MP-HTLC: Enabling Blockchain Interoperability through a Multiparty Implementation of the HTLC

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

MP-HTLC: Enabling Blockchain Interoperability through a Multiparty Implementation of the HTLC

Fadi Barbàra | Claudio Schifanella

Computer Science Department, University of Turin, Turin, Italy

Correspondence
Email: fadi.barbara@unito.it

Summary

The idea of Hash Time-Lock contracts (HTLCs) has been around from 2013. Nowadays these contracts power the majority of atomic swaps making decentralized exchange of tokens possible. On the other hand, HTLCs also have some flaws. For example they can only be instantiated between two parties. This is highly inefficient when many participants want to exchange tokens between the same pair of blockchains at the same time, because the number of transactions increases linearly in the number of participants. To solve this problem, in this paper we present MP-HTLC. MP-HTLC lets multiple users exchange tokens on different blockchains in a single instantiation of the protocol without any leader election. We prove that in case of a UTXO-based blockchain the number of transactions remains constant regardless the number of participants. We are able to maintain the security assumptions of HTLCs using multiparty computation in the creation of the secret preimage and threshold signatures to manage transaction signing. We also present an implementation for each of the aspects of the protocol.

KEYWORDS:
blockchain, interoperability, HTLC, threshold signatures, Bitcoin, Ethereum, Polygon

1 | INTRODUCTION

Nowadaysthere are thousands of projects that use a blockchain and tokens to encourage users to use the project[†]. The tokens in many of these projects have a use only within the project in which they are created (so-called *utility tokens*), but they have an intrinsic value that can be exported. As a practical matter, it is necessary to be able to exchange them for other tokens or coins in other blockchains. The most common example is the exchange of a utility token for ethers or bitcoin.

This is today done via centralized exchanges, but they are vulnerable to many active adversarial attacks. An example of these attacks is the MtGox case[1]. Therefore it is better to exchange tokens in decentralized ways as in the case of the implementation of traditional finance on a blockchain (decentralized finance, or DeFi).

The exchange of tokens sitting in different blockchains requires that at least one blockchain is able to read and verify the states of the other blockchain. This process is called *interoperability*. But DeFi isn't the only place where interoperability is needed. Interoperability is also needed to obtain scalability, in particular in sharding protocols, and both in state and payment channels. In sharding, different committees (i.e. subset of validators or miners) process a different subset of transaction. The committee periodically publish a proof to the parent chain, therefore inheriting the security of the parent chain. An example of this method is Plasma[2], in which the parent chain is Ethereum. In practice the committees create different ledgers. For sake of simplicity we

[†]see for example coingecko.com for a list of tokens or coins

call these ledgers *subchains*. When there is a transaction between two sub-chains, it is important for the one group of committees to be able to verify the states of the other subchain in order to process the transaction. That is the reason why interoperability is needed in sharding protocols.

Analogous arguments can be done for state and payment channels. For example, in a payment channel like Lightning Network [3] the users create off-chain ledgers. Generally this ledgers are called *channels*. However, when there must be a transaction between two channels or a transaction between the parent chain (Bitcoin in this case) and the channel (e.g. during the opening or closing of the channel) it is important to be able to verify the states of the channel.

This last example introduces another problem of current decentralized methods of token or coin exchange. In fact, nowadays the most used method is the Hash Time-Lock Contract (HTLC) [4], but this method has two strong downsides. The first one is that it can be instantiated only between two parties. Consequently multiple participants need to instantiate a contract for each couple of participants: when there is a multitude of participants that wants to exchange tokens between the same couple of blockchain, the HTLC inner mechanics requires a new instance of the contract for each couple of participants. This has large extra costs since performing a single instance of the protocol among many participants is less expensive than creating many instances of the same protocol, both from a communication point of view and from a fee point of view for sending the multiple transactions. We prove this in Section 7.1.

In addition to inefficiency issues, HTLCs suffer also of privacy issues. In fact if a new instance is created each time for each pair of participants, by keeping track of the multiple instances is possible to keep track of which participants are involved: a passive observer of the two blockchains involved can be sure that participant A has interacted with participant B.

To solve these problems we present MP-HTLC. In MP-HTLC a group of participants can exchange tokens between the same couple of blockchains at once. That's accomplished using Secure Multiparty Computation (MPC) [5] and Threshold Signatures (see for example [6] for a threshold signature scheme applied in a blockchain environment). Using MP-HTLC, participants save on fees by participating all together in a single instance of the protocol and sharing the costs. They also gain privacy, as being part of an anonymity set (i.e. the group of participants in one blockchain) they are protected against an external blockchain observers who cannot link a participant in one group with a participant in the second group. To achieve this result, we use a previous work called DMix [7]. In DMix the authors create a protocol acting as a *de facto* decentralized mixer in a UTXO blockchain such as the Bitcoin one. The DMix protocol is split into three phases: in the first phase the participants create an address thanks to a distributed key creation protocol; in the second phase they send their funds to the address and in the third one they redeem their funds sending them to newly created addresses. In the third phase participants use n-of-n threshold signatures. We use the ideas of the DMix protocol to obtain privacy in the MP-HTLC cross-chain communication protocol. Furthermore, by using threshold signatures, participants create a single signature regardless of how many there are. This is an additional layer of protection against an outside observer who cannot know for sure which transactions on the blockchain stand alone or are part of a token exchange with MP-HTLC and it is also a cost saving improvement in case of multiple participants.

We also present two use cases. the first use case is a Bitcoin-Ethereum transfer: this way we describe the interoperability concept applied to the token exchange. For this case we propose an implementation. The second use case is a Ethereum-Polygon exchange: this way we describe the interoperability concept applied to the scalability problem. Furthermore, to prove the feasibility of MP-HTLC we present also the implementation of the case between Bitcoin and Ethereum.
In summary, our research contributions are:

- we present MP-HTLC, an interoperability system between two blockchains capable of managing multiple participants in a single instance
- we present use cases showing that MP-HTLC can be used both to achieve scalability and to achieve simple token exchange
- we propose a decentralized and secure alternative to EVM-compatible bridges such as the one between the Binance Smart chain and Ethereum and the one between Polygon and Ethereum
- we propose an implementation in the case of Bitcoin-Ethereum[‡]
- we present a cost and attack analysis showing that MP-HTLC is better than legacy HTLC in case there are more participants simultaneously available to an exchange in the same blockchain pair

[‡]Is is possible to see the implementation here: https://github.com/disnocen/mp-htlc

The rest of the paper is organized as follows. In Section 2 we describe the related works. In Section 3 we describe the preliminaries necessary to understand the functioning of MP-HTLC and in section 4 we describe the general functioning. In sections 5 and 6 we present the use cases. In Section 7 we present the cost and attack analysis and in Section 8 we conclude.

2 | RELATED WORKS

There are three major strands of research in the blockchain ecosystem that relate to interoperability. For details see the paper of Zamyatin[8], which describes the topological and technical level of the different methodologies to achieve interoperability between two different blockchains, and the paper of Belchior[9], which describes all the solutions currently used to date.

The first line of research is token exchange. This type of interoperability dates back to 2016 with the report created by Vitalik Buterin[10]. This report explains the various differences between different types of blockchain and related implementations of "bridges" between them. The second strand of research has to do with the problem of blockchain scalability. It is a consequence of sharding, i.e. the division of validators/miners into different committees, each specialized in processing a subset of all transactions. The third and last strand of research related to interoperability on blockchain is related to layer-two (L2) technologies.

In the following, we delve into each of these lines of research, give indications of where interoperability is needed and the proposed solutions.

2.1 | Token Exchange

Cross-chain token exchange, also called cross-chain asset transfers, rely on an atomic three-phases procedure: asset locking on a source blockchain, blockchain-transfer commitment, and finally a creation of a representation of the asset on a target blockchain[11,12]. This last step can be also called *release*. As in Zamyatin[8], we call this three-steps procedure cross-chain communication (CCC). The methods to obtain CCC belong to four categories: sidechains, relays, notary schemes and atomic swaps via HTLC.

Sidechains were introduced by Back et al.[13] to enable innovation while maintaining security in the Bitcoin blockchain. The authors consider another blockchain (called *sidechain*) as an extension of the main blockchain (called *mainchain*). A sidechain allows the offloading of transactions from the mainchain: it processes it, and communicate the outcome of such process back to the mainchain effectively performing a CCC. An example of CCC is the two-way peg[14]. A centralized two-way peg is a single entity that locks and then create of a representation of the asset on a target blockchain. Being centralized it requires trust, but provides efficiency. To solve the trust problem, a majority of multiple entities can lock and release: in this case we talk about federated two-way pegs[15].

Relays are generally on-chain clients that can verify transactions on the blockchain using only block headers, without having its entire state. Examples of relay are the simplified payment verification (SPV) used by light-clients and proposed in the original Bitcoin whitepaper[16], and BTC-Relay[17] which is an Ethereum-based chain-relay that verifies transactions and blocks of the Bitcoin blockchain. A relay is managed by a single entity and can not be used by multiple parties.

The third method to exchange assets is using Notary Schemes. A notary committee monitors multiple chains and triggers transactions in a chain upon an event. The difference between a notary committee and centralized or federated two-way pegs is that the notaries do not create tokens, but they just manage and exchange the token in both chains. Examples are exchanges which basically which just run a matching algorithm to put two parties in communication for an exchange: one party wants to exchange a value v_A. of asset A for a value v_B of asset B while the other party has the exact opposite goal. The exchanges itself can be instantiated as centralized exchanges (CEXs) or decentralized exchanges (DEXs): a CEX has the private keys of the users (for this reason it is also sometime called *custodial*), while a DEX has not

Atomic swaps let users exchange assets, currently using Hash-Time Lock contract (HTLC). Atomic swaps are decentralized and trustless, differently from the methods presented above. HTLCs are explained in detail in Section 3.3 and it is the basis of the interoperability method presented here.

2.2 | Scalability

Blockchains are a particular design of distributed systems, so they stand in the tradition of distributed protocols of replicated state machines. As in any distributed system, nodes must reach an agreement, so consensus is one of the most important problems in the blockchain literature.

One of the problems with current consensus algorithms is that they don't scale. In fact, there are two key metrics to measure *scaling*: transaction throughput (i.e. the maximum rate at which the blockchain can process transactions) and latency (i.e. the time to confirm that a transaction has been included in the blockchain). Generally, both transaction throughput and latency remain constant even as the number of miners and validators (i.e., those who are able to process new information and write it on the blockchain) increases.

One solution is to divide the miners and validators into smaller subsets so that each of these subset processes a distinct subset of the information or set of transactions (called "shards"). This method is called sharding. Its first academic proposal has been Plasma[2] and it is currently on the roadmap for Ethereum 2.0 [§]. The term "sharding" refers to approaches targeting the blockchain's core design, also known as *on-chain solutions*, rather than techniques that delegate to parallel blockchain instances such as sidechains, as in Section 2.1. We model the process of sharding as in[8].

