Lilac: Parallelizing Atomic Cross-Chain Swaps

Donghui Ding*†, Bo Long*†, Feng Zhuo*†, Zhongcheng Li*†, Hanwen Zhang*†¶, Chen Tian‡, Yi Sun*†§✉
* Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
† University of Chinese Academy of Sciences, Beijing, China
‡ Nanjing University, Nanjing, China
§ Shandong Key Laboratory of Blockchain Finance, Jinan, China
¶ Institute of Blockchain R&D of ICT, Hainan, China
✉corresponding author
{dingdonghui, longbo20g, zhuofeng19s, zcli, hwzhang, sunyi}@ict.ac.cn, {tianchen}@nju.edu.cn

***Abstract*—Hashed Timelock Contract (HTLC) is a widely-used protocol for cross-chain asset swaps. However, it relies on serial asset-locking to guarantee atomicity, which causes high latency and poor fairness. Aiming at the drawbacks of HTLC, we propose Lilac, a cross-chain asset swap protocol that supports parallel asset-locking. Lilac replaces the unique asset-unlocking credential in HTLC with multiple sub-credentials generated by all participating users, and the sequence of sub-credentials is used as the complete asset-unlocking credential. Users obtain the complete credential only when all assets have been locked, and the credential construction process is independent of the order in which assets are locked, so atomicity can be guaranteed when users lock their assets in parallel. Experiments show when a swap involves 2 to 4 blockchains, Lilac reduces the swap latency by 36.75% to 62.20%. Moreover, Lilac reduces the waiting time gap between different users so the fairness of a swap is improved.**

***Index Terms*—blockchain, cross-chain swap, atomicity, HTLC, swap latency, fairness**

I. Introduction

A variety of blockchain platforms featured by different types of digital assets have emerged in recent years, and these digital assets can circulate reliably within independent blockchains designed for specific industrial applications. However, there are also scenarios in which assets circulate across multiple industrial sectors, so **achieving efficient and secure asset swaps between different blockchains (i.e., cross-chain swaps) has become an urgent demand.**

For instance, we assume there are three blockchains: the copyright chain, tourism chain and game chain. Each blockchain carries a different type of assets: copyrights, hotel vouchers and game credits. Now users of these blockchains have following intentions for swapping:

- Alice holds accounts on the **copyright chain** and **game chain**. She took some photos during her trip, and used the copyright chain to manage the **photo copyrights**. Now she hopes to trade these copyrights for the **game credits** on the game chain.
- Bob holds accounts on the **tourism chain** and **copyright chain**. He runs a hotel in the city where Alice has visited, and issues **vouchers** on the tourism chain. Now he plans to trade some vouchers for Alice's photo copyrights for advertising.
- Carol holds accounts on the **game chain** and **tourism chain**. She owns the **game credits** Alice wants on the game chain. At present, she is also planning a trip and hopes to trade these game credits for the **vouchers** issued by Bob on the tourism chain.

Based on the above intentions, a cross-chain swap involving three blockchains is established. We use the "swap graph" in Fig.1 to describe this swap, in which each vertex represents a user participating in the swap, and each directed edge represents a transfer operation on a blockchain. As is shown in Fig.1, this swap is jointly achieved by multiple "transfer operations" occurring on three blockchains: on the copyright chain, Alice transfers the copyrights to Bob; on the tourism chain, Bob transfers the vouchers to Carol; and on the game chain, Carol transfers game credits to Alice. Each user acts as the **sender** (losing the asset) in the transfer operation on one blockchain, and the **receiver** (obtaining the asset) in the transfer operation on another blockchain. For example, Alice loses the copyrights on the copyright chain, but obtains game credits on the game chain.

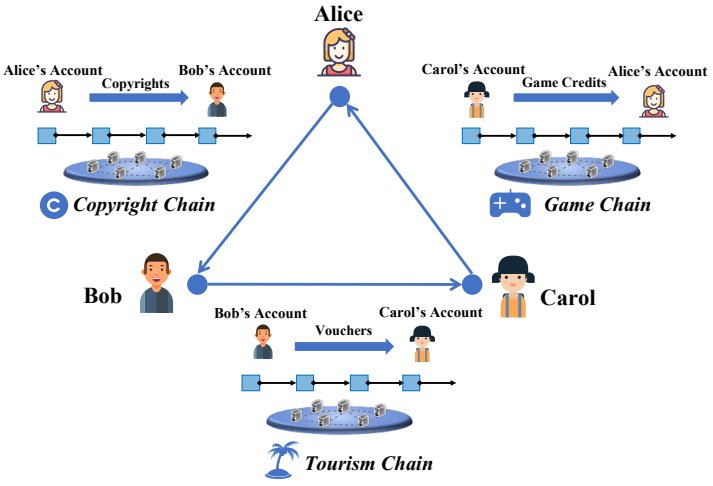


Fig. 1: Instance of Swap Graph

The core of a cross-chain swap is to guarantee **atomicity**, which means **either all the transfer operations in this swap are successfully executed, or none of them are executed.** However, atomicity is challenging in practice. This is because when a user executes a transfer operation, it is difficult to

978-1-6654-9792-3/22/$31.00 ©2022 IEEE

predict whether other users will also execute the transfer operations as agreed. For example, once Alice has transferred the copyrights to Bob on the copyright chain, but Carol refuses to transfer game credits to Alice on the game chain, Alice will suffer from an economic loss.

