4-Swap: Achieving Grief-Free and Bribery-Safe Atomic Swaps Using Four Transactions

Kirti Singh
Indian Institute of Technology Bombay, India
Institute for Development and Research in Banking Technology, Hyderabad, India

Vinay J. Ribeiro
Indian Institute of Technology Bombay, India

Susmita Mandal
Institute for Development and Research in Banking Technology, Hyderabad, India

Abstract

Cross-chain asset exchange is crucial for blockchain interoperability. Existing solutions rely on trusted third parties and risk asset loss, or use decentralized alternatives like atomic swaps, which suffer from grief attacks. Griefing occurs when a party prematurely exits, locking the counterparty's assets until a timelock expires. Hedged Atomic Swaps mitigate griefing by introducing a penalty premium; however, they increase the number of transactions from four (as in Tier Nolan's swap) to six, which in turn introduces new griefing risks. Grief-Free (GF) Swap reduces this to five transactions by consolidating assets and premiums on a single chain. However, no existing protocol achieves grief-free asset exchange in just four transactions.

This paper presents 4-Swap, the first cross-chain atomic swap protocol that is both grief-free and bribery-safe, while completing asset exchange in just four transactions. By combining the griefing premium and principal into a single transaction per chain, 4-Swap reduces on-chain transactions, leading to faster execution compared to previous grief-free solutions. It is fully compatible with Bitcoin and operates without the need for any new opcodes. A game-theoretic analysis shows that rational participants have no incentive to deviate from the protocol, ensuring robust compliance and security.

2012 ACM Subject Classification Security and privacy → Security protocols

Keywords and phrases Atomic Swaps, Griefing, Bribery, HTLC

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.32

Related Version *Extended Version*: https://arxiv.org/abs/2508.04641 [17]

1 Introduction

The trade of cross-chain assets has been a significant problem in blockchains. Centralized exchanges such as Binance, Coinbase, and CoinDCX solve this problem by taking ownership of users' assets and exchanging them among the interested parties. Using centralized exchanges makes the exchange efficient and easy. However, the involvement of a trusted third party leads to the risk of users losing their assets, as happened in the MTGox hack [6, 16] and FTX collapse [7]. Thus, there is a need for trustless exchange protocols. Some cross-chain decentralized exchange solutions exist, such as two-way pegged tokens [3] and using a third chain [22]. However, they face some issues: pegged tokens do not give native tokens, and using a third chain increases the overall overhead for the exchange.

Atomic swaps solve the issues by providing native tokens without needing a third coordinating chain. Tier Nolan introduced it in a Bitcoin forum in 2013 [15, 14]. It typically involves four transactions, the locking of tokens by the two parties, which later claim each others' tokens before a timelock expires. Atomic swaps primarily use Hashed Time-Locked Contracts (HTLCs), which provide a mechanism to lock assets on the blockchain with a

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin; Article No. 32; pp. 32:1–32:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

hashlock and to refund the locked assets after the timelock expires. However, they can also leverage multi-signature transactions [8, 10], adaptor signatures [10, 18], and verifiable timed signatures [18] for implementation. Although Atomic Swap provides a decentralized exchange solution, it suffers from the problem of grief.

Griefing is a common problem in Atomic Swaps wherein a party who locks the tokens waits for the timelock to expire when the counterparty abandons the exchange. To deter this premiums are used as collateral. In 2021, Xue and Herlihy [21] used the concept of cross-locking of griefing premium in the two chains to solve griefing. However, this led to griefing on the premium, and the number of transactions in the swap increased to six.

In 2022, Nadahalli *et al.* proposed combining the locking of griefing premium with the assets in one transaction [13]. Thus, the problem of griefing on the premium is solved. However, they failed to combine the premium on both chains, requiring five transactions for a grief-free atomic swap. Mazumdar also solves the griefing problem by allowing the participating parties to leave the exchange whenever they like, increasing the total number of transactions to eight [11]. This leads to our question: is achieving a grief-free atomic swap in only four transactions possible?

We present the 4-Swap (or 4S, for short) protocol in two versions designed to simplify the exposition of this complex protocol. By breaking it down into incremental versions, we aim to make its functionality and security properties more accessible. We begin with 4S-v1, which operates under the simplifying assumption of zero delay in transactions being added to the blockchain. In this idealized setting, a published transaction is immediately included without waiting in the mempool, effectively eliminating the risks of bribery or censorship attacks. This version allows us to focus on the core mechanics of the protocol, which achieves grief-free execution in just four steps. It does so by leveraging cross-locking of principals and premiums, as seen in prior works [21, 13]. We introduce two novel concepts of cross-publishing of locking transactions and a new timelock in the initiating chain for early refunds. These concepts help combine the principal and griefing premiums for both parties in both chains, reducing the total number of transactions to just four.

Building on this foundation, we present the next version that removes the zero-delay assumption, thus accounting for real-world conditions where transaction delays and susceptibility to bribery attacks are practical concerns. To address this, we integrate the concept of mutual destruction from [19] coupled in both the chains, ensuring the protocol remains bribery-safe. The final version retains its simplicity and efficiency, maintaining grief-free and bribery safety within a four-step process. We do not claim the protocol to be bribery free as it does not considers the reverse bribery attacks where the miners incentivizes the involved parties to publish the conflicting transactions. The protocol's game-theoretical analysis demonstrates that it consistently follows its intended execution path under the assumption of rational behaviour from all parties. We perform a security analysis of the protocol using the UC Framework [4], and present possible redeem paths in the extended version [17], which also includes Bitcoin scripts showing that the protocol requires no new opcodes.

Our Contributions

1. We propose 4-Swap, the first grief-free and bribery-safe atomic swap protocol that uses only four transactions, matching the number of transactions in the TN-Swap.
2. We then present a game-theoretical analysis of our protocol to show that participants have no incentive to deviate from the protocol under rational behaviour.
3. We perform a security analysis of the protocol using the UC framework [4] in the extended version of this paper [17].

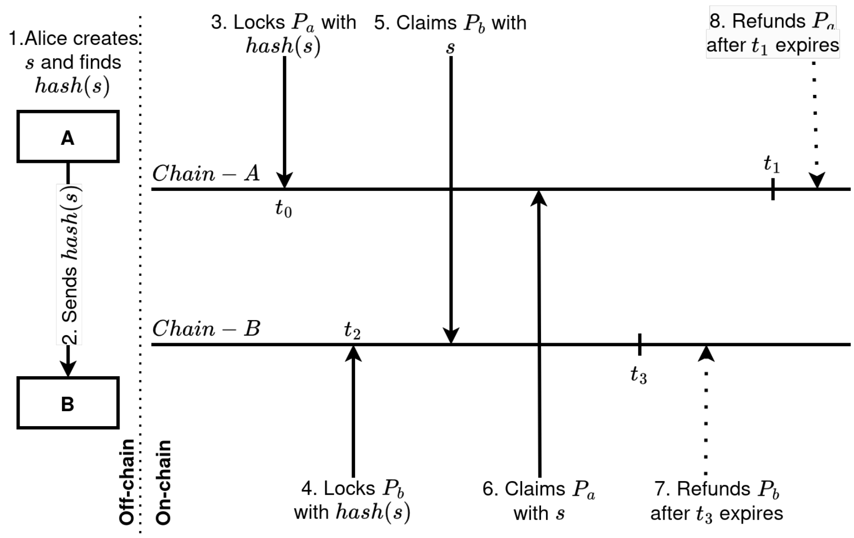

Figure 1 Atomic Swap Protocol.

2 Background

2.1 Atomic Swaps – TN Swap

Atomic swap is a decentralized method of performing cross-chain exchanges. Tier Nolan first introduced it in 2013 [15]. It typically uses two HTLCs with the same hash to lock the tokens. The parties on the successful locking claim each other's tokens before the set timeout expires in their respective chains [9].

Consider A and B, two parties interested in exchanging tokens P_a on $Chain$-A and P_b on $Chain$-B, respectively. They must follow the following steps for a successful atomic swap exchange, as shown in Figure 1.

1. A generates secret s, finds $\mathcal{H}(s)$ and sends $\mathcal{H}(s)$ to B.
2. A locks her tokens on $Chain$-A with $\mathcal{H}(s)$ setting up a refund timelock of t_1.
3. B locks his tokens on $Chain$-B with same $\mathcal{H}(s)$ setting a refund timelock of t_3.
4. A claims B's locked tokens on $Chain$-B with s.
5. B uses the revealed preimage s to claim A's tokens.
6. If A does not claim B's tokens, B can refund after t_3.
7. If B does not claim A's tokens, A can refund after t_1.

The protocol ensures that no party loses their tokens in the exchange. Moreover, the gap in two timelocks ensures that B gets enough time to claim A's token when A has already claimed B's token. Although atomic swap solves the problem of decentralized cross-chain asset exchange, it gives rise to the problem of griefing.

2.1.1 Griefing Attack

Griefing occurs when a party locks the tokens, and the counterparty exits the exchange, forcing the counterparty to wait until the refund timelock expires to retrieve their tokens.

