Big data analytics in smart grids: a review

Research Article

Fadi Barbara, Enrico Guglielmino, Nadir Murru*, and Claudio Schifanella

BTLE: Atomic swaps with time-lock puzzles

https://doi.org/10.1515/jmc-2024-0044
received November 29, 2024; accepted February 12, 2025

Abstract: We present BTLE (Broadcast Time-Lock Exchange Protocol), a two-step protocol that aims to decentralize exchange of funds between two blockchains in scenarios similar to online exchanges. BTLE leverages time-lock puzzles to achieve that. In the first phase, the BTLE-MA protocol allows for a matching between a market maker and one of the competing market takers. In the second phase, the BTLE-AS algorithm allows the exchange between the market maker and the winning market taker. It is not necessary to use both the BTLE-MA and BTLE-AS algorithms in a decentralized-exchange scenario: existing atomic swaps based on hashed time-lock contract (HTLC) can benefit from BTLE-MA and can be adapted to an exchange where there are multiple possible participants. Moreover, BTLE computations are off-chain, so BTLE can be used in those blockchain pairs where at least one of the two does not have a scripting language or where the pair do not have the same hash function in common. This solves a limitation of HTLC-based atomic swaps. We also propose a new time-lock puzzle based on Pell conic calculations as an alternative to the classical time-lock puzzle of Rivest et al. BTLE has been implemented and tested. Experiments demonstrate that this new time-lock puzzle based on the Pell conic is superior for the intended goal. With an N-bit modulus of 2,000 bits, the RSW-TL approach resolves the puzzle in approximately 100 s, whereas our BM-TL method requires over 4,000 s, significantly reducing the number of squaring operations needed.

Keywords: time lock puzzle, blockchain, atomic swaps, decentralized exchange

MSC 2020: 11T71

1 Introduction

With the birth of the Internet and the beginning of message exchanges, the next problem was that of value exchange. The first projects that aimed to solve “remote exchange of value” were centralized since it was not possible to solve the problem of double spending in a decentralized way. Formally, it has been shown that it is possible to create a decentralized randomized Byzantine agreement based on the Bitcoin protocol [1].

Bitcoin was the first system capable of solving this problem in a decentralized manner. Its history can be traced through numerous academic articles, such as the ones on proof of work [2,3] and the one on backlinking to maintain data integrity [4], which all together give rise to the system we nowadays call blockchain.

Starting with Bitcoin, various projects have sprung up to solve the new problems that the Bitcoin project presented [5]. One of the first problems was that of scalability. In fact, a decentralized system such as Bitcoin had the ability to operate only 7 transactions per second (tx/s). For this reason, in 2015, a scaling proposal was based on the idea of operating exchanges on parallel blockchains, to balance the number of transactions on

*** Corresponding author: Nadir Murru,** Dipartimento di Matematica, Università di Trento, Via Sommarive 14, 38123, Povo (TN), Italy, e-mail: nadir.murru@unitn.it
Fadi Barbara: Dipartimento di Informatica, Università di Torino, Via Pessinetto, 12, 10149, Torino (TO), Italy, e-mail: fadi.barbara@unito.it
Enrico Guglielmino: Dipartimento di Scienze Matematiche, Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129, Torino (TO), Italy, e-mail: enrico.guglielmino@polito.it
Claudio Schifanella: Dipartimento di Informatica, Università di Torino, Via Pessinetto, 12, 10149, Torino (TO), Italy, e-mail: claudio.schifanella@unito.it

each blockchain [6]. This idea was further explored in the following years, e. g., in the liquid [7] and plasma projects [8].

A distinct project, Ethereum, was designed to facilitate distributed computation rather than merely transferring value. Unlike Bitcoin, which primarily serves as a decentralized digital currency, Ethereum utilizes a distributed ledger to tracks operations generated by programs on a shared virtual machine, maintaining different states. This platform was the first to implement the concept of smart contracts, an idea proposed by Szabo in 1997 [9].

With the emergence of numerous projects based on blockchain, another problem came to the surface: the problem of communication between these different distributed systems. Formally, given two blockchains, the problem consists in the ability to operate transactions between one blockchain and another. This problem is generally referred to as cross chain communication (CCC) [10].

One of the first works that systematizes the knowledge of this problem was that of Buterin in 2016 [11]. This report divided the methods of coin exchange into two categories: centralized and decentralized. Centralized methods were those in which participants sent funds to a third-party entity that was considered trusted, as it was the keeper of the private keys of said funds. These methods are sometimes also called custodial. Examples of these centralized entities are online exchanges. The advantages of the centralized methods are their ease of implementation, their ease of use and their speed. However, this comes at the expense of user security. In fact, the custodian can carry out attacks such as censorship or literally run away with the funds.

The second method of coin transfer, the decentralized one, allows participants to exchange coins without the use of a third party entity. In these cases the parties who want to exchange funds generate and solve cryptographic problems to lock or unlock the funds on either blockchain. A decentralized method, while safer for users, currently occurs at the expense of ease of use [12].

The difference between centralized and decentralized methods is, however, a distinction of convenience: some projects are a middle ground between the two methods. For example, some bridges use a group of people, usually called notaries, as a trusted third party. In this case, the notaries must reach a consensus before moving funds. This method is safer for the user as more parties need to be nonhonest for him to lose money. On the other hand, in an anonymous system, it is not possible to know if these parties are colluding so they act as a single entity. So we can see that choosing a centralized or decentralized method comes with different costs and different benefits [12].

The underlying problem with decentralized methods is demonstrated in the impossibility result of a fair exchange. This result discovered by Asokan et al. [13] states that in an asynchronous context, such as blockchain, it is impossible to have a secure and fair exchange without a trusted third party. The trusted third party operates a synchronization work between the two parties. To bypass this outcome, there are two methods. The first one is to make the trusted party more decentralized: this is the approach of the notaries, which cannot achieve complete decentralization because of the problems explained earlier. The second is to introduce synchronization methods that do not require a trusted party, leaving the method completely decentralized, or peer-to-peer. In this article, we will only deal with fully decentralized methods for exchanging funds.

The most used method today to achieve synchronicity between two blockchains is based on hashed time-lock contract (HTLC) [14,15]. We briefly explain its working here. A HTLC system uses four transactions between two parties (Alice and Bob) to exchange funds from a blockchain B_A to a blockchain B_B. Whether the participants are honest, only two of these transactions are published, one for each blockchain. In the case that both Alice and Bob are honest, the two transactions actually exchange funds to Bob and Alice, respectively. In the case where Alice or Bob (or both) are dishonest, however, the two transactions are structured in such a way that the parties can take back the transferred funds and lose nothing but time in the process.

While heavily used, this is not the only method to achieve synchronicity: another method is based on time-lock puzzles. The method proposed by Rivest et al. [16] aims to obtain time-bounded encryption: the goal is to obtain a method to make a message remain encrypted only for a predictable amount of time. A time-lock puzzle is also described as a “time capsule” for a message. In our case, the time-lock puzzle is used by Alice and Bob to reveal part of a private key to obtain the counterpart funds while maintaining security. For example, if Alice behaves dishonestly, Bob is able to redeem her funds, knowing that Alice cannot steal them before a certain time defined by Bob himself in the time-lock puzzle. Unfortunately, however, there are few systems

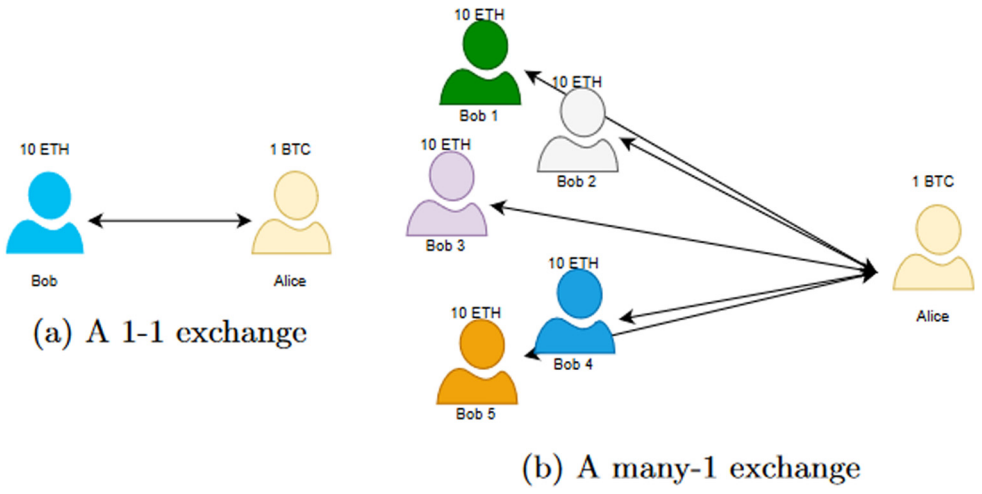

Figure 1: In a 1-1 exchange there are only two parties involved. In a many-1 exchange, multiple parties want to exchange the same amount of funds. In this case, a matching algorithm is used to choose Alice's partner. An example of a many-1 exchange are the online (crypto-)currency markets. (a) A 1-1 exchange, and (b) a many-1 exchange.

to implement a time-lock puzzle other than Rivest's method. In this article, starting from the preliminary work [17], we propose a new method to obtain time-lock puzzles starting from Rivest's method, laying the groundwork for further study to create new puzzles in different environments.