Hashed Timelock Contract (HTLC) is a widely-used cross-chain swap protocol [1] [2] that guarantees atomicity. This protocol adopts **Two-Phase Commit (2PC)** [3], which divides a swap into the **prepare phase** and the **commit phase**. Moreover, an escrow contract (or an **escrow** for short) is deployed on each blockchain, and one of the users (e.g., Alice) is elected as the **initiator**, who will generate a globally unique credential for all users to obtain their required assets.

In the prepare phase, starting from the initiator, the sender of each transfer operation **serially** locks her/his asset into the escrow based on **the order of the swap graph** (In our instance, this order is "Alice $\rightarrow$ Bob $\rightarrow$ Carol"). Since the swap graph forms a cycle (Fig.1), once the asset required by the initiator (game credits in our instance) is locked, all of other assets involved in this swap will also be locked. In other words, all users have been ready to execute the transfer operations. This swap then enters the commit phase. The initiator releases the credential, and the receiver in each transfer operation uses this credential to obtain (or "unlock") her/his required asset from the escrow. However, if any user fails to lock the asset, the initiator will not release the credential, and each locked asset will be withdrawn by its original sender after timeout.

In HTLC, the key for atomicity guarantee is **serial asset-locking**. This is because the unique credential for all users to unlock their required assets is held by the initiator, and can only be released after all assets have been locked (i.e., in the commit phase). If each user serially locks the asset, the initiator will learn the current phase of a swap by observing whether her/his required asset has been locked. However, serial asset-locking has two drawbacks:

- **High swap latency.** When a swap involves multiple blockchains and especially includes "public chains" such as Bitcoin or Ethereum, the swap latency may reach dozens of minutes or even several hours.
- **Poor fairness.** If we define the time taken by each user from locking her/his originally-owned asset on one blockchain to unlocking her/his required asset on another blockchain as the "**waiting time**", the waiting time of users who lock the assets earlier (e.g., the initiator) will be longer. This may make users unwilling to become the initiator and thus reduce the swap success rate.

Aimed at the drawbacks of serial asset-locking in HTLC, the contribution of this paper is to propose **Lilac, a novel cross-chain swap protocol that supports parallel asset-locking**. Lilac decomposes the unique credential in HTLC: **Each user generates a random value as the sub-credential, and the sequence composed of all random values is used as the complete credential for all users to obtain the locked assets.** In each transfer operation of a swap, if the sender has locked the asset, the receiver will release her/his random value as confirmation. Once all random values are released, it means that all assets in this swap have been locked. Moreover, the complete credential is revealed, by which all users obtain their required assets.

In Lilac, the construction process of the credential is independent of the order in which assets are locked, so atomicity is guaranteed when users lock the assets in parallel. Therefore, **Lilac effectively reduces the swap latency.** In addition, the time taken by each user from assets-locking to reveal of the credential is close to other users and only depends on the highest transaction confirmation latency, **so Lilac reduces the waiting time gap between different users, thus improving the fairness of a swap.**

Our experiments show the swap latency of Lilac is reduced by 36.75% to 62.20% compared with HTLC when a swap involves 2 to 4 blockchains. Meanwhile, Lilac effectively reduces the variance of user waiting time in each swap and thus improves the fairness.

The rest of this paper is organized as follows. Section II describes the detailed design of Lilac. Section III analyzes the security of Lilac. Section IV evaluates the performance of Lilac using experiments. Section V introduces the related work of this paper. Finally we conclude our work in Section VI.

II. Design of Lilac

This Section describes the detailed design of Lilac. Subsection II-A introduces the formal notations used by this paper and the assumption based on which Lilac is constructed. Subsection II-B describes the basic workflow of Lilac. Subsection II-C theoretically analyzes the swap latency and user waiting time of HTLC and Lilac.

A. Formal Notations and Assumptions

1. Formal Notations: To accurately specify a cross-chain swap involving n blockchains, we utilize the formal notations in Table I. As an instance of these notations, for the swap introduced in Section I, B=<copyright chain, tourism chain, game chain>, U=<Alice, Bob, Carol>, and A=<photo copyrights, vouchers, game credits>. Moreover, e_1, e_2 and e_3 represent the escrows deployed on copyright chain, tourism chain and game chain respectively.

2. Assumption: Similar to other cross-chain swap solutions including HTLC, we assume all users are connected to a matching platform. Users send their swapping requirements to this platform, and the platform matches swapping requirements to establish the swap graphs. Different from current cryptocurrency exchanges, the matching platform does not hold assets of users during a swap.

B. Workflow of Lilac

The workflow of Lilac is also summarized as 2PC: In the prepare phase, the sender of each transfer operation temporarily locks the asset into the escrow. Once all of the assets to be swapped have been locked, this swap enters the commit phase, and the receivers can unlock their required assets from the escrows.

