BxTB: cross-chain exchanges of bitcoins for all Bitcoin wrapped tokens

Fadi Barbàra, Claudio Schifanella
Department of Computer Science
University of Turin
Turin, Italy
E-mail: fadi.barbara@unito.it, claudio.schifanella@unito.it

2022 Fourth International Conference on Blockchain Computing and Applications (BCCA) | 978-1-6654-9958-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/BCCA55292.2022.9922019

***Abstract*—While it is possible to exchange tokens whose smart contracts are on the same blockchain, cross-exchanging bitcoins for a Bitcoin wrapped token is still cumbersome. In particular, current methods of exchange are still custodial and perform privacy-threatening controls on the users in order to operate. To solve this problem we present BxTB: cross-chain exchanges of bitcoins for any Bitcoin wrapped tokens. BxTB lets users achieve that by bypassing the mint-and-burn paradigm of current wrapped tokens and cross-exchanging already minted tokens in a P2P way. Instead of relaying on HTLCs and the overhead of communication and slowness due to time-locks, we leverage Stateless SPVs, i.e. proof-of-inclusion of transactions in the Bitcoin chain validated through a smart contract deployed on the other blockchain. Furthermore, since this primitive has not been introduced in the academic literature yet, we formally introduce it and we prove its security.**

***Index Terms*—Bitcoin, wrapped token, cross-chain, stateless SPV, privacy**

I. Introduction

Motivation: Today it is possible to use tokens on the Ethereum blockchain and perform swaps between them: automated decentralized exchange methods are increasingly efficient.

This does not happen in the Bitcoin blockchain: a user who only has bitcoins must either exchange bitcoins manually for ethers (e.g. via Bisq) or use a centralized service (custodial and generally threatening to the user's privacy) to tokenize their bitcoins on the Ethereum blockchain, for example using the wBTC token (see Section II for a more detailed analysis of the current situation). In fact, there are currently no cross-chain *automated* decentralized markets: the existing bridges are mostly between EVM-compatible chains or require manual intervetion during all phases of the cross-exchange.

Goal: Throughout this paper, we present Bitcoin-Cross-Tokenized-Bitcoin, or *BxTB* (pronounced *B-cross-TB*), a framework to overcome this problem. Using BxTB a user A can exchange its bitcoins with a user B in exchange for a Bitcoin wrapped token. B in this way can come into possession of "real" bitcoins without interacting with the custodian of the service.

Although, in theory, the method we present can be used to exchange any token for bitcoin, in cases other than a wrapped-token the protocol must take into account several different assumptions. For example, users A and B must decide an exchange-rate between token and bitcoin, modifying the protocol and requiring an introductory step. A hint of how this can be achieved is presented in the Future Works (Section VI) of this paper.

Contribution: our contribution can be summarized in the following list:

- we present a way to exchange wrapped token without using a mint strategy and consequently bypass privacy-threatening controls using Stateless SPVs
- we give a formal explanation of the primitive Stateless SPV: to date the primitive is informally described and there is no formal discussion on the subject
- we give a proof of the security of the primitive, which can be of independent interest.

Organization: The paper is so organized. The Related Works section in Section II gives a brief summary of the current situation regarding the possibility of swapping coins, with a particular focus on the exchange between the Bitcoin and EVM-compatible blockchains. In Section III we give a brief presentation of the stateful SPVs used by light clients and present more details on how *Stateless* SPVs work. In Section IV we present how the BxTB works: we will delineate how the exchange between participants works and how to prove the validity of stateless SPV proofs. In Section V we analyze the BxTB protocol: in particular, we will prove in Theorem 2 that it is not economically viable to forge the proof if it is based on enough blocks, similar to how normally a transaction is considered "confirmed" in a proof of work blockchain. We also explain the advadntages between BxTB and HTLC, commonly used in atomic swaps. In Section VI we describe how we plan to continue the development of BxTB. Then we conclude.

II. Related Works

Recall the impossibility result proved by Pagnia [1] regarding the possibility of fair exchange, i.e an exchange such that no party can cheat the other, as defined in [2]:

Theorem 1 (Fair-Exchange Impossibility)**.** *It is not possible to have a fair exchange that is tolerant of malicious nodes without a trusted third party in an asynchronous system.*

Zamyatin et al. [3] proved that cross-exchange swaps are equivalent to a fair-exchange. Weakening any of the assumptions of Theorem 1 will create a workaround and permit a

978-1-6654-9958-3/22/$31.00 ©2022 IEEE

fair exchange: One way is to use third parties (as in the case of wrapped tokens). The other is to tackle the asynchronicity assumption via cryptographic methods maintaining decentralization.

In the following we give more details on the methods, which are the base of the proposal of this paper.