Let n be the number of participating nodes on a blockchain and for simplicity we assume they have the same computational power. We also assume that a fraction f of those nodes is controlled by a Byzantine adversary (of course f could be equal to 0 if all nodes are honest). We index the i-th transaction in block j as x_i^j and assume there is a validity boolean-function V which outputs 1 if x_i^j is valid and 0 otherwise.

For a fixed block j, a sharding protocol Π running between n nodes outputs a set X which contains k separate shards (subset of transactions): $X_w = \{x_{w_i}^j\}(1 \leq i \leq |X_i|)$ and $X = \cup X_w$. Each shard must follow the subsequent conditions:

- *Agreement*. given a security parameter λ, honest nodes agree on X with a probability of at least $1 - 2^{(-\lambda)}$
- *Validity*. The agreed shard X_w satisfies the validity function V such that $\forall w \in 1 \dots k, \forall x_{w_i}^j \in X_w, V(x_{w_i}^j) = 1$.
- *Scalability*. The value of k grows almost linearly with the size of the network.

In practice, a sharding algorithm automatically parallelizes the available computation power, by dividing the nodes into several smaller committees, each of them processing a disjoint set of transactions. The sharding protocol Π consists of five critical components:

1. *Committee formation*. Each node is assigned to a committee, after it has been identified e.g. via a public key, an IP address or a proof-of-work (PoW) solution. The identification is needed in permissionless blockchain to prevent the Sybil identity[18]
2. *Overlay setup for committees*. After the committee's formation, each node communicates to discover the identities of other nodes in its committee, generally via a gossip protocol[19].
3. *Intra-committee consensus*. Each node in a committee runs a consensus protocol to agree on a single set of transactions
4. *Cross-shard transaction processing*. the inputs and outputs of a transaction might be in different shards. These transactions are called cross-shard (or inter-shard) transactions
5. *Epoch reconfiguration*. the shards need to be reconfigured after a predetermined time ("epoch") to guarantee the security

Interoperability is needed for Cross-shard transaction processing, which generally represents the vast majority of transactions in a sharding consensus protocol. In both the Unspent Transaction-Output (UTXO) model and the account transaction model it is expected that up to 90% of transactions are cross-sharded, as seen in[20] (UTXO model) and in[21] (account-based model).

Atomic commitments are used to perform cross-chain transactions[8]. Protocols that use this method are RSCoin[22], Chainspace[23], OmniLedger[24], RapidChain[25]. These atomic commitments act as garbage-collectors: this can be seen well in UTXO models where a transaction will never be used again once spent. Another method is the distributed state snapshotting mechanism[26] to record the blockchain's recent status. This mechanism is used by SideCoin[27] and Roller-Chain[28]. Other projects such as Elastico[29], do not provide a clear or separated process to deal with the cross shard transactions.

[§] https://ethereum.org/en/eth2/

2.3 | Layer 2

Layer-two protocols, built on top of (layer-one) blockchains, don't disseminate every transaction to the whole network and exchange authenticated transactions off-chain. They utilize the blockchain only as a recourse for disputes. More formally, using Definition 1 in[8] a "layer-two protocol allows transactions between users through the exchange of authenticated messages via a medium which is outside of, but tethered to, a layer-one blockchain. Authenticated assertions are submitted to the parent-chain only in cases of a dispute, with the parent-chain deciding the outcome of the dispute. Security and non-custodial properties of a layer-two protocol rely on the consensus algorithm of the parent chain." In this case interoperability is required to pass from the blockchain to the off-chain method and vice versa. Of these three mechanisms, the only one that needs interoperability is the channel construction and closing, so in the following we explain how channels work

An off-chain channel between n coequal parties establishes a private peer-to-peer medium. This medium allows the involved parties to consent to state updates exchanging authenticated state transitions off-chain. There are two channel techniques: payment channels, supporting payment interactions; and state channels, supporting arbitrary interaction. A channel's management has three phases: channel establishment, transactions phase and either channel closure or disputes.

During channel establishment all parties open a channel by locking collateral on the blockchain during the transaction phase the parties update the channel's state in a two-step process. First, one party proposes a new state transition by sending a signed transaction and the new state to all other parties. After that, each other party verifies that the state transition is valid and if so it relays to the others. This exchange can be done one or more times. The last phase is either a peaceful closure or a dispute in case of faulty/dishonest parties. Disputes arise if a party does not receive n signatures before a local timeout. In this case the honest party may trigger a dispute enforcing a new state transition on layer-one.

Therefore the creation of a payment channel creates at a practical level a new blockchain. This chain is to all intents and purposes off-chain, but having nodes keeping track of funds, it is a ledger. For this reason, every step from the main chain to the secondary chain requires interoperability between the two ledgers. It is important that the methods for exchanging tokens between layer one and layer two are efficient, secure, and provide privacy guarantees, ideally by making the channel opening or closing transaction indistinguishable from any other transaction on layer one. Current methods do not satisfy these premises: the normal way to open a channel is via a multi-signature and it has been shown that this method can not be considered private[30].

3 | BACKGROUND AND REQUIREMENTS

3.1 | Blockchain, Participants and Network Model

We assume the blockchains constitute an asynchronous system lacking of a global clock across chains. For this reason we rely heavily on the result of Asokan[31] on the impossibility of fair exchange without a third parties: its consequence is that there can't be a cross chain communication protocol tolerant against misbehaving nodes with out a trusted third party (see Corollary 1 in[8]). Our goal is to provide a multi-party locking mechanism to remove the asynchronicity assumption. Note that we rely on chain-dependent clocks on the chain. In particular, time is based on block generation. This hinders a possibility of synchronization across chains. We analyze possible attacks based on this in Section 7.2.

We model blockchain transactions as a tuple

$$tx = (fromAddr, toAddr, amt, \{conds\})$$

where $fromAddr$ is the sending address, $toAddr$ is the receiving address, amt is amount of token sent and $conds$ are the conditions needed to redeem tx. In UTXO-based blockchain, $conds$ is never empty because the redeemer has to present at least a signature as a form of zero knowledge proof of knowledge of private key. On the other hand, we assume $conds = \emptyset$ if tx is in an account-based blockchain. For a difference between the two models see Section 3.2.

The MP-HTLC protocol has multiple participants. We call $P_1, \dots, P_N$ the participants that have funds on BC_1 and $Q_1, \dots, Q_N$ the participants that have funds on BC_2. We model the set $\{P_1, \dots, P_N\}$ as a set of participants containing at least one rational participant, P_r and the set $\{Q_1, \dots, Q_N\}$ as a set of participants containing at least one rational participant, Q_r.

We assume $P_1, \dots, P_N$ are "on the same side", meaning that no P_i is actively dishonest towards $\{P_1, \dots, P_N\} \setminus P_i$, while allowing for passive dishonesty, i.e. willing to read messages which are not sent to it (the honest-but-curious model) and the ability for P_i to go offline and not replying to messages sent by $\{P_1, \dots, P_N\} \setminus P_i$ towards P_i. The same goes for $Q_1, \dots, Q_N$.

Note that this assumption means that there can be no meaningful collusion between $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$: if P_i colludes with one of $Q_1, \dots, Q_N$, then if P_i acts on the result of the collusion then P_i would be actively dishonest towards $P_1, \dots, P_N$.

On the other hand, we allow $P_1, \dots, P_N$ to be actively dishonest towards $Q_1, \dots, Q_N$, and viceversa, as long as this does not result in a situation which is equivalent to collusion. In particular, we do not allow e.g. P_i to bribe or otherwise manipulate the behavior of Q_j since then Q_j would act actively dishonest manner towards $Q_1, \dots, Q_N$. Note that this last requirement intuitively "normal" since it is the natural extension of the HTLC between two parties A and B: in that case it is assumed that A can not control B, since if that were not the case, A could steal from B without having to start a HTLC.

Finally, since we present a HTLC-based protocol, we require that the domain of the hash function has the same size in both chains. Possible attacks that can be performed if this assumption is not met are presented in[32]. In practice all examples will use the SHA256 hashing algorithm, so the domain of the hash function will obviously be the same.

3.2 | Account Based vs UTXO Based

There are two major types of models in the blockchain ecosystem. The first model is called the Account (or Balance) model. In this model, the states related to an account are saved inside the account. A transaction in the account-based model is an instruction for how to transition two or more accounts to the next state. The actual transition is executed by the nodes. The final state is not specified in the transaction

The other methodology, called UTXO model, does not contain the logic and states of an account within the account itself, but is constantly recomputed by scanning the entire blockchain. In this type of model, a user sends transactions that contain the results, not the necessary calculations. These are defined as the output of the new transaction, and are expendable by the recipient if they can meet the conditions. In this model the logic is aimed at verification.

An example of a blockchain with an account-based model is Ethereum while an example of a blockchain with an UTXO-based model is Bitcoin. This is one of the reasons why in Sections 5 we choose to make the case study on the exchange between these two blockchain projects.

Finally, since blockchains in an account model have the ability to save states, it is possible to integrate all the logic of the protocol into a single smart contract. As for a UTXO model the logic must be expressed every time in every translation.

This difference is important for our protocol. In the case of a blockchain with an account model we can make a single smart contract containing all the necessary logic. Instead if we are dealing with a UTXO model we will have to make more transactions with similar logic[¶].

3.3 | Hash-Time Lock Contracts

It is a well known result that a fair exchange (as defined in[31]) in an asynchronous system is impossible without a third party[33,34,35,36]. Zamyatin et al[8] reduce the cross-chain communication (CCC) problem to the fair exchange one, proving that there can't be an asynchronous protocol tolerant against misbehaving nodes without a trusted third party (TTP). So for this reason, to have a secure CCC, we have to either introduce synchrony or a TTP. We want MP-HTLC to be a P2P protocol, so we decided to introduce synchrony. We do that by using a locking mechanism called Hash-Time Lock Contract (HTLC)[4]. In a HTLC a transaction from Alice to Bob has two redeeming paths. In the first one, the transaction is redeemable by Bob providing the preimage of a hash before a certain timeout; in the second path, Alice can redeem the transaction after the timeout, i.e. if Bob did not redeem it before. Thus HTLCs work thanks to the preimage resistance property of hash functions.