In addition to Asokan's impossibility result, as pointed out by Zamyatin et al. [10], most decentralized coin-transfer methods proposed so far in the literature involve only two parties. This works for a single exchange, but makes it difficult to implement the proposed method in a context similar to online exchanges. In fact, in the case of an online exchange, there are many participants who intend to operate the same exchange, at the same time and at the same rate while competing for first place. This is a many-1 setting, instead of a 1-1 as in the case of Alice and Bob explained earlier. A visual representation of the difference is shown in Figure 1. Another assumption generally made and implicit in the methods proposed in literature is that these parties already know each other before making the exchange. In reality, this is a very strong assumption and that makes it difficult to implement the proposed system retaining the same security guarantees. To face this problem, a centralization tweak is operated during the implementation where the algorithm of connection and knowledge of the two participants is operated on a central server. In the context of markets, this connection algorithm is called *matching algorithm*.

It is necessary to study methods that give the possibility to exchange funds in a multitude of contexts without requiring that the participants know each other beforehand, especially at a time when traditional financial methods are moving toward decentralized finance (DeFi). For these reasons, our solution is broadcasted, meaning it deal with multiple participants concurrently in a many-1 environment. Therefore, it is a Broadcast Time-Lock Exchange (BTLE).

In conclusion, our contributions are the following:

- We propose an offline and decentralized matching algorithm;
- We propose a decentralized exchange method (i.e., one that works without the help of a trusted third party) using the time-lock puzzle as a synchronization tool;
- We introduce a new type of time-lock puzzle based on the Pell conic, and we explain how to extend it to become a verifiable delay function (VDF);
- We compare the time-lock puzzle proposed by Rivest et al. and the one mentioned earlier;
- We propose an implementation of both the legacy and the new time lock puzzles[1];

1 https://github.com/disnocen/dex-tlp.

- We prove the security of the method in the hybrid ideal/real world simulation in the presence of static honest-but-curious adversaries;
- We conducted an analysis of the method in terms of time.

2 Related works

This section is divided into three subsections. In the first subsection, we will deal with the presentation of the proposed works in the literature related to the communication between two blockchains. In the second subsection, we will deal with the study of the presentation of results obtained in time-based mapping, while in the third subsection, we will introduce the works on the hash time lock puzzles. This last subsection is necessary to make a comparison between our proposed method and the latter method.

2.1 Cross chain communication

One of the first reports on the CCC issue is the report Buterin wrote in 2016 [11]. At that time, the cross-chain communication problem was also called the *interoperability* problem, hence the name in the report. In that report, Buterin classifies the main methods of interoperability present at the time. Generally, an interoperable system was leveraged to enhance the scalability of the blockchain in question. Here, scalability means the increase in terms of throughout (i.e., the number of transactions per second) correlated to an increase in the number of users. In fact, interoperability gives the possibility of rebalancing the loads between two blockchain and therefore split the work between the two. In this sense, we talk about sidechains, introduced for the first time in [6].

Another way to find interoperability is to build an interoperable blockchain by design. This is the case of the ecosystems dubbed “blockchain of blockchains.” Examples of these ecosystems are Cosmos [18] and Polkadot [19]. In both projects, the idea is to create a hierarchy of blockchains where each exchange of funds is approved through the blockchain at the head of all others.

For a more detailed analysis, see, the work of Zamyatin et al. [10].

2.2 Time release cryptography

Here, we briefly explain what is meant by time-based cryptography. For a general overview of this issue, see the detailed survey by Jaques et al. [20]. In this subsection, we present only the fundamental results related to the BTLE protocol.

Timed primitives came up in several contexts. In the following, we distinguish between a preblokchain phase and a postblockchain phase. The first generation, preblockchain, includes “time capsules” for key escrowing as presented by Bellare et al. [21], time-based cryptographic secrets as presented by Rivest et al. [16] and contract signing as in Boneh and Naor [22]. Of those, only the last two protocols are secure against parallel processing, i.e., they use what has been defined as *inherently secure* function [23].

After the introduction of Bitcoin [24], time based cryptography had a new wave of research. In particular, the postblockchain study of time-based cryptographic protocols is focused on VDFs and *time-lock puzzles* (TLPs). VDFs are a subset of TLPs: the difference is that in VDFs other people can verify the solution without the need to solve the puzzle [23].

The majority of the newer studies have focused on VDFs. Some of them proposed new protocols, such as [25], while others extended the TLP [16], making it a VDF. The works of the latter type are that of Wesolowski [26] and that of Pietrzak [27]. These works are compared in the study by Boneh et al. [28]. Interestingly, the new

wave of research on time-based cryptography has generally left aside the traditional time-lock puzzles: we are aware of only one paper on this topic, i.e., the time-lock puzzle in [29]. However, BTLE does not need any outside verification of the result. That is because our puzzles have an implicit verification: if the solution is right, then the user can retrieve the funds, otherwise it cannot.

2.3 Hash time-lock contracts

To date, the majority of peer-to-peer exchange methods are based on the concept of HTLC. The inventor of HTLCs is considered to be Nolan [14], and they are analyzed in many papers, such as the ones by Herlihy [15] and Miraz and Donald [30]. An HTLC is a contract that uses hash-based and time-based cryptographic constructs to lock and unlock funds on chain. Participants in this kind of contract have to manually redeem funds by generating cryptographic proofs of payment before a certain date to proceed with the protocol. The downside of this is that parties are required to stay online during the whole execution of the protocol. This is difficult for power constrained devices or in places where a stable Internet connectivity cannot be expected.

Another downside of HTLCs is that these cryptographic primitives cannot be implemented on all blockchains. To obtain the kind of lock required by the protocol a scripting language is necessary. In particular, both blockchain scripting languages need to have primitive to compute the same hash function. Many popular blockchain projects do not satisfy these requirements. For example it wouldnot be possible to make a HTLC between Tezos (which uses the Blake2b function as its principal hash function) and Bitcoin (which uses the SHA256 function to make hashing).

Finally, HTLCs can be instantiated only between two participants or entities: it is a 1-1 exchange. This makes it difficult to base a complete exchange algorithm in decentralized markets with a many-1 setting.

In some works, several possibilities have been proposed to overcome some of these limitations, using, for example, attribute verifiable timed commitments as a cryptographic primitive [31] or leveraging relays and adapters [32]. However, none of these approaches simultaneously addresses all the previous issues, and none of them utilizes time-lock puzzles as a cryptographic primitive, which we instead propose in our BTLE protocol, where for the first time a solution of this kind also involving time-lock puzzles is presented.

3 Background

3.1 Time-lock puzzle

First, we recall the classic time-lock puzzle developed by Rivest et al. (RSW-TL) [16], and we elaborate this idea for developing a novel time-lock puzzle based on the group's structure of conics, which will be called BM-TL, since it exploits the RSA-like cryptosystem studied in previous studies [33,34]. Both systems are not parallelizable because they use sequential functions. Thus, there is no advantage in having more CPUs, because all computations are necessarily done only on one core of the CPU. These time-lock puzzles can be considered as a CPU-bound puzzle with a timing function and an implicit verification [35].

Thanks to the sequential operations involved in these time-lock puzzles, it is possible to predict the time to solve them. Here, T is the time such that A wants to keep B busy, S is the number of squaring per unit of time (either done by the RSW-TL method or the BM-TL method), and TS is the number of squaring needed. Clearly, S strongly depends on the processor used.

The time-lock puzzle proposed by Rivest et al. [16] is simple yet effective and exploits repeated squarings. Usually, a time-lock puzzle is used to encrypt a secret sk (for instance, a key of a symmetric cryptosystem) so that it could be recovered only after a fixed amount of time T.