Based on the swap instance in Section I, the rest of this subsection first introduces the procedures before a swap (called

TABLE I: Formal Notations

Formal Notations	Explanations
$B = \langle b_1, b_2, \cdots, b_n \rangle$	Sequence of all blockchains in this swap.
$U = \langle u_1, u_2, \cdots, u_n \rangle$	Sequence of all users participating in this swap.
$A = \langle a_1, a_2, \cdots, a_n \rangle$	Sequence of assets to be swapped. Each asset a_i $(1 \leq i \leq n)$ is issued on b_i, and belongs to u_i before the swap.
$OP = \langle op_1, op_2, \cdots, op_n \rangle$	Sequence of transfer operations included in this swap. In each operation op_i, u_i transfers the ownership of a_i to u_j on the blockchain b_i, and $j = (i+1) \bmod n$, so u_i and u_j are the sender and receiver of op_i respectively.
e_i	Escrow deployed on the blockchain b_i.
$T(i)$	**Timeout moment** specified in e_i. After the sender u_i locks a_i into e_i, if the receiver u_j fails to unlock the asset a_i before $T(i)$, u_i will withdraw a_i from e_i.
t_i^r	Transaction recording latency of b_i (i.e., the time taken by a transaction to be recorded on b_i).
t_i^c	Transaction confirmation latency of b_i (i.e., the time taken by a transaction to be confirmed by b_i). In some blockchains, a transaction needs to wait for several additional blocks before it is considered confirmed, so $t_i^c \geq t_i^r$.
t_{max}^c	$t_{max}^c = \max\{t_1^c, t_2^c, \cdots, t_n^c\}$.

"pre-swap phase"), then describes the workflow of the prepare phase and commit phase respectively, and finally gives the value of *timeout moment* (defined in Table I) of each escrow.

1) Pre-Swap Phase:

Each user with an intention for swapping locally generates a random value s_i, and then calculates the hash value h_i of s_i. As an instance, in Fig.2 Alice, Bob and Carol independently generate the random values s_1, s_2 and s_3, and then calculate the hash values h_1, h_2 and h_3 respectively. When each user u_i sends the swapping requirement to the matching platform, the hash value h_i is included in this requirement as a parameter. The platform matches the requirements from different users to establish the swap graphs. Various matching algorithms have been proposed [4] [5], which can be adopted by the platform.

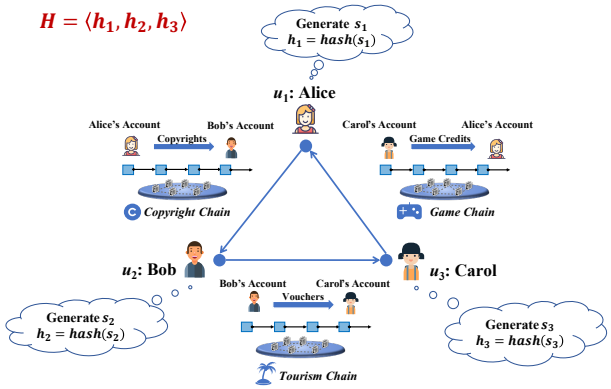


Fig. 2: Instance of Pre-Swap Phase

If a swap graph is established, the platform will push this graph to all users who participate in this swap, so these users will learn the information of this swap in a timely manner. Similar to HTLC, one of the users is elected as the "initiator". Without loss of generality, we let u_1 be the initiator, and assume Alice is elected as the initiator in our instance. Bob and Carol respectively establish network connections with Alice, and all three users read a hash value sequence $H = \langle h_1, h_2, h_3 \rangle$ from the matching platform, which will be used for asset-locking during the prepare phase.

2) Prepare Phase:

In each transfer operation op_i of a swap, the sender u_i broadcasts a "locking transaction" $TX_{locking}^{u_i}$ into the network of the blockchain b_i. This transaction temporarily locks the asset a_i into the escrow e_i, and sets the following unlocking conditions in e_i using the hash value sequence $H = \langle h_1, h_2, \cdots, h_n \rangle$:

1. If the receiver u_j provides the random value sequence $S = \langle s_1, s_2, \cdots, s_n \rangle$ before $T(i)$, and for $\forall k \in [1, n]$, s_k in S and h_k in H satisfy $h_k = \text{hash}(s_k)$, the asset a_i will be transferred to u_j.
2. If the receiver u_j fails to provide the correct sequence S before $T(i)$, u_i will withdraw the asset a_i.

If the receiver u_j is not the initiator (i.e., $j \neq 1$), after the "locking transaction" is confirmed by b_i, u_j will release the random value s_j to the initiator u_1 through the pre-established network connection. The initiator u_1 then locally verifies if received s_j satisfies $h_j = \text{hash}(s_j)$ and rejects fake s_j.