A. Wrapped Tokens

Wrapped Tokens generally follow a *deposit-mint-withdraw* approach. As the name suggests, in this paradigm a user A deposit some coins (tokens) on a specified address on blockchain BC_a and then, thanks to cryptographic methods, a corresponding token is created (*minted*) on blockchain BC_b[1]. The user can exchange the token in blockchain BC_b, or withdraw it. To perform this last operation, the user A *burns*, i.e. destroys, the token on blockchain BC_b and redeems it on BC_a. Generally this burning operation is performed by a service which is generally custodial. In this setting, the service can be considered a trusted third party that users trust not to steal their coins (resp. tokens) or censor transactions. Given the financial operation underlining the service, these custodial services are generally forced to perform checks on the users, such as AML/KYC, before letting them "move" the coins (resp. tokens). We will see when this requirement is not needed when BxTB is used to exchange tokens.

Since we are interested in exchanges between bitcoins on the Bitcon blockchain and their wrapped-token version on other Turing complete chains, we discuss some of the most used (in volume) tokens representing bitcoins. For a comparison and a more detailed explanation see [4].

1) wBTC: Wrapped Bitcoin [5] is most used bitcoin-related token. It was originally deployed on the Ethereum network[2], but it is also instantiated on other blockchains like Tron. Born in 2018 from a partnership between Kyber network, BitGo, and Republic Protocol, the goal of this token is to reflect the value of bitcoins in a 1:1 pegged way using a backed-method, i.e. every token represent a real coin which is held into custody. In the following we introduce details about both the protocol and the actors involved for it to properly function.

Custodian: In this project BitGo manages the custody of the received bitcoins from the users. In this sense, the protocol Wrapped Bitcoin is a custodial service. Since it is custodial, BitGo has to provide the necessary transparency to perform auditing procedures on the collaterized assets.

Merchant: The merchant is the one who receives the bitcoins from the users and then, thanks to the protocol, creates a token on the Ethereum network. The merchant is a intermediary between the user and the custodian. It is also involved in the reverse process, i.e. in destroying the token in the Ethereum blockchain and send the bitcoins to the user on the Bitcoin blockchain.

User: The user is the one who sends or withdraw the bitcoins to the merchant.

The Protocol: At first, the user starts the request to the merchant and if successful the user sends bitcoin to the merchant. The merchant then sends those bitcoins to the custodian. The merchant then creates the token on the Ethereum blockchain. operations on the contract are managed via the "WBTC DAO", a decentralized autonomous organization.

If the user wants to withdraw its own bitcoins, the user asks the merchant to destroy the token on the Ethereum blockchain. Finally the merchant transfers the bitcoins from the custodial to the user on the Bitcoin blockchain.

Analysis: As can be seen from descriptions of the protocol, the protocol is easy to use from the perspective of the user and financially secure in the sense that one wrapped bitcoin will always be redeemable with one bitcoin and vice versa. In this sense, the user will not lose financial power (in bitcoin terms) by using this service. On the other hand, the merchant and the custodians are two third parties which have to be trusted. In fact the merchant could in theory refuse to send money to the custodian. This is mitigated by the fact that the merchant is registered in the DAO and therefore risks legal persecution if it behaves dishonestly. The other third-party, the custodian, is effectively managed from a multisignature. While better than a single point of failure, even if the central authority is shared between multiple parties, they can still collude and steal the funds of the users or censor any redeeming. Finally, this service needs to perform AML/KYC checks on the users, unless exchanged via a decentralized exchange or using the solution provided in this paper.

2) Other: There are other wrapping systems. One is called *renBTC*. The main difference with wBTC is that the custodian is not a single third party but a collection of nodes that run on an external VM, called *DarkNodes* [6]. The nodes are responsible to manage the custody of the received bitcoins from the users, similar to the case of wBTC.

Other methods trade custody (and therefore token-backing) for algorithms. For example, sBTC [7] is a protocol that can be considered non-custodial since generation is handled through a smart contract and project managers do not receive bitcoins directly. sBTC tokens are issued (or 'minted') by those users who hold the Synthetix native cryptocurrency, SNX. In practice, the user deposits (or 'stakes') SNX in a decentralized application which acts as an escrow for the sBTC. The SNX (not bitcoin) therefore acts as collateral for sBTC: the proportion of SNX:sBTC is generally 4:1 for L1 coins, even though this proportion is not fixed and can be changed via a DAO. Currently algorithm issuance is under scrutiny after what happened to the stablecoin UST in the Terra ecosystem[3].

[1]Note that even if the deposited asset can be either native (i.e. a coin) or non-native (i.e. a token), the minted asset on BC_b is inevitably a token

[2]Currently it is the token with the higher market capitalization, meaning it is the project that currently has the higher number of bitcoin minted and not yet burned.

