a withdraw transaction $tx^{j,i+1}_{wtdw}$ under the public key $pk^j_{i,i+1}$ such that $\Sigma_{\mathsf{DS}}.\mathsf{Vrfy}(pk^j_{i,i+1}, tx^{j,i+1}_{wtdw}, \sigma^*) = 1$, then the simulator v_i computes $w^j_{i+1} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Ext}(\sigma^*, \hat{\sigma}^{j,i+1}_{wtdw}, S^j_{i+1})$. If $(S^j_{i+1}, w^j_{i+1}) \notin R$, the simulator aborts. The indistinguishability between Hybrid $\mathcal{H}_3$ and Hybrid $\mathcal{H}_4$ relies on the witness extractability of the adaptor signature scheme Σ_{AS}.

Hybrid $\mathcal{H}_5$: This is identical to Hybrid $\mathcal{H}_4$ with the exception that if, the adversary corrupts participant v_{i+1} and produces σ^*

Round 1: Upon receiving $(\mathsf{name}, a_{i+1}, a_i, pk_{i+1}, sk_{i+1}, pk_i)$ from v_{i+1}, send the tuple $(\mathsf{name}, a_{i+1}, a_i, pk_{i+1}, sk_{i+1}, pk_i, v_{i+1})$ to $\mathcal{S}$ and record it, where $i \in [1,n]$.
Round 2: Upon receiving n input tuples, set $A := (a_1, a_2, \ldots, a_n)$, call the subroutine $\mathtt{Lock}(\mathsf{name}, a_{i+1}, pk_{i+1}, sk_{i+1})$ for each $i \in [1,n]$. If all of the innovations are successful, proceed to Round 3. Otherwise, cancel assets locking by calling the subroutines $\mathtt{UnLock}(\mathsf{name}, a_{i+1}, pk_{i+1})$ for all $i \in [1,n]$.
Round 3: Upon receiving $(\mathsf{abort}, \mathsf{name})$ from v_{i+1}, revert the assets locking by invoking the subroutines $\mathtt{UnLock}(\mathsf{name}, a_{i+1}, pk_{i+1})$ for all $i \in [1,n]$. Otherwise, proceed to round 4.
Round $r = 4 \ldots n+2$**:** Upon receiving $(\mathsf{trade}, \mathsf{name}, a_i, pk_{i+1})$ from v_{i+1}, invoke the subroutine $\mathtt{Transfer}(\mathsf{name}, a_i, pk_{i+1}, r-3)$, remove a_i from A, for some $i \in [1,n]$. Otherwise, invoke the $\mathtt{ReLock}(\mathsf{name}, a_i, r-2)$ subroutine if $a_i \in A$, and proceed to round $r+1$.
Round $n+3$**:** Upon receiving $(\mathsf{trade}, \mathsf{name}, a_i, pk_{i+1})$ from v_{i+1}, invoke the subroutine $\mathtt{Transfer}(\mathsf{name}, a_i, pk_{i+1}, n)$, remove a_i from A, for some $i \in [1,n]$, and proceed to round $n+4$.
Round $n+4$**:** Invoke the $\mathtt{UnLock_n}(\mathsf{name}, a_{i+1}, pk_{i+1})$ subroutine for $a_{i+1} \in A$.
Notify $\mathcal{S}$ about the outcome of the operation.

Subroutine Descriptions:
Lock: On input a tuple $(\mathsf{name}, a, pk, sk)$, transfers the assets a from the source address pk (using sk) to a functionality-controlled address pk^1_{name}, through $\mathtt{Publish}(\mathsf{name}, pk, pk^1_{\mathsf{name}}, a)$. The operation succeeds if $\mathcal{F}_{\mathcal{L}}$ accepts the transaction.
UnLock: On input a tuple (name, a, pk), returns the locked assets a from pk^1_{name} back to the original address pk, using $\mathtt{Publish}(\mathsf{name}, pk^1_{\mathsf{name}}, pk, a)$.
Transfer: On input a tuple $(\mathsf{name}, a, pk, r)$, transfers the locked assets a from pk^r_{name} to the target address pk, using $\mathtt{Publish}(\mathsf{name}, pk^r_{\mathsf{name}}, pk, a)$.
ReLock: On input a tuple (name, a, r), transfers the locked assets a from pk^{r-1}_{name} to the target address pk^r_{name}, using $\mathtt{Publish}(\mathsf{name}, pk^{r-1}_{\mathsf{name}}, pk^r_{\mathsf{name}}, a)$.
UnLock$_\mathbf{n}$: On input a tuple (name, a, pk), returns the locked assets a from pk^n_{name} back to the original address pk, using $\mathtt{Publish}(\mathsf{name}, pk^n_{\mathsf{name}}, pk, a)$.