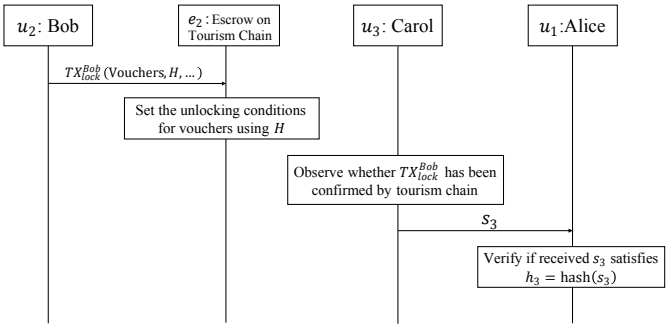


Fig. 3: Workflow of Prepare Phase

As an instance, Fig.3 illustrates the workflow of Lilac when Bob locks his vouchers. Bob broadcasts a locking transaction $TX_{locking}^{\text{Bob}}$ into the network of the tourism chain b_2. This transaction locks his vouchers into the escrow e_2, and sets the unlocking conditions using H, ensuring Carol can only obtain the vouchers before $T(2)$. The client of Carol monitors the events occurring on the tourism chain, and releases s_3 to Alice once observing $TX_{locking}^{\text{Bob}}$ has been confirmed.

Similarly, if Alice locks the photo copyrights on the copyright chain b_1, Bob will release his random value s_2 to Alice. As the initiator, Alice also monitors whether Carol has locked the game credits on the game chain b_3, but needn't release s_1 during the prepare phase.

3) Commit Phase:

If the initiator Alice receives s_2 and s_3 from Bob and Carol, and confirms that Carol has locked the game credits on the game chain b_3, she will know all assets involved in this swap have been locked. Moreover, she can construct the complete sequence $S = \langle s_1, s_2, s_3 \rangle$.

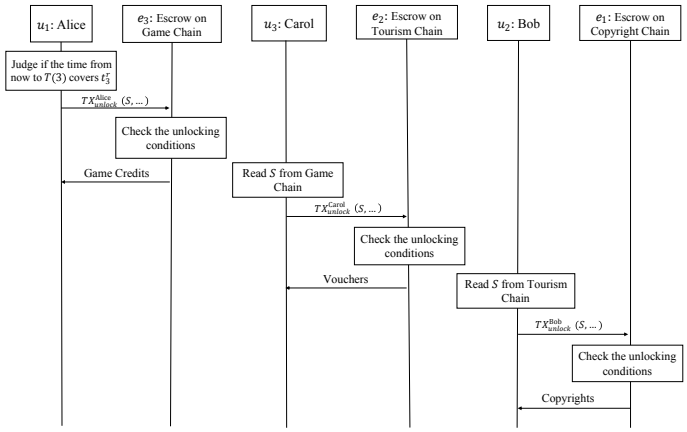


Fig. 4: Workflow of Commit Phase

Fig.4 illustrates the workflow of the commit phase. Before releasing S, Alice judges whether remaining time is adequate to unlock the game credits on the game chain (i.e., whether the time from now to $T(3)$ covers t_3^r). If the time remained by $T(3)$ is adequate, Alice will broadcast an "unlocking transaction" $TX_{unlocking}^{\text{Alice}}$ containing S into the network of the game chain. After this transaction is recorded on the game chain, the escrow e_3 will check whether following unlocking conditions are satisfied:

1. Whether current time is earlier than $T(3)$.
2. For $\forall k \in [1, n]$, whether s_k in S and h_k in H satisfy $h_k = \text{hash}(s_k)$.

If the "unlocking transaction" broadcast by Alice passes the above checks, e_3 will transfer the game credits to Alice. Carol then reads the sequence S from the game chain, and forwards S to the tourism chain b_2 before $T(2)$ to obtain the vouchers locked by Bob. Similarly, Bob forwards the sequence S from the tourism chain to the copyright chain b_1 before $T(1)$ to obtain the photo copyrights locked by Alice.

Compared with HTLC, the evidence that all assets in this swap have been locked changes from "*the asset required by the initiator Alice has been locked* " to "*Alice has received all random values and Carol has locked her game credits*". Alice can construct and then release the complete credential S only in the commit phase, which avoids the adverse situation where only some of the transfer operations in a swap are executed. Moreover, **the credential construction process is independent of the order in which assets are locked, so Lilac guarantees atomicity when assets are locked in parallel.**

In the above swap, if Bob fails to lock his vouchers, Alice will not receive s_3 so she cannot construct complete S, and if Carol fails to lock her game credits, Alice may obtain all random values but is not motivated to release S. In both cases any locked asset a_i will be withdrawn by u_i after $T(i)$.

Finally, the commit phase can be further accelerated: After constructing S, Alice proactively sends S to Bob and Carol through the off-chain network, and then all users can broadcast the "unlocking transactions" in parallel.