[3]See for example https://www.bloomberg.com/opinion/articles/2022-05-12/crypto-crash-contagion-could-go-beyond-bitcoin-ethereum-tether?sref=1kJVNqnU

B. Atomic Swaps

While tokenization of a coin or token let users use pegged tokens on another blockchain, an atomic swap is an exchange of a coin or token for another. The exchange is *atomic* in the sense that parties will always end the protocol owning the same wealth they entered the protocol with. More formally,

Definition 1 (Atomic Swap)**.** *Assume parties A and B want to exchange a certain number of tokens T_a and T_b running on blockchains BC_a and BC_b respectively. Also assume the parties measure their wealth in terms of token T_a and that $\alpha = \frac{T_a}{T_b}$ is the rate of exchange of the tokens. Then a* cross-chain atomic swap *is an exchange of n tokens T_a for $n\alpha$ tokens T_b such that*

- *If the protocol is* successful *then A owns $n\alpha$ tokens T_b and B owns n tokens T_a*
- *If the protocol is* non-successful *then A owns n tokens T_a and B owns $n\alpha$ tokens T_b, i.e. the state of the participants' wealth is unchanged*

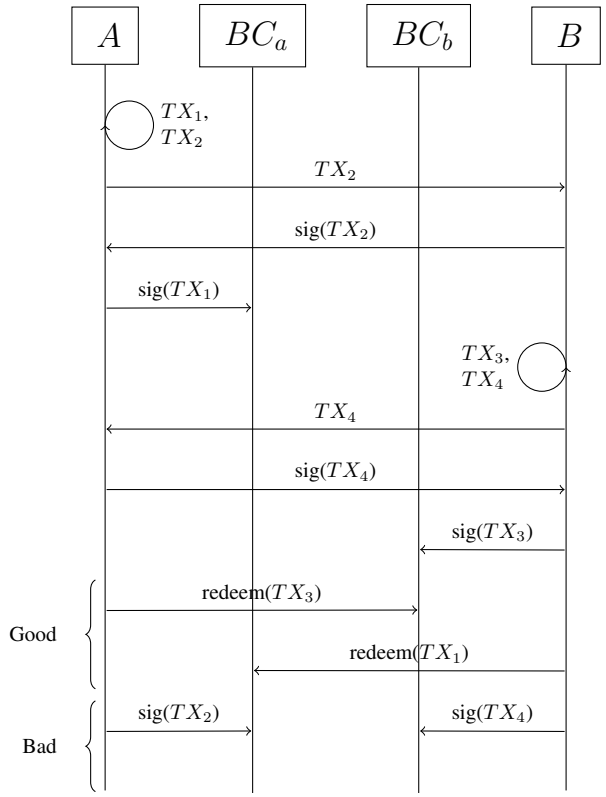

Fig. 1. How a HTLC works

A Hash-Time Lock Contract (HTLC) is a protocol that achieves an atomic swap. The first description of how an HTLC could work is by Tier Nolan [8]. A HTLC needs two phases and four transactions between parties A and B; due to space, we explain here the *meaning* of the transaction leaving out the technical way to implement them. Figure 1 shows a diagram of the operation of an HTLC:

1) TX_1:={Pay nT_a to `<B's public key>` if (x for $H(x)$ known and signed by B) or (signed by A & B)}
2) TX_2:={Pay nT_a from TX_1 to `<A's public key>`, locked 48 hours in the future signed by A}
3) TX_3:={Pay $n\alpha T_b$ to `<A's public key>` if (x for $H(x)$ known and signed by A) or (signed by A & B)}
4) TX_4:={Pay $n\alpha T_b$ from TX_3 to `<B's public key>`, locked 24 hours in the future signed by B}

Note that transactions TX_2 and TX_4 are exchanged and signed by A and B respectively. They are not broadcasted, unless one of the participant is behaving dishonestly (see 'Bad' in Figure 1).

III. Background

Since the goal of the paper is to propose a different way to achive interoperability doing an atomic swap of Bitcoin based wrapped tokens, we provide review on stateful and stateless Simple Payment Verifications related to Bitcoin.

A. Stateful SPV

The first light client has been envisioned by Satoshi Nakamoto in the Bitcoin Whitepaper itself [9]:

> A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in.

The method has originally been called Simple Payment Verification (SPV), but it is commonly referred to "light clients". In practice, a light client let user discern wether a consensus has been reached about a transaction or a set of transaction without making the validation or participating actively into the consensus mechanism. Light-clients are generally off-chain software which connect to nodes.

More complex and performant methods have been created throughout the years. For example, a solution based on Non Interactive Proofs of Proof of Work (NiPoPoW) [10] extracts a "deeper" chain which links together blocks whose nonce would be valid for higher difficulties. More formally, recall PoW based blockchains (and Bitcoin in particular) have a difficulty pararmeter d that is representable as the minimum number of left-most zeroes a block-hash needs to have in order to be considered valid by the nodes of the chain. Nevertheless, a nonce can create a hash representing a number of zeros $d' > d$. The solution proposed by the authors of [10] leverage the higher difficulty of those blocks to link them together. Although interestingng and very performant in principle, the method requires a modification of the bitcoin protocol and can not be instantiated now. Another method is Flyclient [11] which leverages probailistic sampling and Merkle Mountain Range (MMR) [12] commitment of all previous blocks. Fly-client needs some modification to the Bitcoin protocol too.