Figure 9: Ideal functionality $\mathcal{F}_{\mathsf{PS}}$ for atomic swap of assets. The functionality interacts with participants $v_1, v_2, \ldots, v_n$, and the simulator $\mathcal{S}$.

on a withdraw transaction $tx^{x,i+1}_{wtdw}$ under the public key $pk^x_{i,i+1}$ such that $\Sigma_{\mathsf{DS}}.\mathsf{Vrfy}(pk^x_{i,i+1}, tx^{x,i+1}_{wtdw}, \sigma^*) = 1$. The simulator v_i computes

- $w^x_{i+1} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Ext}(\sigma^*, \hat{\sigma}^{x,i+1}_{wtdw}, S^x_{i+1})$ and $(S^x_{i+1}, w^x_{i+1}) \in R$.
- $\sigma^{x+1,i}_{wtdw} \leftarrow \Sigma_{\mathsf{AS}}.\mathsf{Adapt}(\hat{\sigma}^{x+1,i}_{wtdw}, w^x_{i+1} \oplus k_i)$.

If $\Sigma_{\mathsf{DS}}.\mathsf{Vrfy}(pk^{x+1}_{i-1,i}, tx^{x+1,i}_{wtdw}, \sigma^{x+1,i}_{wtdw}) = 0$, the simulator aborts. The indistinguishability between Hybrid $\mathcal{H}_4$ and Hybrid $\mathcal{H}_5$ is based on the pre-signature adaptability of Σ_{AS}.

Hybrid $\mathcal{H}_6$: This is identical to Hybrid $\mathcal{H}_5$ with the exception that if, the adversary corrupts participant v_i and produces σ^* on a re-lock transaction $tx^{x,i}_{rlck}$ under the public key $pk^x_{i,i+1}$ such that $\Sigma_{\mathsf{DS}}.\mathsf{Vrfy}(pk^x_{i,i+1}, tx^{x,i}_{rlck}, \sigma^*) = 1$ before time $x\Delta + t$ for $x \in [1, n-1]$, the simulator aborts. The indistinguishability between Hybrid $\mathcal{H}_5$ and Hybrid $\mathcal{H}_6$ is based on the security of Σ_{DS} and the soundness of the VTD scheme Σ_{VTD}.

Hybrid $\mathcal{H}_7$: This is identical to $\mathcal{H}_6$ with the exception that if, the adversary corrupts v_i and produces any transaction tx^* and the signature σ^* such that $\Sigma_{\mathsf{DS}}.\mathsf{Vrfy}(pk^n_{i,i+1}, tx^*, \sigma^*) = 1$ before time T, the simulator aborts. The indistinguishability between Hybrid $\mathcal{H}_6$ and Hybrid $\mathcal{H}_7$ relies on the security of Σ_{DS} and the soundness of Σ_{VTD}.

Hybrid $\mathcal{H}_8$: This is identical to Hybrid $\mathcal{H}_7$ with the exception that if, the adversary corrupts participant v_i, and the simulator learns $u^{x'}_{i+1} \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{ForceOp}(C^x_{i+1})$ for $x \in [1, n-1]$. The simulator aborts, if $u^{x'}_{i+1} \neq u^x_{i+1}$, where u^x_{i+1} is the correct delayed value of v_{i+1} generated in the preparation phase. The indistinguishability between Hybrid $\mathcal{H}_7$ and Hybrid $\mathcal{H}_8$ relies on the soundness of Σ_{VTD}.