4) Timeout Moment $T(i)$:

$T(i)$ is the key factor that determines whether all transfer operations in a swap are executed. If we assume all users simultaneously broadcast the "locking transactions" at an initial time T_0, and ignore the network delay between two nodes (generally less than 1s), then $T(i)$ is defined by Equation 1:

$$T(i) = (T_0 + t_{max}^c + \Delta_p) + \sum_{k=i}^{n} (t_k^r + \Delta_k) \tag{1}$$

Equation 1 consists of two parts. $(T_0 + t_{max}^c + \Delta_p)$ is the expected ending time of the prepare phase (i.e., the time when all "locking transactions" are confirmed by different blockchains). $\sum_{k=i}^{n}(t_k^r + \Delta_k)$ guarantees Equation 2 holds, so in the commit phase $u_{i+1}\,(1 \leq i < n)$ has adequate time to forward S from b_{i+1} to b_i by broadcasting an "unlocking transaction".

$$T(i) - T(i+1) = t_i^r + \Delta_i, 1 \leq i < n \tag{2}$$

The offsets Δ_p and Δ_k in Equation 1 determine the tolerance of Lilac to abnormal events. If the offsets are small, once a user encounters a device crash the remaining time will be inadequate to support the device recovery, resulting in the failure of this swap. However, current research [6] on HTLC shows that if the time during which assets are locked in escrows is excessively long, users will be motivated to wait and observe the fluctuation of asset price during this time window, and then decide whether to continue the swap. This slows down the swap process and reduces the swap success rate. We leave the design of more accurate $T(i)$ as our future work.

C. Swap Latency & User Waiting Time

Lilac optimizes HTLC in two aspects: **1) lowering the swap latency; 2) reducing the waiting time gap between different users and thus improving the fairness.** In this subsection we will theoretically analyze and compare the swap latency and user waiting time of HTLC and Lilac.

1) Swap Latency:

The **swap latency** is defined as the time from the broadcast of the first "locking transaction" to the confirmation of all "unlocking transactions". Ignoring the abnormal events such as device crashes, we only focus on the minimum swap latency, and use L^H and L^{lic} to denote the minimum swap latency of HTLC and Lilac respectively.

HTLC. In the prepare phase, users serially broadcast the "locking transaction" into different blockchains starting from the initiator u_1. In the commit phase, we assume the initiator proactively sends the credential to other users, and then all users broadcast the "unlocking transactions" in parallel. A swap ends when all "unlocking transactions" are confirmed. L^H thus satisfies Equation 3:

$$L^H = \sum_{k=1}^{n} t_k^c + t_{max}^c \tag{3}$$

Lilac. In the prepare phase, we assume users simultaneously broadcast the "locking transactions", and the prepare phase ends when all "locking transactions" are confirmed. In the commit phase, the workflow of Lilac is the same as that of HTLC. L^{lic} thus satisfies Equation 4:

$$L^{lic} = 2t_{max}^c \tag{4}$$

As $t_{max}^c < \sum_{k=1}^{n} t_k^c$, Lilac effectively lowers the swap latency compared with HTLC.

2) User Waiting Time:

For each user u_i, the **waiting time** is defined as the time from the broadcast of her/his "locking transaction" to the confirmation of her/his "unlocking transaction". We use W_i^H and W_i^{lic} to denote the minimum waiting time of u_i in HTLC and Lilac respectively.

HTLC. In the prepare phase, users serially broadcast the "locking transactions", and when the commit phase begins, all users can broadcast the "unlocking transactions" in parallel, so W_i^H satisfies Equation 5:

$$W_i^H = \begin{cases} \sum_{k=i}^{n} t_k^c + t_{i-1}^c, & 1 < i \leq n \\ \sum_{k=i}^{n} t_k^c + t_n^c, & i = 1 \end{cases} \tag{5}$$

Lilac. If all users simultaneously broadcast the "locking transactions", W_i^{lic} will satisfy Equation 6:

$$W_i^{lic} = \begin{cases} t_{max}^c + t_{i-1}^c, & 1 < i \leq n \\ t_{max}^c + t_n^c, & i = 1 \end{cases} \tag{6}$$

Equation 5 indicates in HTLC the waiting time of u_i during the prepare phase (i.e., $\sum_{k=i}^{n} t_k^c$) is prolonged with the decrease of i. By contrast, Equation 6 indicates in Lilac the waiting time of all users during the prepare phase is equal to t_{max}^c. Lilac thus reduces the waiting time gap between different users and improves the fairness of a swap.

Equations 5 and 6 also show $W_i^{lic} > W_i^H$ may occur if $t_{max}^c > \sum_{k=i}^{n} t_k^c$. To avoid this, in Lilac u_i can defer broadcasting the "locking transaction" for a period of time. In the most ideal case, the waiting time of u_i during the prepare phase is reduced from t_{max}^c to t_i^c, which leads to $W_i^{lic} < W_i^H$.

III. Security Analysis and Discussion

This section introduces the changes of Lilac compared with HTLC, and analyzes security issues caused by such changes.

The main difference between two protocols is that in Lilac all users can lock their assets in parallel. By contrast, in HTLC u_{i+1} locks her/his asset a_{i+1} only after a_i has been locked by u_i. To support parallel asset-locking, Lilac further makes two changes: 1) The credential is generated by all users instead of the single initiator. 2) Off-chain network connections are established between the initiator and other users to transmit sub-credentials (i.e., random values). The above changes cause following security issues.