In Figure 1, there are two possible scenarios of griefing.

- A locks tokens, but B leaves the exchange. A waits until t_1 to refund the locked tokens.
- A and B lock their tokens, but A leaves without claiming B's tokens. Since A only knows the preimage s, B waits until the refund timelock t_3 expires to get back his tokens.

2.1.2 Bribery Attacks

Malicious parties bribe the miners to censor some transactions in the blockchain to gain an unfair advantage. HTLCs suffer from bribery attacks [12]. Suppose A pays B using HTLC and bribes the miners to censor B's claim transaction until the refund timelock expires. After the timelock expires, A can refund the locked tokens, and B fails to get the tokens. Since atomic swaps also use HTLCs, they are prone to bribery attacks. Bribery usually happens by either a party putting up a conflicting transaction(like the refund in HTLC) with a very high transaction fee or putting the refund transaction along with a future transaction that takes input from the refund transaction and can be redeemed by anyone(particularly miners).

In Figure 1, there are two possible scenarios of bribery attacks.

- A and B lock their tokens, but B bribes the miners to censor A's claim transaction until t_3. Since B can see A's claim transaction, he fetches s and claims A's tokens P_a. After t_3, he refunds his tokens P_b and A gets nothing.
- A claims B 's tokens after both successfully locked them. She then bribes the miners to censor B 's claim transaction until t_1. After t_1, A refunds her locked tokens P_a, and B gets nothing.

2.1.3 Bribery Attacks Solutions for HTLCs

In 2021, Tsabary *et al.* [19] proposed MAD-HTLC to remove bribery in HTLC. They modified the HTLC by adding a premium to be locked by the sender in addition to the principal value. The idea was to allow the miners to slash all the locked tokens, disincentivizing the parties to attempt bribery.

Suppose there are two parties where A is paying B using HTLC. The solution works as follows:

- A locks X (principal) and Y (premium) with $H(pre_a)$. Y can be refunded after timelock.
- To refund X, A must reveal pre_b after the timelock.
- B claims X before the timelock by revealing pre_a.
- If A attempts to censor B's claim by bribing miners with a high-fee refund transaction, miners can extract pre_b from A's refund transaction. Anyone with pre_a, pre_b can claim $X + Y$, so miners take the funds. Bribery fails; A gains nothing.

The sender can refund the premium after the timelock expires. Also, to refund the principal tokens locked by the hash of pre_a, the sender has to release a new preimage pre_b after the timelock expires. By bribing miners, the sender can still try to censor the receiver's claimed transaction, which reveals pre_a. However, if the sender tries to bribe the miners by putting in a refund transaction with very high transaction fees, the miners can fetch the pre_b from it. Since MAD-HTLC allows anyone with pre_a and pre_b to claim the tokens and the premium, miners can create their transaction and try to create a block with it. Thus, the sender does not get any benefit from bribing.

2.1.4 Reverse Bribery Attacks

MAD-HTLC does not address scenarios where miners incentivize protocol participants to behave maliciously, a situation known as a reverse bribery attack. This attack was introduced by Wadhwa *et al.* [20] in 2022, along with the proposal of He-HTLC, a protocol designed to mitigate such threats. Unlike MAD-HTLC, He-HTLC introduces a delay in the sender's refund process and allows multiple miners to collectively punish a malicious sender.

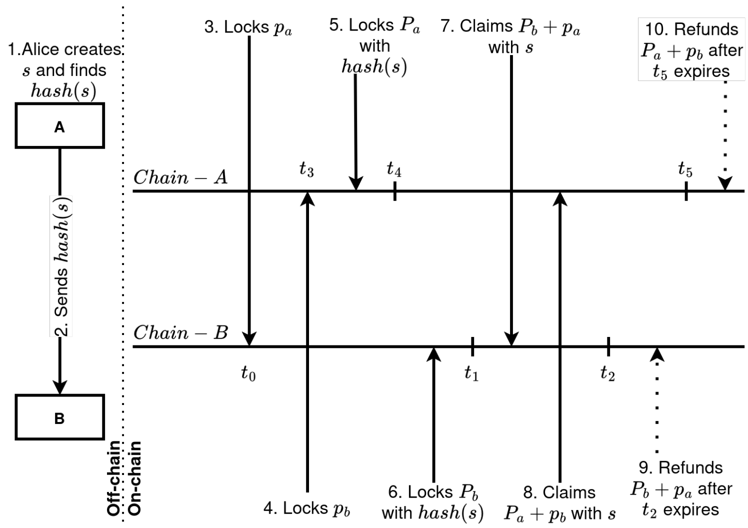

Figure 2 Hedged Atomic Swap Protocol.

Additionally, when a miner redeems funds using pre_a and pre_b, they receive only a portion of the locked tokens; the remainder is burned. This partial reward mechanism discourages collusion between miners and protocol participants.

However, He-HTLC does not offer a construction for atomic swaps within its framework. Rapidash [5], also published in 2022, builds on similar ideas. It introduces a two-step refund mechanism and extends the protocol to support atomic swaps. While this approach solves the reverse bribery attacks, it introduces increased on-chain transaction overhead and fails to be grief-free as one party can abandon the protocol after the counterparty has already locked their assets.

3 Related Works

We now discuss the related work of minimizing the number of on-chain transactions of atomic swaps while keeping it grief-free. Both works assume that parties have some tokens in the chain they want to receive tokens to safeguard from griefing attacks.

3.1 Hedged Atomic Swap

Xue and Herlihy [21] in 2021 proposed a Hedged Atomic Swap protocol that tried to solve the problem of griefing attacks using griefing premiums. The parties have to cross-lock the premium amounts with the principal amounts of the counterparty. The premiums serve as an amount to compensate the parties from griefing. Thus, it introduces the assumption that parties need a small amount of tokens in the other chain where they want to exchange them. The cross-locking helps get the refund premium and claim principal in one step. However, since the protocol requires premium locking, the number of transactions rose to six, two more than the TN-Swap.

Consider A and B, two parties interested in exchanging tokens P_a on *Chain-A* and P_b on *Chain-B*, respectively. A must have p_a on *Chain-B* and B must have p_b on *Chain-A* as griefing premium. They must follow the following steps for a successful atomic swap exchange, as shown in Figure 2.

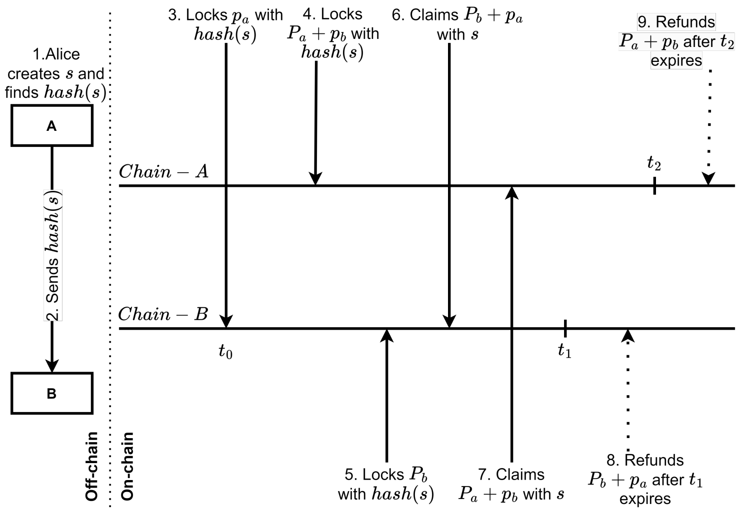

Figure 3 Grief-Free Atomic Swap Protocol.

1. A generates a secret s, finds $\mathcal{H}(s)$.
2. A sends $\mathcal{H}(s)$ to B.
3. A locks her premium p_a on *Chain-B* with $\mathcal{H}(s)$ setting up a timeout of t_1 and refund timelock of t_2 on *Chain-B*. B should lock his principal before t_1 on *Chain-B*.
4. B locks his premium p_b on *Chain-A* with $\mathcal{H}(s)$ setting up a timeout of t_4 and a refund timelock of t_5 on *Chain-A*. t_4 is the timeout for A to lock her principal.
5. A locks her tokens on *Chain-A* with $\mathcal{H}(s)$ before t_4. This locks $P_a + p_b$ until timelock t_5.
6. B locks his tokens on *Chain-B* with $\mathcal{H}(s)$. This locks $P_b + p_a$ until timelock t_2.
7. A claims B's principal P_b and her premium p_a on *Chain-B* with s.
8. B uses the revealed preimage s to claim A's principal P_a and his premium p_b.
9. If A does not claim B's tokens, B can refund $P_b + p_a$ after t_2 expires.
10. If B does not claim A's tokens, A can refund $P_a + p_b$ after t_5 expires.