Hybrid $\mathcal{H}_9$: This is identical to Hybrid $\mathcal{H}_8$ with the exception that if, the adversary corrupts participant v_i, and the simulator learns $sk' \leftarrow \Sigma_{\mathsf{VTD}}.\mathsf{ForceOp}(C^n_{i+1})$. The simulator aborts, if $sk' \neq sk^n_{i,i+1}$, where $sk^n_{i,i+1}$ is the last secret key share of v_{i+1} generated in the preparation phase. The indistinguishability between Hybrid $\mathcal{H}_8$ and Hybrid $\mathcal{H}_9$ relies on the soundness of the VTD scheme Σ_{VTD}.

Simulator $\mathcal{S}$: The execution of $\mathcal{S}$ is defined as the process in Hybrid $\mathcal{H}_9$ while interacting with $\mathcal{F}_{\mathsf{PS}}$. Upon receiving all messages $(\mathsf{name}, a_{i+1}, a_i, pk_{i+1}, sk_{i+1}, pk_i, v_{i+1})$ from $\mathcal{F}_{\mathsf{PS}}$ for all $i \in [1, n]$, $\mathcal{S}$ simulates the adversary's view by processing messages from $\mathcal{F}_{\mathsf{PS}}$. If the simulated view differs from $\mathcal{F}_{\mathsf{PS}}$'s ideal execution, the simulation has encountered an abortion, as discussed in the above hybrids. This concludes the proof. □

C Ethics Considerations on MEV

Malicious miners may manipulate transaction ordering to engage in illicit activities, since blockchains cannot enforce transaction ordering protections [24], [25]. For example, if Alice can execute her transactions after observing Bob's transactions, she can front-run him and profit from it. As introduced in [25], the term miner/maximal extractable value (MEV) refers to the profits that powerful entities (with miners manipulating transaction ordering) can extract by selectively including, excluding, or re-ordering transactions within a new block. MEV extraction is an inherent challenge in blockchain systems and has been widely acknowledged. Existing atomic swap schemes [29, 30, 42, 44], including ours, have risks associated with MEV extraction. Similar to the existing schemes, our work does not solve the MEV extraction issue, but also doesn't make it worse.

MEV extraction in our work. In our work, the MEV extraction mechanism can be exploited to illegitimately extract assets from participants. However, it can succeed only if both of the following conditions are met: (i) the malicious entity (with miners manipulating transaction ordering) must be one of the participants in `ParaSwap`, and (ii) the miners must be selected as the leader (i.e., block proposer) at the refund time. For simplicity, we assume that v_i is the malicious participant, and the other participants are honest. v_i attempts to exchange his/her assets a_i, which is intended for v_{i+1}, in return for the assets a_{i-1} locked by v_{i-1}. In the pre-swap phase of `ParaSwap`, the two participants on each arc jointly generate pre-signatures for the withdraw, re-lock, and refund operations. These pre-signatures strictly define the permitted flow of assets. For example, the assets a_{i-1} can only end up flowing either to v_i (via withdrawal) or v_{i-1} (via refund). The same rule applies to all other asset flows. As a result, the only assets that can flow to v_i are a_i and a_{i-1}. Therefore, the only feasible attack strategy for v_i is to withdraw a_{i-1} while refunding a_i, i.e., preventing v_{i+1} from withdrawing a_i. This results in an imbalanced exchange and enables v_i to obtain both a_{i-1} and a_i, thereby extracting illegitimate value a_i at the expense of v_{i+1}. Specifically, when v_{i+1} submits the corresponding (transaction, signature) pair to the chain to withdraw a_i, the malicious miner can exploit MEV extraction by manipulating transaction ordering, e.g., placing v_i's refund transaction before v_{i+1}'s withdraw transaction. We emphasize that prior atomic swap schemes, such as [29, 30, 42, 44], can be exploited through MEV extraction with the same re-ordering. However, such re-ordering must occur after the pre-defined refund time, because the refund transaction signature can only be generated after this timeout according to the VTD technology.

Danger for non-experienced users. We caution that non-experienced users may face the risk of losing their locked assets due to MEV extraction. Specifically, a malicious participant with miners manipulating transaction ordering could compromise the balance security of honest participants, i.e., if others withdraw an honest participant's locked assets, the participant cannot withdraw the intended assets. We notice that several countermeasures have been developed to mitigate MEV extraction, such as Flashbots [8], SUAVE [9], MEV-Boost [12], and Skip Protocol [14]. We recommend that users perform atomic swaps on platforms equipped with MEV prevention mechanisms.