Long-term locking problem [6] [7]. If a user refuses to lock her/his asset midway due to price fluctuations, assets of other users may be locked in the escrows for a long time and cannot be used until $T(i)$. This problem does not violate atomicity but affects user experience. More users are likely to encounter this problem in Lilac than in HTLC. When u_m refuses to lock her/his asset a_m, in HTLC any u_i $(m < i \leq n)$ will also not lock a_i, so $n-m-1$ users encounter this problem. By contrast, in Lilac the misbehavior of u_m causes up to $n-1$ users to suffer from this problem. However, Lilac accelerates the swap process, so we can set $T(i)$ closer to the current time, thereby shortening the time during which assets are locked in escrows. Moreover, Lilac compresses the time window for users to observe the price fluctuations, and thus reduces the probability that users quit a swap.

Failure of Random Value Transmission Link. The network link between u_i $(i \neq 1)$ and the initiator u_1 may encounter a failure, causing the random value s_i to be lost or tampered with. This issue results in the abort of a swap, but does not violate atomicity. Aiming at this issue, the initiator u_1 should return a confirmation message to u_i after receiving s_i. Moreover, users can reduce the impact of such failure by switching network links and retransmission.

Attack using fake random values. A malicious user u_i $(i \neq 1)$ may send a fake random value s_i' to the initiator u_1. Once u_1 releases the sequence containing s_i', all users except u_i cannot unlock their required assets. However, u_i can secretly construct the correct S, and obtains a_{i-1} without loosing a_i after $T(i)$. Therefore, the initiator should locally verify whether $h_i = \text{hash}(s_i)$ is satisfied for each received s_i and reject fake s_i. To prevent malicious users from sending a large number of fake random values and thus wasting the computation resources, u_1 should only verify random values from authenticated users. If a user continuously sends fake random values, the connection from this user will be cut off. We can achieve secure user authentication by decentralized identification mechanisms [8].

Besides the above issues, if the device of u_{i+1} crashes and fails to recover before $T(i) - t_i^r$, this user may lose a_{i+1} without obtaining a_i. However, HTLC adopts the same timeout mechanism as Lilac, so **the security level of Lilac is not reduced compared with HTLC.** Moreover, Lilac is lightweight and u_{i+1} can easily switch to a new device.

IV. Implementation and Evaluation

We implemented Lilac and HTLC based on Ethereum. The escrows were developed with `Solidity 0.5.0` [9], and the user client was developed with `Python 3.6`. We deployed both protocols on the platform in Table II, and used PoA (Proof of Authority) consensus to flexibly adjust the **block interval** of

each blockchain (i.e., the time interval between the generation of two adjacent blocks in each blockchain). We conducted following experiments to evaluate the performance of Lilac.

TABLE II: Platform of Experiments

Platform	Configuration
CPU	Intel® Xeon® Silver 4114 Processor (2.2 GHz×40 cores)
Operating System	CentOS Linux 7.9 (2009)
Memory Size	64G
Disk Capacity	1T

A. Experiment1: Swap Latency under Different Blockchain Numbers and Block Intervals

In this experiment, we compared the swap latency of HTLC and Lilac under different combinations of blockchain numbers and block intervals. 18 combinations were constructed in total: The blockchain number n was gradually increased from 2 to 4, and the block intervals were set to 25s, 30s, 35s, 40s, 45s and 50s respectively. For each combination we ran two tests, one of which was for HTLC and the other was for Lilac. In each test the block intervals of all n blockchains were equal, and we launched 3 swaps in total. The time interval between each swap was picked up randomly because in practice we cannot predict when a swap occurs. Moreover, once a transaction was recorded on the blockchain, we waited one more block before this transaction was considered confirmed. To accelerate this experiment, in both HTLC and Lilac we let the initiator proactively send the asset-unlocking credential to other users during the commit phase.

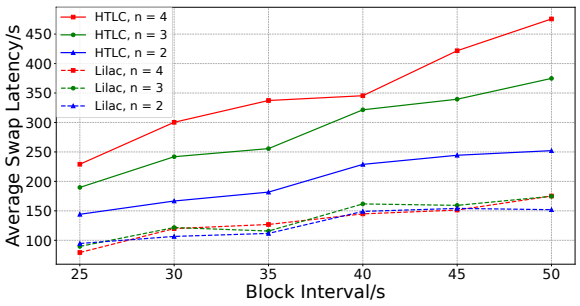


Fig. 5: Swap Latency of HTLC and Lilac

Our experiment results are reflected in Fig.5. For the same blockchain number, the solid line is always located above the dotted line, indicating the swap latency of Lilac is lower than that of HTLC in any combination. Moreover, the swap latency of HTLC grows with the block interval, and the solid lines with larger blockchain numbers are located in the upper part of this figure, indicating the swap latency of HTLC depends on both the transaction confirmation latency and the blockchain number. By contrast, although the swap latency of Lilac also grows with the increase of the block interval, all dotted lines roughly overlap, which indicates the swap latency of Lilac is independent of the blockchain numbers. This is consistent with Equations 3 and 4. In general for the same block interval, the optimization effect of Lilac becomes more obvious with the increase of the blockchain number. When a swap involves 2 blockchains, Lilac reduces the swap latency by 36.75% on average compared with HTLC, and when 4 blockchains are involved, the average reduction reaches 62.20%.