Let A be the entity that creates the time-lock puzzle, A encrypts sk by means of

$$c \equiv sk + a^{2^t} \pmod{n},$$

where $n = pq$, where p and q are prime numbers, $0 < a < n$ is a random number, and t is a positive number. If p and q are known, one can efficiently compute c, observing that 2^t can be reduced modulo $\varphi(n)$, i.e., $e \equiv 2^t \pmod{\varphi(n)}$, and then one just has to compute $a^e \pmod{n}$ with a great reduction of the repeated squares needed for obtaining c. The entity A sends (n, a, t, c) to the entity B that has to recover sk.

Since p and q are kept secret by A, the entity B has to perform t squarings, since the computation of a^{2^t} is believed not to be parallelizable. In fact, multiplication of "big" primes is the same trapdoor function used by the RSA cryptosystem.

The idea of Rivest et al. for creating time-lock puzzles can be easily adapted using different products for performing the powers. Given a field $\mathbb{F}$, the conic

$$C = \{(x, y) \in \mathbb{F} \times \mathbb{F} : x^2 - Dy^2 = 1\},$$

with D square-free, can be equipped with the product

$$(x_1, y_1) \otimes (x_2, y_2) = (x_1x_2 + y_1y_2D, x_1y_2 + x_2y_1),$$

so that $(C, \otimes)$ is an abelian group with identity (1,0) and the inverse of an element (x, y) is $(x, -y)$. Considering $x^2 - D$ an irreducible polynomial over $\mathbb{F}$, one can easily observe that C is isomorphic to the elements of norm 1` of the field $\mathbb{A} \coloneqq \mathbb{F}[X]/(x^2 - D)$. Then we can parametrize the conic by considering $P = \mathbb{A}^*/\mathbb{F}^* \cong \mathbb{F} \cup \{a\}$, where $a \notin \mathbb{F}$ is the point at the infinity. In this way, the induced product over P is given by

$$\begin{cases} a \odot b = \dfrac{D + ab}{a + b}, & \text{if } a + b \neq 0 \\ a \odot b = a, & \text{if } a + b = 0. \end{cases}$$

It is interesting to observe that the same parametrization can be obtained by using the line $y = \frac{1}{m}(x + 1)$.

Considering $\mathbb{F} = \mathbb{Z}_p$ and D not a quadratic residue modulo p, then C is a cyclic group of order $p + 1$, and thus,

$$a^{\odot p+1} \equiv 1 \pmod{p}$$

for every $a \in \mathbb{Z}_p$, where the powers are evaluated with respect to the product $\odot$. Moreover, we can also define the conic C with points whose coordinates belong to the ring $\mathbb{Z}_n$. In this case, we do not have a group structure; however, considering $P = \mathbb{Z}_n \cup a$, with $n = pq$, p and q primes, we have an analogue of the Euler's theorem:

$$a^{\odot \Psi(n)} \equiv 1 \pmod{n}, \quad \forall a \in \mathbb{Z}_n^*,$$

where $\Psi(n) = (p + 1)(q + 1)$ plays the role of the Euler totient function [33]. Now, we have all the tools for defining the BM-TL following the idea of Rivest et al. Indeed, the secret sk can be encrypted by

$$c \equiv sk + a^{\odot 2^t} \pmod{n}.$$

Knowing the factorization of n, one can efficiently compute $a^{\odot 2^t}$ evaluating first $e \equiv 2^t \pmod{\Psi(n)}$ and then $a^e \pmod{n}$. Without knowing the factorization of n, one must perform t squarings with respect to the product $\odot$.

3.2 VDF

In the following, we explain how it is possible to extend the timelock puzzles into a VDF. To do that we apply the method described by Wesolowski [26] to the BM-TL conic.

We say that a VDF implements a function $\mathcal{X} \to \mathcal{Y}$ if it is a tuple of three algorithms [36]:

- **Setup**$(\lambda, t) \to pp$ is a randomized algorithm that takes a security parameter λ and a time bound t, and outputs public parameters pp,
- **Eval**$(pp, x) \to (y, \pi)$ takes an input $x \in \mathcal{X}$ and outputs a $y \in \mathcal{Y}$ and a proof π.

• **Verify**$(pp, x, y, \pi) \rightarrow \{\text{accept}, \text{reject}\}$ outputs accept if y is the correct evaluation of the VDF on input x.

The study by Wesolowski [26] uses the **Setup** and **Eval** phases already described in the study by et al. [16] and Section 3. We are going to describe how to extend its **Verify** phase. There are two possible types of proofs, one interactive and one non-interactive. We start by explaining the interactive one. If Primes(λ) is the set of prime numbers before λ and $h = g^{2^t}$, its steps are as follows:

(1) The verifier sends to the prover a random prime ℓ sampled uniformly from Primes(λ),
(2) The prover computes $q, r \in \mathbb{Z}$ such that $2^t = q\ell + r$ with $0 \leq r < \ell$, and sends $\pi \leftarrow g^q$ to the verifier.
(3) The verifier computes $r \leftarrow 2^t \pmod{\ell}$ and outputs accept if $h = \pi^l g^r$

This conclude the interactive proof. The noninteractive proof uses the Fiat–Shamir heuristic in the random oracle model.

Since the creation and solution of $2^t \pmod{n}$ is the same for both the RSW-TL and BM-TL methods, the work by Wesolowski [26] naturally extends the BM-TLP to a BM-VDF.

3.3 Conditional transactions

The blockchains that have a scripting language allow users to create imperative (as in the case of Ethereum) or verifying (as in the case of Bitcoin) smart contracts. One application of this fact is the construction of conditional transactions, i.e., transactions that can be redeemed by different keys under certain conditions.

In the case of BTLE-AS, we use transactions that use time as a condition (measured in terms of blocks). In particular, we define a (t_1, t_2)-transaction or transaction with parameters (t_1, t_2), such that

• it can be redeemed by a key K_1 before time t_1,
• it can be redeemed by a key K_2 after time t_1 but before time t_2,
• it can be redeemed by a key K_3 after the time t_2.

Specifying how to create a conditional transaction is outside the scope of the article, but examples of this type of transaction can be found in the BIP16 [37] specification for Bitcoin, but are possible in any blockchain project that has a scripting language. We use this kind of transaction in BTLE-AS (Section 4.2.1) where, unless otherwise specified, the keys K_1 and K_3 are the same).

4 Design

In this section, we explain how it is possible to create both a matching algorithm and an atomic swap algorithm based on time-lock puzzles. Both algorithms are part of the BTLE and for simplicity, we call BTLE-MA as the matching algorithm and BTLE-AS as the swap algorithm. Without loss of generality, we can say that the participant who initiates the process is selling its token in exchange of (or *to buy*) the other.

We point out that it is not necessary to use both the BTLE-MA and the BTLE-AS algorithms to obtain a token swap. In particular, it is possible to add to the current swap methods based on HTLC a matching algorithm based on BTLE-MA. This facilitates the partner discovery steps in current atomic swap systems.

In the following, we assume a decentralized platform where participants want to swap tokens. Each participant owns a client that can track the progress of the reference blockchain. We informally call *view* the client's collection and ordering of states of the blockchain. This view is not necessarily the same for all participants due to network lag or possible attacks, such as an Eclipse attack that has the effect of giving a false view of the blockchain blocks. Here, "false" means that the victim's view is different from the view of the

majority of blockchain miners/validators. In the following, we will not focus on the consequences of this attack because it has no real consequences: if the victim has a different view of the blockchain, then it is very difficult for her to have transactions accepted by miners. The existence of different views explains why BTLE-MA is necessary: there cannot be a temporal-based ordering of possible buyers and an asynchronous systems imply the absence of a shared clocks.

We distinguish two classes of participants. The first is the class of *initiators*: this kind initiates the exchange by proposing the deal, i.e., the selling of its token. The initiator corresponds to a *market makers* in traditional centralized markets. The latter is the class of *exchangers*: to this class belong the possible buyers interested in an equivalent and opposite deal but that do not want to start it. These participants correspond to *market taker*.

We assume there is a single initiator which we denote by A (Alice), and many possible exchangers. Since the exchangers represent Alice's partner, following the cryptography tradition we will call them all "Bobs" and, supposing they are indexed and in finite number d, we denote them as $\{B_1, B_2, \ldots, B_d\}$ and we assume d secure channels of communication between A and each one of B_i, $i = 1 \ldots d$. We also assume $\{B_1, B_2, \ldots, B_d\}$ compete at the same price level, as in traditional order books. If not, then A narrows the view to the subset of Bobs with the most favorable exchange rate for A.

