# Tesseract: Real-Time Cryptocurrency Exchange Using Trusted Hardware

Iddo Bentov Yan Ji Fan Zhang Xueyuan Zhao Yungi Li SJTU Cornell University Cornell University Cornell University SJTUAri Juels Lorenz Breidenbach Philip Daian ETH Zürich and Cornell Tech Cornell Tech Cornell Tech

## **Abstract**

We propose Tesseract, a secure real-time cryptocurrency exchange service. Existing centralized exchange designs are vulnerable to theft of funds, while decentralized exchanges cannot offer real-time crosschain trades. All currently deployed exchanges are also vulnerable to frontrunning attacks. overcomes these flaws and achieves a best-of-bothworlds design by using Intel SGX as a trusted execution environment. Furthermore, by running a consensus protocol among SGX-enabled servers, Tesseract mitigates denial-of-service attacks. Tesseract supports not only real-time cross-chain cryptocurrency trades, but also secure tokenization of assets pegged to cryptocurrencies. For instance, Tesseract-tokenized bitcoins can circulate on the Ethereum blockchain for use in smart contracts. We provide a reference implementation of Tesseract that supports Bitcoin, Ethereum, and similar cryptocurrencies.

# 1 Introduction

The rise of Bitcoin [54] has spawned many hundreds of other cryptocurrencies as well as application-specific units of value known as crypto "tokens." This diverse ecosystem of assets has in turn led to a large and dynamic array of cryptocurrency *exchanges*, platforms that allow users to trade different cryptocurrencies against one another and/or for fiat currencies. At the time of writing, the aggregate daily trading volume of cryptocurrency exchanges exceeds \$25 billion.

Unfortunately, cryptocurrency exchanges suffer from a variety of security problems. Currently, the most popular exchanges are *centralized*, meaning that they hold traders' assets while trades are executed. Such exchanges support real-time trading and often automatically match buy and sell orders. They are vulnerable, however, to *theft* of traders' funds (cf.

Section 3.1). In a number of high-profile incidents, funds have been stolen when exchanges were breached or other forms of malfeasance took place, e.g., [25, 6, 38, 52].

Permissionless blockchains, however, are designed specifically to eliminate trust assumptions between transacting parties by avoiding centralization. trust-free cryptocurrency exchange can be realized for transactions across such blockchains in the form of atomic intra-chain or cross-chain swaps (ACCSs) [16], transactions that exchange cryptocurrencies between pairs (or among sets) of users in a fair, all-or-nothing manner. ACCSs, though, require users to wait many minutes (in fact, often hours) for a trade to execute. Additionally, atomic swaps in general aren't sufficient to realize an exchange: A mechanism for matching orders or otherwise performing price discovery is also necessary. (Since ACCSs serve as a useful reference point, we elaborate on the concept and its limitations in Section 2.)

The systemic risk of theft in centralized exchanges has led to the rising popularity of *decentralized* exchanges such as EtherDelta [64] and the soon-to-be-implemented 0x [81], AirSwap [57], and Kyber Network [46]. These systems hold traders' funds and settle transactions in smart contracts, eliminating the risk of theft in centralized exchanges. Unfortunately, they have other drawbacks. Their on-chain settlement means that they cannot support real-time trading. Additionally, while their use of smart contracts conveys an appearance of trustworthiness, they are vulnerable to various frontrunning attacks by exchanges and other users.

Achieving the best of both worlds has been a standing challenge, but a seemingly elusive one. An ideal cryptocurrency exchange would be *real-time* like a centralized exchange, meaning that participants can respond to price fluctuations and alter their positions with low latency. It would support even traders that choose to utilize automated programs for high frequency trading and arbitrage (cf. [7]), who may wish to modify their

positions in milliseconds. At the same time, such an exchange would be *trust-free*, protecting against theft in the way that decentralized exchanges do, but also eliminating frontrunning attacks that exploit blockchain latencies.

In this work, we present Tesseract, a cryptocurrency exchange that achieves this ideal set of properties. Tesseract is *real time*. Traders can rapidly observe the alterations in the buy (a.k.a. "bid") and sell (a.k.a. "ask") orders on the exchange, as well as external events (e.g., [85]), then modify their trading positions in milliseconds. By performing fast *price discovery*, they can drive price convergence so that the gap (a.k.a. "spread") between the bids and asks is small, leading to efficient markets like those in major financial systems. Tesseract also prevents theft of users' funds by exchange operators and hackers as well as a variety of frontrunning attacks present in existing cryptocurrency exchanges.

Tesseract relies on SGX, a trusted execution environment supported by an instruction-set architecture extension in recent-model Intel CPUs [1, 34, 51]. SGX allows applications to execute within a protected environment called an *enclave* that ensures confidentiality and software integrity. It enables Tesseract to behave like a trusted third party, controlling funds without exposing them to theft while preventing frontrunning by the exchange operator. Additionally, Tesseract provides mitigation against denial-of-service (DoS) attacks via a consensus protocol among SGX-enabled nodes.

Tesseract supports *cross-chain* trading in which assets are exchanged across distinct blockchains. Trades within a single blockchain, e.g., exchange of tokens and Ether within Ethereum, can also be important (cf. [67, 18, 60, 19, 15]). While this use case can be achieved at least in part using smart contracts, a significantly simplified variant of Tesseract can offer the added benefit of real-time trading, which smart contracts cannot support. Tesseract also supports a tokenization scheme that allows pegged tokens to ciculate across blockchains, without relying on a human element for security (see Section 7).

The main challenge in the design of Tesseract is dealing with powerful network adversaries. Such adversaries can perform an *eclipse attack* in which an exchange is presented with fake blockchain data. We show how to address this problem by checkpointing trustworthy blocks within the Tesseract application and having it monitor the cumulative difficulty of newly furnished blocks. A network adversary can also suppress messages / transactions issued by the exchange in an attempt to interfere in on-chain settlement of trades, e.g., permitting partial settlement in which cryptocurrency flows to the adversary from a counterparty but not from the adversary, resulting in the adversary stealing

funds. We express a theoretical solution to these network attacks in terms of an ideal functionality called a *refundable multi-input transaction* (RMIT). RMIT provides a conceptual springboard for securely architecting a secure cross-chain exchange. We present a highly efficient realization of RMIT in Tesseract, in a protocol involving a network of multiple SGX-backed nodes running Paxos. While only one node handles assets directly, others can execute or cancel transactions should the main node fail. This protocol enforces a key fairness property we define called *all-or-nothing settlement*.

Our security assumptions around Tesseract are quite conservative. We assume that an adversary (potentially the exchange operator) can gain complete physical access to the host in which the funds are stored and complete control of its network connections. We do assume that the code that we run inside the SGX enclave is secure against side-channel attacks [84], but this code is constant-time and constant-memory. In a sense, the Tesseract exchange still relies on a trusted party in the form of the hardware manufacturer, because the private key that resides inside CPU (and generates signatures for remote attestation) is provisioned by the manufacturer. It can be argued that a weaker yet similar form of trust is required in a practical instantiation of any cryptographic protocol, since the manufacturer may be able to attack the protocol by embedding malicious logic into the hardware. We also, however, incorporate a scheme of double attestation (Section 4.3) that provides an extra layer of defense against a corrupt manufacturer. Thus, Tesseract still requires trust, but to a significantly lesser degree than centralized exchanges and other possible real-time exchange schemes (cf. Section 3).

In summary, our contributions in this paper are as follows:

- We introduce Tesseract, an SGX-backed cryptocurrency exchange that can support a wide variety of transaction types, with cross-chain trading as its primary application.
- We consider attacks by powerful network adversaries that may seek to mount eclipse attacks or suppress transactions to achieve unfair settlement and thus theft of funds. We define a key fairness property called *all-or-nothing settlement* and show how to realize an exchange that achieves this property using as a conceptual building block an ideal functionality called RMIT.
- We present practical techniques to achieve all-ornothing settlement in Tesseract. These techniques include within-enclave blockchain monitoring to prevent eclipse attacks and use of a consensus

group of SGX-backed nodes that can enforce and/or cancel transactions in the case that the main (asset-holding) exchange node becomes unavailable.

 We implement proof-of-concept of Tesseract, describing our parameter and design choices.

# 2 Atomic cross-chain swaps

A secure protocol for ACCSs was given in [16]. We specify an intuitive description of the protocol in Figure 1, demonstrating a swap of bitcoins for litecoins as an example. The main thrust of the protocol  $\Pi_{accs}$  is that Alice can redeem Bob's coins only by publicly revealing her decommitment x on a blockchain, thereby allowing Bob to use x to redeem Alice's coins on the other blockchain. To avoid a race condition, Alice's coins remain locked for  $s_0$  more time than Bob's coins, which should give Bob enough time to learn x and claim Alice's coins. The reason behind the time limits is that an honest party should be able to gain back possession of her money in the case that the other party aborted. We provide a proof of security for  $\Pi_{accs}$  in Appendix A.

The first two steps of  $\Pi_{\rm accs}$  terminate after  $c_0$  and  $f(c_0)$  confirmations on the Bitcoin and Litecoin blockchains, so that the transactions will become irreversible with a high enough probability. The function  $f(\cdot)$  estimates a level of confidence for  $\mathsf{TX}_\mathsf{B}$ 's irreversibility that is on par with that of  $\mathsf{TX}_\mathsf{A}$ . Per Section 3.1, a reasonable choice for  $f(\cdot)$  can be, e.g., f(n) = 3n. Combined with a sensible choice for the parameters  $t_0, s_0$  (see Appendix A), Alice and Bob will need to wait for hours (or perhaps minutes with faster cryptocurrency systems) until the  $\Pi_{\rm accs}$  protocol completes.

In the accompanying illustration (Figure 2), Alice trades  $n_1 = 2$  BTC for Bob's  $n_2 = 600$  LTC. The last block of the Bitcoin blockchain is  $T_1$ , and the last block of the Litecoin blockchain is  $T_2$ . The time limit  $t_0$  is set to about two weeks into the future (i.e., 2000 more blocks in Bitcoin, and 8000 more blocks in Litecoin, as the block creation rate is 4 times faster in Litecoin than in Bitcoin). The extra safety time  $s_0$  is set to 100 Bitcoin blocks, which is  $\approx$  16 hours on average. Note that both Bitcoin and Litecoin allow specification of the time limit in seconds rather than blocks (since valid blocks need to specify a timestamp that is within certain leniency bounds), which adds convenience but not security.