Hedged Atomic Swap handles the two griefing cases mentioned in Section 2.1.1. For the first case where A locks P_a before t_4 and B leaves the exchange, A can refund $P_a + p_b$ after t_5 because B cannot refund his premium after A locks his principal. Since B has not locked his principal, A can refund her premium p_a after t_1. Thus, B loses p_b to A, disincentivizing him from leaving the exchange. For the second case, where A refuses to claim after A and B lock their principal, A and B can be refunded after t_5 and t_2, respectively. Thus, A gets $P_a + p_b$ and B gets $P_b + p_a$ by refunding. To ensure that A is disincentivized from leaving the exchange for this case, p_a is kept larger than p_b. Thus, A loses $p_a - p_b$ by leaving the exchange. However, this introduces the problem of griefing on premiums as the B can leave the swap after A locks her premium, and similarly, A can leave the swap after B locks his premium. This leads to the waiting until their respective refund timelock expires.

3.2 Grief-Free Atomic Swap

In 2022, Nadahalli *et al.* [13] proposed Grief-free atomic swaps that guarantee no griefing while reducing the on-chain transactions to five. The idea is to combine the locking of the premium with the principal. However, it could not match the number of transactions in the TN-Swap as the combination was done only in one chain.

Again, consider the same setup as in the Section 3.1. The following steps help achieve a grief-free atomic swap in five steps, as shown in Figure 3.

1. A generates secret s, finds $\mathcal{H}(s)$.
2. A sends $\mathcal{H}(s)$ to B.
3. A locks her premium p_a with $\mathcal{H}(s)$ setting up a refund timelock of t_1 on *Chain-B*.
4. A and B create a locking transaction $P_a + p_b$ and A publishes it on *Chain-A*. This also sets up refund timelock t_2. If B leaves before creating this, A can refund p_a by using s.
5. B locks his tokens with same $\mathcal{H}(s)$. This locks $P_b + p_a$ on *Chain-B* until timelock t_1.
6. A claims B's principal P_b and her premium p_a on *Chain-B* with s.
7. B uses the revealed preimage s to claim A's principal P_a and his premium p_b.
8. If A does not claim B's tokens, B can refund $P_b + p_a$ after t_1 expires.
9. If B does not claim A's tokens, A can refund $P_a + p_b$ after t_2 expires.

The GF-Swap handles the griefing case similarly to the XH-Swap; the difference is that it combines the premium with the principal and solves the griefing problem on premiums. Table 1 compares various atomic swap protocols along with 4-Swap. We now present the model and our assumptions for the 4-Swap protocol.

Table 1 Comparison of Atomic Swap Protocols. The column *Txns* indicates the total number of on-chain transactions required by the protocol in the worst-case scenario.

Protocol	Griefing Resistance	Bribery Safety	Rev. Bribery Safety	Txns
Tier-Nolan [15]	No	No	No	4
Hedged [21]	Yes	No	No	6
Grief-Free [13]	Yes	No	No	5
Rapidash [5]	No	Yes	Yes	6
4-Swap	Yes	Yes	No	4

4 System Model

We assume two UTXO-based blockchains that allow multi-signature transactions. All the interacting parties, including miners, are assumed to be rational, and they try their best to increase their incentives. The transactions in both blockchains first go into a pool of unconfirmed transactions known as the mempool, from which miners then take some transactions to create a block published in the blockchain. Transactions that utilize the same UTXO (conflicting transactions) can also be published in the mempool, from which only one eventually gets into the blockchain, and the rest become invalid. We also assume synchronous communication and the presence of a global clock. A transaction may be active or inactive based on whether it can be added to a block immediately or not. Thus, the transactions that can be added to a block after some timelock are called inactive transactions; otherwise, if they can be immediately added to a block, they are called active transactions. Both blockchains allow the publishing of both active and inactive transactions, which are stored in the mempool. Additionally, a collision-resistant hash function $\mathcal{H}$ converts inputs of any length to constant-length outputs. Also, all the parties can see the state of both the blockchains along with their mempools. The 4-Swap protocol assumes that each party must hold some tokens on the chain where they want to receive the tokens to safeguard against potential attacks, as done in all related work [21, 13]. Also, we assume that there is a significant difference in the principal and premium amount locked by the participating parties to accommodate fluctuations in the price of tokens. The premium amount is significantly higher than the transaction fees in both chains, and the value gained by exchanging the tokens is less than losing any of the premium. Moreover, the protocol does not consider the reverse bribery attacks i.e when the miners incentivizes the involved parties to publish conflicting transactions.

4.1 UTXO Model

We adopt the notation for UTXO-based blockchains from [2], and use charts to visualize the process and clarify the transaction flow as shown in Figure 4. Transactions are represented as rounded rectangles. Transactions already confirmed on the blockchain are shown with double borders, while those not yet confirmed have single borders. Each transaction (rounded rectangle) includes one or more output boxes (with squared corners) showing the coins assigned to each output. Arrows indicate the conditions for spending them.

Typical conditions are abbreviated for simplicity. Most outputs require signatures from one or more parties, with public keys shown under the arrow. Additional conditions like timelocks are shown above the arrow. For example, a relative timelock, requiring that t rounds have passed since the transaction was published, is represented as $+t$ above the arrow. Based on the number of rounds since the blockchain's creation, absolute timelocks are labelled $> t$.

If the output's spending condition includes multiple options (a disjunction), the condition is written as $\phi = \phi_1 \vee \cdots \vee \phi_n$, where $n \in \mathbb{N}$. The charts depict this using a diamond shape for the output, and each condition ϕ_i is displayed on a separate arrow. It also uses $\vee$ notation on conditions to represent disjunction. In the case of conjunction, it is represented in the chart as $\phi = \phi_1 \wedge \cdots \wedge \phi_n$. Also, any transaction of the form tx^y_x depicts action y happening to funds of x. It does not depict who is publishing the transaction. Certain outputs can be spent by anyone labelled as ACS in Figure 4 without the need for signatures. The grey square represents who can spend the output.

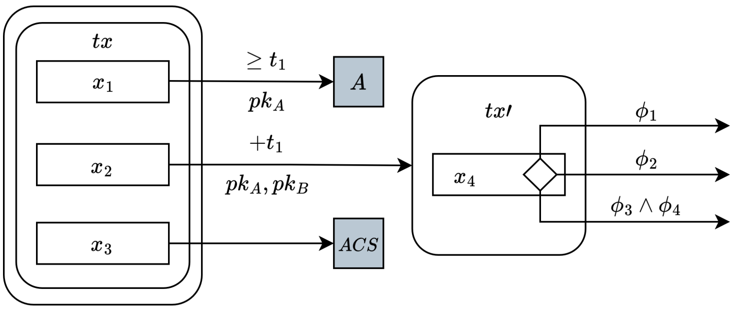

Figure 4 Transaction Schema Representation.

5 4-Swap Protocol

We present the development of the protocol through two versions, detailing the rationale behind each modification leading to the final 4-Swap protocol. The initial version (v1) focuses on facilitating grief-free atomic swaps that require only four on-chain transactions while making some assumptions. The next version then keeps the grief-free property and makes the protocol bribery safe within four steps. The notations used in the protocol are listed in Table 2 for two parties A and B on two chains *Chain-A* and *Chain-B*.

Table 2 Notation used in the protocol.

Notation	Description
$\mathcal{H}(s)$	Hash of some secret s
s_m	Main secret created by B
P_a	Principal amount A wants to exchange on $Chain\text{-}A$
p_a	Griefing premium of A on $Chain\text{-}B$
x_a	Bribery premium of A on $Chain\text{-}A$
P_b	Principal amount B wants to exchange on $Chain\text{-}B$
p_b	Griefing premium of B on $Chain\text{-}A$
x_b	Bribery premium of B on $Chain\text{-}B$
t_i	Timestamps in the chains
tx_x^y	Transaction that performs action y on tokens of x
s_{r1}	Refund secret created by A
s_{r2}	Refund secret created by B
s_{br}	Bribery secret created by A
s_e	Early execution secret created by A

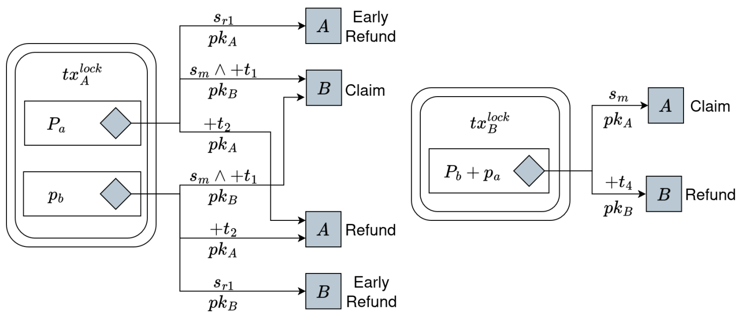

Figure 5 Transaction Schema for $4S\text{-}v1$.

5.1 4S-v1

In this initial version (v1), we relax the constraints typically associated with transaction processing by assuming that transactions are added to the blockchain immediately upon publication, with zero delay. This assumption is essential for considering conflicting transactions using the same UTXO. It ensures that the first active transaction put on the blockchain amongst many active or inactive conflicting transactions gets into it and can never be censored, eliminating the possibility of griefing. The transaction schema of $4S - v1$ is provided in Figure 5.