By definition, any light client assumes that the chain with the most PoW solutions is the one that follows the rules of the network and will eventually be accepted by the majority

of miners. Consequently, a light client has to store at least a specific subset of the headers of the blockchain in order to be able to evaluate the presence of transactions: in this sense, those kind of light clients are considered *stateful*.

This has been a brief introduction. For more detail and a complete systematization of knowledge on the matter see e.g. [13]

B. Stateless SPV

The second kind of light client is the stateless one. It does not store any headers of the blockchain. These light clients perform checks on the headers provided by the users.

Stateless SPV have been presented by Jason Prestwich as a way to solve the problem of relay-maintenance in the Ethereum blockchain[4]. Stateless SPVs are currently proposed for alternative versions of relays but there is no formal explanation of how them work[5]. Goal of this section is to fill this gap.

For our purposes we can model the typical Bitcoin block as a set of five fields: magic number, block size, block header (or simply "header" from now on), transaction counter and transaction list. As in the light-clients case, we are not interested in the transaction list or the other constants, but only in the block header, since this is what SPVs (both stateful and stateless) are interested in.

The fields of the block header are presented in Table I and we refer to the header of i-th block $\mathcal{B}_i$ as $\mathcal{H}_i$ and to its fields via dot-specification, e.g. the nonce field of the i-th block header is $\mathcal{H}_i.nonce$. The leader election phase of the Proof of Work uses the header to compute a hash. If this hash is less than a target L, then the block can be included in the blockchain. Note that the lower the target L the lower the probability of finding the right hash. Consequently the lower L, the more *work* (i.e. nonce update and rehash) needed by the miners.

Let h_i be the valid hash of the header $\mathcal{H}_i$ and note that h_i is included in $\mathcal{H}_{i+1}$ as $\mathcal{H}_{i+1}.hashPrevBlock$ and that its numerical representation of h_i has to be lower than L.

Assume a user U broadcasts a transaction tx on the Bitcoin blockchain. Slightly abusing language, we identify the transaction with its transaction hash. Assume tx is included in block $\mathcal{B}_N$ among other $k-1$ transactions. Then the vector vec_N containing the transaction and the Merkle tree's branches contains the data needed to verify that transaction tx is included in block $\mathcal{B}_N$ by comparing the Merkle tree hash with $\mathcal{H}_N.hashMerkleRoot$. For example, with reference to figure 2, the vector vec_N would be:

$$vec_N = [tx, H_0, H_{23}, H_{4567}] \tag{1}$$

Assume a smart contract SC is capable of evaluation hash functions and make integer comparisons. Therefore SC can reliably check if tx has been confirmed in the Bitcoin blockchain via the evaluation of a proof π such that:

$$\pi = [vec_N, \mathcal{H}_N, \mathcal{H}_{N+1}, \ldots, \mathcal{H}_{N+n_{blocks}}] \tag{2}$$

where n_{blocks} is the number of blocks needed to be sufficiently secure that π has not been forged. See Section V-B for a more detailed security proof.

TABLE I
BLOCK HEADER FIELDS

Field	Description
Version	The version of the block.
$hashPrevBlock$	The hash of the previous block in the chain.
$hashMerkleRoot$	The hash of the Merkle root of the transactions.
Time	The time the block was created.
nBits	The target of the block (compressed form).
$nonce$	The nonce of the block.

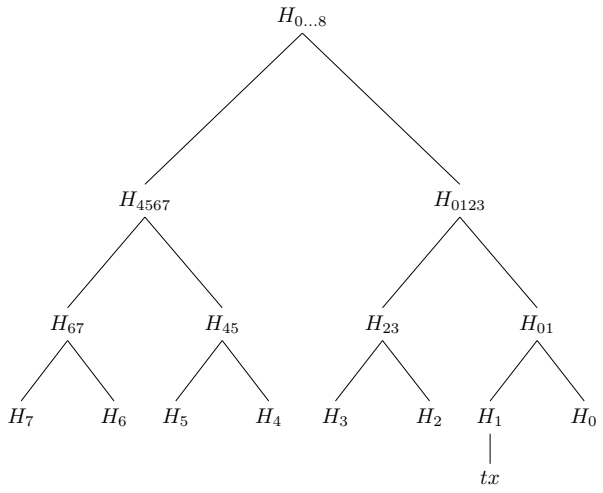

Fig. 2. Particular merkle tree structure of a block with transaction tx. In this case the merkle leafs needed for the proof π are (H_0, H_{23}, H_{4567})

IV. DESIGN

A. Setting and Notation

We assume the existence of a decentralized-application platform PLAT which acts as a match-making algorithm between parties and provides a secure channel between parties[6]. The goal of PLAT is to match parties who want to exchange bitcoins for a wrapped equivalent on a Turing-complete blockchain, we will call *Tchain*. We call this exemplary

[4]See for example [14] which is currently the most popular relay on the Ethereum network. Despite this, development is stopped at 2017 (last commit) and as of May 19th 2022 the last transaction is from 1325 days ago. The main reason seems to be the fact that maintain a relay is costly and there is no rational incentive to do so.