Note that the set $\{B_1, B_2, \ldots, B_d\}$ is completely determined by the view of A since we assume that A's view of the blockchain is the correct one. Consequently, there may be other potential exchangers or some have withdrawn and A's view of potential buyers is not synchronized yet. In the protocol description, we will see why neither of these two cases is a problem.

Finally, in the following, we treat the time-lock puzzle TLP as a blackbox, which takes two inputs and then outputs a cryptographic puzzle, thanks to how we modeled the time-lock puzzle primitive in Section 3. The solution of the puzzle is the cleartext itself. This shows that in theory, the system can use any kind of timelock puzzle, including those described in Section 3. In Section 6, we will see which one is better from a practical point of view.

4.1 BTLE-MA

In this round, Alice has to choose the exchanger among $\{B_1, B_2, \ldots, B_d\} \leftarrow GetExchangersList()$. Here, *GetExchangersList*() is a routine on the platform that returns the addresses of the exchangers and a way to communicate with them. How to implement it is outside the scope of this article.

BTLE-MA is explained in pseudo-code in Algorithm 1. As seen in the figure, A starts by generating a random message $rand_i$, $i = 1 \ldots d$ for each participant in $\{B_1, B_2, \ldots, B_d\}$. Then A associates this message to the intended receiver, $rand_i \leftrightarrow B_i$, $i = 1, \ldots, d$. The inputs of each time-lock puzzle are the message $rand_i$ and a time in seconds. Note that in a time-lock puzzle $TB_i = TLP(rand_i, time)$, the random message is different for each B_i, but *time* is equal for all participants: all potential exchangers must have the same chance of being able to find the solution at the same time. The randomness of the winner is determined by unpredictable factors such as network latency or the puzzle real solving time. In this sense, the choice of B_j is truly random. The output is a tuple that represent the cryptographic puzzle (Section 3). A performs this subroutine for all B_i, $i = 1, \ldots, d$, and then it sends all the puzzles. Finally, A waits for a solution.

When A receives the first solution (i.e., the cleartext of the random message) $\widehat{rand_j}$ from some B_j, A checks that it is a valid message. Practically, this means that A verifies that the cleartext $\widehat{rand_j}$ is equal to the message $rand_j$ associated with B_j. If that is the case, A accepts the solution and B_j is the winner: from now on, A will proceed to communicate only with B_j and discards all other solutions. If the message is not valid, A waits for another solution.

Algorithm 1 BTLE-MA routine

1: $\{B_1, B_2, \ldots, B_d\} \leftarrow GetExchangersList()$
2: **for** $i = 1, \ldots, d$ **do**
3: $rand_i \leftarrow RandGen()$
4: $map(rand_i \leftrightarrow B_i)$
5: $TB_i \leftarrow TLP(rand_i, time)$
6: **end for**
7: **for** $i = 1, \ldots, d$ **do**
8: $Send : TB_i \longrightarrow B_i$
9: **end for**
10: $AcceptedSolution$ = `False`
11: **while** $AcceptedSolution$ = `False` **do**
12: $waitSolution()$
13: **if** $VerifySolution(sol)$ == `True` **then**
14: $AcceptedSolution$ = `True`
15: $WinnerSolution = sol$
16: **end if**
17: **end while**
18: $Winner = map(WinnerSolution)$
19: return $Winner$

We stress the fact that the entirety of the BTLE-MA routine runs off-chain. Therefore, it is not costly nor slow.

4.2 BTLE-AS

Let Bob be the winner of BTLE-MA, and we denote it as B. This means that A will deal only with B, as in a classical token exchange. The goal of the parties in BTLE-AS is exchange their tokens in an atomic way.

In the following, we present the notation used to understand the protocol. At the beginning of BTLE-AS, A owns 1 `coin1`, and B owns 1 `coin2`[2]. To enforce the atomicity of the protocol, we consider the ending of the BTLE-AS protocol as "successful" only if one of this two possible endings occurs:

(1) A has 1 `coin2` B has 1 `coin1`,
(2) A has 1 `coin1` B has 1 `coin2`.

Case 1 means that both A and B were honest during the execution of the protocol, while Case 2 happens if either A or B has been dishonest or faulty.

Intuitively BTLE-AS aims to exchange private keys between A and B, so that they can move their newly acquired fund to other addresses. Of course, simply exchanging private keys would easily lead to possible theft: if A is not honest, A can still use his key to move his funds even if it sent the key to B, effectively stealing B's funds. Therefore, we split a private key into two parts and we use the homomorphism of both the sum and external product in the elliptic curve group. Formally, given two secret keys sk^A, sk^B, an elliptic curve generator g and the relative public key pk^A and pk^B, then the key sum is homomorphic:

$$(sk^A + sk^B)g = pk^A + pk^B = (sk^A g) + (sk^B g). \tag{1}$$

2 The real exchange rates between the two tokens and how they are decided are beyond the scope of this article. We assume this kind of information can be exchanged either in the channel during BTLE-MA or the UI implementation of the protocol.

Beside not sending the time-lock puzzle, A has another way to cheat B. In fact, A could create and send to B the time-lock puzzle of a random number instead of its private key sk_A. To avoid this problem, we introduce a zero-knowledge proof of the fact that the time-lock puzzle is hiding the right secret key, of course without revealing it.

In the following, we introduce the swap and the proof protocols.

4.2.1 The swap

We explain the exchange of tokens `coin1` and `coin2` with reference to Table 1, with notation in Table 2.

First, A and B create the key pairs sk_2^A, pk_2^A and sk_1^B, pk_1^B, respectively, where sk_i^j, pk_i^j are related to blockchain BC_i, $i = 1, 2$ and user $j = A, B$. These key pairs are, respectively, the first of the two shares of A and B to redeem the exchanged funds. After this step, A and B exchange the public keys pk_2^A and pk_1^B. If this first part is successful, both A and B create ephemeral keys (sk_1^A, pk_1^A and sk_2^B, pk_2^B respectively) that represent the second of the two shares to redeem the exchanged funds. At this point, A and B can create new public keys (called PK_1^B and PK_2^A respectively) to which they can send the funds. This transaction is a $(\frac{\lambda}{2}, t)$-conditional transaction as explained in Section 3.3, where t is a generic deadline the participants A and B can agree on or can be included in a specification. Because of the way the keys PK_1^B and PK_2^A and consequently the addresses are built, neither of the two participants can redeem the funds in either blockchains at this point of the exchange. For example, A needs to know sk_2^B to redeem coins in the address for PK_2^A. For this reason, in the second part of the exchange, A and B exchange the time-lock puzzles TLP_A and TLP_B and their respective proofs, explained in Section 4.2.2. Once the time-lock puzzles are opened/solved, A gets sk_2^B and B gets sk_1^A. By using λ as time unit, we observe that the second time-lock puzzle is sent later (by 1/4 of the time unit) with

Table 1: Protocol execution between Alice and Bob for a successful swap

Alice (`coin1`→`coin2`)		**Bob** (`coin2`→`coin1`)
$sk_2^A \xleftarrow{\$} [1, l_2\text{–}1]$, $pk_2^A = sk_2^A G_2$		$sk_1^B \xleftarrow{\$} [1, l_1\text{–}1]$, $pk_1^B = sk_1^B G_1$
$rm_A \leftarrow GenRandomMessage()$		$rm_B \leftarrow GenRandomMessage()$
	$pk_2^A, rm_A \rightarrow$	
	$pk_1^B, rm_B \leftarrow$	
$sk_1^A \xleftarrow{\$} [1, l_1\text{–}1]$, $pk_1^A = sk_1^A G_1$		$sk_2^B \xleftarrow{\$} [1, l_2\text{–}1]$, $pk_2^B = sk_2^B G_2$
$\text{PK}_1^B = pk_1^A + pk_1^B = (sk_1^A + sk_1^B)G_1$		$\text{PK}_2^A = pk_2^A + pk_2^B = (sk_2^A + sk_2^B)G_2$
$\text{hash}_{A\to B} \leftarrow \text{SendCondTx}(\frac{\lambda}{2}, t, \text{PK}_1^A \to \text{PK}_1^B)$		$\text{hash}_{B\to A} \leftarrow \text{SendCondTx}(\frac{\lambda}{2}, t, \text{PK}_2^B \to \text{PK}_2^A)$
		$T_B \leftarrow TLP_B(sk_2^B, \lambda)$
		$\pi_B \leftarrow \textbf{ProofGen}(sk_2^B, rm_A)$
	$T_B, \text{hash}_{B\to A}, \pi_B \leftarrow$	
$T_A \leftarrow TLP_A(sk_1^A, \frac{3}{4}\lambda)$		
$\pi_A \leftarrow \textbf{ProofGen}(sk_1^A, rm_B)$		
	$T_A, \text{hash}_{A\to B}, \pi_A \rightarrow$	
$pk_2^B = \text{PK}_2^A\text{–}pk_2^A$		$pk_1^A = \text{PK}_1^B\text{–}pk_1^B$
ProofV er(rm_A, pk_2^B, π_B)		**ProofV er**(rm_B, pk_1^A, π_A)
open T_B and redeem `coin2`		open T_A and redeem `coin1`