Atomic swaps use griefing premiums to prevent griefing attacks, but locking these premiums separately increases the number of on-chain transactions [13]. To reduce this, we combine the premium with the principal amount, limiting the total number of on-chain transactions to four. The combination is done crosswise: A's premium is locked with B's principal, and B's premium is locked with A's principal, ensuring the premium is returned during the claim transaction. The premium structure is designed such that the party claiming first has to pay a higher griefing premium, as it can cause grief to the counterparty by not claiming. In the first case of griefing, if a party quits after the counterparty has already locked, the counterparty can refund its principal and the quitter's premium, thereby penalizing the quitter. However, in the second case, if a party quits before claiming (after both have locked), both parties can refund each other's premiums after the refund timelock. Thus, the first claimer has to lock a higher premium to disincentivize it from quitting the exchange.

We introduce cross-publishing of locking transactions and an option for an early refund for the party whose tokens get locked first. Cross-publishing means only B can publish A's locking transaction, and only A can publish B's. Cross-publishing helps to let the party know that their locking transaction is put on the chain by the counterparty, and they should do something based on it if they have the counterparty's locking transaction. Suppose B publishes A's locking transaction; if A has B's locking transaction, he should publish it; otherwise, he should immediately do an early refund on the locking transaction B put on the chain. If A makes an early refund of the principal amount, B should also be able to make the refund on the premium locked in the locking transaction; thus, another secret s_{r1}, which, when revealed during the early refund of A's principal amount, helps B to early refund his premium. s_{r1} just indicates that A is trying to refund his tokens. To ensure that A gets enough time to make an early refund, we introduce a new timelock t_1 as shown in Figure 6, which helps to restrict B from claiming until the timelock expires. Cross-publishing also eliminates the first case of griefing, where a party leaves after the counterparty locks the tokens. If A leaves after B publishes A's locking transaction, B can claim A's principal after t_1. Also, suppose A publishes B's locking transaction first by moving away from the protocol. In that case, B can either put A's locking transaction or wait until refund timelock to get back the principal and A's griefing premium. Thus, the protocol is grief-free, as shown in Lemma 1. We define the 4-Swap v1 protocol in three phases of setup, initiation and redeeming as in [19].

The protocol 4S-v1 describes an atomic swap between participants A and B, with B initiating the process. The principal amounts are denoted as P_a for A and P_b for B, while p_a and p_b represent the respective griefing premium amounts. t_1 is the timelock for the early refund on *Chain-A* and t_2, t_4 are the refund timelocks for *Chain-A* and *Chain-B* respectively. The following are the steps:

1. **Setup Phase**
   a. B generates a secret s_m, finds $\mathcal{H}(s_m)$ and sends $\mathcal{H}(s_m)$, UTXO p_b to A.
   b. A generates a secret s_{r1}, finds $\mathcal{H}(s_{r1})$, creates multi-sig transaction tx_A^{lock} that locks $P_a + p_b$ with $\mathcal{H}(s_m)$ and sets up two refund timelocks t_1(early refund) and t_2. s_{r1} is required if A wants to refund the locked tokens early. A then sends $tx_{lock,A}$ (only signed by A), $\mathcal{H}(s_r1)$ and UTXO p_a to B.
   c. B creates a multi-sig transaction tx_B^{lock} that locks $P_b + p_a$ with $\mathcal{H}(s_m)$ and sets up refund timelock t_4. B then sends $tx_{lock,B}$ (only signed by B) to A. Nothing has been published in either of the chain until now.
2. **Initiation Phase**
   a. B publishes tx_A^{lock} to *Chain-A*, which is immediately included in a block.
   b. A then publishes tx_B^{lock} to *Chain-B* which is immediately included in a block.
3. **Redeeming Phase**
   a. If A creates and publishes a transaction for early refunding P_a using s_{r1}, B can also publish a transaction for early refunding p_b using the revealed s_{r1}.
   b. If A does not make an early refund, B waits for t_1 to expire and creates and publishes a transaction to claim P_a and p_b.
   c. If B publishes tx_A^{claim}, then A also creates and publishes tx_B^{claim} to claim P_b and p_a
   d. If B doesn't claim by t_2, A creates and publishes a refund transaction to get $P_a + p_b$.
   e. If A doesn't claim by t_4, B creates and publishes a refund transaction to get $P_b + p_a$.

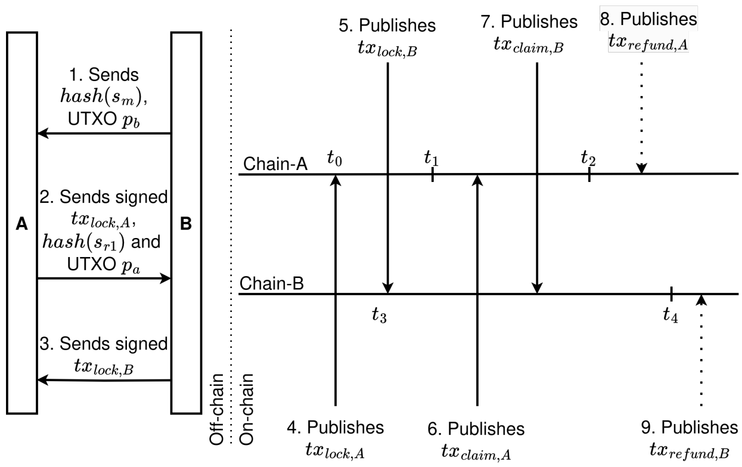

Figure 6 Transaction flow for $4S$-$v1$ and $4S$, with $4S$ differing slightly in off-chain interactions, as detailed in Section 5.2.

▶ **Lemma 1** (Griefing Lemma). *If any party abandons the protocol after tokens of the counterparty are locked on either of the two blockchains, the abandoning party incurs a penalty.*

Proof. Griefing can occur in two scenarios: first, if party B publishes party A's locking transaction, but A abandons the protocol, and second, if B abandons the protocol after both locking transactions are confirmed. In the first case, after the timelock t_1, B can submit a claim transaction because it knows the secret s, thereby retrieving the total amount $P_a + p_b$, where P_a and p_b are the tokens locked by A and B, respectively. Consequently, A incurs a loss of P_a. In the second case, if B abandons the protocol after both locking transactions are confirmed, A can refund its locked tokens $P_a + p_b$ after the timelock t_2, but A still incurs a penalty of p_a, the tokens it initially locked. To discourage B from abandoning the protocol, the value of p_b is set to be greater than p_a, ensuring that B faces a larger penalty in such scenarios. This mechanism ensures the protocol's resilience against griefing attacks. ◀

5.2 4-Swap Protocol Details

In this section, we remove the assumption of zero delay in adding transactions to the blockchain after publication. In practice, transactions first enter the mempool and are confirmed only when miners include them in a block. To address this delay and ensure protocol security, we incorporate the concept of mutual destruction from [19]. This approach disincentivizes participants from publishing conflicting transactions(bribery scenarios) in the mempool. However, we do not address reverse bribery attacks, as seen in HTLCs [20] because it would increase the total on-chain transactions.

Mutual destruction works by embedding secrets within conflicting transactions. If these secrets enter the mempool together, any participant (typically miners) can use them by creating a transaction claiming the entire locked amount, making both conflicting transactions invalid. This mechanism discourages parties from attempting bribery. To implement this in the 4S protocol, we introduce the following two secrets in addition to s_{r1} and the main secret s_m:

- s_{br}: A creates s_{br} for claiming B's tokens. It helps the protocol to know that A is claiming.
- s_{r2}: B creates s_{r2} for refunding its tokens. It helps the protocol to know that B is refunded.

Thus, whenever a pair of claim and refund transactions are published together in the blockchains, miner can create their slashing transaction to claim all the locked tokens. The transaction schema and all possible redeem paths for 4-Swap are shown in Figure 7 and Figure 8. This also covers the bribery case where an early refund transaction of A is censored by claim transaction of B (p_b works as bribery premium of B).