Atomic swaps (a particular case of CCC) leverage HTLCs in the following way:

1. Suppose Alice has 0.1 BTC and she wants to exchange it with someone for 1 ETH, and suppose Bob has the inverse need
2. If Alice and Bob agree on the exchange, Alice produces a secret r uniformly at random and creates a hash-time locked transaction, such as the one described before, with hash $h = H(r)$ (where H is a hash function) and timeout Δ with Bob's address on the Bitcoin blockchain as the recipient
3. Bob uses h to create a hash-time locked transaction directed to Alice's address on the Ethereum blockchain with timeout $\Delta' < \Delta$
4. Alice redeems the transaction on Ethereum before Δ; by doing that she provides r, which is visible to anybody on the Ethereum blockchain

[¶]The interested reader can find additional information at https://bitcoin.stackexchange.com/a/49872 and at https://eth.wiki/en/fundamentals/design-rationale#accounts-and-not-utxos

5. Bob uses r to redeem the transaction on Bitcoin before Δ

If Alice misbehaves by not following through the protocol after Step 2, then Bob can redeem his own transaction after Δ'. The swap is atomic in the sense that either Alice and Bob get their new coins, or they get their old ones: there is no way one party can lose anything.

3.4 | Threshold Signatures

A threshold signature scheme (TSS) lets a group of participants compute a signature together, without learning information about the private key. Generally in a (t, n)-TSS, n participants hold their distinct private keys (called *key shares*) and any set of $t + 1 \leq n$ distinct participants can produce a valid signature, while any subset of at most t participants can't.

MP-HTLC needs a (n, n)-TSS, so that all participants are needed to produce a signature. This is done to avoid possible collusion between the participants who can in this way steal money from the minority of people. Reasoning of this type are common in the literature, see for example[37].

More formally a TSS has always two algorithms:

- *Thresh-DKGen*, a distributed key generation (DKG) protocol. If the protocol is successful, each participant has its private share key sk_i of the whole private key sk
- *Thresh-Sig*, a distributed signing protocol. If the protocol is successful, it returns a valid signature of a message.

In MP-HTLC, we use threshold signatures to mask the number of participants behind a transaction. In fact, only one signature is produced in a TSS, whatever the number of participants. This is different from multisignatures which produce one signature per participant. For a more detailed description see[38].

For the proof of concept (PoC) presented in this paper, we used the ZenGo ECDSA-TSS library[#]. That's because both Bitcoin and Ethereum currently use the ECDSA signature scheme. In any case, both projects will change the signature scheme. Bitcoin is currently switching to a Schnorr signature scheme[39], while Ethereum is switching to BLS[40]. Both Schnorr[41] and BLS[42] signatures have a variant on which it is possible to create a TSS in Bitcoin and Ethereum. In particular, it is currently recommended[43] to use MuSig2 for the Bitcoin threshold scheme

It is important to note that other signature schemes, such as EdDSA, easily support TSS. In particular, a technical note has recently been published on a TSS applied to Monero Ring Signatures, which are called Thring Signatures[44]. This means that MP-HTLC can in principle be instantiated also between Monero and another blockchain using another synchrony assumption (e.g. signature-based contracts as explained in[45] instead of HTLC as said in Section 3.3).

3.5 | MPC

Similar to HTLC, MP-HTLC also has a secret creation phase. This secret cannot be created by a single participant, who would become a kind of leader in the protocol, decreasing its decentralization and security. For this reason the creation of the secret must be a joint computation performed by all parties involved. That means that we are in a distributed computing scenario and secure multi-party computation (MPC) is used to to be sure that the joint computations are secure: the goal of MPC is for a group of participants to learn the correct output of some previously agreed function applied to their private inputs without revealing anything else. To practically create the secret, the participants will use MPC to sum their private inputs.

In MPC security is defined using the *real-ideal paradigm*. In the ideal world parties interact with a third party which is completely trusted, cannot be attacked and would never betray any of the parties. Of course we can't use the same model for the real world, because no entity can be absolutely trusted. In the real world the third part is replaced by a protocol. Intuitively, this protocol is considered secure if any consequences caused by an adversary in the protocol in the real world could also be achieved in the ideal world.

There are two different adversary models commonly used for MPC: *semi-honest* and *malicious* adversary. A semi-honest adversary is one who corrupts parties but follows the protocol as specified. This is the case where parties can collude by pooling their views (i.e. private inputs) together. This kind of adversary is considered passive, because they cannot act on the knowledge gained. Another name for semi-honest is *honest but curious* adversary.

[#]See the GitHub repository at https://github.com/ZenGo-X/multi-party-ecdsa

The other kind of adversary is the *malicious* (also called active) one. This kind of adversary may cause corrupted parties to deviate arbitrarily from the prescribed protocol. A malicious adversary has all powers of a semi-honest one, but it can also act on the knowledge it gains manipulating the outputs of the protocol. For more details see for example[46].

To implement this part of the protocol, we used the framework proposed by[47]. The framework proposes several methods to perform MPC protocols between two or more participants in different types of adversaries. Given the modularity of MP-HTLC, the choice of the specific MPC protocol does not affect the other routines. For this reason the proof of concept proposed in this paper uses the Tale protocol to do the sum.

3.6 | P2SH Bitcoin Transactions

In Bitcoin, a transaction tx is a transfer of value (*coins*) published to the network. Many transactions are collected into blocks. Bitcoin uses a Unspent Transaction output (UTXO) model, see Section 3.2. In this model, each transaction tx references previous transactions $tx_1, \dots, tx_n$: it is said that these transactions are *spent*. Transaction tx is currently unspent and the one or more addresses it sends money to are the *outputs* of tx. The spent outputs of $tx_1, \dots, tx_n$ are said to be the *inputs* of tx. So the input(s) of a transaction are the output(s) of previous transaction(s).

Bitcoin supports very basic smart contracts which allow different types of scripts. Scripts are divided into standard and non-standard which contain the logic to redeem a transaction. By default non-standard scripts are not accepted by miners who do not include them in their blocks, even though they are valid. This is to avoid possible DDoS[48].

Currently the standard types are

- Pay To Public Key Hash (P2PKH)
- Pay To Script Hash (P2SH)
- Multisig
- Pubkey
- Null Data

For more details on current standard scripts see the page dedicated to bitcoin developers[‖].

Each transaction is divided into two parts. One part determines the conditions needed to redeem the transaction and it's called `scriptPubKey`. The other part contains data to satisfy those conditions and it's called `scriptSig`.

We only use P2SH transactions in the course of the protocol. This transaction type allows users to embed non-standard scripts into a standard script creating an acceptable transaction. This allows a sender to fund any arbitrary transaction, no matter how complicated its script. The format of this type of transaction is as follows[49]:

```
OP_HASH160 [20-byte-hash-value] OP_EQUAL
```

where `[20-byte-hash-value]` is the hash of the serialized non standard script.

This kind of transaction can be redeemed by a standard `scriptSig`:

```
...data... {serialized script}
```

We use this construct in two phases of the protocol. We explain in the phase sections the particular script.

3.7 | Partially Signed Bitcoin Transactions

Some times multiple parties need to cooperate to produce a transaction. Examples include multisignature setups, transferring funds form or to cold or hardware wallets, and CoinJoin transactions. In these cases, one party A may need to exchange an unsigned or partially-signed transaction with another party B so that A and B together can send these funds to a third party C. Originally this process was wallet-implementation dependent, making it hard for people who use different wallet softwares to

[‖] See https://developer.bitcoin.org/devguide/transactions.html

exchange these partially signed transactions. This problem has been solved with BIP174[50] and BIP370[51]. Those BIPs create an interchange format for Bitcoin transactions called Partially Signed Bitcoin Transaction (PSBT).

To give an example of how PSBT work, we describe the process in the case mentioned before, i.e. where a party A and a party B need to send a payment to C using inputs from both A and B. The construction goes through the following steps[**]:

1. Without loss of generality, A proposes a particular transaction to be created by building a PSBT ptx that contains certain inputs and outputs; no additional metadata is needed. A sends ptx to B
2. B adds information about the UTXOs being spent by the transaction to ptx. In particular B adds its inputs
3. Signers A and B inspect the PSBT transaction ptx and its metadata to decide whether they agree with the transaction.
4. If they agree, they produce a partial signature for the inputs for which they have relevant keys. Without loss of generality A sends the current state of ptx to B
5. B runs an extractor protocol to produce a valid Bitcoin transaction from ptx if all inputs are finalized.

In practice PSBTs let Bitcoin users aggregate inputs and outputs in a single transaction. This helps them reduce fees by requiring them to broadcast only one transaction to the chain instead of many. We take advantage of this fact during the Commit Phase in the Bitcoin transaction to make only one signature (see Section 5.2)

PSBT are already working on both Bitcoin, Bitcoin Cash and its subsequent forks[††]. A version of PSBT called Partially Created Zcash Transactions (PCZT) are currently in the process of being finalized[‡‡].

4 | DESIGN

The goal is to make a multiparty-swap protocol between two blockchains minimizing the number of transactions needed while ensuring the same level of security. We do that by generalizing the HTLC to multiple participants (multiparty-HTLC) with the same secret. In the following we will be blockchain agnostic. We call blockchain BC_1 the first blockchain and blockchain BC_2 the second blockchain. The N participants in blockchain BC_1 will be $P_1, \dots, P_N$ and the N participants in blockchain BC_2 will be $Q_1, \dots, Q_N$. In the course of the explanation, $P_1, \dots, P_N$ will also be called the initiators and $Q_1, \dots, Q_N$ the finalizers. The protocol is easily extendable to the case where a P_i serves both Q_i and Q_j or where a P_i is served by Q_i and Q_j for some i, j, so that there is a different number of participants in the two blockchains, say N in BC_1 and N' in BC_2. In fact, we can represent a participant as dual, instead of a single participant with multiple funds, so that we end up having the same number of participants in both blockchains.

The only assumption on both blockchains is that they are able to understand the same hash function and have a basic scripting language. An example blockchain is Bitcoin, but also Ethereum or Tezos are supported. On the other hand blockchains like Monero or Komodo cannot be used as they do not have a scripting language.

In the course of explaining the design in the protocol, we will refer to the two types of blockchain: account-based blockchains and UTXO-based blockchains. The protocol is divided into three phases: Precommit, Commit and Redeem. Table 1 gives an overview of the whole protocol and the different steps needed in the UTXO and Account based models.

4.1 | Precommitting Phase

In the first part of the protocol, there are two objectives:

- *Creation of the secret*: this part is similar to the creation of the secret in HTLC and is done only by one group of participants.
- *Creation of the aggregate public key*: this is done by everyone and it is necessary for all participants to have the same possibility to manipulate the money.

In the following we will explain in detail these two subroutines. This phase is carried out in parallel by the two groups.