Table 2: Notation used in the explanation of the token exchange protocol

BC_1, BC_2	Blockchain 1 and 2 with tokens `coin1` and `coin2`
G_1,G_2	The base point for the elliptic curve of BC_1 and BC_2
l_1,l_2	The base point order for the elliptic curve of BC_1 and BC_2
PK_1^A	Public key for the address on blockchain BC_1, where A has the coins
PK_1^B	Public key for the address on blockchain BC_1, where B receives the new coins
PK_2^A	Public key for the address on blockchain BC_2, where A receives the new coins
PK_2^B	Public key for the address on blockchain BC_2, where B has the coins
$sk_{1,2}^A$, $pk_{1,2}^A$	Shares created by A for blockchain $BC_{1,2}$
$sk_{1,2}^B$, $pk_{1,2}^B$	Shares created by B for blockchain $BC_{1,2}$

a shorter opening time (i.e., 3/4 of the time unit). This ensures that both participants, A and B, open the puzzle at about the same time. As an example, consider the conditional transaction sent from Bob to Alice:

$$\mathrm{CondTx}\left(\frac{\lambda}{2}, t, \mathrm{PK}_2^B \to \mathrm{PK}_2^A\right).$$

This conditional transaction is redeemable under the following conditions:

(1) By the holder of the secret key associated with PK_2^B before time $\frac{\lambda}{2}$;

(2) By the holder of the secret key associated with PK_2^A after time $\frac{\lambda}{2}$ but before time t;

(3) By the holder of the secret key associated with PK_2^B after time t.

The mechanism ensures that the conditional transaction operates correctly under the following conditions:

- If Alice does not send her conditional transaction before time $\frac{\lambda}{4}$, Bob retains the ability to redeem his transaction until time $\frac{\lambda}{2}$, thereby safeguarding his funds.
- If Alice sends her conditional transaction before time $\frac{\lambda}{4}$, she is granted a designated window to redeem Bob's funds, specifically between time $\frac{\lambda}{2}$ and t. After time t, if Alice has not redeemed the funds, Bob is permitted to withdraw them, thereby ensuring a protective measure against potential delays or malicious intent on Alice's part.

These considerations are similarly applicable to the transaction initiated by Alice toward Bob.

4.2.2 The Proof

In the previous section, we saw how the exchange works, and we treated the proof as a blackbox. In this section, we explain in details how the proof works. Here, we explain how to do this assuming that the users will use one of the two time-lock puzzles described in Section 3. In particular, we assume that for each protocol there exist two functions **ProofGen**() and **ProofVer**() such that:

- **ProofGen**(sk, $rand$) is a deterministic algorithm that, given a secret key sk and a random message $rand$, creates a proof π.
- **ProofVer**(π, pk) is a deterministic algorithm that, given the public key pk with respect to sk and the proof π, outputs 1 if π is valid and 0 otherwise.

We start with the explanation of **ProofGen**. In case of the RSW-TL and the BM-TL time-lock puzzles, c is obtained from the formula,

$$c = sk + a^{2^t}, \quad c = sk + a^{\odot 2^t},$$

respectively, where sk is the message (the secret key) the user wants to encapsulate in the puzzle, a a random number and t is the number of squarings.

To generate the proof, we leverage the fact that digital signatures ensure unforgeability, meaning that only the holder of the private key can produce a valid signature for a given message. Unforgeability does not necessarily imply that no information about the private key can be leaked (unlike zero-knowledge proofs, which guarantee no leakage at all), as some schemes may tolerate partial leakage without compromising security. However, in our scenario, the unforgeability of the signature is sufficient to ensure the validity of the proof, as only the correct holder of the private key can generate a valid signature. Thus, let **SigGen**(m, s) be the signature routine of a digital signature scheme such that given a message m and a secret key s it outputs a signature σ, and let **SigVer**(m, σ, S) be the respective verification algorithm such that given a message m, a public key S and a signature σ outputs True if σ is the signature of message m, and False otherwise. Finally, let P a prover and V a verifier, similar to every zero-knowledge protocol. Then the protocol works in the following way, and the case of the RSW-TL can be seen in Algorithm 2.

Algorithm 2 ProofGen(sk, rm)

$c = sk + a^{2^t}$
$A \leftarrow$ **PubkeyGen**(a^{2^t})
$\sigma \leftarrow$ **SigGen**(rm, c)
$\pi = (\sigma, A)$
Send(π)

Assume there is an algorithm $X =$ **PubkeyGen**(x) that given a number x considers x as a ephemeral secret key and outputs the relative ephemeral public key X. When P creates the time-lock puzzle c, it also computes the public key $A =$ **PubkeyGen**(a^{2^t}) in the case of RSW-TL, or $A =$ **PubkeyGen**$(a^{\odot 2^t})$ in the case of BM-TL. Then P computes $\sigma =$ **Sig**(rm, c), a signature of the random message rm using the time-lock puzzle result c acting as ephemeral secret key. The proof is $\pi = (\sigma, A)$. This concludes the **ProofGen** routine.

Upon receiving the time-lock puzzle an the proof π, V starts the **ProofVer** routine as seen in Algorithm 3. V first checks that the proof works. To do that, V uses the public key $pk + A$ to verify that the message signed in σ is rm (recall that V already has pk as per the swap algorithm) using the **SigVer**$(\sigma, pk + A)$ routine. This works because $c = sk + a^{2^t}$, pk is the public key corresponding to sk and $A =$ **PubkeyGen**(a^{2^t}) by construction. Consequently, by using the property of homomorphism of the sum in elliptic curve cryptography, we have:

$$PubKey = pk + A = g \cdot sk + g \cdot a^{2^t} = g(sk + a^{2^t}) = g \cdot c.$$

If the verification checks, then V is sure P's time-lock puzzle is correctly built and then proceeds in solving it.

Algorithm 3 ProofVer(rm, pk, π)

$\sigma = \pi[0]$; $A = \pi[1]$
$Pubkey = pk + A$
$Bool \leftarrow$ **SigVer**$(rm, \sigma, Pubkey)$
if $Bool$ == `True`
    Accept π
else
    Reject π
end if

5 Security

The security of the proposed scheme relies on the difficulty of factoring large composite numbers, a well-known cryptographic assumption referred to as the *factoring assumption*. In this section, we provide a security proof in the hybrid ideal/real-world simulation model, which considers a *static honest-but-curious* adversary $\mathcal{A}$. In this context, *static* means that the adversary is given a fixed set of parties to control from the outset; honest parties remain honest, and corrupted parties remain corrupted throughout the protocol. The *honest-but-curious* (or semi-honest, or passive) adversarial model assumes that even corrupted parties correctly follow the protocol specifications without deviation. However, the adversary gains access to the internal state of all corrupted parties, including the transcript of all messages received during the protocol execution. The adversary then attempts to use this information to derive data that should remain private.

To formalize the problem, a two-party protocol is defined as a random process that maps pairs of inputs to pairs of outputs, one for each party. This process is referred to as a *functionality* and is represented as follows:

$$f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*,$$
$$(x, y) \mapsto (f_1(x, y), f_2(x, y))$$

The first party, with input x, aims to obtain $f_1(x, y)$, while the second party, with input y, aims to obtain $f_2(x, y)$.