Since the long confirmation time in decentralized networks makes  $\Pi_{accs}$  slow, it is likely that the agreed upon price (in the example,  $n_2/n_1 = 300$  LTC per BTC) was decided by observing the prices in real-time exchanges. This implies that the parties cannot respond to price fluctuations in a fair manner: if Bob is rational then he may cancel the trade after the first step (if the

#### Protocol $\Pi_{accs}$

- 1. Alice samples a random  $x \in \{0,1\}^{\lambda}$ , computes a hash commitment  $Y = \mathsf{hash}(x)$ , and broadcasts a transaction  $\mathsf{TX}_{\mathsf{A}}$  that spends  $n_1$  BTC into an output script that dictates:
  - Alice can gain back possession of her  $n_1$  BTC after  $c_0 + t_0 + s_0$  blocks.
  - Bob can redeem the n<sub>1</sub> BTC by supplying a preimage of Y and signing with his secret key.
- After TX<sub>A</sub> is buried under c<sub>0</sub> extra blocks and therefore becomes irreversible w.h.p., Bob broadcasts a transaction TX<sub>B</sub> that spends his n<sub>2</sub> LTC into an output script that dictates:
  - Bob can gain back possession of his  $n_2$  LTC after  $4t_0$  blocks.
  - Alice can redeem the n<sub>2</sub> LTC by supplying a preimage of Y and signing with her secret key.
- 3. After  $\mathsf{TX}_\mathsf{B}$  is buried under  $f(c_0)$  extra blocks and therefore becomes irreversible w.h.p., Alice redeems the  $n_2$  LTC of Bob by revealing x.
- 4. Bob redeems the  $n_1$  BTC of Alice by supplying x.

Figure 1: Protocol for an atomic cross-chain swap.

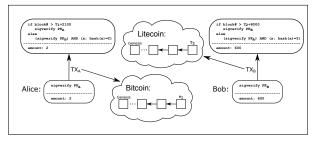


Figure 2: Illustration of an atomic cross-chain swap.

market price of LTC went up), and if Alice is rational then she may cancel the trade after the second step (if the market price of BTC went up). Another implication is that  $\Pi_{accs}$  by itself is not a complete trading solution, because real-time exchanges are still needed for price discovery.

A matching service for ACCSs was established in 2015, though it became defunct due to lack of usage [47].

# 3 Cryptocurrency Exchanges

We describe several alternative designs for a real-time cryptocurrency exchange, and also survey non-real-time designs. See Table 1 for a summary comparison between Tesseract and the alternatives.

# 3.1 Centralized Exchange

In a centralized cryptocurrency exchange, users transfer ownership of their funds to the sole control of the exchange administrator. This transfer of ownership (a.k.a. deposit) is done via an on-chain transaction that may take a long time to be confirmed, according to a confidence parameter that the exchange administrator set. Most exchanges accept a Bitcoin transfer by waiting 1 hour on average (6 PoW confirmations).

The business model of a centralized exchange can be described as a "goose that lays golden eggs". That is to say, the exchange administrator may run away with all the funds that the users deposited (usually by claiming "I was hacked"), and the disincentive to doing so is that the exchange collects a fee from each trade between the users. Most exchanges also charge a withdrawal fee, and some exchanges collect fees even when the users place bid and ask orders.

Still, there have been many thefts of funds from centralized exchanges (cf. [25]). About 650,000 bitcoins were lost when the MtGox exchange shut down in February 2014 [38], and the users of the Bitfinex exchange lost 120,000 bitcoins in August 2016 [6].

# 3.2 Exchange with Off-chain Channels and TTP

In this design, each user establishes off-chain bidirectional payment channels [61, 22, 50] with a semi-TTP server *S*, one channel for each cryptocurrency that the user wishes to trade in. This produces a hub and spoke network structure, see Figure 3 for an illustration of trading among the Bitcoin, Ethereum, and Litecoin cryptocurrencies.

The traders will then communicate their bid and ask orders to *S*. Whenever the orders of two traders match, they will send an instant off-chain payment to *S*, and *S* will route the funds of one trader to the other.

It is better for each individual to trade in small amounts, because the TTP can always steal the most recent amount that was funneled through *S*. However, this recommendation is in conflict with the common behavior of large traders, who frequently create big bid/ask "walls".

In any case, even if the amount in each trade is small, the risk of theft by a corrupt TTP remains high. This

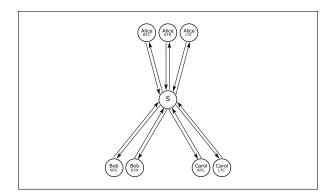


Figure 3: Exchange via off-chain channels.

is because the aggregate amount that all the traders funnel through S at a particular point in time can be substantial. As an example that does not involve an exchange but demonstrates this point, the online wallet service Inputs.io made it attractive for users to deposit small amounts, and then ran away with more than 4000 bitcoins [52].

Another major drawback of this approach is that the TTP has to lock matching collateral for each off-chain payment channel of each trader, due to nature of off-chain bi-directional channels. It is therefore likely that the exchange service would need to impose high fees on its users.

# 3.3 Non-real-time Exchanges

There are many non-real-time exchanges. We mention a few to illustrate important points in the design space.

ShapeShift [72] is a centralized matching service that mitigates the risks associated with a full-fledged exchange by necessitating that each trader will deposit only a small amount of cryptocurrency for a short period of time. If a quick match is available then ShapeShift will execute the trade, otherwise it will immediately refund the cryptocurrency to the trader (i.e., via a transaction on the blockchain). ShapeShift does not support real-time trades and price discovery. It fetches the current prices from centralized exchanges.

Since ShapeShift is rather popular, the aggregated amount of funds that can be stolen is likely to be substantial. In this sense, ShapeShift does not solve the systemic risk that centralized exchanges entail.

EtherDelta [64] is a non-real-time non-cross-chain decentralized exchange that has been operational since July 2016, with quite a significant amount of popularity—particularly for the first listings of Initial Coin Offerings (ICO). However, EtherDelta is vulnerable to frontrunning attacks, see [12].

BitShares [70] offers a cryptocurrency exchange that does not rely on trusted parties. It is not real-time,

Table 1.	Comparison	of Cryptocurrency	Exchanges

	Trust	DoS	Collateral	Front- running	Price Discovery
Centralized	yes	minor	no	yes	yes
TTP/channels (Section 3.2)	semi	minor	from TTP	yes	yes
ShapeShift	semi	minor	no	yes	no
Tesseract	SGX	minor	no	no	yes

but relatively fast due to a delegated proof-of-stake consensus protocol in which blocks are created every few seconds by central committee members (who may engage in frontrunning attacks, see Section 4.2).

Traders first convert their cryptocurrency to IOUs in the BitShares system, and later convert these IOUs to the native BitShares cryptocurrency (BTS) according to an up-to-date exchange rate that is set by elected representatives that the BitShares stakeholders voted for. See [70, Section 2] and [76] regarding the risk of market manipulation with this approach.

The BTS cryptocurrency that traders ultimately obtain can be exchanged for other cryptocurrencies by means that are again external to the BitShares system — centralized exchanges (a.k.a. gateways) are commonly used for this task.

**Exchange Based on Mutual Distrust:** Instead of relying on trusted hardware, it would be possible in principle to operate an exchange service (similar to Tesseract) as a logical server that is implemented via multiple physical servers that are distrustful of each other.

Traders will need to send their bid/ask requests using threshold encryption [23] in order to avoid frontrunning attacks (see Section 4.2), and the physical servers will run a Byzantine consensus protocol and sign the settlement transactions (cf. Section 4) with a threshold signature scheme [30]. An honest majority among the physical servers can guarantee protection from theft.

Since the physical servers would need to reside in different geographical locations to provide meaningful security, and since Byzantine agreement with threshold decryption has to be performed for each of the users' orders, the latency of a mutual distrust based exchange would probably be measured in seconds (depending on the number of physical servers). By contrast, the responsiveness of Tesseract can be measured in milliseconds.

# 4 The Tesseract Design

The Tesseract exchange achieves its security and performance goals by relying on a *trusted execution environment*, specifically SGX. Intel *Software Guard Extensions* (SGX) is a hardware architecture that enables code execution in an isolated, tamper-free environment. Intel SGX can also attest that an output represents the result of such an execution, and allows remote users to make sure that the attestation is correct. The *remote attestation* feature is essential for Tesseract, for reasons that will soon become clear (cf. Section 4.3 for further discussion). For more information on the SGX architecture, see [1, 35, 34, 51].

The operation of Tesseract is illustrated in Figure 4. The enclave code is hardcoded with the hash of the Bitcoin genesis block, or a more recent "checkpoint" block of the Bitcoin blockchain. When the execution starts, the enclave receives the latest block headers from an untrusted Bitcoin client that runs on the same server machine. Each header has its PoW validated against the difficulty rule of the Bitcoin protocol, and is then added to a FIFO queue that is stored inside the enclave. The size of the queue is set according to a parameter that specifies the maximum time window that the enclave maintains. For instance, 8064 Bitcoin block headers would correspond to a 2-month window (when header 8065 is added the first header will be removed, and so on). The enclave will also maintain the same kind of queue for every other cryptocurrency that is supported by the Tesseract exchange service. We note that Bitcoin and Litecoin block headers are 80 bytes each, and an Ethereum block header is  $\approx 512$  bytes.

After initialization, the enclave invokes a key generation procedure to create a keypair (sk, pk) for each supported cryptocurrency. The randomness that we feed to the key generator is obtained by concatenating several sources: the RDRAND instruction that  $sgx\_read\_rand()$  uses for hardware-based randomness, the hashes of the latest blockchain blocks, OS provided randomness via /dev/random, and the SGX trusted clock. Each of these sources increases the entropy of the random data, and therefore reduces the likelihood that an adversary will have knowledge of the secret key sk.