Lines in Fig.5 are not rigorously straight. This is because the transaction confirmation latency t_i^c depends on multiple factors including the block interval and the time at which this transaction is broadcast. We randomly launched the swaps in our experiment to simulate real situations, which added uncertainty to the value of t_i^c. Moreover, after a transaction is sent to the network of b_i, it usually needn't wait the entire block interval before being recorded on b_i if b_i is not congested, so t_i^c is less than two block intervals when one more confirmation block is required. Therefore, the swap latency of Lilac is less than four block intervals given $L^{lic} = 2t_{max}^c$. For instance, L^{lic}<200s holds in Fig.5 when the block interval reaches 50s.

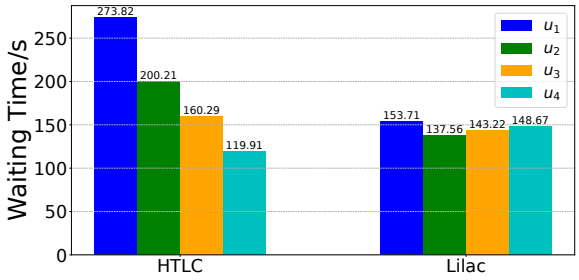
B. Experiment2: User Waiting Time under Different Block Intervals

Experiment2 compared the user waiting time of HTLC and Lilac. We considered the cross-chain swaps involving 4 blockchains, and constructed 3 combinations in total: The block intervals in each combination were (40s, 40s, 40s, 40s), (20s, 30s, 40s, 50s) and (50s, 40s, 30s, 20s) respectively. We ran 2 tests for each combination, one of which was for HTLC and the other was for Lilac. 3 swaps were launched in each test, and we measured the waiting time of u_i $(1 \leq i \leq 4)$ in these swaps. Finally, we took the average of each user's waiting time. Our experiment results are reflected in Fig.6.

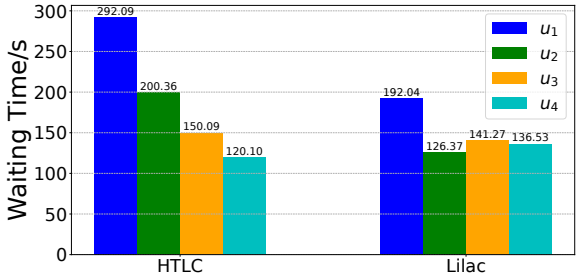
As shown in Fig.6, Lilac effectively reduces the waiting time gap between different users, and thus improves the fairness compared with HTLC. According to our calculation, in Lilac the population variances of user waiting time are 36.32s^2, 644.78s^2 and 754.88s^2. By contrast the population variances in HTLC are 3229.45s^2, 4251.34s^2 and 5518.64s^2. For HTLC, the waiting time of the initiator u_1 in all 3 combinations is significantly longer than that of other users, which may make users unwilling to become the initiator and thus reduce the swap success rate.

Fig.6a shows the waiting time of all users in Lilac is close when the combination is (40s, 40s, 40s, 40s), indicating our protocol performs best in terms of the fairness when block intervals of all involving blockchains are equal. Fig.6b shows in Lilac the waiting time of u_1 is the longest among all 4 users, because the waiting time of all users during the prepare phase is equal to t_{max}^c (Equation 6) and u_1 will send an "unlocking transaction" to b_4 with the block interval of 50s, causing u_1 to wait the longest time during the commit phase. Similarly, Fig.6c shows the waiting time of u_2 in Lilac is the longest among all 4 users.

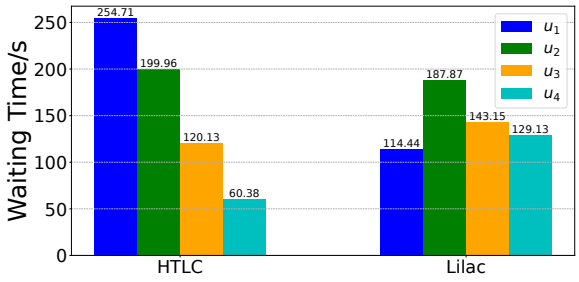
The waiting time of u_4 in Lilac is longer than in HTLC for all 3 combinations. This is because in Lilac the waiting



(a) User Waiting Time of (40s, 40s, 40s, 40s)



(b) User Waiting Time of (20s, 30s, 40s, 50s)



(c) User Waiting Time of (50s, 40s, 30s, 20s)

Fig. 6: User Waiting Time of HTLC and Lilac

time of u_4 during the prepare phase is equal to t^c_{max} while in HTLC such value is equal to $\bar{t^c_4}$, the average transaction confirmation latency of b_4 (Equation 5). Generally $t^c_{max} > \bar{t^c_4}$ holds. Nevertheless, the average user waiting time of Lilac is shorter than that of HTLC in all 3 combinations. Moreover, u_4 can defer broadcasting the "locking transaction" in Lilac to shorten her/his waiting time as specified in Subsection II-C2.