We begin with the following notation:

- Let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality, and let π be a two-party protocol for computing f.
- The view of the i-th party ($i \in \{1, 2\}$) during an execution of π on input (x, y) and security parameter λ is denoted by $\mathsf{view}^i_\pi(x, y, \lambda)$.
- The output of the i-th party ($i \in \{1, 2\}$) during an execution of π on input (x, y) and security parameter λ is denoted by $\mathsf{output}^i_\pi(x, y, \lambda)$.
- The joint output of both parties is denoted by:

$$\mathsf{output}_\pi(x, y, \lambda) = (\mathsf{output}^1_\pi(x, y, \lambda), \mathsf{output}^2_\pi(x, y, \lambda)).$$

We now present the following definition from the study by Hazay and Lindell [38]:

Definition 5.1. (Security in the presence of honest-but-curious adversaries) Let $f = (f_1, f_2)$ be a *functionality*. We say that π securely computes f in the presence of static *honest-but-curious* adversaries if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that:

$$\{(\mathcal{S}(1^\lambda, x, f_1(x, y)), f(x, y))\}_{x,y,\lambda} \overset{c}{\equiv} \{(\mathsf{view}^1_\pi(x, y, \lambda), \mathsf{output}_\pi(x, y, \lambda))\}_{x,y,\lambda},$$
$$\{(\mathcal{S}(1^\lambda, y, f_2(x, y)), f(x, y))\}_{x,y,\lambda} \overset{c}{\equiv} \{(\mathsf{view}^2_\pi(x, y, \lambda), \mathsf{output}_\pi(x, y, \lambda))\}_{x,y,\lambda},$$

for all $x, y \in \{0,1\}^*$ such that $|x| = |y|$, and $\lambda \in \mathbb{N}$.

To prove security, we show that the ideal model, presented in Algorithm 4, can simulate the execution of the real-world protocol BTLE-AS. We refer to the ideal model as BM-AS-IDEAL, which operates with parties in the set $\mathcal{P} = \{A, B\}$. We restrict our analysis to scenarios with a single malicious party. We denote the corrupted party, controlled by the adversary $\mathcal{A}$, as P_I, and the honest party as P_J. Finally, *sid* denotes a unique session identifier, which ensures that each protocol execution is distinct, preventing replay attacks and preserving the integrity of the communication.

Algorithm 4 BM-AS-IDEAL

1: Upon receiving **Setup**$(sid, l_1, l_2, G_1, G_2)$ from some party $\mathcal{U} \in \mathcal{P}$, check that *sid* hasn't been previously used and

- Generate sk_i^P, pk_i^P, rm_P for $i = 1, 2$ and $p \in \mathcal{P}$
- Send $pk_2^A, rm_A, sk_1^B, pk_1^B, \text{PK}_2^A$ to B
- Send $pk_1^B, rm_B, sk_2^A, pk_2^A, \text{PK}_1^B$ to A

2: Upon receiving **Clean**$(sid, P \in \mathcal{P}, txhash, \lambda, t)$ from some party $\mathcal{U} \in \mathcal{P}$, if t has not passed and
3: **if** second time *sid* used and *txhash* is correct for party P **then**
4: Start *time*
5: Publish to the other party $\bar{P}$:

- $T_P \leftarrow TLP_P(sk_i^P, \lambda)$ with $i = 2$ if $P == B$, $i = 1$ otherwise
- $\pi_P \leftarrow$ **ProofGen**$(sk_i^P, rm_{\bar{P}})$
- *txhash*

6: **else**
7: **if** third time *sid* used and *txhash* is correct for party P and $time < \frac{\lambda}{4}$ **then**
8: Publish to the other party $\bar{P}$:

- $T_P \leftarrow TLP_P(sk_i^P, \frac{3}{4}\lambda)$ with $i = 2$ if $P = =B$, $i = 1$ otherwise
- $\pi_P \leftarrow$ **ProofGen**$(sk_i^P, rm_{\bar{P}})$
- *txhash*

9: **end if**
10: **end if**

In Algorithm 4, the functionality f can be interpreted as a comprehensive set of operations performed by the trusted third party, which includes:

- The secure generation and distribution of public–private key pairs.
- The verification and validation of the transaction hash (*txhash*).
- The secure generation and distribution of time lock puzzles TLP_P along with their corresponding proofs π_P.

Moreover, to ensure proper synchronization of timing-related actions, especially in the context of cross-chain interactions, we introduce a *global clock model*. This model is specifically tailored to provide a unified, blockchain-native time reference for participants operating across different blockchains. Finally, for the purposes of this analysis, we assume that the devices used by the parties are not compromised and securely manage all cryptographic operations, such as storing private keys and generating random values.

5.1 Global clock model

In the context of cross-chain atomic swaps between Blockchain_A and Blockchain_B, we address the need for a unified time reference for both participants, Alice and Bob, by proposing a *global clock model*. This model operates based on block counts rather than real-world time, providing a blockchain-native mechanism to synchronize actions across chains. Alice and Bob determine that a specified time interval has passed by observing the insertion of a predefined number of blocks.

Let:

- T_A denote the average block insertion time on Blockchain_A,
- T_B denote the average block insertion time on Blockchain_B.

Since the global clock is based on the slower of the two blockchains, we assume, without loss of generality, that Blockchain_A is the slower blockchain. Consequently, the global clock operates using Blockchain_A as its reference.

At the start of the swap process, let h_k^A and h_l^B represent the block heights of Blockchain_A and Blockchain_B, respectively. We define the key time parameters as follows:

- A transition of λ units of time corresponds to the insertion of the block at height $h_{k+4\lambda}^A$.
- A transition of t units of time corresponds to the insertion of the block at height h_{k+4t}^A.

These definitions use a coefficient of 4 to enable Alice and Bob to check for intermediate time milestones, specifically at $\frac{\lambda}{4}$ and $\frac{3\lambda}{4}$. This ensures that even if λ is not divisible by 4, there is a corresponding block insertion for each fractional milestone, allowing precise synchronization.

Although each blockchain maintains its independent block count, the global clock increments in sync with the block count of Blockchain_A. This ensures that both Alice and Bob remain synchronized in their actions throughout the atomic swap process.

To implement this global clock model effectively:

- Both Alice and Bob must have active addresses on both Blockchain_A and Blockchain_B.
- Each participant must be capable of monitoring block insertions on both blockchains, either through node connections or reliable notifications.

Thus, for reliable synchronization, we also assume that the faster blockchain (Blockchain_B) is notified each time a new block is added to Blockchain_A. This ensures a decentralized, blockchain-native timing system tailored for this specific application.

5.2 Proof of security

We are now ready to proceed with the proof of the following theorem:

Theorem 5.1. (Security of the BM-AS protocol) *Let the functionality* BM-AS-IDEAL *and protocol BM-AS be as defined in* Definition 5.1. *The BM-AS protocol securely computes the BM-AS-IDEAL in the presence of one honest-but-curious adversary.*

Proof. Let $\mathcal{A}$ be a static honest-but-curious adversary. We construct a simulator $\mathcal{S}$ who invokes $\mathcal{A}$ internally to simulate the execution of the real protocol, while interacting with BM-AS-IDEAL in the ideal world.

We assume that Bob (P_I) is the corrupted party controlled by $\mathcal{A}$, while Alice (P_J) is the honest one (the case where Alice is corrupted and Bob is honest can be handled analogously). This assumption is reflected in the simulation steps below, where $\mathcal{S}$ replicates the actions of $\mathcal{A}$ in the real protocol.

The simulator $\mathcal{S}$:

(1) $\mathcal{S}$ sends **Setup**$(sid, l_1, l_2, G_1, G_2)$ to BM-AS-IDEAL and receives the keys and random message.

(2) $\mathcal{S}$ simulates the sending of the conditional transaction with parameters $(\frac{\lambda}{2}, t)$ and publishes the output $txhash_I$ to P_J.

(3) $\mathcal{S}$ simulates TLP_I and $\textbf{ProofGen}_I$ and publishes the output to P_J.

(4) $\mathcal{S}$ monitors the chain until the following occurs:

   (a) P_J sends $txhash_J$, TLP_J, and $\textbf{ProofGen}_J$ before time $\frac{\lambda}{4}$:
   P_I opens TLP_J and redeems the money from the transaction with hash $txhash_J$.

From the standpoint of $\mathcal{A}$, the simulated and real executions are identical:

- Step 2 in simulation is parallel to Step 7, 10 of BM-AS protocol.
- Step 3 in simulation is parallel to Step 8, 9, 10 of BM-AS protocol.
- The creation of the conditional transaction is the same.
- Step 4(a) in the simulation is parallel to the Step 13, 16 in BM-AS.

$\mathcal{S}$ measures the times λ and t using the global clock model presented in 5.1. Thus, we have proved that for any adversary $\mathcal{A}$ in the real model, there exists a simulator $\mathcal{S}$ in the ideal model that can effectively simulate the execution of the real protocol, ensuring that both the adversary's view and the joint outputs of the honest and corrupted parties in the real-world execution are computationally indistinguishable from those in the ideal-world execution. Consequently, the BM-AS protocol securely computes the BM-AS-IDEAL functionality in the presence of one honest-but-curious adversary, as stated in the theorem. □

Remark on Sybil attacks: A potential challenge for the proposed protocol is its susceptibility to Sybil attacks, where malicious actors create multiple fake identities to disrupt the matching process or exploit time-lock puzzles without completing exchanges. However, we assume that adversaries are rational and aim to maximize their gains. In this context, Sybil attacks are inefficient, as splitting computational resources across multiple identities reduces the ability to solve time-lock puzzles effectively, thereby lowering the adversary's potential reward. While this work does not include specific mechanisms to resist Sybil attacks, exploring such protections could be an interesting direction for future research.