The enclave will then attest [35] that a public key pk is its deposit address, for each cryptocurrency. The attestation to these public keys should be published through multiple services (such as websites, IPFS [11], and even Bitcoin and other blockchains). As an example, Figure 4 shows two such deposit addresses  $PK_{SGXBTC}$ ,  $PK_{SGXLTC}$ , for Bitcoin and Litecoin. The anti-DoS component that we describe in Section 6 is also useful for making sure that the attested deposit addresses will be publicly known.

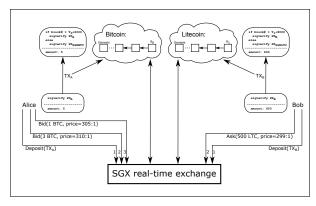


Figure 4: Illustration of deposits followed by bids/asks.

In fact, it is better if the deposit address is a hash of the public key, as this increases security and reduces the size of unspent outputs on the public ledger. For example, a 257-bit compressed ECDSA public key gives 128 bits of security at most, while 160-bit hash digest of the 257-bit public key will give 160 bits of security (if the hash function is preimage-resistant). This is done in our implementation via P2SH [2] (P2WPK/P2WSH [42] can be used post-SegWit). Note that there is no point in mounting a collision attack on a scriptless address [5]. The settlement transaction (see next) will expose the public key, but potential attacks would then have a short timeframe until the transaction becomes irreversible. Hence, for maximal security the enclave will generate and attest to a fresh deposit address after each settlement.

When a new user wishes to open a Tesseract account, she first needs to deposit a significant enough amount into a deposit address of the exchange. deposit transaction is confirmed on the blockchain, the (GUI client of the) user will transform the confirmed deposit into evidence that will be sent to the enclave. This evidence consists of the transaction that spends the coins into a deposit address of Tesseract, as well as an authentication path that consists of the sibling nodes in the Merkle tree whose root is stored in a block header, and the index of that block. Tesseract will credit the user's account (in the enclave) after verifying that the deposit transaction is valid, that the block B that contains the deposit belongs to the enclave's headers queue, and that B is buried under enough additional confirmations (see Section 4.1 for security analysis). Tesseract also protects against replay attacks, by requiring strictly increasing block indices for the user's deposits. In Figure 4, the evidence that Alice provides is Deposit( $TX_A$ ).

As shown in Figure 4, the output of a valid deposit transaction needs to specify a time limit (e.g., two weeks). Before the limit is reached, only the enclave can spend the deposit amount (for a Bitcoin deposit,

this public key PK<sub>SGXBTC</sub> is hardcoded in the output and the spending is done by creating a signature with the corresponding secret key SK<sub>SGXBTC</sub>). After the time limit, the user can gain back control of her money by signing with a secret key that only she knows. In cryptocurrencies such as Bitcoin and Litecoin, the time limit can be expressed in the output script via the CHECKLOCKTIMEVERIFY instruction [77]. Technically, SK<sub>SGXBTC</sub> can still spend the output after the time limit (since Bitcoin transactions should be *reorg safe* [55, 77]), but this is not guaranteed because the user may also spend the output then. This deposit format ensures that the funds will safely be restored to the user if the Tesseract server becomes unavailable.

We note that the enclave is hardcoded with the current difficulty parameter of each PoW-based blockchain. At the beginning of the execution, the enclave will fetch blocks from genesis (or the more recent checkpoint), and verify that the chain reaches a block of the hardcoded difficulty level. This prevents an adversary (who has physical control of the Tesseract server) from feeding a low-difficulty fake chain to the enclave. The enclave updates the PoW difficulty level by inspecting the timestamps of block headers in the FIFO queue and applying the consensus rules of the cryptocurrency system (the queue size must be at least as the adjustment interval, which is 2016 for Bitcoin). This implies that an adversary cannot feed low-difficulty blocks to the enclave at a later time. The users of the Tesseract exchange can gain extra security by inspecting the latest block of each traded cryptocurrency and verifying (via remote attestation) that the enclave has the latest blocks, see Section 4.1 for details.

Malicious users may try to carry out a DoS attack on the Tesseract server by attempting to open many new accounts while providing fake deposits as evidence. Currently, Bitcoin blocks contain less than 4000 transactions, which implies that the authentication path requires 12 or fewer sibling nodes of the Merkle tree, and hence 12 invocations of a hash function. Thus, the time complexity of verifying the validity of a deposit is quite low. To further mitigate the prospects of a DoS attack, the enclave may require a moderate PoW done on the entire evidence data of the deposit (that the user will compute on her own), or simply limit the number of new account requests per timeframe.

One reason that the enclave maintains a queue of headers and fetches the additional block confirmations from the queue — as opposed to asking the user to concatenate the extra confirmations as part of the evidence of the deposit — is that the queue provides an undisputed point of reference in the form of the genesis (or checkpoint) block. That is to say, if there are two blockchains that use the same hash function for PoW and

have a similar difficulty level, then a malicious user could deceive the enclave into accepting a deposit transaction that was confirmed on an incorrect blockchain. This approach also reduces the communication complexity between the Tesseract server and remote users.

After the user registers with Tesseract, her deposited amount is credited into her account entry in the array of users that is stored inside the enclave. Next, the user will be able to trade in real-time with other users who opened a Tesseract account, by sending bid/ask orders to the Tesseract server via a secure channel (see Section 4.2). If the user wishes to deposit other currencies into her account, she can then send similar authentication paths as evidence.

In Figure 4, Bob opens an account with Deposit(TX<sub>B</sub>), and then asks to sell 500 LTC for the price of 299 LTC per BTC. Since Alice's bids are with a price of 305 LTC per BTC and higher, there is no match yet, and the requests of Alice and Bob are recorded in the order book kept inside the enclave. Each user can request her recent trading history via the secure channel, and cancel her pending orders. The Tesseract server publishes an anonymized version of the order book (i.e., price and volume of each order, without usernames) with remote attestation; hence anyone can observe the price spread of the exchange. Since order book updates can occur at a very rapid rate, we reduce the amount of SGX attestations via delayed randomized checkpoints: the enclave always outputs the anonymized order book without a signature, and outputs a delayed attestation (that include an incremental counter) only for randomly selected data points. The administrator of the Tesseract server provides her part of the double attestation for all the data points (using HTTPS, see Section 4.3). Thus, an administrator that publishes fake order book data repeatedly will be detected w.h.p.

Real-time trading among the users will cause frequent updates to the balances of their accounts inside the enclave, but these updates are not reflected on the actual cryptocurrency systems yet. If nothing else were to happen, the entire process would just be a sandbox or playground, as the users will simply claim their original money after the time limit of their deposits is reached. Therefore, from time to time (e.g., once a day) Tesseract will broadcast to the cryptocurrency networks "settlement" transactions that commit the current account balances of the users. See Figure 6 for an illustration, and Section 5 regarding a secure settlement protocol.

The enclave extends the time limit of each user's output in the settlement transactions that it constructs (e.g., if the user could control the output in 5 days before the settlement, then she could control the output in 19 days after the settlement). This allows uninterrupted

trading by active traders. To minimize the size of the settlement transactions, users who did not trade are not included in the inputs and outputs. When some of a user's funds are in an output whose time limit is about to expire, the user will be prohibited from trading. The user is permitted to send a renewal request *before* the expiration, in case she was unlucky and none of her trade orders were matched (renewal after the expiration can be exploited by malicious users who would create conflicting transactions near the time limit). The user can also request an early withdrawal of some of her funds. This is done by directing the enclave to prepare an output that is controlled only by the user, in the next settlement.

In our design, the Tesseract exchange collects a proportional fee for each successful trade (e.g., 0.1% from both ends of a trade), and a flat fee for early withdrawal and renewal requests. The exchange limits the total number of pending orders that a user may have in the order book, and users who flood the exchange with an excessive number of orders may be penalized (by confiscating some of their funds) or blacklisted for a period of time. The fees that Tesseract collects are needed in order to pay miner fees for the settlement transactions.

Forthcoming Bitcoin support for aggregated Schnorr signatures [82] will enable Tesseract to attach a single signature to the settlement transaction, instead of one signature for every input. This implies that the settlement transaction size can be halved, which is significant for large transactions (e.g., with 1000 traders the transaction size will 64 kilobytes smaller). It is also likely that miners will impose a considerably lower fee for a large settlement transaction with a single aggregated signature. Let us note that signature aggregation is required in principle if the enclave refreshes its deposit address after each settlement, since the aggregated signature will need to be verified against different public keys.

In case of a forthcoming hard fork of the kind that created Ethereum Classic or Bitcoin Cash, users should secure themselves against replay attacks (cf. [49, Section 2.4]) by withdrawing their coins from the Tesseract exchange. The users may switch to a new version of Tesseract with updated code that supports the hard fork (or completely new cryptocurrencies), and which can be deployed at a later time. Our implementation has dynamic support for ERC20 tokens, hence no switch is needed when new ERC20 tokens are introduced (a user can create new order book pairs, for a fee).

In Appendix B we provide excerpts of our reference code that corresponds to the above description.

# 4.1 Eclipse Attacks

Let us assume an adversary  $\mathcal{A}$  that controls some  $p < \frac{1}{2}$  fraction of the computational power of some blockchain the enclave interacts with, and also has physical access to the Tesseract server. Thus,  $\mathcal{A}$  can cut the communication between the enclave and any blockchain network, and feed the enclave with fake blocks.

Assuming a naive enclave implementation,  $\mathcal{A}$  can mount an *Eclipse attack* [33] as the following example illustrates:  $\mathcal{A}$  cuts the enclave off from the Bitcoin network and presents it with a fake blockchain containing a deposit transaction  $\mathsf{TX}_\mathsf{fake}$ . As a result, the enclave credits  $\mathcal{A}$  with a higher Bitcoin balance, which  $\mathcal{A}$  trades for Litecoin inside the enclave. When the enclave publishes the next settlement transactions on the two blockchains,  $\mathcal{A}$  will have traded her fake Bitcoin for real Litecoin: The Bitcoin settlement transaction will not be valid because it spends an output from  $\mathsf{TX}_\mathsf{fake}$  which was never included in the real Bitcoin blockchain. However, the Litecoin settlement transaction will be valid, resulting in  $\mathcal{A}$  profiting.