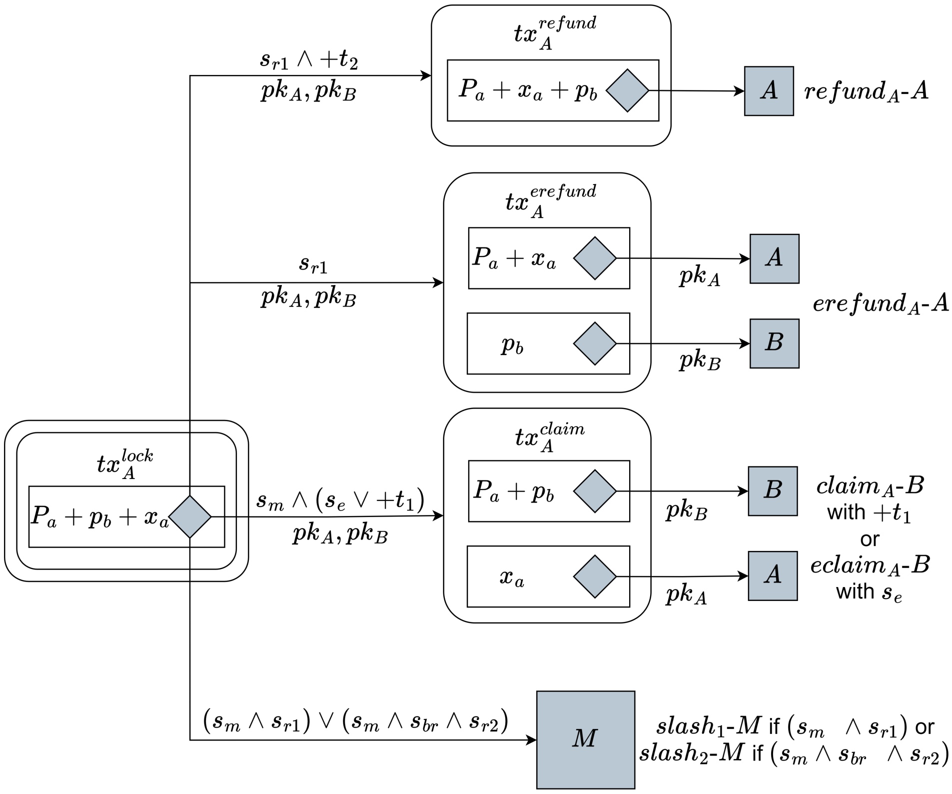

Figure 7 Transaction Schema of $4S$ for A.

Additionally, in the setup shown in Figure 6, even when both parties act honestly, B must wait for t_1 to expire before claiming the funds. We introduce another secret, s_e, generated by A for early execution. Once both locking transactions are confirmed on-chain, A can share s_e with B, allowing B to claim the funds without waiting for t_1 to expire.

However, griefing premiums alone are insufficient to address the bribery issue due to the following scenarios:

- After B successfully claims A's tokens in *Chain-A*, B may attempt to bribe miners by publishing conflicting refund transactions, as B no longer has anything at risk.
- B's claim transaction remains in the mempool, while A observes the corresponding secret from the mempool of *Chain-A*. If A's claim transaction is confirmed on *Chain-B* first, A may attempt to bribe miners by submitting a conflicting refund transaction since A similarly has nothing to lose.

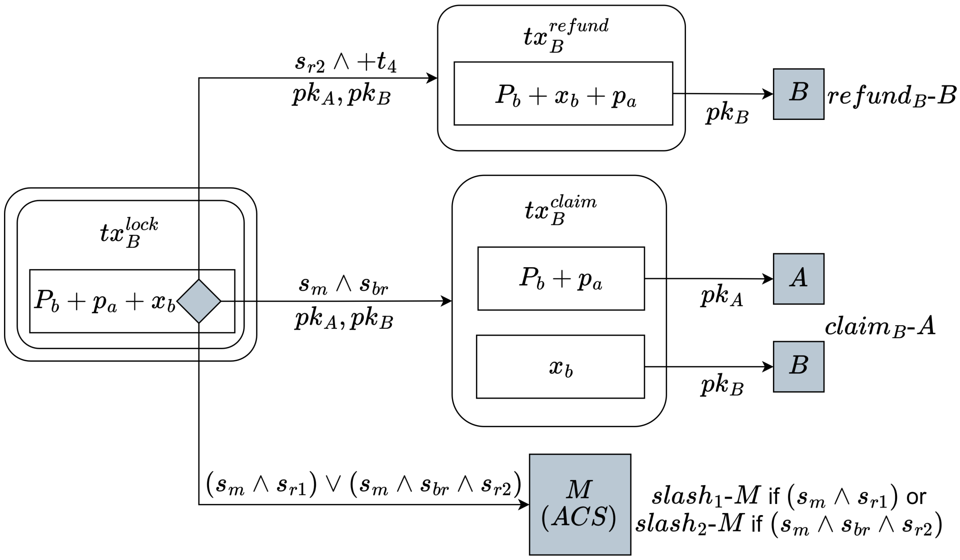

Figure 8 Transaction Schema of $4S$ for B.

The protocol introduces two bribery premiums to mitigate the abovementioned risks: x_a for party A and x_b for party B. These premiums are not cross-locked, meaning each party must contribute an additional amount alongside their principal to be a bribery premium. Moreover, we incorporate static claim transactions in the swap. In the setup phase, both parties jointly create their claim transactions, which cannot be altered during the protocol's runtime. These static claim transactions facilitate the exchange of principal amounts while returning the bribery premiums within the same transaction. This approach addresses the bribery risks in the abovementioned scenarios. If a party attempts bribery, they risk losing their bribery premium if the counterparty cannot claim it due to the bribery attempt. We also create static early refund and refund transactions for party A on *Chain-A* and static refunds for B on *Chain-B* to reduce the overhead of multiple redeem paths. $slash_1$ and $slash_2$ in Figure 7 and Figure 8 represent the redeem paths available to miners on individual locking transactions in case of bribery. The protocol flow is the same as Figure 6, but just in off-chain interactions, the claim, refund and early refund transactions are created and signed before signing the locking transactions. When a party(miner) uses s_m, s_{r1} to slash, it comes under $slash_1$, otherwise if it slashes by s_m, s_{br}, s_{r2} it comes under $slash_2$. We now present the $4S$ protocol in three phases.

Protocol $4S$ defines an atomic swap between participants A and B, initiated by B. Principal amounts are P_a for A and P_b for B, with griefing premiums p_a and p_b, and bribery premiums x_a and x_b, respectively. Figure 7 and Figure 8 shows the detailed construction of the two locking transactions and how the subsequent transactions will be created. The following are the required steps:

1. **Setup Phase**
   - **a.** B generates secrets s_m, s_{r2}, finds $\mathcal{H}(s_m), \mathcal{H}(s_{r2})$ and sends $\mathcal{H}(s_m)$, UTXO p_b to A.
   - **b.** A generates secrets s_{r1}, s_e, s_{br}, finds $\mathcal{H}(s_{r1})$, $\mathcal{H}(s_e)$, $\mathcal{H}(s_{br})$, creates multi-sig transaction tx_A^{lock} that locks $P_a + p_b$ with all hashes with different conditions and sets up two refund timelocks t_1(early refund) and t_2. s_{r1} is required if A wants to refund the locked tokens early. A also creates multi-signature transactions tx_A^{claim}, $tx_A^{erefund}$,

tx_A^{refund} using tx_A^{lock} and signs it. The tx_A^{claim}, $tx_A^{erefund}$ and tx_A^{refund} are completely signed before signing tx_A^{lock}. A then sends tx_A^{lock} (only signed by A), tx_A^{claim}, $\mathcal{H}(s_{r1})$, $\mathcal{H}(s_e)$, $\mathcal{H}(s_{br})$ and UTXO p_a to B.

c. B creates a multi-sig transaction tx_B^{lock} that locks $P_b + p_a$ with all hashes with different conditions and sets up refund timelock t_4. B also creates a multi-signature transaction tx_B^{claim} and tx_B^{refund} using tx_B^{lock} and signs it. B then sends the tx_B^{lock} and tx_B^{claim} to A. The tx_B^{claim} and tx_B^{refund} are completely signed before signing tx_B^{lock}. B then sends $tx_{lock,B}$ (signed by B) to A. Nothing has been published in either chain until now.

2. Initiation Phase

a. B publishes tx_A^{lock} to *Chain*-A.

b. A then publishes tx_B^{lock} to *Chain*-B.

3. Redeeming Phase

a. If A and publishes $tx_A^{erefund}$ for early refunding $P_a + x_a$ using s_{r1}, B also gets back p_b.

b. If $tx_A^{erefund}$ is not published, B waits for t_1 to expire and publishes tx_A^{claim} to claim P_a and p_b. This also refunds x_a to A.

c. B can publish without waiting if A releases the early execution secret s_e to B.

d. If B publishes tx_A^{claim}, then A also creates and publishes tx_B^{claim} to claim P_b and p_a. this also refunds x_b to B.

e. If B does not claim before t_2, A creates and publishes tx_A^{refund} to get P_a, p_b and x_a.

f. If A does not claim before t_4, B creates and publishes tx_B^{refund} to get P_b, p_a and x_b.

Reducing the on-chain transactions to four helps in reducing the overall time taken by the exchange in both the worst case as well as for the best case. For the best case, the parties follow the protocol and the early execution secret s_e is also shared. The protocol thus just requires four transactions, each followed by some confirmation delay. For the worst case, if party B leaves after both the locking transactions have been published, both parties have to wait until the refund timelocks expire in both the chains to refund their tokens. The refund timelocks entered will be smaller than the refund timelocks in the earlier works [21, 13] as it requires fewer on-chain transactions.