[**] See https://github.com/bitcoin/bitcoin/blob/master/doc/psbt.md#psbt-in-general for details
[††] https://docs.bitcoincashnode.org/doc/release-notes/release-notes-0.20.6/
[‡‡] https://github.com/zcash/zcash/issues/2542

	UTXO Based	Account Based
Precommitting Phase	Secret Creation (MPC)	Secret Creation (MPC)
	Aggregate PubKey Creation	Aggregate PubKey Creation
		Smart Contract deployment
Committing Phase	P2SH-tx sending to AggPubKey	Smart Contract funding
Redeeming Phase	P2SH-tx sending to receivers	Smart Contract activation
		Smart Contract sends transactions to receivers

TABLE 1 Phases of the MP-HTLC protocol in the UTXO and Account based models. The Account based model needs more steps than the UTXO one. Note that the Secret Creation step via Multiparty computation methods is performed by Initializers only.

Secret creation

Participants $P_1, \dots, P_N$ undertake a process of creating a secret. To do so, participants use a multiparty computation protocol with the goal of obtaining an output from hidden inputs. Formally, $P_1, \dots, P_N$ from private inputs $(x_1, \dots, x_n)$ create an output $y = f(x_1, \dots, x_n)$. Examples for a function f are $sum(x_1, \dots, x_n)$, $max(x_1, \dots, x_n)$ or $min(x_1 \dots, x_n)$. To be more concrete, let's assume that f is the sum of the private inputs.

After creating this secret number, all participants can hash it, similar to what happens in a HTLC. The secret is therefore y and the hash is $h := H(y)$, where H is a hash function that can be computed on both blockchain BC_1 and blockchain BC_2

Aggregated Key

Participants of both blockchains create an aggregate key using threshold signatures. Threshold signatures are supported by the major signature schemes that are used (or planning to be used) in each blockchain. Examples of digital signature schemes that support threshold signatures are ECDSA, Schnorr, EdDSA, and BLS signature. This is necessary because participants need a shared key to manage the transfer of coins: if the key weren't shared among all people, then there would be a leader. And this is not acceptable since we want a peer-to-peer protocol.

Remember that the aggregate key is inherently public, as explained in Section 3.3, and that we don't know its private key and that the public key is different from the address on the blockchain. In fact, the address on the blockchain is a derivation of the public key. In the case of Bitcoin for example the address is a derivative of the script used within the transaction.

In this part we see the difference between a blockchain with an account-based model and one with a UTXO-based model. A blockchain with UTXO-based model uses particular constructs within the transaction, while a blockchain with account-based model deploys a Smart Contract.

Whatever mechanism is used in this phase, it must have the ability to be inactive for this phase. In fact, if it were active the other participants could take the tokens without giving their own in return. Inactivity can be achieved with time-locks or activation switches in the smart contract.

This mechanism must have two possibilities of redemption of funds. The first is to be used in case all participants are honest. In practice this means that the money goes from the group of participants $P_1, \dots, P_N$ to $Q_1, \dots, Q_N$ or vice versa. The second way of redemption is instead used in the event that the other participants are dishonest: assuming that the mechanism is initiated by $P_1, \dots, P_N$, the second way gives them the possibility to take back their tokens in case $Q_1, \dots, Q_N$ behave dishonestly.

4.2 | Commit Phase

At this phase, the parties commit to sending money. Participants in both sets send transactions, whether this is a particular transaction or a transaction to the smart contract, in order to prove that they have the tokens and are willing to continue the protocol. In order not to confuse the transactions that occur in the different phases, we will call these transactions the *inTx* transactions, while the other transactions in the Redeeming Phase will be called the *outTx* transactions. Transactions *inTx* can be redeemed by $P_1, \dots, P_N$ together before a time-out Δ, and by the single P_i after that, similarly to the normal HTLC case (see Section 3.3)

These transactions will be understood in detail in Section 5 where we present the case study between Bitcoin and Ethereum. This is the end of the Commit Phase.

	Bitcoin	Ethereum
Precommitting Phase	Secret Creation (MPC)	Aggregate PubKey Creation
	Aggregate PubKey Creation	Smart Contract deployment
Committing Phase	P2SH-tx sending to AggPubKey	Smart Contract funding
Redeeming Phase	P2SH-tx sending to receivers	
	Smart Contract activation	
	Smart Contract sends transactions to receivers	

TABLE 2 Steps for the MP-HTLC protocol between the Bitcoin and Ethereum blockchain. The Precommit and Commit phases are done in parallel. The Redeeming phase is sequential.

4.3 | Redeem Phase

The last step is performed sequentially. Each step is done first by one group of participants and then by the other group of participants, when the latter receives confirmation that the intermediate step has been done. The goal of the participants in this step is to redeem all the tokens.

The protocol initiators $P_1, \dots, P_N$ perform the first *outTxes* to redeem the transactions or smart contract of the finalizers $Q_1, \dots, Q_N$.

In the case of a blockchain with UTXO model, this type of transaction also has paths to be redeemed: the first if the finalizers are honest and the second in case they are dishonest. As in the Commit Phase, paths are time dependent, i.e. transactions *outTx* can be redeemed by $Q_1, \dots, Q_N$ before a time-out Δ', and by $P_1, \dots, P_N$ after that, again similarly to the normal HTLC case (see Section 3.3).

These transactions can be signed with the aggregate key created in the first step, contain h and can be redeemed by the finalizers using the y secret. In the case of a blockchain capable of supporting Smart Contracts, the logic is already all built into it and since the Smart Contract is already deployed in the first phase, this transaction simply consists of activating it (see functions `activationSwitch` and `isSha256Preimage` in Section 5).

After finalizers see the transactions made by initializers, they can proceed with their transactions and redeem the funds on the other blockchain, having seen the secret y in the blockchain.

5 | ETHEREUM-BITCOIN CASE STUDY

For the purpose of this paper, we create a proof of concept of a MP-HTLC between the Bitcoin and the Ethereum blockchains. We made this choice because the former blockchain has a UTXO-based model while the latter has an account-based model.

Throughout this section, the notation changes slightly from the previous section to make it more intuitive according to the case study. Table 2 give a schematic view of what happens. For easiness of explanation we distinguish between “bitcoiners", participants who have bitcoin and want ethers, and “etherers”, participants who have ethers and want bitcoin. Formally, we assume there are $2N$ participants such that $B_1, \dots, B_N$ are bitcoiners and $E_1, \dots, E_N$ are etherers. We proceed with the description of the three phases of the MP-HTLC protocol.

5.1 | Precommitting Phase

In the first phase we assume that each B_i has already chosen an E_i and exchanged with it all the necessary information to proceed to a classical atomic swap (e.g. pubkeys, BTC/ETH rate, timeouts, and amount of coins). This part is outside the scope of the paper and can be performed via public billboards, order-books or, theoretically, even with Automated Market Makers.

In this phase and the next one, bitcoiners and etherers proceed in parallel without interaction. For ease of explanation, however, we first explain what happens between bitcoiners and then what happens between etherers.

Bitcoiners

Bitcoiners have two goals for the precommitting phase: to create a secret and its hash for the MP-HTLC, and to create a shared public key. Bitcoiners $B_1, \dots, B_N$ use a distributed key generation mechanism $Thresh - DKGen$ to create an aggregate public key $AggPk^{BTC}$.

Then bitcoiners $B_1, \dots, B_N$ create a secret all together (see Section 3.3). It is necessary that the secret is created by all bitcoiners, otherwise there would be a leader. For this reason the secret s is created in multiparty computation, as described in Section 3.5. From the secret s, all participants can build the same hash $h = H(s)$ (where $H(\cdot)$ is a hash function) and use it in the second set of transactions in the Redeem Phase.

After that, each B_i is responsible to send h to E_i. Although it is not necessary for everyone to do this from a technical point of view, this overhead is necessary from a social point of view: this way each etherer E_i does not have to trust other etherers $E_1, \dots, E_{i-1}, E_{i+1}, \dots E_N$.

Etherers

Etherers have two goals for the precommitting phase: to create a shared public key, and to create and (after receiving h) deploy a smart contract SC. The smart contract contains the necessary logic for either receiving and sending funds based on the presentation of a preimage of the hash h or returning of funds to the owners. This smart contract is the analogue of the P2SH transactions made by bitcoiners. Etherers first use the same distributed key generation mechanism used by bitcoiners to create an aggregate public key $AggPk^{ETH}$.

This step is also necessary in an account-based blockchain like Ethereum. This is because many of the operations can only be done by the person who creates the smart contract. Formally the smart contract must be deployed on the blockchain using the address associated with the public key $AggPk^{ETH}$.

Deploying a smart contract means publishing a transaction to a particular address. For this reason it is possible to use threshold signatures of $E_1, \dots, E_N$ to sign and publish it.

The main functions of the smart contract [§§] are:

- `activationSwitch`: this function makes the smart contract active; before SC is activated, sending a preimage of the hash (however correct) does not allow the release of the coins contained in the smart contract
- `isSha256Preimage`: this function takes care of sending funds to bitcoiners upon presentation of the right h hash preimage if SC has been previously activated via the `activationSwitch` function

As of now the smart contract is deployed but unfunded and not activated. Both the funds and the activation arrive in the Commit Phase. This concludes the precommitting phase

5.2 | Commit Phase

Bitcoiners

In this phase, the goal of Bitcoiners is to fund $AggPk^{BTC}$ with a transaction. Using PSBT (see Section 3.7) $B_1, \dots, B_N$ are able to send only one transaction with multiple inputs and only one output. We call this transaction ptx

Using both the aggregate public key $AggPk^{BTC}$ and the hash h, each bitcoiner B_i creates a P2SH transaction (see Section 3.6 we call ${}_{in}tx_i^{BTC}$. The locking script of this transaction has two redeeming paths. The first path allows redemption before a timeout ${}_{in}\Delta$ by signing the redeeming transaction via a threshold signature. That's because, for this redeeming path, the redeeming transaction has to be valid when verified by $AggPk^{BTC}$. The second redeeming path allows B_i to redeem ${}_{in}tx_i^{BTC}$ as the timeout ${}_{in}\Delta$ passes using a signature verifiable by some public key Pk_i^{BTC} owned (together with the related private key) by B_i. In the Bitcoin Script language, this output script looks similar to the following one where we put `InDelta=`${}_{in}\Delta$, `AggPkBtc=`$AggPk^{BTC}$ and `BiPkBtc=`Pk_i^{BTC}:

```
IF
    <now + InDelta> CHECKLOCKTIMEVERIFY DROP
     <BiPkBtc> CHECKSIGVERIFY
ELSE
```

[§§] See the smart contract at https://github.com/disnocen/mp-htlc/blob/master/Interop.sol

```
    <AggPkBtc> CHECKSIGVERIFY
ENDIF
```

These transactions are P2SH scripts that produce an address $SHAddr_i^{BTC}$. Note that $SHAddr_i^{BTC}$ is different for script. The address format doesn't take amounts, private keys or transaction hashes into consideration. However, given that public key Pk_i^{BTC} is different for each B_i in the second redeeming path, the redeeming script is different. Therefore there are multiple outputs in the PSBT ptx.