[5]We checked in famous repository of paper such as DBLP, Scopus, Google Scholar and IEEE. Yet, it is possible to see the original talk by the author at https://youtu.be/njGSFAOz7F8. An informal explanation of how Stateless SPV work and their security is proposed in https://ethresear.ch/t/stateless-spv-proofs-and-economic-security/5451

[6]This can be implemented with end-to-end encrypted communication so that PLAT never knows anything more about the exchange,see e.g. [15]

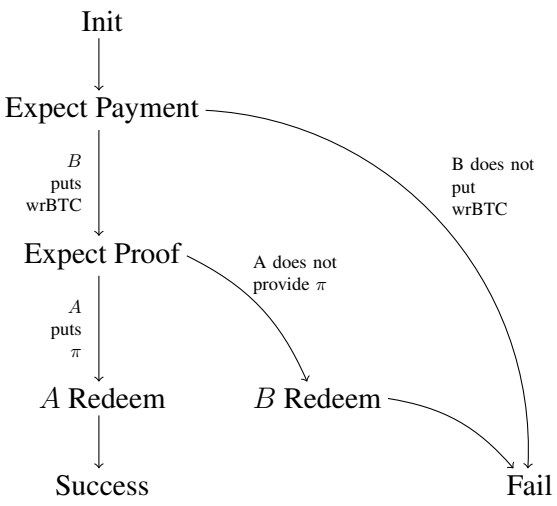

Fig. 3. State succession of Algorithm 1

wrapped token *wrBTC* and we assume it is build on the model of those explained in Section II-A.

After the match, PLAT interacts with a smart contract SC deployed on Tchain. The smart code's pseudocode can be seen in Algorithm 1. Note that we do not explain how the matching works, since it is outside the scope of this paper: we assume the match has been done and we describe how parties interact to complete the protocol.

We also assume two parties. Alice, A, that owns bitcoins and wants to exchange them for wrBTC on Tchain. Bob, B, that owns wrBTC and wants to exchange it for bitcoins. For simplicity, we assume the exchange amount is amt, the timeout on the exchange is T and the number of blocks to build the stateless SPV proof π is n_{blocks} (See Equation (5) for a practical way to decide on n_{blocks}) . We assume that parties agree on those parameters during the match-making phase, non-necessarily without suggestions or defaults provided by the platform PLAT. Note that there is no need to agree on an exchange rate since the currencies are supposed pegged. We briefly mention a way to exchange non-pegged tokens in Section VI of this paper.

We denote the addresses of parties in the following way. Since both parties needs to have two addresses (one on Bitcoin and one on Tchain), we denote the addresses of Alice and Bob on Bitcoin as $AddrA_{\mathrm{BTC}}$ and $AddrB_{\mathrm{BTC}}$ respectively and on Tchan as $AddrA_{\mathrm{TCH}}$ and $AddrB_{\mathrm{TCH}}$ respectively. The exchange of meaningful data between A and B, e.g. addresses and amount, can be public or exchanged privately in the channel. From a practical point of view this does not change anything from a privacy point of viewd: all information becomes public once the smart contract is used by the parties.

B. The Swap

We assume that B initialize the smart contract SC by calling the PREPROCESS (line 1) and the INIT functions (line 14). The latter function put set the State equal to Init.

Algorithm 1 PLAT's smart contract SC

Require: $AddrA_{\mathrm{BTC}}$, $AddrB_{\mathrm{BTC}}$ and $AddrA_{\mathrm{TCH}}$, $AddrB_{\mathrm{TCH}}$ are valid addresses
Require: amount amt is a positive integer
Require: timeout T is a positive integer
1: **function** PREPROCESS($AddrA_{\mathrm{BTC}}$, $AddrB_{\mathrm{BTC}}$, $AddrA_{\mathrm{TCH}}$, $AddrB_{\mathrm{TCH}}$, amt, T , L)
2: Struct Exchange = {
3: AddrABTC = $AddrA_{\mathrm{BTC}}$,
4: AddrBBTC = $AddrB_{\mathrm{BTC}}$,
5: AddrATCH = $AddrA_{\mathrm{TCH}}$,
6: AddrBTCH = $AddrB_{\mathrm{TCH}}$,
7: amt = amt,
8: T = T
9: L_{max} = L ▷ L_{max} is the maximum target accepted as difficulty in headers
10: }
11: States = { Init, ExpectPayment, ExpectProof, Redeemable, Success, Fail }
12: setState(sid,Init)
13: **end function**
14: **function** INIT(sid, Exchange)
15: **if** State == Init **then**
16: map sid to Exchange
17: setState(sid,ExpectPayment)
18: **end if**
19: **end function**
20: **function** RECEIVEWRBTC(sid, Exchange)
21: **if** `msg.sender` == AddrBTCH &
22: `msg.value` == amt &
23: State == ExpectPayment **then**
24: Event(Exchange, sid,received, {AddrBTCH, amt})
25: setState(sid,ExpectProof)
26: **end if**
27: **end function**
28: **function** RECEIVEPROOF(sid, Exchange, π)
29: **if** `msg.sender` == AddrATCH & time $<$ T &
30: Verify(π, Exchange)== True & State == Init **then** ▷ Verification is explained Section IV-C
31: send(AddrATCH, amt)
32: Event(Exchange, sid,sent, {AddrATCH, amt})
33: setState(sid,Redeemable)
34: REDEEMWRBTC(sid, Exchange)
35: **end if**
36: **end function**
37: **function** REDEEMWRBTC(sid, Exchange)
38: **if** `msg.sender` == AddrATCH &
39: time $<$ T & State == Redeemable **then**
40: send(AddrATCH, amt)
41: Event(Exchange, sid,sent, {AddrATCH, amt})
42: setState(sid,Success)
43: **else if** `msg.sender` == AddrBTCH &
44: time $>$ T & State == ExpectProof **then**
45: send(AddrBTCH, amt)
46: Event(Exchange, sid,withd, {AddrBTCH, amt})
47: setState(sid,Fail)
48: **end if**
49: **end function**