The security analysis of the protocol is detailed in the extended version of the paper [17]. We now present a game-theoretical analysis of the 4-Swap protocol to ensure that the protocol remains in the equilibrium state in the presence of rational parties and miners

6 Game Theoretical Analysis

The protocol is modelled as an Extensive Form Perfect Information Game, where all parties know the latest states of blockchains and their mempools [1]. The game consists of three phases: Setup, Claim, and Refund. The Setup phase covers the protocol's setup and locking steps, while the Claim and Refund phases focus on claiming and refunding locked funds.

The game involves three rational players: A, B, and the miner M. The miner M is initially assumed to be the strongest and only miner on both blockchains, and is responsible for mining all blocks. Since M controls the blockchain, it can always accept bribes to censor transactions. However, we demonstrate that the protocol effectively incentivizes honest behaviour, ensuring that bribery is not rational for any party.

The game progresses in rounds. In each round, A and B choose their actions, after which the miner M confirms transactions and advances the game to the next round. The complete game tree is constructed by combining the trees for the Base, Claim, and Refund trees in Figure 9, Figure 10 and Figure 11.

Initially, we analyze the protocol under the assumption of a single powerful miner, M. However, we assume this miner M does not intentionally delay the confirmation of transactions without bribery. We later show that this is true in the case of a multi-miner setup because of opportunity cost. We then generalize the model to a multi-miner setup across both chains. The analysis shows that the actions and utilities of A and B remain consistent even in the multi-miner setting, illustrating the protocol's robustness and ability to maintain incentives for honest participation under varying conditions.

▶ **Definition 2** (Perfect Information Extensive Form Game). *A game with perfect information in extensive form can be described as a tree structure and is formally defined by the tuple* (N, H, P, A_i, u_i) *where* $i \in N$*:*

- N*: A finite set of n players, labeled as* $N = \{1, 2, \ldots, n\}$*. Each decision point in the game that is not terminal is associated with a specific player* $I \in N$*, responsible for choosing that stage.*
- H*: The collection of all possible action sequences, called histories, where each sequence* h *leads to a specific node within the game tree. A subset* $Z \subseteq H$ *represents terminal histories, which correspond to the endpoints of every possible playthrough, also known as the tree's leaf nodes.*
- P*: A mapping called the player function, which determines the player assigned to decide each non-terminal history* $h \in H \setminus Z$*.*
- A_i*: A function that specifies the set of actions available to a player* i *at a given history* h*, denoted as* $A_i(h)$*. This set represents the choices available to player* i *at that point in the game. Each branch extending from a node in the tree corresponds to one of these actions.*
- u_i*: A function representing the payoffs for each player* i*, defined as* $u_i : Z \to \mathbb{R}$*. This function assigns a real number to each terminal history* $z \in Z$*, which reflects the utility that player* i *receives if the game ends at that particular terminal state.*

The parties A, B, M will resort to different actions at different game histories based on the utilities they will get from the actions. A strategy for a party is the sequence of actions it will take as the game proceeds based on the actions already taken by all the parties. The set of all such strategies of all the players is known as the strategy profile. We now formally present the definition of strategy profile.

▶ **Definition 3** (Strategy Profile). *In an extensive form game with perfect information, a strategy profile specifies the action* $a \in A_i(h)$ *chosen by each player* $i \in N$ *at every history* h *when required to act. Specifically, for each player* $i \in N$*, a strategy* s_i *is a function mapping the set of histories* $H_i = \{h \in H : P(h) = i\}$ *to the set of actions* A_i*. This mapping satisfies* $s_i(h) \in A_i(h)$ *for all* $h \in H_i$*. A strategy profile is a collection of strategies for all players,* $s = (s_1, s_2, \ldots, s_n)$*.*

Since the game is a perfect information game, all the parties know the best action for the other parties. Since parties are rational, they will try to increase their utility in each subgame of the main game. Subgame Perfect Nash Equilibrium gives the path of the game from which the parties will not deviate, as it gives them the best utility. We now define the Subgame-Perfect Nash Equilibrium.

▶ **Definition 4** (Subgame Perfect Nash Equlibrium). *A strategy profile* $s^* = (s_1^*, s_2^*, \ldots, s_n^*)$ *qualifies as a Subgame Perfect Nash Equilibrium (SPNE) if, in every subgame* G' *of the game* G*, each player's strategy* s_i^* *is the optimal response to the strategies of all other players.*

More specifically, let H' *represent the set of all histories within the subgame* G'*. For each player* i*, the strategy* s_i^* *is deemed optimal in* G' *if the following condition holds:* $u_i(s_i^*, s_{-i}^*; h) \geq u_i(s_i, s_{-i}^*; h)$*, for all strategies* s_i *available to player* i *in* G'*, and for every* *history $h \in H'$. Here, s^*_{-i} represents the strategies of all players other than i in the SPNE, and $u_i(s_i, s^*_{-i}; h)$ denotes player i's payoff when the strategy profile (s_i, s^*_{-i}) is followed in the subgame starting at history h. A strategy profile is an SPNE if it ensures a Nash Equilibrium in every subgame.*

We first define the slashing lemma and then prove the game-theoretic safety of 4-Swap.

▶ **Lemma 5** (Slashing Lemma). *Slashing is an action available to the miner M, which can take away all the locked tokens of the parties A and B. For any $h \in H_M$ of subgame G' of our game where M is miner, if $\exists\ a = slash \in A_M(h)$, then $u_M(h, slash) > u_M(h, !slash)$.*

Proof. The miner only gets the option to slash the locked tokens if there is a bribery scenario of censoring a current transaction by a conflicting transaction that becomes valid after a certain timelock. The miners will only censor the transaction if they get enough bribes from the bribing party to censor it. Assuming the lemma does not hold, at least one other action exists for the miner M, which gives it a better utility than slashing. Therefore, the bribe amount should be more than the tokens locked in the chains. Assuming the parties are rational, such a bribe will not be published. Thus, no action can provide a better utility than slashing, and the assumption that the lemma does not hold is false. Hence, rational miners will always choose to slash both parties' locked tokens. ◀

6.1 Game Trees

The game proceeds in rounds with parties putting all their available choices before a miner comes to confirm some transaction, and then the next round starts. The game is divided into three parts for better representation. The base tree is the starting point of the tree and starts with A sharing the tx^A_{lock} with B or not. This tree gives information about different scenarios of early refunds and reflects that the best strategy is not to deviate from the honest path. The claim tree assumes both parties have locked their tokens in their chains. The claim tree starts with B, choosing whether to publish the claim transaction or not. The refund tree focuses on parties wanting to return their tokens without claiming. In any of the trees, wherever a miner gets the option to slash, it chooses that, and thus, all the other available options are not shown in the tree to reduce its size. The nodes in the tree represent the party that takes the action, and the action taken by that party is represented by the arrow that takes it to another node, which takes the next action. After all the transitions are finished, the leaf represents the utilities of the parties A, B and M. In Figure 9, Figure 10 and Figure 11, the utilities of all the parties in different game paths are represented at the leaf nodes. The utilities in *Chain-A* and *Chain-B* are represented with a superscript of 1 and 2, respectively, and f represents the transaction fees the miner gets on confirming some transaction. The claim tree and refund tree are together called as redeem tree. The game trees do not directly represent the opportunity cost, but if two paths give the same utility, the path with the least delay is preferred because of the lesser loss of opportunity cost. Ideally, B would prefer P_a over P_b and vice-versa for A as they are interested in the exchange. In case of refund this interest of parties changes and thus the difference in griefing premium $(p_b - p_a)$ helps in dis-incentivizing the parties from leaving.

6.1.1 Base Tree

The base tree begins with A deciding whether to send her locking transaction tx^{lock}_A to B. If not, the protocol ends without affecting either party's tokens. If sent, B can either publish tx^{lock}_A or respond with tx^{lock}_B. If B only publishes tx^{lock}_A, miners (M) confirm it and earn a