Then all the $B_1, \dots, B_N$ complete together the PSBT-sequence and broadcast the transaction. they get transaction hash ${}_{in}txHash^{BTC}$ and they inform $E_1, \dots, E_N$ about this hash.

Etherers

Having created the smart contract SC in the precommitting phase with all the necessary logic, this phase is easier for etherers. $E_1, \dots, E_N$ just send their funds (plus fees) to the smart contract. Then they wait for bitcoiners $B_1, \dots, B_N$ in the redeem phase. Now SC is funded, but still inactive

5.3 | Redeem Phase

This phase is sequential. If each bitcoiner B_i informs E_i about ${}_{in}txi^{BTC}$ and if the etherers complete their part of the precommitting and committing phases (visible on the blockchain, so they can keep track), then all participants start the last phase of the protocol. The goal of this phase is the actual transfer of funds from $SHAddr_i^{BTC}, \quad i = 1, \dots, N$ to etherers $E_1, \dots, E_N$ on the Bitcoin blockchain and from the smart contract SC to bitcoiners $B_1, \dots, B_N$.

At the formal level, $B_1, \dots, B_N$ must make a single transaction ${}_{out}tx^{BTC}$ that has as input the transaction outputs identified by the ${}_{in}txHash^{BTC}$ and as output Bitcoin addresses redeemable by $E_1, \dots, E_N$. $B_1, \dots, B_N$ must then produce one threshold signature to publish this transaction.

The output addresses in this transaction are derived from P2SH transactions as in the Commit Phase. In fact each output has two redeeming paths as well (one before and one after a timeout we call ${}_{out}\Delta$, which is generally different from ${}_{in}\Delta$) to defend bitcoiners from possible non-activation by etherers. So this transaction is not a standard one and needs to be encapsulated in a P2SH transaction. For each i-th output, the first redeeming path expires at timeout ${}_{out}\Delta$ and needs the preimage of hash h, as well as the signature verifiable from the public key provided by E_i. The second redeeming path becomes active after the timeout ${}_{out}\Delta$ and needs only the signature verifiable by the public key of B_i. We stress that ${}_{out}tx^{BTC}$ is a single transaction with multiple output addresses.

For each output address, its locking script is (where `OutDelta`=${}_{out}\Delta$, `h`=h, `EiPkBtc` is the Bitcoin public key provided by E_i and `BiPkBtc`=Pk_i^{BTC}):

```
IF
    <now + OutDelta> CHECKLOCKTIMEVERIFY DROP
     <BiPkBtc> CHECKSIGVERIFY
ELSE
    SHA256 h EQUAL <EiPkBtc> CHECKSIGVERIFY
ENDIF
```

The etherers know when the transaction they are interested in has been published by the bitcoiners because knowing ${}_{out}\Delta$ they can compose the P2SH script and therefore they can monitor the P2SH address. When etherers see the address funded, they activate the smart contract using the `activationSwitch` function. After the activation they wait for one of $B_1, \dots, B_N$ to send the preimage into the smart contract. This triggers the sending of coins to the addresses of bitcoiners $B_1, \dots, B_N$ on Ethereum.

When this happens, each etherer E_i can learn about the preimage by parsing the blockchain. With this preimage, each E_i can redeem the i-th output of ${}_{out}tx^{BTC}$.

The protocol is then concluded.

5.4 | Implementation

We propose an implementation of this use case. The link can be found here[¶¶].

To implement the threshold signature we used the ZenGo project[##]. We modified the demo so that it would sign particular digests of transactions from the input. In particular we used the distributed key generation and the distribute signing protocol from[52]. This is one possible choice of many and we could use other threshold protocols. Then we proceeded to create scripts for both Bitcoin and Ethereum. To explain what we did we split this explanation into two parts

Bitcoin

We used the bitcoin-libs Python 3 library to create the transactions. We modified the library to make the transaction signed by the ZenGo library.

Namely we did five steps:

- we create the aggregated key using the distributed key generation algorithm of the rust library and derived a P2PKH address; we call this address $AggAddrBTC$
- we sent coins to this $AggAddrBTC$
- we created a transaction to send the money to another address, we call this address $NewAddr$, and we get the digest.
- we used the rust library's API to sign in a distributed way the digest
- we sent the transaction[|||]

Ethereum

In the case of Ethereum we made these steps:

- we create the aggregated key using the distributed key generation algorithm of the rust library and derived an Ethereum address; we call this address $AggAddrETH$
- we created a smart contract able to manage the funds, as explained in the previous subsection; the owner of the smart contract is the $AggAddrETH$
- we modified the `ethereumjs/tx` creating a new function we called `signTh`: this function calls the rust library to sign the transactions so that it is verifiable using the aggregated public key behind $AggAddrETH$

6 | EVM-COMPATIBLE BLOCKCHAINS CASE STUDY

Here we present another use case, namely a Ethereum-Polygon token exchange. After reading the section, it will be easy to see how this way to exchange tokens between blockchains can be used for any couple of EVM-compatible blockchain, effectively creating an alternative to centralized bridges such as the Binance smart chain bridge or the Polygon one. This use case is different from the previous one in two ways. The first difference concerns the model of the blockchains involved. In fact, both the Ethereum and Polygon blockchains are account-based: in this case MP-HTLC is smart contract based on both sides. The second difference is in terms of purpose. As stated by the Polygon whitepaper, the project aims to be an interoperable sidechain of Ethereum to achieve scalability. Polygon has faster block-creation time and cheaper fees. Currently many blockchain DApps are creating Polygon version of their services, especially DeFi services like AAVE [†], SushiSwap [‡] or Uniswap[53]. Note that the idea of using new EVM-compatible blockchains to solve Ethereum's scalability problems is not new and is also used by blockchain-based video games such as Axie Infinity [§] which created its own EVM-compatible blockchain for game management.

[¶¶] https://github.com/disnocen/mp-htlc
[##] https://github.com/ZenGo-X/multi-party-ecdsa
[|||] You can see the transaction at this URL: https://live.blockcypher.com/btc-testnet/tx/40b0dcfdfc6461f79aabf88660f50adc03a32c0a3c40b5d4af5ddf161243909b/
[†] https://aave.com
[‡] https://sushi.com
[§] https://axieinfinity.com

Of course Polygon already provides a *bridge* to exchange tokens between the Polygon network and Ethereum, but this bridge is centralized and therefore is subject to possible censorship and theft. Here we propose a decentralized alternative using MP-HTLC.

To understand why our alternative can substitute the whole bridge, the reader should recall two things. The first one is that every token on a EVM-compatible blockchain are the result of the deploy of a smart contract which then manages the balances of the users. The second thing to remember it that given two EVM-compatible smart contracts SC_A and SC_B, it is possible to call a function in SC_B from SC_A. We use the notation $SC_B.functionName(\cdot)$ to symbolize the call from SC_A. Therefore the MP-HTLC protocol can be used to exchange tokens based on smart contracts that have a version deployed on the Ethereum blockchain and one deployed in the Polygon blockchain. Some examples of these tokens are the USDT token or the WrappedBTC token[¶].

We model the group of participants in this way. We assume there are N participants that own some amount of $TokenA$ on the Ethereum blockchain and want the same amount of $TokenA$ on the Polygon blockchain. We denote these participant as $E_1^A, \dots, E_N^A$ and we call them etherers, as in Section 5. Similarly we assume there are N participants that own some amount of $TokenA$ on the Polygon blockchain and want the same amount of $TokenA$ on the Ethereum blockchain. We denote these participant as $P_1^A, \dots, P_N^A$ and we call them polygoners. So there are $2N$ participants in total

We proceed with the description of the three phases of the MP-HTLC protocol in the case of EVM-compatibility blockchains.

6.1 | Precommitting Phase

In the first phase we assume that each E_i^A has already chosen an P_i^A and exchanged with it all the necessary information to proceed to a classical atomic swap (e.g. pubkeys, timeouts, and amount of coins). Note that there is no rate of exchange for $TokenA$ because we are assuming this token is pegged. As in the previous use case, how the participants exchange this data is outside the scope of the paper and can be performed via secure chats.

In this phase and the next one, etherers and polygoners proceed in parallel without interaction. For ease of explanation, however, we first explain what happens between etherers then what happens between polygoners, so in practice without loss of generality we assume etherers will start the process.

Etherers

Etherers have three goals for the precommitting phase: to create a secret and its hash for the multi-HTLC, to create a shared public key and to use this public key $AggPk^{ETH}$ as the identity (address) for deploying the smart contract SC_{ETH} that containing the necessary logic for managing the funds.

In practice etherers $E_1^A, \dots, E_N^A$ use the distributed key generation mechanism $Thresh-DKGen$ to create an aggregate public key $AggPk^{ETH}$ and MPC to create a secret all together, as in the previous use case. From the secret s, all participants can build the same hash $h = H(s)$ and use it in the second set of transactions in the Redeem Phase.

After that, each E_i^A is responsible to send h to P_i^A. The smart contract is the same as the one in the previous use case. The only difference is the function to send money. As explained before, there is a function call from SC_{ETH} to the smart contract of $TokenA$ on Ethereum. Formally the function is:

$$TokenA_{ETH}.send(_to_, _amount)$$

which is called if the `checkHash` function returns `true`

Polygoners

Polygoners have two goals for the precommitting phase: to create a shared public key, and to create and (after receiving h) deploy the smart contract SC_{POL}. Polygoners first use the same distributed key generation mechanism used by etherers to create an aggregate public key $AggPk^{POL}$. 1

The smart contract is the same as the one in the Ethereum case. The function call from SC_{POL} to the smart contract of $TokenA$ on Polygon is:

$$TokenA_{POL}.send(to, amount)$$

which is called if the `checkHash` function on SC_{POL} returns `true`

[¶]Wrapped tokens are pegged to the original blockchain but transferable on the Ethereum or Polygon blockchain.

As of now both smart contracts are deployed but unfunded and not activated. Both the funds and the activation arrive in the Commit Phase. This concludes the precommitting phase

6.2 | Commit Phase

In this phase participants on both chains send funds to the send the funds to their respective smart contracts. Now SC_{ETH} and SC_{POL} are funded, but still inactive

6.3 | Redeem Phase

Similarly as the Redeem Phase of Section 5, participants activate their relative smart contract. After doing that etherers redeem the funds in SC_{POL} by sending the preimage of h. Polygoners are therefore able to get the funds from SC_{ETH}.

7 | ANALYSIS

In the following we present a cost analysis where we prove that MP-HTLC can be superior to HTLC in terms of number of transactions needed and an attack analysis. Finally we present a note on the security of the protocol.

7.1 | Cost Analysis

In this section we analyze the costs of our MP-HTLC. We compare it with the standard HTLC. We prove that as long as there are at least two participants on a UTXO-based blockchain, then it is more cost effective to use MP-HTLC instead of instantiating different HTLCs.