After the initialization, B is required to put amt wrBTC in the smart contract (line 20) to advance the protocol[7]. If B does that before the timeout, the State of SC is set to ExpectProof. Note that A has not provided any bitcoin at this point: even if B fails this step, A does not lose anything.

If B behaves honestly, A broadcasts a transaction from $AddrA_{\mathrm{BTC}}$ to $AddrB_{\mathrm{BTC}}$ on the Bitcoin blockchain. After waiting for the creation of n_{blocks} blocks on the Bitcoin blockchain, A builds the proof π. After that, A calls SC's function RECEIVEPROOF (line 28) from $AddrA_{\mathrm{TCH}}$: one of the inputs of the call is the the proof π. If A provides a correct π before the timeout (see Section IV-C and Algorithm 2 to see how the verification works), the State of SC is set to Redeem (see Figure 3 to see how States of SC change) and A can redeem the wrBTC (line 39). On the other hand, if A fails to provide a correct π the state of SC ramains ExpectProof: after the timeout, B can redeem the wrBTC (line 44).

The protocol is atomic (see Definition 1) and we show that in Section V-A.

C. Proof Validation

We are left to explain how the statless SPV proof π is validated. The verification exploits the fact that it is highly expensive to produce a formally valid proof in the dishonest case: see Theorem 2 in Section V-B.

The first thing to check is if the transaction tx is included in the Merkle tree of block $\mathcal{B}_N$. It is easy to build the vector vec_N (see Equation (1)) for A which has access to the Bitcoin blockchain. The Merkle root hash is included in the header $\mathcal{H}_N$, see Table I, and to perform this check the function only needs the correct hash function.

If tx is included in the Merkle tree, then the vec_N is valid and it is possible to proceed and chek if the subsequent headers are built one upon the other with a sufficient difficulty. With reference to Algorithm 2, this is performed from line 7 through 15.

Algorithm 2 Verification procedure of SC

Require: $\pi = [vec_N, \mathcal{H}_N, \mathcal{H}_{N+1}, \ldots, \mathcal{H}_{N+n_{blocks}}]$ ▷ Equation (2)
Require: $\pi.vec_N = [tx, H_i, H_{jk}, \ldots]$ ▷ Equation (1)
Require: Exchange as defined in Algorithm 1
Require: $H(\cdot)$ hashing function of Bitcoin

```
1:  function VERIFY(π,Exchange)
2:      vec = π.vec_N
3:      VerifyMerkleTree(vec,H_N.hashMerkleRoot)
4:      H_prev = π.H_N
5:      L_max = Exchange.L_max
6:      assert H_prev.nBits ≤ L_max   ▷ Check that the target is not too large
7:      for i = 1, ..., n_blocks do
8:          H_cur = π.H_{N+i}
9:          assert H_cur.nBits ≤ L_max
10:         if H(H_prev) == H_cur.hashPrevBlock then
11:             H_prev = H_cur
12:         else
13:             return False
14:         end if
15:     end for
16:     return True
17: end function
```

V. ANALYSIS

A. Atomicity

To prove that the protocol is atomic, we need to show that no party would incur in financial loss in case of abort or incorrect execution. We do that with reference to Figure 3.

Of course if no initialization is done (Init), then neither A nor B incur in financial loss since there is no deployemnt of capital. After that, the smart contract expect a payment from B (ExpectPayment). Assuming incorrect execution, at this step, B does not put any wrBTC in the smart contract. Similar to the case of the initialization, A does not incur in financial loss since A has not deployed capital yet.

[7]Note that the PREPROCESS, INIT and RECEIVEWRBTC can in principle be collapsed into only one function: B would certainly save fome fees doing that. We decided to keep this functions separated in Algorithm 1 for two reasons. The first one is that keeping the funcion separated help the understanding of the protocol. The second reason is that this way the platform PLAT could theoretically initialize SC in place of B, possibly as promotion to make users save on fees.