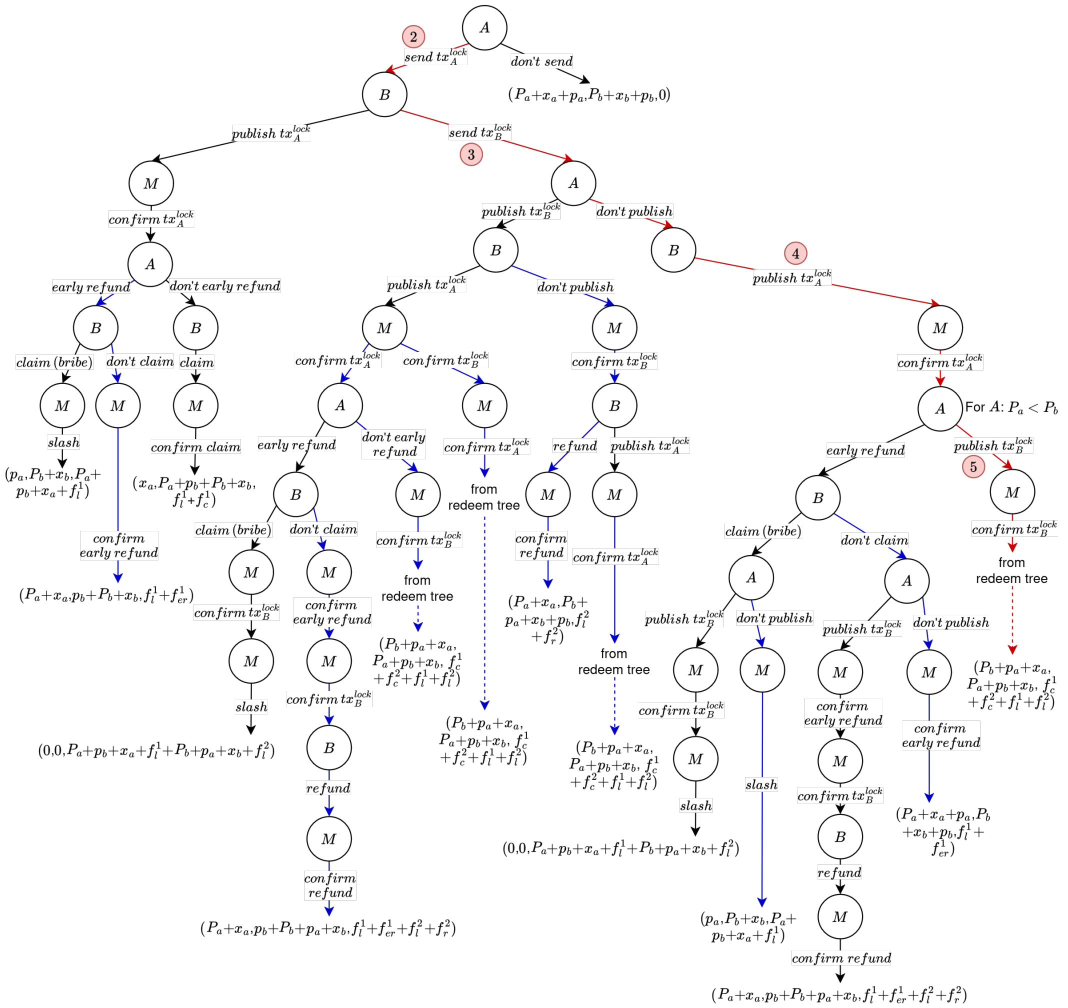

Figure 9 SPNE – Base Tree: Blue lines show sub-game optimal strategies; red lines show main game optimal strategies. Red path numbers correspond to transaction steps in Figure 6.

fee f_l^1, and A then chooses whether to publish an early refund. If A does so and B bribes miners to censor it, M slashes all locked tokens. Otherwise, A is refunded and M earns f_{er}^1. If A doesn't refund early, B can claim the funds after a timelock, gaining P_a, while A loses.

If B sends tx_B^{lock}, A then decides whether to publish it. If she does and B withholds tx_A^{lock}, M confirms tx_B^{lock} and gains f_l^2. B can now either refund, gaining both his funds and A's griefing premium, or proceed with the exchange. By backward induction, refunding yields higher utility. If both publish their locking transactions, miners confirm one first, and parties act accordingly. In any bribery attempt, miners benefit from slashing, and both parties incur losses.

Transaction fees f_{er}^1, f_r^1 are for early and regular refunds on *Chain-A*, and f_l^2, f_r^2 are for locking and refunding on *Chain-B*. The redeem tree shows equilibrium utilities after locking, further explored in the claim tree. If parties exchange locking transactions and A publishes first, backward induction shows B prefers refunding to gain the griefing premium.

Finally, if both parties exchange transactions and B publishes tx_A^{lock} first (confirmed with fee f_l^2), A can either early refund, publish tx_B^{lock} or do nothing(not shown in Figure 9). Since early refunding results in fee loss, backward induction favours continuing the exchange by publishing tx_B^{lock}. The path where A does not do anything will lead to B claiming the locked assets as it knows the main secret.

6.1.2 Claim Tree

The claim tree represents the subgame where both parties have published and locked their tokens on the chain. It starts with B having the option to publish a claim transaction or leave the exchange. Leaving the exchange means eventual refining by both parties, which we will discuss in the refund tree. If B publishes a claim transaction revealing the secret s_m, A can publish the claim transaction. In subsequent sub-games, whenever a party A or B gets an option to bribe by submitting the refund transaction on the chain, the best strategy for the miners is to slash; thus, all rational parties will refrain from doing so. Thus, the best strategy is to claim each other's tokens without refunding after publishing the claim transactions. f_c^1 represents the transaction fees of claim transaction on *Chain-A* and f_c^2, f_r^2 represents claim and refund transaction fees on *Chain-B*.

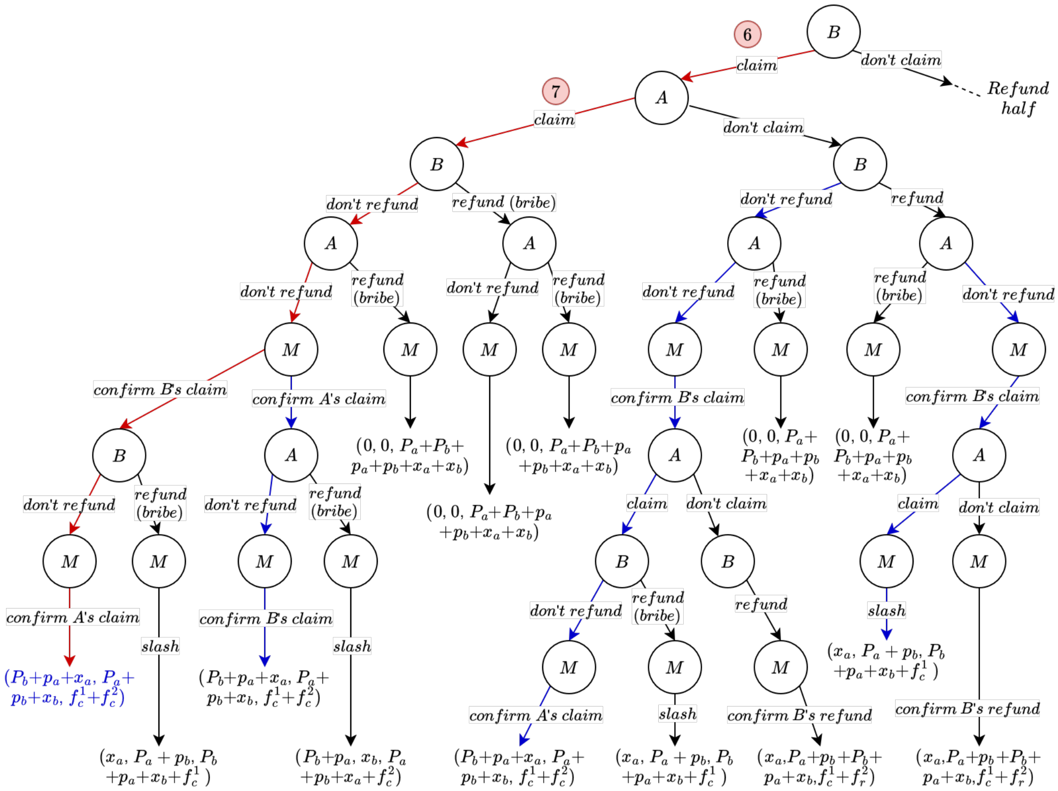

Figure 10 SPNE – Claim Half.

6.1.3 Refund Tree

When B does not publish a claim transaction, A either publishes a refund transaction or does not. If she does not publish the refund transaction, B will publish his refund transaction, which is eventually confirmed by miners by gaining f_r^2. After this, B can still try to claim

A's transaction by publishing a claim, which can be contested by A by publishing a refund, but the miners' best strategy would be to slash. If A publishes a refund after B does not publish a claim transaction, the best strategy for B is to also submit a refund transaction and avoid slashing actions for the miner. Thus, using backward induction, we can see from the refund tree that the best strategy is when both parties publish refund transactions without publishing their claim transactions.

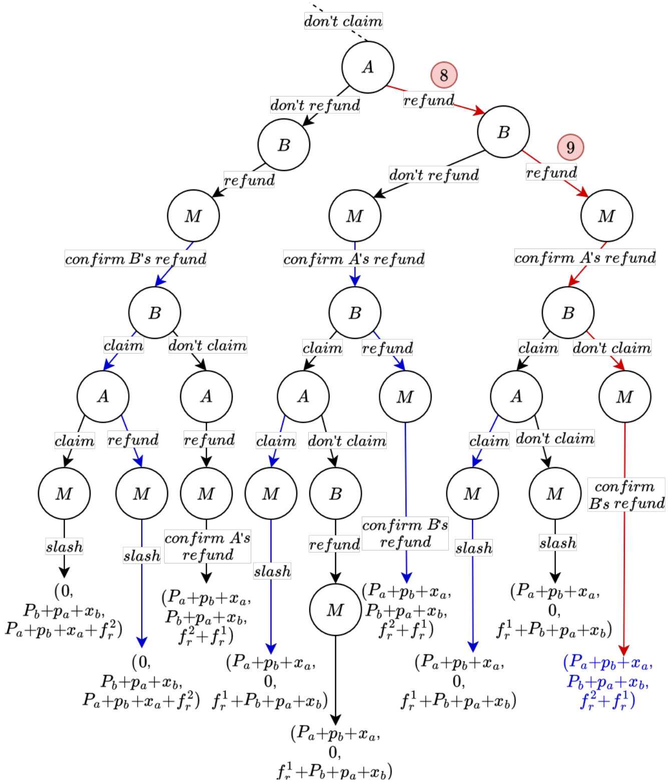

Figure 11 SPNE – Refund Half.