To defend against this attack, we rely on the fact that the rate at which  $\mathcal{A}$  can feed fake blocks to the enclave is at least twice slower than in the absence of an attack. (Since  $p < \frac{1}{2}$ .) Assuming that SGX has a trusted clock, the enclave can impose a rule that requires waiting for additional confirmations if the blocks arrive too slowly. We note that the Tesseract enclave is assumed to be running without interruptions, since our enclave code disallows rollbacks [75, 48] by design (cf. Section 6 regarding our approach to resiliency).

The time between every two consecutive Bitcoin blocks is an exponentially distributed random variable. Hence, for a rule that dictates whether blocks arrive too slowly we should consider the sum of exponential random variables, known as the Erlang distribution. Let n be the number of blocks that a deposit needs to be buried under before it is credited by the enclave. Let  $\delta$  be the multiplicative slowness factor by which blocks are allowed to arrive. E.g.,  $\delta = 3$  means that blocks that arrive 3 times slower than the expected time (or more slowly than that) will trigger the enclave to wait for n extra block confirmations before accepting any deposits.

Setting  $\delta$  to a high value reduces the probability of a false positive (i.e., a rejected deposit when no attack is taking place and the honest chain growth was unluckily slow during some timeframe). However, a high  $\delta$  also increases the prospects of an attack. For any  $\delta > 1$ , it is possible to set a large enough n so that the probability of a successful attack becomes negligible. However, a large n implies that honest users need to wait for a long time before their deposit is confirmed, which makes the Tesseract exchange service unattractive.

Table 2: Deposit confidence vs false positives

p	δ	n	$\Pr[\operatorname{Erlang}(n,p) \leq \delta n]$	$\Pr[\operatorname{Erlang}(n,1) > \delta n]$
$\frac{1}{10}$	2	60	$2^{-75}$	$2^{-31}$
$\frac{1}{10}$	2	120	$2^{-145}$	$2^{-58}$
$\frac{1}{5}$	1.5	120	$2^{-92}$	$2^{-21}$
$\frac{1}{4}$	1.3	120	$2^{-82}$	$2^{-10}$

In Table 2 we provide exemplary concrete parameters for n and  $\delta$ . E.g., the third row of Table 2 shows that with n = 120 (20 hours on average in Bitcoin) and  $\delta = 1.5$ :

- An adversary with computational power  $p \le \frac{1}{5}$  can mount a successful eclipse attack on the enclave with probability  $2^{-92}$  or smaller.
- On expectation, an honest user will need to wait for extra confirmations once in every ≈ 2 million deposits that she makes.

While the concrete parameters that can be obtained are already quite reasonable, let us stress that prudent users of the Tesseract exchange will not be exposed to eclipse attacks at all. Any user can simply compare the latest blocks in the actual cryptocurrency networks with the latest blocks that Tesseract enclave publishes (with remote attestation), and cancel her bids/asks in case of a discrepancy. In the example above, the honest  $P_j$  will avoid  $P_i$ 's attack by observing that the latest Bitcoin blocks that Tesseract published are inconsistent with the real Bitcoin network, and refuse to trade her LTC for BTC. Our practical instantiation of Tesseract has another layer of security that further protects (incautious) users from eclipse attacks, see Section 5.3.

#### 4.2 Secure Communication

For each user who has already opened an account with Tesseract, we establish a secure channel (TLS) when the user wishes to communicate with the enclave. The reasons for a channel with authenticated encryption are the following:

• Fast identification: The authenticated messages in the TLS Record Protocol are computed via symmetric-key operations, after the initial key exchange (done via public-key operations in the Handshake Protocol) to establish the channel. Since symmetric-key operations are an order of magnitude faster than public-key operations, a persistent TLS connection delivers performance suitable for real-time trades.

• Frontrunning prevention: An adversary can try to inspect the entire communication flow that arrives at the Tesseract server, learn information regarding real-time actions of other users, and exploit this information to perform trades. Encrypted communication avoids such attacks.

An example of a frontrunning attack is shown in Figure 5. There, Alice believes that the BTC price is going to rise. Therefore, she places an order to buy 10 BTC at \$870 each, so that any of the current sellers will match her order first. On the other hand, Bob believes that the price of BTC is going to drop, and he therefore places an order to sell his 10 BTC for a price that is as low as \$820. Given the public order book, Bob's intention is thus to sell 2 BTC for \$850, 5 BTC for \$840, and 3 BTC for \$820. If the trades are executed in this order, it will be to the benefit of Bob, because he will actually sell 10 BTC to Alice for \$870 each. However, an adversary with this knowledge can permute the orders and insert her own new orders. In this scenario, the adversary would be guaranteed to gain  $\$10 \cdot (870 - 851) = \$190$ , by buying Bob's 10 BTC cheaply and then selling it to Alice.

Since all users send encrypted messages through their secure channels, an adversary with a physical control of the Tesseract server cannot frontrun other users. To the best of our knowledge, all the other designs of cryptocurrency exchanges are exposed to these kinds of frontrunning attacks.

We note that an adversary may still observe patterns of communication at the IP-level and try to learn information about the traders. An IP-level anonymizer (e.g., Tor [24]) is inapplicable as a mitigation technique against such adversaries, since users wish to perform real-time trades. However, the user's client can randomly inject dummy data into the TLS channel (which would be ignored on arrival), thereby making it more difficult to track communication patterns. Furthermore, in future versions of Tesseract we plan to allow users to upload an algorithmic trading program to their enclave account (for a fee), that will enable them to issue multiple trading orders without communication with the server. The use of automated trading programs is quite popular in centralized exchanges (cf. [7]), although these automated traders do communicate each of their orders to the server.

#### 4.3 Double Attestation

Several reputable providers may wish to offer different variants of the Tesseract service (perhaps with their own tokenized coins and fiat assets, see Sections 7 and 8). This raises the following question: does a single entity (i.e., the hardware manufacturer) have the power to compromise the security of all the Tesseract-based platforms, simultaneously?

Order Book (BTC/USD)			
	Buying	Selling	
	Price Volume	Price Volume	
	\$850 2	\$890 3	
	\$840 5	\$906 5	
	\$820 5	\$945 4	
	Arrival of new ord 1. Alice: bu 2. Bob: sel	y(\$870, 10)	
	Frontrunning:		
	<ol> <li>Adversary:</li> </ol>	buy(\$851, 10)	
	2. Bob:	sell(\$820, 10)	
	<ol><li>Alice:</li></ol>	buy(\$870, 10)	
	<ol> <li>Adversary:</li> </ol>	sell(\$870, 10)	

Figure 5: Example of frontrunning.

No such single entity exists with regard to centralized exchanges (cf. Section 3.1), because these exchanges are independent of one another. That is to say, a security breach of one centralized exchange will not have a direct impact on the users of the other centralized exchanges.

For trusted hardware with remote attestation support, the plain way that the manufacturer can break security is by attesting to fraudulent data. In our context, suppose for example that there are two Tesseractbased exchanges  $X_1, X_2$  that invite users to deposit their funds to PK<sub>SGXBTC1</sub> and PK<sub>SGXBTC2</sub>, respectively. If Intel has knowledge of the secret signing keys  $sk_1, sk_2$  that are embedded into the CPUs of  $X_1$  and  $X_2$ , then it can forge signatures that attest to fresh ephemeral public keys PK'<sub>SGXBTC1</sub>, PK'<sub>SGXBTC2</sub> that Intel would generate together with the corresponding secret keys  $SK'_{SGXBTC1}, SK'_{SGXBTC2}$ . Thus, Intel will be able deceive users into sending their deposits to  $PK'_{SGXBTC1}$ ,  $PK'_{SGXBTC2}$ , and then steal funds that users wished to deposit to  $X_1, X_2$ .

The manufacturer may also break security by embedding malicious logic into the hardware. For instance, whenever an application executes code that generates a (supposedly) random secret key, the key will actually be generated in a way that can be predicted by the manufacturer. While this attack would be easy enough if there were one assembly opcode that generates a random key (the malicious opcode can use a randomness source with low entropy), it is far more difficult to achieve predictable behavior for any application-level code that is executed by a general-purpose CPU.

Another attack vector that the hardware manufacturer may attempt is simply to send the data that a CPU generates over the network (to the manufacturer's address), without consent or knowledge of the administrator of the server computer. This is indeed a concern with Intel's Management Engine (see [63]), but it is not an inherent defect of the trusted hardware model (and hopefully the Management Engine will soon have

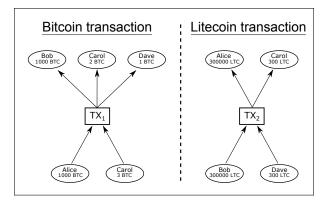


Figure 6: The cross-chain settlement problem.

an opt-out option).

Similarly to [71], the Tesseract platform protects against false remote attestation by attaching a secondary signature – created by the administrator of the platform – to the attested data. Following the above example, the users of  $X_1$  (resp.  $X_2$ ) will take into consideration the reputation of the administrator of  $X_1$  (resp.  $X_2$ ), and reject the attested data unless it was signed both by the SGX CPU and by the reputable administrator. This means that the hardware manufacturer alone cannot attack all Tesseract-based exchanges, since the manufacturer has to collude with the administrator of an exchange in order to create a fraudulent attestation. This implies that Tesseract is strictly more secure than centralized exchanges.

The double attestation mechanism is also efficient, since the secondary signature is rarely needed. Specifically, the secondary signature is required only once for the identity public key of the enclave (which is the hardware-associated public key of Section 6.1), and this identity can then establish the TLS channel with each user. All further communication in a TLS channel (e.g., bid/ask orders) is done without attestation. For non-user-specific data such as real-time updates to the public order book, the secondary signature is already implicit if HTTPS is used to view this data.