V. Related Work

Achieving atomic cross-chain swaps has become a common concern in the blockchain industry. A variety of protocols supporting atomic swaps have been proposed, which are divided into two categories:

1. Protocols based on the third-party coordinator. The coordinator owns the global view of each swap and triggers correct transfer operations on involving blockchains. Such protocols are further categorized into two approaches. One uses a single notary as the coordinator, and its typical instances include centralized cryptocurrency exchanges and Tesseract [10], a protocol based on Trusted Execution Environment (TEE). This approach supports efficient asset swaps, but requires the notaries be highly credible, otherwise the notaries may be malicious or hacked. The other introduces a third-party blockchain as the coordinator, and its typical instances are XCLAIM [11] and AC^{3}WN [12]. This approach reduces the security risks compared with centralized notaries, but highly relies on third-party facilities: A new "coordinating chain" needs to be introduced, and multiple "relay nodes" should be set up to deliver the coordinating messages and necessary verification information (such as block headers) between the "coordinating chain" and other blockchains.

2. Hashed Timelock Contract (HTLC). This protocol was first proposed in 2013 and initially used for asset-swaps between two blockchains [1]. In 2018 Herlihy [2] extended this protocol to the swaps between multiple blockchains. HTLC does not rely on third-party coordinators[1] and achieves swaps only by the cooperation of participating users. Currently it has served as the core components of various asset-swapping platforms [13] [14]. Lilac optimizes HTLC while the advantage of HTLC is not compromised.

As is specified in Section III, in a swap where the asset price fluctuates significantly, a user may quit this swap and thus causes the "long-term locking problem". Recently this issue has attracted increasing attentions. Using game theory, Xu *et al.* [6] obtain the quantitative relation between the swap success rate and other factors including the degree of price fluctuation. Han *et al.* [15] and Xue *et al.* [7] introduce the pledge mechanism into HTLC. If a user refuses to lock the asset, pledge of this user will be confiscated, thus weakening users' motivation to quit. Lilac provides another solution to this issue: It compresses the time window for users to observe price fluctuations, so the swap success rate is increased.

VI. Conclusion

In this paper, we optimize HTLC, and propose Lilac, a novel protocol that supports parallel asset-locking. Compared with HTLC, Lilac effectively reduces the swap latency and improves the fairness in terms of user waiting time. In our future work, we will design a more accurate *timeout moment* $T(i)$ to increase the swap success rate.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (2020YFB1005403); National Natural Science Foundation of China (61972382); Natural Science Foundation of Inner Mongolia (2020MS06017); Key Research and Development Program of Hainan Province (ZDYF2021GXJS008).

[1]The matching platform in HTLC is different from the coordinators because this platform does not control the transfer operations on any blockchain.

References

[1] T. Nolan. (2013) Re: Alt chains and atomic transfers. [Online]. Available: https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949

[2] M. Herlihy, "Atomic cross-chain swaps," in *ACM PoDC 2018*, Egham, United Kingdom, Jul. 2018, pp. 245–254.

[3] B. W. Lampson and H. E. Sturgi, "Crash recovery in a distributed data storage system," Xerox Palo Alto Research Center, Palo Alto, CA, USA, Tech. Rep., Jun. 1979.

[4] R. Anderson, I. Ashlagi, D. Gamarnik, and Y. Kanoria, "A dynamic model of barter exchange," in *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms*, San Diego, CA, USA, Jan. 2015, pp. 1925–1933.

[5] R. M. Kaplan, "An improved algorithm for multi-way trading for exchange and barter," *Electronic Commerce Research and Applications*, vol. 10, no. 1, pp. 67–74, 2011.

[6] J. Xu, D. Ackerer, and A. Dubovitskaya, "A game-theoretic analysis of cross-chain atomic swaps with htlcs," in *2021 IEEE 41st International Conference on Distributed Computing Systems*, Washington DC, USA, Jul. 2021, pp. 584–594.

[7] Y. Xue and M. Herlihy, "Hedging against sore loser attacks in cross-chain transactions," in *ACM PoDC 2021*, Virtual Event, Italy, Jul. 2021, pp. 155–164.

[8] (2021) Decentralized identifiers (dids) v1.0. [Online]. Available: https://www.w3.org/TR/did-core/

[9] (2021) Solidity v0.5.0. [Online]. Available: https://docs.soliditylang.org/en/v0.5.0/

[10] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, London, United Kingdom, Nov. 2019, pp. 1521–1538.

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

[12] V. Zakhary, D. Agrawal, and A. E. Abbadi, "Atomic commitment across blockchains," *Proceedings of the VLDB Endowment*, vol. 13, no. 9, pp. 1319–1331, 2020.

[13] (2020) Atomicdex and atomic swaps. [Online]. Available: https://developers.komodoplatform.com/basic-docs/start-here/core-technology-discussions/atomicdex.html#introduction

[14] (2017) Decred: Atomic swaps. [Online]. Available: https://docs.decred.org/advanced/atomic-swap/

[15] R. Han, H. Lin, and J. Yu, "On the optionality and fairness of atomic swaps," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, Zurich, Switzerland, Oct. 2019.