On the other hand, if B does put the right amount of wrBTC in SC, then A is expected to make a payment from $AddrA_{\mathrm{BTC}}$ to $AddrB_{\mathrm{BTC}}$ on the Bitcoin blockchain, then build and send the proof π to SC. In this case, if A does not make a payment before the timeout or if A provides an incorrect proof (we show in Theorem 2 the sufficient condition for B to be sure A will not be able to creat an invalid proof which is accepted), then B does not incur in financial loss since B will be able to redeem the wrBTC from SC by calling the REDEEMWRBTC function.

Finally, if A and B follow the protocol correctly, then A can redeem the wrBTC on Tchain by putting π in the RECEIVEPROOF function, since the REDEEMWRBTC function is called from RECEIVEPROOF.

B. Security

We want to prove that it is economically secure to validate a transaction with no stored block header but only with the n_{blocks} headers provided by the user (which of course can potentially be forged by it). By *economically secure* we mean that the cost of forging a proof is higher than the amount of the transaction the user needs to prove the inclusion of. Formally:

Definition 2. *A proof π defined as Equation* (2) *of transaction tx with amount amt is* (n_{blocks}, p^*)-economically secure *if the expected cost of having a probability of p^* of producing π is higher than amt.*

To prove it, we will leverage the $nBits$ field of the block header (see Table I). Also we assume that the difficulty/target do not change between block N and $N + n_{blocks}$. This is not

a big constrain since the difficulty changes once every 2016 blocks in Bitcoin and n_{blocks} is generally less than 10 (note that a Bitcoin transaction is generally considered confirmed after 6 blocks). Furthermore, it will be easy to see that *if* the difficulty changes between block N and $N + n_{blocks}$, then taking $\max(\mathcal{H}_N.nBits, \mathcal{H}_{N+n_{blocks}}.nBits)$ as difficulty solves the problem.

We briefly formalize the leader election phase of the Nakamoto consensus protocol which is based on the HashCash system [16]: Given a target L, the *work* of a miner is to find the nonce $nonce$ for a block $\mathcal{B}$ such that[8] $H(\mathcal{B}) < L$, where H is the hash function used by the system, SHA256 in the Bitcoin case. In this case we say that $nonce$ is a valid nonce for block $\mathcal{B}$. Valid outputs for the hash function used in Bitcoin range from 0 to $2^{256} - 1$, but L is much less than that. So the number of available hashes for block $\mathcal{B}$ is L and the probability of finding one is $p = L/2^{256}$. "Extracting" hashes is therefore a Bernoulli process of probability (and mean) p. To give an idea with real values, on May 25, 2022, the difficulty is $d = 31251101365711$ [9]. The target L can be computed from the difficulty by doing $L = 2^{224}/d$ (the exact computation to arrive at this relation between L and d can be seen in [17]). Therefore $p \approx 7 \times 10^{-24}$. The probability of 1 success in n Bernoulli trials is $p^* = np(1-p)^{n-1}$. For example, with the values presented above, we need $n = 10^{26}$ trials to be $\approx 60\%$ sure of finding a valid nonce. Consequently, assuming C to be the cost of computing a *terahash* (i.e. 10^{12} trials), being 60% sure of producing one single header (i.e finding a nonce) has an expected cost $EC_1^{0.6} = 10^{-12} \cdot n \cdot C$.

We can now prove that:

Theorem 2. *Given a target $\mathcal{H}.nBits = L$, let $p = L/2^{256}$ and p^* be the probability of success in finding a suitable hash for one header $\mathcal{H}$ after n_{p^*} trials. Assuming C to be the cost of computing a terahash, then the expected cost $EC^{p^*}_{n_{blocks}}$ in being p^* sure of finding n_{blocks} nonces for n_{blocks} headers is:*

$$EC^{p^*}_{n_{blocks}} = n_{blocks} \cdot 10^{-12} \cdot n_{p^*} \cdot C \tag{3}$$

Then a sufficient condition to for a protocol between rational participants to achieve (n_{blocks}, p^*)-economic security for a proof π of transaction tx with amount amt is

$$EC^{p^*}_{n_{blocks}} \geq amt \tag{4}$$

Proof. It is easy to compute $EC^{p^*}_{n_{blocks}}$ of Equation (3) noting that all trials are independent events. To see the sufficiency of condition in Equation (4), first note that generally the cost of producing hashes is shared by all the miners in the honest case and therefore comes at a cost of ≈ 0 for the honest user (it only has to pay the fees of transaction tx)

On the other hand, if the user is dishonest and needs to forge π, then it incurs a potential loss: it has to bear the expected cost $EC^{p^*}_{n_{blocks}}$ without being completely sure of finding a solution within a certain timeout, since there is still a probability of $1 - p^*$ of *not* producing the proof even after incurring in the $EC^{p^*}_{n_{blocks}}$ cost[10]. Therefore, the user has no interest in forging a proof π in the case of Equation (4) since it is expected to lose more than it has to gain. □

Since Equation (4) is not operational, Equation (5) gives a practical way for participant B, or PLAT, to enforce economic security of the protocol for any amount amt of every transaction:

$$n_{blocks} \geq \frac{amt}{10^{-12} \cdot n_{p^*} \cdot C} \tag{5}$$

C. Privacy and Communication

BxTB is as private as the HTLC described in Section II-B. In HTLCs in particular the hash created by the initiator A acts a indexing of the transactions in Bitcoin and in the other chain used for the swap. Using this index it is possible to learn A and B's addresses in both blcokchains and the amount exchanged. These are the same information an external observer and the platform PLAT can gather by looking at the smart contract SC.