#### 5 Atomic Cross-Chain Settlements

Assume first that Tesseract only supports the trading of digital assets (cf. Section 8) that circulate within a single cryptocurrency. In this case, the publication of each settlement transaction — that reflects the account balances of the users after trading in a time period — does not entail the risk of an adversary stealing funds from honest users. The reason is that an invalid deposit (see Section 4.1) or blockage of the settlement will amount just to a DoS attack, since all the users will claim their prior funds after the time limit in the output of

their original deposit (or the last settlement transaction) expires.

On the other hand, trading among multiple cryptocurrency systems (that are independent of one another) may allow an adversary to steal funds from honest users. We provide an illustration of the risk in Figure 6. Suppose for instance that 1 BTC is worth \$2000, and also that the market price of 1 BTC is 300 LTC. In the illustration, Alice and Bob traded 1000 BTC (i.e., \$2 million worth of BTC) for 300000 LTC (i.e., \$2 million worth of LTC), while Carol and Dave traded 1 BTC for 300 LTC. Thus, the enclave will construct and sign the Bitcoin and Litecoin settlement transactions, and attempt to broadcast the settlements to the Bitcoin and Litecoin networks. An adversary with physical access to the Tesseract server can collude with Alice and intercept the Bitcoin settlement transaction when it leaves the CPU but before it is broadcast to the Bitcoin network, and let the Litecoin settlement transaction go through and reach the Litecoin network. The result is that the transfer of ownership of \$2 million worth of LTC from Bob to Alice will be committed on the Litecoin system, while the transfer of ownership of \$2 million worth of BTC will never occur. In effect, Bob lost \$2 million worth of funds to Alice.

Let us provide security definitions that capture the above fairness problem.

**Definition 1** (All-or-nothing settlement). Given the transaction  $tx_1$  for system  $C_A$  and the transaction  $tx_2$  for system  $C_B$ , an all-or-nothing cross-chain settlement is a protocol that guarantees that

- 1. Both  $tx_1$  will become confirmed on system  $C_A$  and  $tx_2$  will become confirmed on system  $C_B$ , or
- 2. Neither  $tx_1$  will become confirmed on system  $C_A$  nor will  $tx_2$  become confirmed on system  $C_B$ .

In our context,  $C_A$  and  $C_B$  are cryptocurrencies. We stress that parties that execute the consensus protocol for  $C_A$  may be unaware of the existence of  $C_B$ , and vice versa.

Notice that Definition 1 does not imply that honest users are fully protected against financial loss. Specifically, an adversary  $\mathcal{A}$  that prevents both  $tx_1$  and  $tx_2$  from being confirmed may benefit at the expense of honest users:  $\mathcal{A}$  may wish to renege on a trade after observing some external events and/or price fluctuations that worked to her disadvantage. Still, Definition 1 implies better security than that of the commonplace centralized exchanges (cf. Section 3.1), because the users of such centralized exchanges run not only the risk that their trades will be reversed but also the risk that their initial funds will be stolen.

**Definition 2** (Unprivileged settlement). Let  $U_1^{\text{in}}, U_2^{\text{in}}$  denote the sets of users in the inputs of the transactions  $tx_1, tx_2$ , and let  $U_1^{\text{out}}, U_2^{\text{out}}$  denote the sets of users in the outputs of  $tx_1, tx_2$ . Let  $U = U_1^{\text{in}} \cup U_2^{\text{in}} \cup U_1^{\text{out}} \cup U_2^{\text{out}}$ . An unprivileged cross-chain settlement is a protocol that satisfies Definition 1 in the presence of an adversary A who can obtain any information that every user  $P \in U$  accesses, at the moment that the information was accessed.

In essence, Definition 2 implies that honest traders cannot utilize secret data during the settlement protocol (such as picking a secret  $x \in \{0,1\}^{\lambda}$  in the first step of the ACCS protocol in Section 2), because  $\mathcal{A}$  could break the security by gaining access to any sensitive data that honest traders attempt to use. Thus, Definition 2 captures a rushing adversary who has physical control over the SGX server and can intercept all the data that leaves the CPU, before honest users have an opportunity to make use of this data in a secure fashion. Note that Definition 2 does not permit  $\mathcal{A}$  to observe the secret keys that enable honest users to spend their funds, as long as honest users do not access their secret keys during the settlement protocol.

In fact, Definition 2 gives A more power than a realworld adversary with physical control over the SGX server. Consider for instance a protocol where in the first step the enclave encrypts data using Carol's public key, and attempts to send the encrypted data to Carol over the network. In that case, A will not be able to obtain the data that Carol accesses; the only action available to A is to mount a DoS attack and not let the protocol make progress. The motivation for the more conservative definition is that we wish to support settlement transactions among a large number of users (e.g., thousands) and multiple cryptocurrency systems, where the users can be anonymous and can create Sybil accounts. In this setting, it is difficult to design a secure protocol that sends sensitive data to rational users (with the expectation that they will act in their own selfinterest), due to the possibility of malicious coalitions with Sybils who would be willing to sacrifice some of their funds. For this reason, Definition 2 denies the enclave the power to communicate privately with individual users.

Thus, intricate solutions to the all-or-nothing settlement problem are needed mainly because our goal is to support many anonymous traders. Let us in fact demonstrate that with a few users, the all-or-nothing settlement problem can become easy. In Figure 7, Alice and Bob again wish to trade \$2 million worth of BTC for LTC, but they are the only users of the Tesseract exchange. Here, the enclave prepares the settlement transactions  $\mathsf{TX}_1, \mathsf{TX}_2$  that keep the enclave in control in the next two weeks (2000 blocks where  $T_1$  is the head

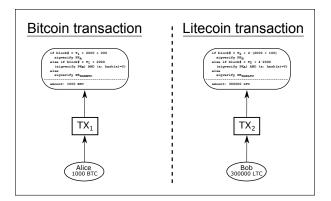


Figure 7: Settlement with two parties.

of the Bitcoin blockchain, and 8000 blocks where  $T_2$  is the head of the Litecoin blockchain). This enables Alice and Bob to continue to trade, if they wish to. The secret data  $x \in \{0,1\}^{\lambda}$  is generated inside the enclave. After the enclave receives evidence that  $\mathsf{TX}_1$  and  $\mathsf{TX}_2$  are both confirmed, it sends x in encrypted form *only* to Alice, over a secure channel. After the two weeks, the outputs can be redeemed using x, otherwise the timeouts allow the funds to be returned to each user. As with the ACCS protocol (cf. Section 2), the timeout in  $\mathsf{TX}_1$  is longer, so Bob will have enough time to redeem the 1000 BTC after Alice reveals x, spending 300000 LTC.

Let us note that Definition 2 does not give  $\mathcal{A}$  the power to observe secret information inside the enclave. In the Tesseract implementation, this is justified because we use a constant-time constant-memory library for cryptographic operations [83], reducing the potential for side-channels greatly.

We now present solutions to the all-or-nothing settlement problem, in a setting that involves many anonymous traders.

## 5.1 Naive Protocols

To clarify why an intricate protocol is needed, we first describe a simple protocol  $\Pi_{\text{simp}}$  that relies on N extra servers  $S_1, S_2, \ldots, S_N$  that are supposedly reputable. See Figure 8.

The cryptocurrency systems  $C_1$  and  $C_2$  can be for example Bitcoin and Litecoin as in Figure 6. The embedding of K into  $\mathsf{TX}_1$  and  $\mathsf{TX}_2$  can be done with the OP\_RETURN script instruction [8], which allows storing arbitrary data on the blockchain as an unspendable output (for a small fee). It is not possible to mount a malleability attack that removes K from  $\mathsf{TX}_1$  or  $\mathsf{TX}_2$ , because the signatures for  $\mathsf{TX}_1$  and  $\mathsf{TX}_2$  are over the entire transaction data (i.e., data that includes the OP\_RETURN output).

Since information that is published on a blockchain

#### Protocol $\Pi_{\text{simp}}$

- 1. The enclave picks a symmetric key  $K \in \{0,1\}^{\lambda}$ .
- 2. The enclave embeds K into  $\mathsf{TX}_1, \mathsf{TX}_2$ .
- 3. The enclave sends  $ct = \mathsf{encrypt}_K(\mathsf{TX}_1, \mathsf{TX}_2)$  to  $S_1, S_2, \dots, S_N$ .
- 4. The enclave waits for acknowledgements from  $S_1, S_2, \dots, S_N$ .
- 5. The enclave broadcasts  $TX_1$  to  $C_1$  and  $TX_2$  to  $C_2$ .
- 6. Each  $S_j$  that sees  $\mathsf{TX}_i$  but not  $\mathsf{TX}_{3-i}$  will fetch K from  $\mathsf{TX}_i$ , decrypt ct, and broadcast  $\mathsf{TX}_{3-i}$  to  $\mathcal{C}_{3-i}$ .

Figure 8: Naive protocol for fair settlement.

becomes publicly available, the idea behind  $\Pi_{\text{simp}}$  is that any non-corrupt server  $S_i$  will be able to impose fairness by fetching K from a public blockchain and decrypting the ciphertext ct, because ct is already in  $S_i$ 's possession.

Unfortunately,  $\Pi_{\text{simp}}$  is insecure, due to a race condition. The adversary  $\mathcal{A}$  can intercept both  $\mathsf{TX}_1$  and  $\mathsf{TX}_2$ , but broadcast neither of them initially. Since the users' outputs must have a time limit (see Section 4),  $\mathcal{A}$  will wait until an input (that belongs to a corrupt user  $P_j$ ) in  $\mathsf{TX}_i$  is about to expire, and then broadcast  $\mathsf{TX}_{3-i}$ . Then,  $\mathcal{A}$  will instruct  $P_j$  to spend that input, thereby invalidating  $\mathsf{TX}_i$ . Hence, even if all of the servers  $S_1, S_2, \ldots, S_N$  are honest, they may not have enough time to fetch K from  $\mathsf{TX}_{3-i}$  and broadcast the decrypted  $\mathsf{TX}_i$ .