In the standard HTLC as defined in[4] (and explained in Section 3.3) there is a best case scenario and a worst case scenario. Still, because the parties exchange signatures and do not broadcast transaction, then the number of transaction is the same regardless of the best/worst scenario. In fact, according to the HTLC protocol there are two transactions published on chain to start the exchange (one per party, and therefore one per blockchain), and two other transactions published on chain if the exchange is successful. Therefore the parties will publish a total of four transactions per instantiation of the protocol

For the MP-HTLC protocol there are two different implementation of the swap. One for the UTXO model and one for the account based model. We count the number of transactions for each phase in both the UTXO and account model assuming there are n participants on each blockchain. In Table 3 we put a summary of the number of transaction for each phase and each model.

Precommit Phase

In the UTXO model there is no need to do any transaction at this point, so in this phase there are 0 transactions done by these participants. On the other hand, participants on an account-based blockchain have to deploy a smart contract. This is the first transaction they have to do.

Commit Phase

Thanks to PSBT (see Section 3.7), participants in the UTXO model broadcast only one transaction, regardless of the number of inputs. On the other hand, in the account-base model all participants have to fund smart contract on the blockchain. Therefore there is 1 transaction in the UTXO model and there are n transaction the account model model.

Redeem Phase

In this last phase, participants on UTXO-based blockchain have a clear advantage. In fact it is really easy to batch transactions in UTXO-based blockchain because transactions can have multiple outputs.

Batching is currently impossible on account-based blockchain, even if there are multiple proposals that aim to address and solve this problem, especially for EVM-compatible blockchain[54,55,56,57]. If it will be possible to batch more transactions into one, then the number of transaction in an account-based blockchain will be the same as the number of transactions needed in a UTXO-based blockchain.

	Number of Transactions						
	HTLC			MP-HTLC			
	1st phase	2nd phase	TOT	Precommit	Commit	Redeem	TOT
Acct. based	n	n	$2n$	1	n	$n+1$	$2n+2$
UTXO based	n	n	$2n$	0	1	1	**2**

TABLE 3 The table describes the number of transactions required in the two implementations. The number of participants n is meant for each blockchain. This means that a token exchange needs a total of $2n$ participants between the two blockchains.

7.2 | Attack Analysis

Multiple-blockchains modeling is more complicated than treating each part singularly, as there is extra complexity underlying the cross-chain communication. In fact, even if each blockchain has its own security model, these models can be different between blockchains. For example: the Bitcoin blockchain Proof-of-Work-based consensus algorithm is proved to work correctly only if the adversarial computation power is less than 33%[58]. On the other hand, some Proof-of-Stake-based consensus algorithms are proved to work correctly only if the total adversarial stake is less than 33%. Even if the two values are the same, the cost of attaining them is potentially different.

We decided to address the two blockchain attacks that have reorganization of blocks (also called *reorgs*) as a consequence: other attack would have no effect on the transactions. Note that there is no 51% attack. This is because it is not a real attack formally, but it is a state of the blockchain nodes that allows the implementation of real attacks, such as censorship or double spending. So in this sense, we do not directly mention the 51% attack, but we do mention its effects.

Double Spending

This is the most famous blockchain attack and it is even mentioned in the Satoshi's Bitcoin paper[16]. The goal is to trick two different entities (e.g. two different merchants) into assuming they have been paid, using the same coins (technically, the same transaction output(s) in a UTXO-model blockchain, see Section 3.2). When this attack is successful, only one of those transactions is definitely accepted by the blockchain network, but the attacker receives both products from both vendors.

This is how this attack would play out in the MP-HTLC protocol. Adversary party A claims it wants to exchange v_A coins of blockchain BC_A for an equivalent amount of v_B coins of blockchain BC_B. These coins are currently owned by party B. So A starts participating to the protocol and it sends v_A to the aggregated address $AggAddr_A$ on BC_A, but it also sends v_A to another BC_A-address $OthAddr_A$. If A can trick B into believing it is honestly behaving and B continues to follow the protocol and A's double-spend attempt is successful, then A would get v_B coins of blockchain BC_B and retain v_A coins in the BC_A-address $OthAddr_A$. B would lose its v_B coins of blockchain BC_B without gaining any v_A coins.

The easiest way to deal with this attack is by waiting for the confirmation period. As explained in Section 3.1 there is no global or shared clock. So "time" is defined as the passing of blocks' generation. In this case, participant have to wait for the confirmation period of their chain before assuming the current phase is finished and starting the next phase of the protocol. Other works, such as Sai et al.[59], propose methods to punish the double spend problem across two blockchains.

Similar to the Double spend attack, the Finney and Brute force attacks aim to spend the same coins twice. they are different on the social level. For example, the Brute-Force attack assumes the attacker can control a lot of computing power and it will use this power to perform the double spend. Because the mechanics of the attack are the same, we don't further analyze these variation.

Selfish Mining and Selfish Endorsing

Eyal and Sirer propose the selfish mining attack[58], where a malicious miners is incentivized to deviate from the honest protocol withholding valid blocks until it mines two consecutive blocks. If this strategy is successful, then the miner gets double the reward because both of its blocks gets accepted forming the longer chain[#]. In the following years Sapirshtein et al.[60] identify the optimal selfish-mining policy, Nayak et al.[61], consider network attacks, and Kwon et al.[62], consider the impact of selfish mining in the context of mining pools. Still, these works treat selfish mining in the context of a Proof-of-Work consensus algorithm.

[#]Technically in both the GHOST and Nakamoto consensus protocols, the winning chain is the chain with the most work. In our scenario these two concepts are equivalent, so we model it using the more intuitive one

The work of Neuder et al. present the *selfish behavior* attack[63], that is the selfish mining attack in the context of a Proof-of-Stake blockchain (Tezos in particular). This attack incentivize rational bakers ‖ to ignore the longest-chain rule and to create a separate two-block fork faster than the rest of the network can publish two blocks.

This attack would disrupt the procedure of the protocol because some transactions could be effectively erased from the blockchain. The solution is similar to the solution for the Double spend attack: parties should wait for the confirmation period of both blockchains before advancing to the next phase.

7.3 | A Note on Correctness

Goal of this section is to prove that the MP-HTLC protocol is correct in the same sense a HTLC is correct. Intuitively, that is because MP-HTLC follows the same construction of HTLCs, if we abstract the participants on one side as a single entity. Yet, we have to prove that this abstractions is well defined, meaning that we have to prove that (using the general notation of Section 4) a cheating participant between $P_1, \dots, P_N$ or $Q_1, \dots, Q_N$ would not endanger the correctness of the protocol. We will prove it by showing that if there is a dishonest participant $\widehat{P}_j$ between $P_1, \dots, P_N$, then the protocol is still correct if $Q_1, \dots, Q_N$ treat $P_1, \dots, P_N$ as all dishonest participants and abort the protocol. The same goes if there is a dishonest participant $\widehat{Q_j}$ between $Q_1, \dots, Q_N$.

We do that in two steps. In the first step we assume participants can only cheat by missing timeouts. Then we let participants cheat by sending arbitrary messages. Note that in Section 3.1 we stated that $P_1, \dots, P_N$ may be actively dishonest only towards $Q_1, \dots, Q_N$ and passively dishonest towards all participants in the MP-HTLC protocol, and that the same goes for $Q_1, \dots, Q_N$.

We need to reason around the concept of "time", an essential concept to talk about timeouts in a meaningful way. Throughout the paper we assumed asynchronous systems and a block-based time tracking (see Section 3.1). Assuming a constant rate of block production for each blockchain (e.g. at most 15 blocks produced in Ethereum per one block produced in Bitcoin) is enough to ensure correctness of the HTLC protocol between two parties (single entities) Alice and Bob. The question is under which assumptions this reasoning holds true in the case of multiple parties for each blockchain.

The main source of problems comes from the fact that in normal HTLCs between two parties A and B, one entity A either respects or does not respect the timeout, with no middle cases. In case A does not respect the timeout, then B acts accordingly. On the other hand, in case of multiple participants $P_1, \dots, P_N$ it is possible to have a situation in which *some* participants have been honest and respected the timeout, while others did not. We denote as $\widehat{P}_j$ the dishonest participant in the set $\{P_1, \dots, P_N\}$. In such a case, we have implicitly considered the protocol as aborting, i.e. we assumed that participants $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ stop the protocol and redeem the coins they already invested. We have to prove that this behavior does not affect the correctness of the protocol.

We start by stating what *correct* means in MP-HTLC and then we prove that the protocol is correct even in the aforementioned case.

Definition 7.1. [Correctness] Let $P_1, \dots, P_N$ be participants in blockchain BC_1, powered by coin c_1, who owns amounts $a_1^1, \dots, a_N^1$ of coin c_1 respectively. Similarly, let $Q_1, \dots, Q_N$ be participants in blockchain BC_2, powered by coin c_2, who owns amounts $a_1^2, \dots, a_N^2$ of coin c_2 respectively. Finally, assume $c_1 = \rho c_2$ with ρ the agreed exchange rate between c_1 and c_2 and $\frac{a_i^1}{a_i^2} = \frac{\rho c_2}{c_1}$, so that the value $a_i^1 c_1$ has the same (monetary) value as $a_i^2 c_2$ for each $i = 1 \dots N$. Then the protocol as described in Section 4 is *correct* under the model of Section 3.1 if and only if at the end of the protocol:

1. Either any rational participant in $P_1, \dots, P_N$ obtained amounts $a_1^2, \dots, a_N^2$ of coin c_2 and any rational participant in $Q_1, \dots, Q_N$ obtained amounts $a_1^1, \dots, a_N^1$ of coin c_1
2. Or any rational participant in $P_1, \dots, P_N$ still own amounts $a_1^1, \dots, a_N^1$ of coin c_1 and any rational participant in $Q_1, \dots, Q_N$ own amounts $a_1^2, \dots, a_N^2$ of coin c_2

In other words, Definition 7.1 states that the MP-HTLC protocol is correct if and only if participants either get the amounts of coins they want (Point 1 of the definition) or they still own the initial amount of funds (Point 2 of the definition), in accordance with the requirements of a general CCC[8] as presented in Section 3.3. We are ready to prove that MP-HTLC is correct in the case of participants missing timeouts.

‖ bakers are the analogous of miners in the Proof-of-Stake chains. Their goal is to produce blocks and their probability to be chosen is related to how much token they hold (*bake* or *stake*)

Proposition 7.1. Assume participants $P_1, \dots, P_N$ need to exchange an amount $a_1^1, \dots, a_N^1$ of coin c_1 on blockchain BC_1 for an amount $a_1^2, \dots, a_N^2$ of coin c_2 on blockchain BC_2, and assume participants $Q_1, \dots, Q_N$ have an opposite need. Assume there is at least one rational participant P_r among $P_1, \dots, P_N$ and at least one rational participant Q_r among $Q_1, \dots, Q_N$.