Remark on postquantum primitives: While it is well known that the factoring assumption is susceptible to quantum attacks via Shor's algorithm, this assumption remains justified by its current practical security within blockchain interoperability. Despite the rapid advancements in quantum computing, the deployment of post-quantum cryptographic schemes is still an ongoing process, and many standardized blockchain infrastructures continue to rely on prequantum assumptions, particularly elliptic curve cryptography (ECC) for key generation and digital signatures. Since ECC is not quantum resistant, existing blockchain systems will require substantial upgrades to maintain security in a postquantum era. Quantum threats extend beyond factoring-based schemes, as highlighted by Fernandez-Carames and Fraga-Lamas [39], which discusses the vulnerabilities of both public-key cryptography and hash functions against quantum attacks, including those based on Shor's algorithm. This underscores the need for quantum-resistant cryptographic primitives in blockchain ecosystems.

Finally, an interesting direction for future research would be to extend the current security proof to the malicious adversary model, addressing potential deviations from the protocol and demonstrating robustness against actively adversarial behaviors.

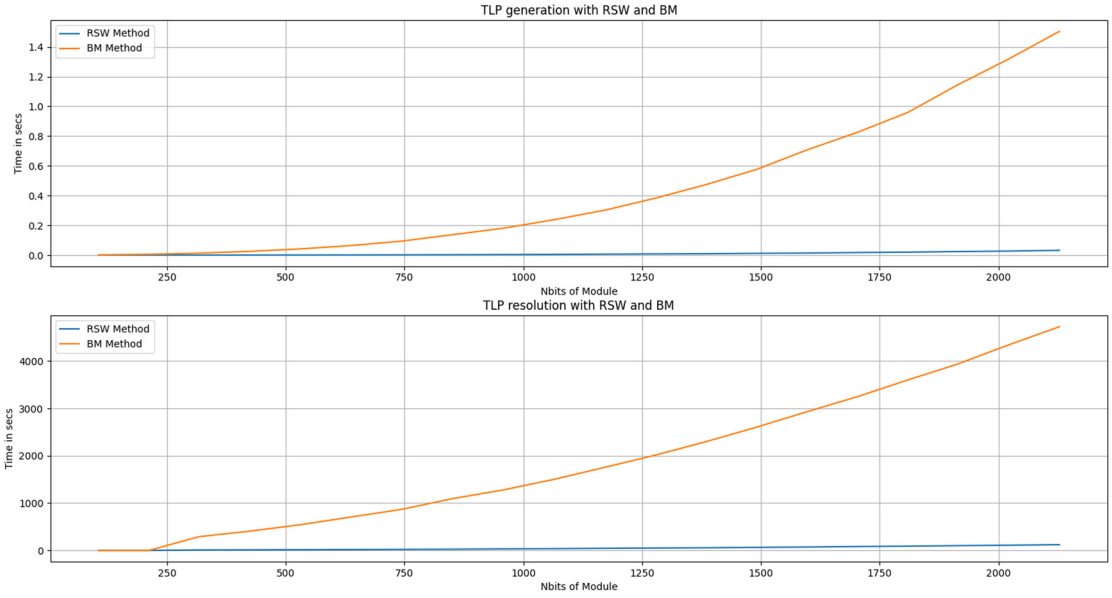

Figure 2: Comparison of generation (above) and resolution (below) times of the time-lock puzzles in the RSW-TL and BM-TL cases. As seen in figure, for a scenario like the one presented in this article, on a practical level, only BM-TL can be used.

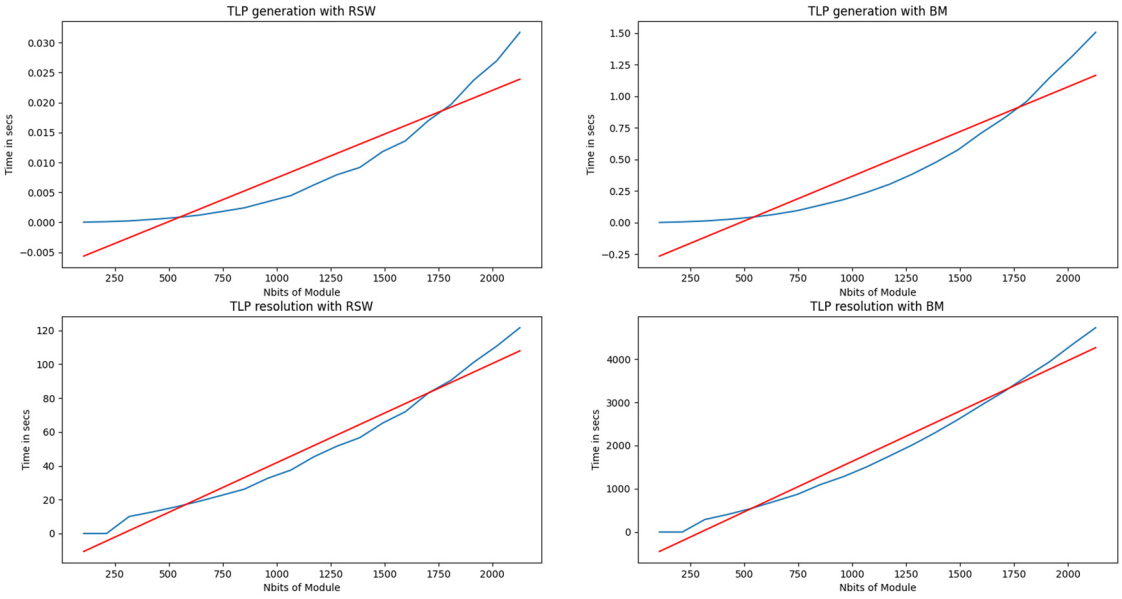

Figure 3: Comparison of generation (above) and resolution (below) times of the RSW (left) and BM (right) time-lock puzzles. It is clear that the growth in terms of seconds with respect to the magnitude in bits of the modulo N is not linear. We also note that the times in the BM-TL case are about 30 times greater than the times in RSW-TL.

6 Discussion

In this section, we analyze the exchange protocol considering three points of view. The first is that of security, then we will look at the protocol from a fee point of view, and finally, we will look at the speed of the protocols. We conclude that the best time-lock puzzle protocol for this purpose is the time-lock puzzle based on the Bellini-Murru protocol (BM-TL).

Both the time-lock puzzles are based on already analyzed methods. In particular, both the RSW and BM method derive their security from the difficult problem of number factorization.

BTLE is competitive with the classic atomic swap in terms of fees. This is because no on-chain operations are required: the exchange of random messages in BTLE-MA, the exchange of private key shares in BTLE-AS and the computation needed for the generation an resolution of the time-lock puzzles are done off-chain. The only transaction per blockchain is the transaction that participants make to send funds to another address. This is an advantage over the number of transactions required in an HTLC.

Another advantage of BTLE over HTLC is that the time-lock puzzle system is more flexible. While in an HTLC, it is necessary to decide the waiting time between transactions as a function of the timing of the creation of blocks in both blockchains, in BTLE, the timing can be shorter since BTLE does not have to do with blocks (there are no transactions sent other than the funding transaction if both parties are honest), but the only delays are related to the latency of the internet network, which are obviously much shorter.

We conclude this section with advice on choosing a time-lock puzzles between RSW-TL and BM-TL based on the tests we conducted. As seen in the comparison in Figure 2, using a squaring number $t = 10^7$ the BM-TL takes much longer than the RSW-TL method. In particular, it can be seen how with a number N such that its length in bits is about 2,000 bits, the resolution time of the time-lock puzzle is $\tilde{1}00$ s in the case of RSW-TL, while it exceeds 4,000 s (just over an hour) in the BM-TL case. Figure 3 highlights this result. Furthermore, the growth in terms of seconds in both generation and resolution of the time-lock puzzle is not linear in terms of the number of bits in the modulo N.

This makes it more practical to use BM-TL as fewer total squarings are required. If participants want to wait less than an hour, they can use a N module of less than 2,000 bits or decrease t. Test results can be found

in the Github repository.[3] There, it is possible to find the results of the tests made with both BM-TL and RSW-TL, on a core of different machines (Apple M1, AMD Series 7, Intel Xeon Skylake IBRS).