If the cryptocurrency systems  $C_1, C_2$  allowed transactions to embed large arbitrary data, then it would have also been possible to eliminate the reliance on  $S_1, S_2, \ldots, S_N$ . Briefly, each  $\mathsf{TX}_i$  will embed the  $\mathsf{TX}_{3-i}$  data in a designated output, the enclave will broadcast both  $\mathsf{TX}_1$  and  $\mathsf{TX}_2$ , and any user would then have the opportunity to enforce fairness. This would bloat  $C_i$  with the entire  $\mathsf{TX}_{3-i}$  data, which is undesirable — there are risks associated with a popular decentralized cryptocurrency that allows embedding of large data (e.g., illegal content). In any event, this approach is insecure due to the same race condition that  $\Pi_{\text{simp}}$  exhibits.

In the following section, we give a theoretical protocol  $\Pi_{\text{theo}}$  that avoids the race condition, using scripts with PoW-based logic that ensures the occurrence of certain conditions on another blockchain.

# **5.2** Theoretical Protocol

Let us present a theoretical protocol for the all-or-nothing settlement problem, which solves the race condition that Section 5.1 elaborates upon. Following Section 5 and Figure 6, we condition the second settlement

Functionality RMIT (refundable multi-input transaction)

Notation: let C be a cryptocurrency system.

Upon receiving  $tx = (\{in_1, ..., in_k\}, \{out_1, ..., out_n\}, \phi_1, \phi_2)$ 

- 1. Verify  $\forall j \in [k]$ :  $in_j$  is unspent in C.
  - If the verification failed then abort.
- 2. Verify  $\sum_{j=1}^{k} \operatorname{amount}(in_j) \ge \sum_{j=1}^{n} \operatorname{amount}(out_j)$ .
  - If the verification failed then abort.
- 3. Make  $\{in_1, \ldots, in_k\}$  unspendable in C.
- 4. Wait to receive a witness w
  - (a) If  $\phi_1(w) = 1$  then commit  $\{out_1, \dots, out_n\}$  to C, and terminate.
  - (b) If  $\phi_2(w) = 1$  then make  $\{in_1, ..., in_k\}$  spendable in C, and terminate.
  - (c) Otherwise, return to Step 4.

Figure 9: The ideal functionality RMIT.

transaction  $\mathsf{TX}_2$  on the result of the first settlement transaction  $\mathsf{TX}_1$ , by constraining  $\mathsf{TX}_2$  with PoW-based predicates that verify certain events' occurences on another blockchain.

As we will see, this approach is problematic with the current Bitcoin protocol. Thus, we first describe the settlement protocol in an hybrid world that has an ideal "refundable multi-input transaction" (RMIT) functionality, defined in Figure 9.

The description of  $TX_1$ ,  $TX_2$  is outlined in Figure 10. We use the notation  $TX_{i,j}$  to denote that  $TX_i$  was updated by supplying w that satisfied  $\phi_i$ . The secrets  $x_1 \in \{0,1\}^{\lambda}, x_2 \in \{0,1\}^{\lambda}$  are generated inside the enclave. The predicates  $\phi'_1, \phi'_2$  are specified in Figure 11. To elaborate, the hardcoded parameter  $D_0$  specifies a difficulty level for PoW mining,  $\ell_1$  is an upper bound on the length of an authentication path of a Merkle tree, and  $\ell_2$  is a PoW confidence parameter. The input witness w for  $\phi'_1$  consists of up to  $\ell_1$  sibling hash values  $v_i$  in the authentication path (with direction  $d_i \in \{'L', 'R'\}$ ) for the leaf transaction y, together with exactly  $\ell_2$  block headers  $H_1, H_2, \dots, H_{\ell_2}$ . The predicate  $\phi'_1$  will verify that  $\mathsf{TX}_{1,1}$ is in a leaf that reaches some root value r, and that ris extended by valid proofs of work  $H_1, H_2, \dots, H_{\ell_2}$  that meet the difficulty level  $D_0$ . The input witness w for  $\phi_2$ does the same, but also verifies that there is a valid PoW chain of at least  $\ell_3$  blocks between the hardcoded  $b_1$  and  $TX_{1,2}$ .

We describe the theoretical protocol  $\Pi_{theo}$  for all-ornothing settlement in Figure 12. Note that the enclave

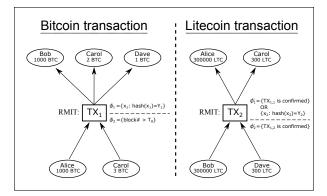


Figure 10: Theoretical fair settlement transactions.

constructs  $\mathsf{TX}_2$  only after it receives the evidence that  $\mathsf{TX}_1$  was confirmed in the end of Step 1, by hardcoding  $b_1$  as the hash of the block in which  $\mathsf{TX}_1$  resides.

Essentially,  $\Pi_{\text{theo}}$  avoids the race condition by first making sure that  $\mathsf{TX}_1$  was resolved on the cryptocurrency system  $\mathcal{C}_1$  either by committing the output or by committing the inputs, and then allowing  $\mathsf{TX}_2$  to commit accordingly in the cryptocurrency system  $\mathcal{C}_2$ . If  $\mathcal{A}$  carries out a DoS attack before  $x_1$  is released in Step 3, then the users will gain possession of their inputs in the  $\mathcal{C}_1$  after block  $T_0$  is reached (see Figure 10), which would be followed by the miners of  $\mathcal{C}_1$  starting to create a witness w that satisfies  $\phi_2'(w) = 1$  and thus allowing the users to gain possession of their inputs in  $\mathcal{C}_2$ . If the enclave exposes  $x_1$  in Step 3, then it is still the case that the miners of  $\mathcal{C}_1$  will be harnessed to resolve  $\mathsf{TX}_1$  in one of the two possible ways.

In the case that no attack is taking place, the enclave will release  $x_2$  in Step 4, thereby allowing the settlement to complete quickly and without asking the miners of  $C_2$  to evaluate a complex condition that relates to another blockchain.

However, the assumption regarding the computational power of  $\mathcal{A}$  has to be slightly less conservative in comparison to the power that is needed to mount a classical double-spending attack [68], because  $\Pi_{\text{theo}}$  enables  $\mathcal{A}$  to gain a minor head start that depends on the parameter  $T_0$ . Specifically,  $\mathcal{A}$  can intercept  $x_1$  in Step 3 and use her own computational power (and  $x_1$ ) to create a hidden chain  $w_1$  that spends  $\mathsf{TX}_1$  into  $\mathsf{TX}_{1,1}$ . The miners of  $\mathcal{C}_1$  will create the witness  $w_2$  in which  $\mathsf{TX}_1$  is spent into  $\mathsf{TX}_{1,2}$ , but they will only begin to work on  $w_2$  after block  $T_0$  is reached.

The success probability of an attack with a duration of  $T_1$  blocks for the head start is

$$\sum_{k=0}^{\infty} \Big( \Pr[\operatorname{NegBin}(T_1, p) = k] \cdot \Pr[\operatorname{NegBin}(\ell_2, p) \ge \ell_2 - k] \Big).$$

The first negative binomial variable counts the number

# Predicate $\phi_1'$

Hardcoded parameters:  $\mathsf{TX}_1, D_0, \ell_1, \ell_2$ Input:  $w = ((v_1, d_1), (v_2, d_2), \dots, (v_k, d_k), y, H_1, H_2, \dots, H_{\ell_1})$ 

- 1. Embed  $hash(TX_{1,1})$  into y
- 2. For j = 1 to  $min(k, \ell_1)$ 
  - If  $d_j = L$  then  $y := \mathsf{hash}(y, v_j)$  else  $y := \mathsf{hash}(v_j, y)$
- 3. For j = 1 to  $\ell_2$ 
  - Embed y into  $H_i$  and compute  $y := hash(H_i)$
  - If  $y > D_0$  then return false
- 4. return true

# Predicate $\phi_2'$

Hardcoded parameters:  $\mathsf{TX}_1, D_0, \ell_1, \ell_2, \ell_3, b_1$ Input:  $w = (G_1, \dots, G_n, (v_1, d_1), \dots, (v_k, d_k), y, H_1, \dots, H_{\ell_2})$ 

- 1.  $z := b_1$
- 2. For j = 1 to  $\max(n, \ell_3)$ 
  - Embed z into  $G_i$  and compute  $z := hash(G_i)$
  - If  $z > D_0$  then return false
- 3. Embed  $hash(TX_{1,2})$  into y
- 4. For j = 1 to  $min(k, \ell_1)$ 
  - If  $d_j = L$  then  $y := \mathsf{hash}(y, v_j)$  else  $y := \mathsf{hash}(v_j, y)$
- 5. Embed y into  $H_1$  and compute  $y := hash(H_1)$
- 6. If  $y \neq z$  then return false
- 7. For j = 2 to  $\ell_2$ 
  - Embed y into  $H_i$  and compute  $y := hash(H_i)$
  - If  $y > D_0$  then return false
- 8. return true

Figure 11: The cryptocurrency scripts  $\phi'_1, \phi'_2$ .

of blocks that  $\mathcal{A}$  creates during the time that the honest miners are creating  $T_1$  blocks. This corresponds to the head start, because these  $T_1$  blocks will not contribute to the witness that the predicate  $\phi_2'$  requires. The second negative binomial variable counts the number of blocks that  $\mathcal{A}$  creates while the honest miners are creating  $\ell_2$  blocks. If  $\mathcal{A}$  can extend her head start to reach  $\ell_2$  or more blocks before the honest miners, then the attack succeeds.

In Table 3, we give exemplary figures for the attack on  $\Pi_{\text{theo}}$ . For easy comparison, we also include the success probability without a head start (i.e.,  $T_1 = 0$ ), which is

#### Protocol $\Pi_{theo}$

- The enclave releases TX<sub>1</sub> and waits for evidence that it was confirmed on the system C<sub>1</sub>.
- The enclave releases TX<sub>2</sub> and waits for evidence that it was confirmed on the system C<sub>2</sub>.
- The enclave releases x<sub>1</sub> and waits for evidence that TX<sub>1,1</sub> was confirmed on the system C<sub>1</sub>.
- 4. The enclave releases  $x_2$ .

Figure 12: Theoretical protocol for fair settlement.