To prove that our protocol is game-theoretically safe, we need to show that the protocol holds the Subgame Perfect Nash Equilibrium at the honest(expected) path in the game tree.

▶ **Theorem 6** (SPNE). *The Subgame Perfect Nash Equilibrium for our game is the strategy profile* $s^*(A, B, M)$ = (*send* tx_A^{lock}, *don't publish, publish* tx_B^{lock}), (*send* tx_B^{lock}, *publish* tx_A^{lock}, *claim, don't refund, don't refund*), ((*confirm* tx_A^{lock}, *confirm* tx_B^{lock}, *confirm B's claim, confirm A's claim*) *or* (*confirm* tx_A^{lock}, *confirm* tx_B^{lock}, *confirm A's claim, confirm B's claim*)).

Proof. Lemma 5 ensures parties will never try to bribe the miners because of slashing. We apply backward induction at every subgame of the setup, claim, and refund phases of the game tree until we get the SPNE for the entire game. Figure 9, Figure 10 and Figure 11 show the SPNE of the entire 4-Swap game. Thus, parties can never get more utility than the prescribed(honest) strategy. ◀

Theorem 6 shows that the honest course of action is the SPNE of our game. We now show that the strongest miner M can be generalized to a multi-miner setup in both chains while ensuring the protocol's safety.

▶ **Lemma 7** (Miner's Dilemma). *In the case of a multi-miner setup, a miner does not wait for the parties to publish conflicting transactions for slashing opportunities.*

Proof. Lemma 5 proves that slashing gives the maximum utility to the miners. However, in the presence of rational parties, parties know that slashing is the best action for the miners, and thus, they will refrain from publishing conflicting transactions. Thus, the miner will lose the opportunity cost to some other miner by waiting for the conflicting transaction and hence does not wait for the same. ◀

▶ **Theorem 8.** *In the 4-Swap protocol, rational parties A and B maximize their utilities by following honest actions, even in a multi-miner setup.*

Proof. Replacing a single all-powerful miner with arbitrary miners in a multi-miner setup yields identical utilities for parties A and B at all game tree leaf nodes. Lemma 5 shows that the slashing mechanism deters any miner, regardless of hashing power, from delaying transactions due to bribery. Rational miners thus enforce the slashing mechanism when applicable. In non-bribery scenarios, miners can increase their rewards by confirming the only valid transaction (on any particular chain). Thus, arbitrary miners make the same decisions as an all-powerful miner. While the rewards in the game tree may be distributed among multiple miners in a multi-miner setup, this distribution does not affect the utilities of A and B. Consequently, the rational strategies for A and B remain aligned with honest behaviour, preserving the protocol's game-theoretic safety in a multi-miner environment. ◀

7 Discussion

To make the 4-Swap protocol reverse bribery safe along with griefing safety, we can leverage the idea of splitting the refund transaction into two steps, such that whenever a refund transaction is published, there is sufficient time for miners to claim a portion and burn the remaining tokens [20, 5]. Splitting the transaction in two steps helps in providing a punishing window, thereby solving the reverse bribery. In 4-Swap there are three bribery cases: early refund of A getting censored by claim of B, claim of B getting censored by refund of A and claim of A getting censored by refund of B. To handle all these cases, all the transactions that cause conflicts have to be split up. Thus, the claim of B and the two main refund transactions also have to be split up into two transactions. Thus for the worst case where both parties refund, we will have six transactions and in best case where parties exchange their tokens, we will have five transactions. This still leads to an open question whether we can extend this construction to be reverse bribery safe in just four transactions.

8 Conclusion

We propose 4-Swap, the first atomic swap protocol to achieve both grief-free and bribery-resistant atomic swaps using only four steps. To our knowledge, this is the first work that combines principals and premiums in both chains while matching the number of transactions in TN-Swap, leading to faster execution of the protocol compared to previous grief-free atomic swaps. A game-theoretical analysis of the protocol shows that the honest execution path constitutes a stable equilibrium, and thus satisfies the goals of being grief-free and safe against bribery attacks.

References

1 Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and Subhra Mazumdar. Securing lightning channels against rational miners. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 393–407, 2024. doi:10.1145/3658644.3670373.

2 Lukas Aumayr, Matteo Maffei, Oğuzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. Bitcoin-compatible virtual channels. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 901–918, 2021. doi:10.1109/SP40001.2021.00097.

3 Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *ACM Comput. Surv.*, 54(8):168:1–168:41, 2022. doi:10.1145/3471140.

4 Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007. doi:10.1007/978-3-540-70936-7_4.

5 Hao Chung, Elisaweta Masserova, Elaine Shi, and Sri AravindaKrishnan Thyagarajan. Rapidash: Atomic swaps secure under user-miner collusion. *Cryptology ePrint Archive*, 2022.

6 Yuanjun Ding and Weili Chen. Probing the mystery of cryptocurrency exchange: The case study based on mt.gox. In *International Conference on Service Science, ICSS 2022, Zhuhai, China, May 13-15, 2022*, pages 297–304. IEEE, 2022. doi:10.1109/ICSS55994.2022.00053.

7 Shange Fu, Qin Wang, Jiangshan Yu, and Shiping Chen. FTX collapse: A ponzi story. In Aleksander Essex, Shin'ichiro Matsuo, Oksana Kulyk, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez, Sam Werner, Andrea Bracciali, and Geoff Goodell, editors, *Financial Cryptography and Data Security. FC 2023 International Workshops - Voting, CoDecFin, DeFi, WTSC, Bol, Brač, Croatia, May 5, 2023, Revised Selected Papers*, volume 13953 of *Lecture Notes in Computer Science*, pages 208–215. Springer, 2023. doi:10.1007/978-3-031-48806-1_14.

8 Maurice Herlihy. Atomic cross-chain swaps. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 245–254. ACM, 2018. URL: https://dl.acm.org/citation.cfm?id=3212736.

9 Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 245–254, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3212734.3212736.

10 Philipp Hoenisch, Subhra Mazumdar, Pedro Moreno-Sanchez, and Sushmita Ruj. Lightswap: An atomic swap does not require timeouts at both blockchains. In Joaquín García-Alfaro, Guillermo Navarro-Arribas, and Nicola Dragoni, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2022 International Workshops, DPM 2022 and CBT 2022, Copenhagen, Denmark, September 26-30, 2022, Revised Selected Papers*, volume 13619 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2022. doi:10.1007/978-3-031-25734-6_14.

11 Subhra Mazumdar. Towards faster settlement in htlc-based cross-chain atomic swaps. In *4th IEEE International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications, TPS-ISA 2022, Atlanta, GA, USA, December 14-17, 2022*, pages 295–304. IEEE, 2022. doi:10.1109/TPS-ISA56441.2022.00043.

12 Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Timelocked bribing. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I*, pages 53–72, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-662-64322-8_3.

13 Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Grief-free atomic swaps. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2022, Shanghai, China, May 2-5, 2022*, pages 1–9. IEEE, 2022. doi:10.1109/ICBC54727.2022.9805490.

14 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.

15 Tier Nolan. Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949, 2013. Accessed: 2024-05-20.

16 Sandeep Rao and Corbet Shaen. Mt. gox–the fall of a giant. *Understanding Cryptocurrency Fraud, edited by Shaen Corbet*, pages 71–82, 2022.

17 Kirti Singh, Vinay J. Ribeiro, and Susmita Mandal. 4-swap: Achieving grief-free and bribery-safe atomic swaps using four transactions, 2025. doi:10.48550/arXiv.2508.04641.

18 Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1299–1316. IEEE, 2022. doi:10.1109/SP46214.2022.9833731.

19 Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. MAD-HTLC: because HTLC is crazy-cheap to attack. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1230–1248. IEEE, 2021. doi:10.1109/SP40001.2021.00080.

20 Sarisht Wadhwa, Jannis Stoeter, Fan Zhang, and Kartik Nayak. He-htlc: Revisiting incentives in HTLC. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. URL: https://www.ndss-symposium.org/ndss-paper/he-htlc-revisiting-incentives-in-htlc/.

21 Yingjie Xue and Maurice Herlihy. Hedging against sore loser attacks in cross-chain transactions. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 155–164. ACM, 2021. doi:10.1145/3465084.3467904.

22 Ryan Zarick, Bryan Pellegrino, and Caleb Banister. Layerzero: Trustless omnichain interoperability protocol. *CoRR*, abs/2110.13871, 2021. arXiv:2110.13871.