On the other hand, BxTB achieve the same results the HTLC achieves but with less transactions, even in the worst case. In fact, BxBT requires two transactions in the worst case instead of four as in the HTLC case: One transaction from B to SC to initialize the smart contract and deposit the tokens; One transaction from B to SC to withdraw the funds after the timeout. The worst case is considered to be the case where A does not follow the protocol, since if B does not follow the protocol then no transaction is being done by definition (B does not initialize the smart contract and/or does not deposit the tokens in SC).

VI. Future Works and Perspectives

As we mentioned before, BxTB can in principle work for any couple of tokens, from a Proof-of-Work blockchain to a wrapped-token on any Turing complete blockchain. In fact, assuming the hash function(s) used for the leader election is implemented in the smart-contract language of Tchain, the basic checks are trivial to make and shared by all Proof-of-Work blockchains. Some changes to the protocol are needed: for example, if the tokens are not pegged, then the parties need to agree on the rate of the exchange as in a general atomic swap (see Section II-B). Technically this rate can be decided while posting the order on PLAT, but some considerations on privacy and communication-complexities should be studied.

BxTB can also be extended by forcing A to put some collateral. In the currenct design of BxTB, A can potentially abort the protocol after the initialization or intentionally put an invalid proof. While that would not hurt B financially (assuming n_{blocks} has been chosen in accordance to Equation (5)), it is still a problem since B's capital is blocked for a

[8]The nonce is part of the block header and therefore part of the block, for this reason it does not appear explicitly, see Table I.

[9]See https://bitinfocharts.com/comparison/bitcoin-difficulty.html

[10]It is possible that $EC^{p^*}_{n_{blocks}}$ and amt do not share the same currency to be evaluated. In this case we assume amt is evaluated using the currency $EC^{p^*}_{n_{blocks}}$ is evaluated.

certain amount of time. This is a common problem in atomic swaps and it is present in current implementations of HTLC too (see e.g. [18]). The collateral would act as an incentive for A to continue the protocol, since it can be slashed in cases where A puts an invalid proof or simply does not follow through the protocol.

In the future we plan to do a full implementation of the protocol assuming an exchange between Bitcoin and a Bitcoin wrapped-token using the smart-contract language of Solidity in a private instance of Ethereum.

VII. Conclusions

We presented BxTB, a new protocol to exchange bitcoin wrapped tokens without incurring in privacy threatening controls. We achive exchanges leveraging stateless SPVs, practically describing how to create a light client in a smsart ocntract. We proved the economic security of the protocol and we achieve atomicity using less transactions between parties even in the worst case when one party is dishonest.

References

[1] Henning Pagnia and Felix C Gartner. On the impossibility of fair exchange without a trusted third party. Technical report, Citeseer, address, 1999.

[2] N. Asokan. *Fairness in electronic commerce*. PhD thesis, IBM, 1998.

[3] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. Sok: Communication across distributed ledgers. 12675:3–36, 2021.

[4] Giulio Caldarelli. Wrapping Trust for Interoperability: A Preliminary Study of Wrapped Tokens. *Information*, 13(1):6, January 2022.

[5] Kyber Network, BitGo Inc, and Republic Protocol. Wbtc Whitepaper v0.2. https://wbtc.network/assets/wrapped-tokens-whitepaper.pdf, 2019.

[6] Ross Pure and Zian-Loong Wang. Renvm secure multiparty computation. https://github.com/renproject/rzl-mpc-specification/blob/master/z0_spec.pdf, 2020.

[7] Synthetix Team. Synthetix litepaper. https://docs.synthetix.io/litepaper/, 2022.

[8] T. Nolan. Alt chains and atomic transfers, 2013.

[9] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Whitepaper*, page 9, 2008.

[10] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 61–78, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[11] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. FlyClient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 928–946, 2020.

[12] Peter K. Todd and Zindros Dyonis. Merkle mountain ranges, December 2018.

[13] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. Sok: Blockchain light clients. *IACR Cryptol. ePrint Arch.*, page 1657, 2021.

[14] Ethereum. Btc relay. https://github.com/ethereum/btcrelay, 2021.

[15] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2019.

[16] Adam Back. Hashcash - A Denial of Service Counter-Measure. page 10, 2002.

[17] Rhys Bowden, H. Paul Keeler, Anthony E. Krzesinski, and Peter G. Taylor. Block arrivals in the bitcoin blockchain. *CoRR*, abs/1801.07447, 2018.

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