Table 3: Breaking the security of  $\Pi_{\text{theo}}$ 

p	$T_1$	$\ell_2$	with head start	with $T_1 = 0$
$\frac{1}{3}$	6	50	0.0016	0.0003
$\frac{1}{5}$	10	50	$2^{-30}$	$2^{-37}$
$\frac{1}{5}$	6	50	$2^{-33}$	$2^{-37}$
$\frac{1}{5}$	6	100	$2^{-65}$	$2^{-69}$
$\frac{1}{10}$	20	50	$2^{-64}$	$2^{-79}$
$\frac{1}{10}$	10	50	$2^{-71}$	$2^{-79}$
$\frac{1}{10}$	10	100	$2^{-145}$	$2^{-153}$

simply the probability  $Pr[NegBin(\ell_2, p) \ge \ell_2]$ .

For the opposite attack, A may intercept  $x_1$  in Step 3 and then create a hidden chain  $w_2$  that excludes  $x_1$ . With this attack strategy, A will broadcast  $x_1$  to  $C_1$  right before the timeout  $T_0$  is reached, in hope that her hidden chain  $w_2$  will outcompete the chain that the miners of  $C_1$  begin to create. This attack vector is mitigated by disallowing a precomputation of  $w_2$ . Specifically, the enclave hardcodes  $b_1$  into  $\mathsf{TX}_2$ , and the predicate  $\phi_2'$  verifies that  $b_1$  is buried under at least  $\ell_3$  PoW blocks.

The parameter  $\ell_3$  should be set to  $2\ell_2 + T_1$ . This gives a time span of  $T_1$  blocks to update  $\mathsf{TX}_1$  into  $\mathsf{TX}_{1,1}$ , after the enclave received the evidence that  $\mathsf{TX}_1, \mathsf{TX}_2$  were confirmed and thus revealed  $x_1$ . The parameter  $T_1$  should not be too low, to avoid the cancellation of the settlements in case of a short network outage or a slow chain growth in  $C_2$  relative to  $C_1$ .

In the current Bitcoin network,  $\ell_1=12$  suffices, hence the predicates  $\phi_1',\phi_2'$  require  $\leq 12+\ell_2+\ell_3$  hash invocations for confidence level  $\ell_2$ . Given that the complexity of ECDSA signature verification is an order of magnitude higher than that of invoking a hash function, moderate values such as  $\ell_2=50$ ,  $T_1=10$ ,  $\ell_3=2\ell_2+T_1=110$  imply that Bitcoin miners can validate the scripts  $\phi_1',\phi_2'$  for a mild fee. These parameters for PoW-based SPV proofs can be even better if the

cryptocurrency system supports NIPoPoW [36].

It is unlikely that  $\Pi_{\text{theo}}$  will be vulnerable to an attack that embeds a transaction that spends  $\mathsf{TX}_1$  into  $\mathsf{TX}_{1,1}$  or  $\mathsf{TX}_{1,2}$  in another cryptocurrency system  $\mathcal{C}_3$ , where  $\mathcal{C}_3$  has the same PoW hash function and the same difficulty level. The reason is that the txid hash of  $\mathsf{TX}_1$  in the leaf of the Merkle tree is determined according to the prior history that goes back to the genesis block of  $\mathcal{C}_1$ . Unless  $\mathcal{C}_3$  allows the input of a transaction to consist of arbitrary data,  $\mathcal{A}$  will need to mount a preimage attack that creates valid transaction in  $\mathcal{C}_3$  with a particular value (i.e., the txid of  $\mathsf{TX}_1$ ) as its hash.

The main obstacle to an implementation of  $\Pi_{theo}$  in Bitcoin is the RMIT functionality. It is possible to implement the specific RMIT that  $\Pi_{theo}$  requires by creating a transaction  $tx_{init}$  that spends the inputs into a single output that is controlled by the secret signing key of Tesseract, and creating a refund transaction  $tx_{\text{refund}}$  that has locktime [80] of  $T_0$  and spends the output of  $tx_{init}$  back into the inputs. After the enclave receives evidence that  $tx_{refund}$  is publicly available, it will broadcast  $tx_{init}$  to the Bitcoin network. When the execution of  $\Pi_{\text{theo}}$  reaches Step 3 and the enclave needs to release  $x_1$ , it will broadcast a transaction  $tx_{commit}$ that spends the output of  $tx_{init}$  into the desired outputs. The only problem with this procedure is that there is no good way to make  $tx_{refund}$  publicly available while relying on the security of Bitcoin alone. In a purely theoretical sense, it is possible to make  $tx_{refund}$  available by storing it as arbitrary data on the Bitcoin blockchain using OP\_RETURN, but this will be very costly because the size of  $tx_{refund}$  can be dozens of kilobytes and the capacity of an OP\_RETURN output is only 80 bytes. An efficient version of RMIT can be done via a Bitcoin protocol fork: an initial transaction will mark both the inputs and the new outputs as unspendable in the UTXO set, and a subsequent transaction will supply a witness to  $\phi_1$  or  $\phi_2$  and thereby ask the miners to make either the inputs or the outputs spendable (for a fee). An Ethereum implementation of a RMIT contract is possible, but it should be noted that  $\Pi_{theo}$  (and its generalization to more than two systems) requires RMIT support by all the cryptocurrency systems that are involved in the settlement.

Our analysis of  $\Pi_{theo}$  gives the essential security arguments for a protocol that enables an all-or-nothing settlement. A formal security proof of  $\Pi_{theo}$  (as well as  $\Pi_{prac}$  of Section 5.3 and  $\Pi_{RTExch}$  of Section 6) requires a rigorous model for the cryptocurrency consensus system — such as GKL [29] or PSS [58] — together with a rigorous model that is rich enough to express the scripting language that controls the users' coins (see, e.g., [53]). In Appendix A we provide a formal security proof (under certain assumptions) for the

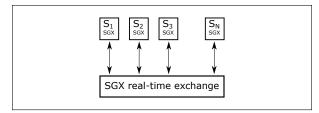


Figure 13: Practical fair settlement.

ACCSs protocol of Section 2, that also serves to show several of the ingredients that a proof for  $\Pi_{theo}$  needs to incorporate.

# 5.3 Practical Protocol

The theoretical protocol  $\Pi_{theo}$  of Section 5.2 is resilient against an adversary who has total access to the server machine, except for the data that is inside the SGX CPU. Here, we present a practical protocol  $\Pi_{prac}$  for the allor-nothing settlement problem that relaxes this resiliency aspect, but in fact offers better security in other respects.

Our strategy is to distribute the trust among N additional servers that are all running SGX enclaves (see Figure 13), and ensure that  $\Pi_{\text{prac}}$  satisfies Definition 2 if there exists at least one server  $S_j \in \{S_1, S_2, \ldots, S_N\}$  that is beyond the reach of the adversary  $\mathcal{A}$ . That is to say, we assume that  $S_j$  can communicate with cryptocurrencies  $\mathcal{C}_1, \mathcal{C}_2$  without interference.

The main idea of  $\Pi_{prac}$  is to emulate the essential characteristic of the theoretical protocol  $\Pi_{theo}$ , which is to wait for a proof that the settlement transaction  $\mathsf{TX}_1$  was either committed to  $\mathcal{C}_1$  or cancelled, and then do the same for the settlement transaction  $\mathsf{TX}_2$ .

The settlement protocol  $\Pi_{\text{prac}}$  that Tesseract and the servers  $S_1, S_2, \ldots, S_N$  execute is specified in Figure 14. As a prerequisite, the Tesseract server and  $S_1, S_2, \ldots, S_N$  need to share a symmetric secret key K that is known only to their enclaves. The transactions  $\mathsf{TX}_1^c, \mathsf{TX}_2^c$  are "cancellation" transactions that invalidate the settlement transactions  $\mathsf{TX}_1, \mathsf{TX}_2$ , respectively. In Bitcoin,  $\mathsf{TX}_i^c$  can be implemented simply by spending one of the inputs of  $\mathsf{TX}_i$  into a new output that is identical to that input (this will cause  $\mathsf{TX}_i, \mathsf{TX}_i^c$  to conflict with each other).

Thus, the protocol  $\Pi_{prac}$  seeks to preserve the property that  $\mathsf{TX}_2$  remains confidential inside the enclaves for as long as  $\mathsf{TX}_1$  is not yet confirmed. This property avoids the risk that  $\mathsf{TX}_i, \mathsf{TX}_{3-i}^c$  will compete for confirmations at the same time, as that can easily violate the all-ornothing requirement.

In the case that at least one server  $S_i$  is not under physical attack, we have that either  $\mathsf{TX}_1$  or  $\mathsf{TX}_1^c$  will be broadcast to  $\mathcal{C}_1$  within  $T_1$  blocks. As a consequence, either  $\mathsf{TX}_1$  or  $\mathsf{TX}_1^c$  will be confirmed after  $T_1 + \ell_2$  blocks.

## Protocol $\Pi_{prac}$

- 1. Tesseract sends ct = encrypt $_K(\mathsf{TX}_1,\mathsf{TX}_2,\mathsf{TX}_1^c,\mathsf{TX}_2^c)$  to  $S_1,S_2,\ldots,S_N$ .
- 2. For every  $i \in [N]$ , Tesseract waits for acknowledgement from  $S_i$  that it received ct.
- 3. Tesseract broadcasts  $TX_1$  to  $C_1$ .
- Starting from the time at which it received *ct* in Step

   each server S<sub>i</sub> ∈ {S<sub>1</sub>, S<sub>2</sub>,...,S<sub>N</sub>} inspects the next blocks of C<sub>1</sub>
  - If  $S_i$  does not see  $\mathsf{TX}_1$  on  $\mathcal{C}_1$  within  $T_1$  blocks, then it broadcasts  $\mathsf{TX}_1^\mathsf{c}$  to  $\mathcal{C}_1$ .
  - If  $S_i$  sees that  $\mathsf{TX}_1$  has  $\ell_2$  extra confirmations on  $\mathcal{C}_1$ , then it broadcasts  $\mathsf{TX}_2$  to  $\mathcal{C}_2$ .
  - If  $S_i$  sees that  $\mathsf{TX}_1^c$  has  $\ell_2$  extra confirmations on  $\mathcal{C}_1$ , then it broadcasts  $\mathsf{TX}_2^c$  to  $\mathcal{C}_2$ .