Assume $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ agree on using MP-HTLC and that participant $\widehat{P}_j$ is dishonest and does not follow the protocol by missing the timeout Δ, as defined in Section 4.2. Then if all participants $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ abort the protocol, then the protocol MP-HTLC is correct in the sense of Point 2 of Definition 7.1. This is valid also in the case participant $\widehat{Q_j}$ is dishonest and does not follow the protocol by missing the timeout Δ' as defined in Section 4.3.

Proof. Recall that we assumed participants can get the current state of both blockchains BC_1 and BC_2 at any time.

Assume dishonest participant $\widehat{P}_j$ does not follow the protocol at the Commit Phase (Section 4.2) and by the time the timeout Δ passes $\widehat{P}_j$ has not sent its own *inTx* transaction. Then all participants in the set $\{P_1, \dots, P_N\} \setminus \{\widehat{P}_j\}$ can redeem their own *inTx* transaction. Furthermore, participants $Q_1, \dots, Q_N$ see this behavior by observing the blockchain BC_1. Since there is at least one rational participant in the $\{Q_1, \dots, Q_N\}$ set it is clear that it is impossible for them to conclude the Redeem Phase, since Q_r will not participate in signing the *outTx* (Section 4.3), forcing $Q_1, \dots, Q_N$ to redeem their *inTx* transaction. Therefore the protocol aborts and $P_1, \dots, P_N$ own amounts $a_1^1, \dots, a_N^1$ of coin c_1 respectively and $Q_1, \dots, Q_N$ own amounts $a_1^2, \dots, a_N^2$ of coin c_2 respectively, which is the case presented in Point 2 of Definition 7.1. Note that this reasoning also applies to the case where dishonest participant $\widehat{Q_j}$ has not sent its own *inTx* transaction on BC_2 by the time the timeout Δ passes in the Commit Phase, since rational participant P_r will not sign *outTx*.

On the other hand, assume all participants $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ completed the Pre-commit and Commit Phases successfully, and they are in the Redeem Phase. Then $P_1, \dots, P_N$ have to sign a transaction together using threshold signatures as in Section 3.4, regardless of whether they are in a UTXO based blockchain or in a Account based blockchain. Since $\widehat{P}_j$ can only cheat by missing deadlines, we assume $\widehat{P}_j$ participates in it. Assume for the same reason that $Q_1, \dots, Q_N$ also follow the protocol. The only deadline $\widehat{P}_j$ can miss is the redeeming of *outTx* in BC_2. But in this case $\widehat{P}_j$ would not be rational and the correctness definition does not apply to him since $\widehat{P}_j$ goes against his own interest. Interestingly, Q_j if rational will be able to redeem its own coins on BC_1 regardless, since Q_j can surely see the preimage of the hash from rational participant P_r: which is the case presented in Point 1 of Definition 7.1.

□

After proving that if one participant misses a deadline then the protocol is still correct, we now see that this remains true for any kind of dishonest behavior, provided it is compliant with the model in Section 3.1. We do that by showing that any dishonest behavior involving transactions is equivalent to missing a timeout and therefore already covered in Proposition 7.1.

Proposition 7.2. Assume a setting similar to Proposition 7.1 where $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ are allowed to have dishonest behavior according to the model in Section 3.1. Then the protocol as described in Section 4 is *correct*, as in the Definition 7.1.

Proof. The easiest way to prove that is by going through each of the phases. In the Pre-commit phase there is no transaction involved, so any cheating has no effect on any fund.

In the Commit phase, the only transactions are the one analyzed in Proposition 7.1 and are "on the same side", meaning $P_1, \dots, P_N$ do transactions that are redeemed by either $P_1, \dots, P_N$ together with a threshold signature or each participant will redeem its own. All possible kinds of dishonest behavior in this phase have to be equivalent to missing the timeout Δ: dishonest participant $\widehat{P}_j$ can not steal funds from P_i in any way since it is not allowed by the model described in Section 3.1. Therefore the whole Commit Phase for $P_1, \dots, P_N$ is covered by Proposition 7.1. The same works also for $Q_1, \dots, Q_N$.

Finally, regarding the Redeeming Phase, $P_1, \dots, P_N$ are required to essentially send funds to $Q_1, \dots, Q_N$ and viceversa. Assume a dishonest participant $\widehat{P}_j$ which blocks this step: then the redeeming phase can not start, which is basically a missed timeout and parties can redeem their own funds. As in Proposition 7.1 then, the protocol remains correct thanks to Point 1 of Definition 7.1. □

7.4 | A Note on Security

While the HTLC is vulnerable to bribery attacks (see e.g.[64] and[65]), recently Tsabary et al. presented MAD-HTLC[66]. In MAD-HTLC the authors modify the classic HTLC adding a *redeem path* that benefits the miner in case Bob is dishonest but benign (i.e. rational and willing not to follow the protocol in case of potential reward but prefers to act honestly for the same reward). This contract is called MH-Dep.

To prevent the case where Bob is spiteful (i.e. dishonest even if he would lose the reward) the authors introduce a collateral contract (MD-Col). MD-Col can be redeemed by Bob if he's honest, or by any miner if Bob tries to be dishonest.

Using MAD-HTLC instead of HTLC in a cross-chain communication doesn't provide atomicity anymore: MAD-HTLC uses the mutual assure destruction (MAD) principle where if a party misbehave, then all parties lose everything.

MP-HTLC can obtain synchrony with both MAD-HTLC and HTLC thanks to the modularity of the construction, but for ease of explanation we presented the MP-HTLC protocol using HTLC.

7.5 | Other synchrony methods

Throughout the paper, we described MP-HTLC using HTLC as a method to achieve synchrony between two blockchains (i.e., two asynchronous systems). The downside is that HTLCs require both chains to support the same hash function and that's not always the case. For example, Tezos supports the Blake2 hash function, Bitcoin the SHA256 function and Monero don't support any hash function as it does not have a scripting language.

Fortunately, HTLCs aren't the only locking mechanism for getting synchrony out of the system. Other methods are:

- *Signature Based Locks*. Instead of relying on the preimage resistance property, signature based lock rely on the discrete logarithm problem. In this case, after creating a secret r, Alice computes $Y = g^r$ and embeds it in the creation of a digital signature[67]. it is possible to prove discrete logarithm equality even across different groups[45], which means that a variation of MP-HTLC supporting Monero is possible and we are working on it right now.
- *Time Lock Puzzles*. In this case Alice weakly encrypts the secret r so that Bob can decrypt it in a finite and predetermined (by Alice) time using basically a brute force algorithm. An example is found in[68].

It is possible to combine locking mechanism. For example it is possible to prove that a discrete logarithm is equal to a preimage via zero-knowledge proofs[69]. This enables the implementation of MP-HTLC between a HTLC-capable blockchain such as Bitcoin or Ethereum and a non-HTLC-capable blockchain such as Monero that is capable of Signature Based Locks.

8 | CONCLUSIONS

In this paper we presented a method to achieve hash-time-lock contracts between multiple participants in the same instance in order to decrease costs and increase user privacy. We have also presented two use cases. The first between Bitcoin and Ethereum, where we also linked the code used to instantiate the protocol and produce a test transaction. The other use case is done between two EVM-compatible blockchain.

To prove that the MP-HTLC protocol satisfies correctness as the HTLCs, we described a suitable model where the participants can be abstracted as one rational entity and proved correctness in that model. To achieve that, we assumed participants on one blockchain are not willing to perform active dishonest acts to each other. The rationale has been that one single participant would act in its own self-interest if rational, and we decided to treat the whole group as rational. Surprisingly, we proved that under these assumptions it is possible to treat the whole group as rational by requiring only *one* rational party per group of participants. We argued that for protocol correctness purposes requiring one rational party for each group allows for the treatment of the whole group of participants $P_1, \dots, P_N$ and $Q_1, \dots, Q_N$ as two separate single entities.

STATEMENTS

- All authors declare that they have no conflicts of interest
- Data sharing is not applicable to this article as no new data were created or analyzed in this study.

References

1. Decker C, Wattenhofer R. Bitcoin Transaction Malleability and MtGox. In: Kutylowski M, Vaidya J., eds. *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*. 8713 of *Lecture Notes in Computer Science*. organization. Springer; 2014; address: 313–326

2. Poon J, Buterin V. Plasma: Scalable autonomous smart contracts. *White paper* 2017.

3. Poon J, Drya T. The Bitcoin Lighning network. *Whitepaper* 2015.

4. Nolan T. Alt Chains and Atomic Transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949; .

5. Shamir A. How to Share a Secret. tech. rep., MIT; address: 1979.

6. Nick J, Ruffing T, Seurin Y. MuSig2: Simple Two-Round Schnorr Multi-signatures. In: Malkin T, Peikert C., eds. *Advances in Cryptology – CRYPTO 2021*organization. Springer International Publishing; 2021.

7. Barbara F, Schifanella C. DMix: Decentralized Mixer for Unlinkability. *2nd Conference of Blockchain Research and Applications* 2020.

8. Zamyatin A, Al-Bassam M, Zindros D, et al. SoK: Communication Across Distributed Ledgers. *IACR Cryptol. ePrint Arch.* 2019; 2019: 1128.

9. Belchior R, Vasconcelos A, Guerreiro S, Correia M. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. *CoRR* 2020; abs/2005.14282.

10. Buterin V. Chain Interoperability. *R3 Research Paper* 2016.

11. Fynn E, Bessani A, Pedone F. Smart Contracts on the Move. In: organization. IEEE; 2020: 233–244

12. Hargreaves M, Hardjono T, Belchior R. Open Digital Asset Protocol. Internet-Draft draft-hargreaves-odap-02, Internet Engineering Task Force; address: 2021. Work in Progress.

13. Back A, Corallo M, Dashjr L, et al. Enabling Blockchain Innovations with Pegged Sidechains. *Whitepaper* 2014.

14. Singh A, Click K, Parizi RM, Zhang Q, Dehghantanha A, Choo KR. Sidechain technologies in blockchain networks: An examination and state-of-the-art review. *J. Netw. Comput. Appl.* 2020; 149. doi: 10.1016/j.jnca.2019.102471

15. Dilley J, Poelstra A, Wilkins J, Piekarska M, Gorlick B, Friedenbach M. Strong Federations: An Interoperable Blockchain Solution to Centralized Third Party Risks. *CoRR* 2016; abs/1612.05491.

16. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. *Whitepaper* 2008.