7 Conclusions

In this article we presented BTLE, a two-phase protocol for obtaining atomic swaps between two blockchains based on time-lock puzzles instead of HTLC, which (differently from HTLC) can be used in both a 1-1 swap and a many-1 swap as found in online exchanges. In this article, we have proved the feasibility of the protocol and we proposed an implementation of all the functions of BTLE that can be found online[4]. In particular, time-lock puzzle constructions for both the RSW-TL and the BM-TL are presented and implemented. Furthermore, we tested BTLE using the two types of time-lock puzzles, and we published the results. Our protocol is subject to quantum attacks. To address these challenges, future adaptations of our protocol could explore alternative time-lock constructions based on quantum-safe assumptions, such as lattice-based cryptography or hash-based time-lock puzzles. This research direction remains crucial, especially as postquantum cryptographic standards see increasing adoption and gradual integration into blockchain architectures.

Acknowledgement: The authors thank the referee for her/his suggestions to improve this work. This work was partially presented at CIFRIS24, the Congress of the association De Cifris www.decifris.it/cifris24.

Funding information: None declared.

Author contributions: All authors have accepted responsibility for the entire content of this manuscript and approved its submission.

Conflict of interest: The authors declare no competing interests.

Ethical approval: The conducted research is not related to either human or animals use.

Data availability statement: Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

References

[1] Garay JA, Kiayias A, Leonardos N. The Bitcoin backbone protocol: analysis and applications. In: Oswald E, Fischlin M, editors. Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26–30, 2015, Proceedings, Part II. vol. 9057 of Lecture Notes in Computer Science. Springer; 2015. p. 281–310. doi: https://doi.org/10.1007/978-3-662-46803-6_10.

[2] Dwork C, Naor M. Pricing via processing or combatting junk mail. In: Brickell EF, editor. Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings. vol. 740 of Lecture Notes in Computer Science. Springer; 1992. p. 139–47. doi: https://doi.org/10.1007/3-540-48071-4_10.

[3] Back A. Hashcash - A denial of service counter-measure; 1992. p. 10.

[4] Haber S, Stornetta WS. How to time-stamp a digital document. J Cryptol. 1991;3(2):99–111. doi: https://doi.org/10.1007/BF00196791.

[5] Poelstra A. On Stake and Consensus; 2015. https://www.semanticscholar.org/paper/On-Stake-and-Consensus-Poelstra/5fc38a9b1301cd65604e67e1a424a1241f66cecf.

3 See files whose name starts with generation − values − from − test at `https://github.com/disnocen/dex-tlp`.

4 https://github.com/disnocen/dex-tlp.

[6] Back A, Corallo M, Dashjr L, Friedenbach M, Maxwell G, Miller A, et al. Enabling blockchain innovations with pegged sidechains. Whitepaper. 2014:25.
[7] Dilley J, Poelstra A, Wilkins J, Piekarska M, Gorlick B, Friedenbach M. Strong federations: An interoperable blockchain solution to centralized third-party risks. arXiv:161205491 [cs]. 2017 Jan.
[8] Poon J, Buterin V. Plasma: Scalable autonomous smart contracts. White paper. 2017.
[9] Szabo N. Formalizing and securing relationships on public networks. First Monday. 1997;2(9). doi: 10.5210/fm.v2i9.548.
[10] Zamyatin A, Al-Bassam M, Zindros D, Kokoris-Kogias E, Moreno-Sanchez P, Kiayias A, et al. SoK: Communication across distributed ledgers. ePrint IACR. 2018:17.
[11] Buterin V. Chain interoperability. R3 Research Paper. 2016.
[12] Qin K, Zhou L, Afonin Y, Lazzaretti L, Gervais A. CeFi vs. DeFi - Comparing Centralized to Decentralized Finance. CoRR; 2021. https://arxiv.org/abs/2106.08157.
[13] Asokan N, Shoup V, Waidner M. Asynchronous protocols for optimistic fair exchange. In: Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3–6, 1998, Proceedings. organization. IEEE Computer Society; 1998. p. 86–99. doi: https://doi.org/10.1109/SECPRI.1998.674826.
[14] Nolan T. Alt chains and atomic transfers. 2013. https://archive.ph/wWzna.
[15] Herlihy M. Atomic cross-chain swaps. In: Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing. Egham United Kingdom: ACM; 2018. p. 245–54.
[16] Rivest RL, Shamir A, Wagner DA. Time-lock puzzles and timed-release crypto; 1996.
[17] Barbara F, Murru N, Schifanella C. Towards a broadcast time-lock based token exchange protocol. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 13098. Springer; 2022. p. 243–54.
[18] Kwon J, Buchman E. Cosmos Whitepaper; 2016. https://cosmos.network/cosmos-whitepaper.pdf.
[19] Wood DG. POLKADOT: Vision for a heterogeneous multi-chain framework. Whitepaper. 2016.
[20] Jaques S, Montgomery H, Roy A. Time-release cryptography from minimal circuit assumptions; 2020. 755.
[21] Bellare M, Goldwasser S. Encapsulated key escrow. Massachusetts Institute of Technology. Series/Report no. MIT-LCS-TR-688; 1996.
[22] Boneh D, Naor M. Timed commitments. In: Goos G, Hartmanis J, van Leeuwen J, Bellare M, editors. Advances in Cryptology - CRYPTO 2000. vol. 1880. Berlin, Heidelberg: Springer Berlin Heidelberg; 2000. p. 236–54.
[23] Boneh D, Bonneau J, Bünz B, Fisch B. Verifiable delay functions. In: Shacham H, Boldyreva A, editors. Advances in Cryptology - CRYPTO 2018. vol. 10991. Cham: Springer International Publishing; 2018. p. 757–88.
[24] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. Whitepaper. 2008:9.
[25] Malavolta G, Thyagarajan SAK. Homomorphic time-lock puzzles and applications. In: Advances in Cryptology - CRYPTO 2019. vol. 11692. Cham: Springer International Publishing; 2019. p. 620–49.
[26] Wesolowski B. Efficient verifiable delay functions. J Cryptol. 2020;33(4):2113–47. doi: https://doi.org/10.1007/s00145-020-09364-x.
[27] Pietrzak K. Simple verifiable delay functions. In: 10th Innovations in Theoretical Computer Science Conference (ITCS 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik; 2018.
[28] Boneh D, Bonneau J, Bünz B, Fisch B. Verifiable delay functions. In: Shacham H, Boldyreva A, editors. Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I. vol. 10991 of Lecture Notes in Computer Science. Springer; 2018. p. 757–88.
[29] Liu J, Jager T, Kakvi SA, Warinschi B. How to build time-lock encryption. Des Codes Cryptogr. 2018;86(11):2549–86. doi: https://doi.org/10.1007/s10623-018-0461-x.
[30] Miraz MH, Donald DC. Atomic cross-chain swaps: Development, trajectory and potential of non-monetary digital token swap facilities. Ann Emerg Tech Comput. 2019 Jan;3(1):42–50.
[31] Manevich Y, Akavia A. Cross chain atomic swaps in the absence of time via attribute verifiable timed commitments. In: IEEE 7th European Symposium on Security and Privacy; 2022. p. 606–26.
[32] Lys L, Micoulet A, Potop-Butucaru M. R-SWAP: Relay based atomic cross-chain swap protocol. In: Proceedings of the 6th International Symposium on Algorithmic Aspects Cloud Computer. 2021. p. 18–37.
[33] Bellini E, Murru N. An efficient and secure RSA-like cryptosystem exploiting Rédei rational functions over conics. Finite Fields Appl. 2016 May;39:179–94.
[34] Bellini E, Murru N, Scala AJD, Elia M. Group law on affine conics and applications to cryptography. Comput Appl Math. 2021;409:125537.
[35] Ali IM, Caprolu M, Di Pietro R. Foundations, properties, and security applications of puzzles: a survey. 2020 Apr. arXiv:190410164 [cs].
[36] Boneh D, Bünz M, Di Pietro R. A survey of two verifiable delay functions. Technical Report; 2018. p. 712. http://eprint.iacr.org/2018/712.
[37] Andresen G. BIP-16: Pay to Script Hash; 2013. Github. https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki.
[38] Hazay C, Lindell Y. Efficient secure two-party protocols - techniques and constructions. Information Security and Cryptography. Springer; 2010. doi: https://doi.org/10.1007/978-3-642-14303-8.
[39] Fernandez-Carames TM, Fraga-Lamas P. Towards post-quantum blockchain: A review on blockchain cryptography resistant to quantum computing attacks. IEEE Access. 2020;8:21091–116.