17. Ethereum . BTC Relay. https://github.com/ethereum/btcrelay; 2021.

18. Douceur JR. The Sybil Attack. In: Druschel P, Kaashoek MF, Rowstron AIT., eds. *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*. 2429 of *Lecture Notes in Computer Science*. organization. Springer; 2002; address: 251–260

19. Ganesh AJ, Kermarrec A, Massoulié L. Peer-to-Peer Membership Management for Gossip-Based Protocols. *IEEE Trans. Computers* 2003; 52(2): 139–149. doi: 10.1109/TC.2003.1176982

20. Kokoris-Kogias E, Jovanovic P, Gasser L, Gailly N, Syta E, Ford B. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In: organization. IEEE Computer Society; 2018; address: 583–598

21. Wang J, Wang H. Monoxide: Scale out Blockchains with Asynchronous Consensus Zones. In: Lorch JR, Yu M. , eds. *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*organization. USENIX Association; 2019; address: 95–112.

22. Danezis G, Meiklejohn S. Centrally Banked Cryptocurrencies. In: organization. The Internet Society; 2016; address.

23. Al-Bassam M, Sonnino A, Bano S, Hrycyszyn D, Danezis G. Chainspace: A Sharded Smart Contracts Platform. In: organization. The Internet Society; 2018; address.

24. Kokoris-Kogias E, Jovanovic P, Gasser L, Gailly N, Syta E, Ford B. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In: organization. IEEE Computer Society; 2018; address: 583–598

25. Zamani M, Movahedi M, Raykova M. RapidChain: Scaling Blockchain via Full Sharding. In: Lie D, Mannan M, Backes M, Wang X., eds. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*organization. ACM; 2018; address: 931–948

26. Chandy KM, Lamport L. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Trans. Comput. Syst.* 1985; 3(1): 63–75. doi: 10.1145/214451.214456

27. Krug J, Peterson J. Sidecoin: a snapshot mechanism for bootstrapping a blockchain. *CoRR* 2015; abs/1501.01039.

28. Paiva J, Leitão J, Rodrigues LET. Rollerchain: A DHT for Efficient Replication. In: IEEE Computer Society; 2013: 17–24

29. Luu L, Narayanan V, Zheng C, Baweja K, Gilbert S, Saxena P. A Secure Sharding Protocol For Open Blockchains. In: Weippl ER, Katzenbeisser S, Kruegel C, Myers AC, Halevi S., eds. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*ACM; 2016: 17–30

30. Tikhomirov S, Pickhardt R, Biryukov A, Nowostawski M. Probing Channel Balances in the Lightning Network. *CoRR* 2020; abs/2004.00333.

31. Asokan N, Shoup V, Waidner M. Asynchronous Protocols for Optimistic Fair Exchange. In: organization. IEEE Computer Society; 1998; address: 86–99

32. Malkhi D, Reiter MK. Byzantine Quorum Systems. In: Leighton FT, Shor PW., eds. *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*ACM; 1997: 569–578

33. Even S, Goldreich O, Lempel A. A Randomized Protocol for Signing Contracts. In: Chaum D, Rivest RL, Sherman AT., eds. *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*Plenum Press, New York; 1982: 205–210

34. Even S, Yacobi Y. Relations among public key signature systems. tech. rep., Computer Science Department, Technion; 1980.

35. Pagnia H, Gartner FC. On the impossibility of fair exchange without a trusted third party. tech. rep., Citeseer; address: 1999.

36. Yao AC. How to Generate and Exchange Secrets (Extended Abstract). In: IEEE Computer Society; 1986: 162–167

37. Maxwell G, Poelstra A, Seurin Y, Wuille P. Simple Schnorr multi-signatures with applications to Bitcoin. *Designs, Codes and Cryptography* 2019; 87(9): 2139–2164. doi: 10.1007/s10623-019-00608-x

38. Aumasson J, Hamelink A, Shlomovits O. A Survey of ECDSA Threshold Signing. *IACR Cryptol. ePrint Arch.* 2020; 2020: 1390.

39. Pieter Wuille TR. BIP-340: Schnorr Signatures for secp256k1. GitHub; .

40. Ryan D. GitHub; .

41. Schnorr C. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* 1991; 4(3): 161–174. doi: 10.1007/BF00196725

42. Boneh D, Lynn B, Shacham H. Short Signatures from the Weil Pairing. *J. Cryptol.* 2004; 17(4): 297–319. doi: 10.1007/s00145-004-0314-9

43. Nick J. GitHub; .

44. Goodell B, Noether S. Thring Signatures and their Applications to Spender-Ambiguous Digital Currencies. *IACR Cryptol. ePrint Arch.* 2018; 2018: 774.

45. Sarang N. Discrete logarithm equality across groups. tech. rep., Monero Research Lab; address: 2018.

46. Evans D, Kolesnikov V, Rosulek M. A Pragmatic Introduction to Secure Multi-Party Computation. *Found. Trends Priv. Secur.* 2018; 2(2-3): 70–246. doi: 10.1561/3300000019

47. Keller M. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In: Ligatti J, Ou X, Katz J, Vigna G., eds. *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*ACM; 2020: 1575–1590

48. Bistarelli S, Mercanti I, Santini F. An Analysis of Non-standard Transactions. *Frontiers Blockchain* 2019; 2: 7. doi: 10.3389/fbloc.2019.00007

49. Andresen G. BIP-16: Pay to Script Hash. Github; .

50. Chow A. BIP-174: Partially Signed Bitcoin Transaction Format. Github; .

51. Chow A. BIP-370: PSBT Version 2. Github; .

52. Gennaro R, Goldfeder S. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. In: Lie D, Mannan M, Backes M, Wang X., eds. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*ACM; 2018: 1179–1194

53. Adams h, Zinsmeister N, Salem M, Keefer R, Robinson D. Uniswap v3 Core. *Whitepaper* 2021.

54. Zoltu M. EIP-2711: Sponsored, expiring and batch transactions. [DRAFT]. Github; 2020.

55. Matt . EIP-3005: Batched meta transactions [DRAFT]. Github; 2020.

56. Matt . Native Meta-Transaction Proposal Roundup. EthResear.ch; 2020.

57. Wang Y, Zhang Q, Li K, et al. iBatch: saving Ethereum fees via secure and cost-effective batching of smart-contract invocations. In: Spinellis D, Gousios G, Chechik M, Penta MD., eds. *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*organization. ACM; 2021; address: 566–577

58. Eyal I, Sirer EG. Majority is not enough: bitcoin mining is vulnerable. *Commun. ACM* 2018; 61(7): 95–102. doi: 10.1145/3212998

59. Sai K, Tipper D. Disincentivizing Double Spend Attacks Across Interoperable Blockchains. In: organization. IEEE; 2019; address: 36–45

60. Sapirshtein A, Sompolinsky Y, Zohar A. Optimal Selfish Mining Strategies in Bitcoin. In: Grossklags J, Preneel B., eds. *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*. 9603 of *Lecture Notes in Computer Science*. organization. Springer; 2016; address: 515–532

61. Nayak K, Kumar S, Miller A, Shi E. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. In: organization. IEEE; 2016; address: 305–320

62. Kwon Y, Kim D, Son Y, Vasserman EY, Kim Y. Be Selfish and Avoid Dilemmas: Fork After Withholding (FAW) Attacks on Bitcoin. In: Thuraisingham BM, Evans D, Malkin T, Xu D., eds. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*organization. ACM; 2017; address: 195–209

63. Neuder M, Moroz DJ, Rao R, Parkes DC. Selfish Behavior in the Tezos Proof-of-Stake Protocol. *CoRR* 2019; abs/1912.02954.

64. Judmayer A, Stifter N, Zamyatin A, et al. Pay to win: Cheap, crowdfundable, cross-chain algorithmic incentive manipulation attacks on pow cryptocurrencies. tech. rep., Cryptology ePrint Archive; address: 2019.

65. Winzer F, Herd B, Faust S. Temporary Censorship Attacks in the Presence of Rational Miners. In: organization. IEEE; 2019; address: 357–366

66. Tsabary I, Yechieli M, Eyal I. MAD-HTLC: Because HTLC is Crazy-Cheap to Attack. *CoRR* 2020; abs/2006.12031.

67. Hoenisch P, Pino dLS. Atomic Swaps between Bitcoin and Monero. *CoRR* 2021; abs/2101.12332.

68. Rivest RL, Shamir A, Wagner DA. Time-lock puzzles and timed-release crypto. Tech. Rep. Technical memo MIT/LCS/TR-684, MIT Laboratory for Computer Science; address: 1996.

69. Chase M, Ganesh C, Mohassel P. Efficient Zero-Knowledge Proof of Algebraic and Non-Algebraic Statements with Applications to Privacy Preserving Credentials. In: Robshaw M, Katz J., eds. *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*. 9816 of *Lecture Notes in Computer Science*. organization. Springer; 2016; address: 499–530

70. Zamyatin A, Harz D, Lind J, Panayiotou P, Gervais A, Knottenbelt WJ. XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets. In: organization. IEEE; 2019; address: 193–210

71. Grigg I. EOS-An Introduction. In: organization. ; 2017; address.

72. Damgård I, Pastro V, Smart NP, Zakarias S. Multiparty Computation from Somewhat Homomorphic Encryption. In: Safavi-Naini R, Canetti R., eds. *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. 7417 of *Lecture Notes in Computer Science*. Springer; 2012: 643–662

73. Ben-Or M, Goldreich O, Micali S, Rivest RL. A fair protocol for signing contracts. *IEEE Trans. Inf. Theory* 1990; 36(1): 40–46. doi: 10.1109/18.50372

74. Gray J. Notes on Data Base Operating Systems. In: Flynn MJ, Gray J, Jones AK, et al., eds. *Operating Systems, An Advanced Course*. 60 of *Lecture Notes in Computer Science*. Springer; 1978: 393–481

75. Cachin C, Kursawe K, Shoup V. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *Journal of Cryptology* 2005; 18(3): 219–246. doi: 10.1007/s00145-005-0318-0

76. Nick J, Ruffing T, Seurin Y. MuSig2: Simple Two-Round Schnorr Multi-Signatures. *IACR Cryptol. ePrint Arch.* 2020; 2020: 1261.

77. Deshpande A, Herlihy M. Privacy-Preserving Cross-Chain Atomic Swaps. In: Bernhard M, Bracciali A, Camp LJ, et al., eds. *Financial Cryptography and Data Security - FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers*. 12063 of *Lecture Notes in Computer Science*. organization. Springer; 2020; address: 540–549

78. Judmayer A, Stifter N, Zamyatin A, et al. Pay-To-Win: Incentive Attacks on Proof-of-Work Cryptocurrencies. *IACR Cryptol. ePrint Arch.* 2019; 2019: 775.

79. Kanani J, Arjun A, Nailwal S, Bjelic M. Polygon Whitepaper. *Whitepaper* 2021.