Figure 14: Practical protocol for fair settlement.

This allows  $S_i$  or one of the other non-adversarial servers to broadcast the appropriate transaction (i.e.,  $TX_2$  or  $TX_2^c$ ) to the cryptocurrency system  $C_2$ , causing it to be confirmed too.

The adversary A may attempt to mount a race attack with a head start of  $T_1$  blocks, by eclipsing one of the servers  $S_i$ . The attack can proceed as follows:

- A intercepts the data TX<sub>1</sub> that Tesseract reveals in Step 3 of Π<sub>prac</sub>, and deactivates the Tesseract server.
- 2. A eclipses the server  $S_j$ , and feeds it with a fake blockchain (generated by A herself) that contains  $TX_1$ .
- When the enclave of S<sub>j</sub> becomes convinced that TX<sub>1</sub> was confirmed, it releases TX<sub>2</sub>.
- 4.  $\mathcal{A}$  waits until  $\mathsf{TX}_1^c$  is confirmed on  $\mathcal{C}_1$ , and then broadcasts  $\mathsf{TX}_2$  to  $\mathcal{C}_2$ .

As with  $\Pi_{\text{theo}}$ , the reason that  $\mathcal{A}$  obtains a head start is that the honest participants wait for a duration of  $T_1$  blocks before they attempt to invalidate  $\mathsf{TX}_1$ , whereas  $\mathcal{A}$  begins to create her fake chain immediately — see Section 5.2 and Table 3 for analysis. Note that the purpose of the cancellation transaction  $\mathsf{TX}_2^c$  is to defeat this race attack, in the case that  $\mathcal{A}$  fails to generate  $\ell_2$  blocks while the honest network generates  $T_1 + \ell_2$  blocks.

In fact, it is more difficult for  $\mathcal{A}$  to exploit the head start and attack  $\Pi_{\text{prac}}$ , than it is to attack  $\Pi_{\text{theo}}$ . This is because  $\Pi_{\text{prac}}$  can specify the precise duration  $T_1$ , and  $\Pi_{\text{theo}}$  has to estimate  $T_1$  by setting  $T_0$  in the predicate  $\phi_2$ . This estimation should use a lenient bound (that will likely give  $\mathcal{A}$  a larger head start), as otherwise the variance of the block generation process can cause  $\phi_2$  to be triggered and thus abort the settlement.

Notice that  $\mathcal{A}$  cannot mount an eclipse attack before Step 3 of  $\Pi_{\text{prac}}$  is reached. Only the Tesseract enclave can produce the data  $\mathsf{TX}_1$ , and it will do so only after receiving all the acknowledgements from  $S_1, S_2, \ldots, S_N$  in Step 2. Therefore, an eclipse attack will be thwarted if at least one non-adversarial server  $S_i \in \{S_1, S_2, \ldots, S_N\}$  is present, because  $S_i$  will broadcast the invalidation transactions  $\mathsf{TX}_1^c, \mathsf{TX}_2^c$  to ensure the all-ornothing guarantee of Definition 1.

In practice, it is preferable that the Tesseract enclave will wait for acknowledgements from only a constant fraction of the servers  $S_i \in \{S_1, S_2, ..., S_N\}$ , so that  $\mathcal{A}$  will not be able to deny service by preventing a single acknowledgement from reaching Tesseract in Step 2 of the settlement procedure. Our practical approach can in fact make Tesseract resistant to DoS attacks in a broader sense, see Section 6.

Another advantage of  $\Pi_{\text{prac}}$  is that it can support other cryptocurrency systems besides a PoW blockchain. This is because the servers  $S_1, S_2, \ldots, S_N$  can run a full node inside their enclave, whereas the predicates  $\phi_1', \phi_2'$  lack the power to express the irreversibility condition of a more complex cryptocurrency system.

Irrespective of the settlement procedure, the Tesseract exchange server can fetch from  $S_1, S_2, \ldots, S_N$  the heights of their longest chains (e.g., once every 30 minutes), and refuse to confirm users' deposits if less than N/2 of the servers respond. This would avert fake deposits from being confirmed due to an eclipse attack, without relying on the prudence of the users.

# 5.4 Settlement with One Secure Processor

Is it possible to devise a workable protocol for all-ornothing settlement that utilizes servers  $S_1, S_2, \ldots, S_N$  that do not have SGX processors, such that the protocol is secure if at least one of the servers is isolated from the adversary? If the round complexity can depend on a security parameter, then protocols that accomplish this task are indeed possible.

The basic idea is to rely on the gradual release technique [9, 31] to reveal  $TX_1, TX_2$  simultaneously. The Tesseract enclave can generate a fresh symmetric key  $K \in \{0,1\}^{\lambda}$ , send the ciphertext  $ct = \text{encrypt}_K(TX_1, TX_2)$  to  $S_1, S_2, \ldots, S_N$ , and wait for acknowledgements from  $S_1, S_2, \ldots, S_N$  that they received ct. Then, Tesseract can send each of the  $\lambda$  bits of K, and wait for acknowledgements from  $S_1, S_2, \ldots, S_N$  after each bit is received.

We can improve upon the basic idea by letting the SGX enclave assume the role of a trusted dealer, and combine a *fair* secret sharing protocol with the gradual release technique. To this end, we employ the fair secret reconstruction protocol of Lin and Harn [43]. The

# Protocol $\Pi_{grad}$

- 1. The enclave picks a random symmetric key  $K \in \{0, 1\}^{\lambda}$  such that  $K = (w_1, w_2, \dots, w_q), w_i \in \{0, 1\}^m, \lambda = q \cdot m$ .
- 2. The enclave creates  $ct = \mathsf{encrypt}_K(\mathsf{TX}_1, \mathsf{TX}_2)$ .
- For w<sub>i</sub> ∈ {w<sub>1</sub>, w<sub>2</sub>,...,w<sub>q</sub>}, the enclave creates N-outof-N additive secret shares of w<sub>i</sub>, including dummy secrets:
  - The enclave picks a random  $\alpha \in [s-1]$ .
  - For  $\ell \in [d] \setminus \{\alpha, \alpha+1\}$ 
    - For each  $j \in [N]$ , the enclave picks random shares  $x_i^{\ell,j} \in \mathbb{GF}(2^{\lambda})$  conditioned upon  $\sum_{j=1}^N x_i^{\ell,j} \neq 0$ .
  - For  $\ell = \alpha + 1$ 
    - For each  $j \in [N]$ , the enclave picks random shares  $x_i^{\ell,j} \in \mathbb{GF}(2^{\lambda})$  conditioned upon  $\sum_{j=1}^N x_i^{\ell,j} = 0$ .
  - For  $\ell = \alpha$ 
    - For each  $j \in [N]$ , the enclave picks random shares  $x_i^{\ell,j} \in \mathbb{GF}(2^{\lambda})$  conditioned upon the m least significant bits of  $\sum_{j=1}^{N} x_i^{\ell,j}$  being equal to  $w_i$  and  $\sum_{j=1}^{N} x_i^{\ell,j} \neq 0$ .
- 4. For every  $i \in [q], \ell \in [d], j \in [N]$ 
  - The enclave creates a signature  $\sigma_i^{\ell,j}$  for the share  $x_i^{\ell,j}$ .
- 5. For each  $j \in [N]$ 
  - The enclave sends  $(ct, \{x_i^{\ell,j}, \sigma_i^{\ell,j}\}_{i \in [q], \ell \in [d]})$  to  $S_j$  via a secure channel.
- 6. For each  $i \in [q]$ 
  - The servers  $S_1, S_2, \ldots, S_N$  reconstruct  $w_i$  by sending their signed shares in the sequential order  $(x_i^{1,1}, \sigma_i^{1,1}), \ldots, (x_i^{1,N}, \sigma_i^{1,N}), (x_i^{2,1}, \sigma_i^{2,1}), \ldots$ , until the indicator  $\sum_{j=1}^N x_i^{\alpha+1,j} = 0$  is found.
  - If some corrupt server  $S_j$  does not send  $(x_i^{\ell,j}, \sigma_i^{\ell,j})$  within  $\tau$  time, then the honest servers begin an exhaustive search for the key K.
- 7. Any server that decrypts ct will broadcast  $TX_1$  to  $C_1$  and  $TX_2$  to  $C_2$ .

Figure 15: Gradual protocol for fair settlement.

combined protocol  $\Pi_{\rm grad}$  is parameterized according to a decoys amount  $d \geq 2$ , batching value  $m \geq 1$ , and a timeout  $\tau$  (for example  $\tau = 10$  minutes). See Figure 15 for the description of  $\Pi_{\rm grad}$ .

It is inherently the case that the adversary A can

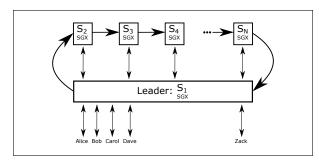


Figure 16: Fairness with anti-DoS protection.

recognize whether a potential secret key K' is equal to K, by attempting to decrypt the structured ciphertext ct. Thus, if  $\mathcal{A}$  can brute-force the unrevealed bits of K, she does not need to let Step 6 of  $\Pi_{\mathrm{grad}}$  progress until an indicator value  $\sum_{j=1}^N x_i^{\alpha+1,j} = 0$  becomes known. The adversary  $\mathcal{A}$  may try to guess  $\alpha$  and learn an m-bit value of  $\sum_{j=1}^N x_i^{\alpha,j} \neq 0$  that the honest servers do not know, but the success probability of guessing  $\alpha$  correctly is  $\frac{1}{d-1}$ . Furthermore, under the assumption that  $\mathcal{A}$  cannot breaks K within  $\tau$  time in order to verify whether her guess was correct, she must execute Step 6 honestly if she wishes that the other servers will help to reveal the next bits of K. Note that this is the case even if  $\mathcal{A}$  corrupts N-1 of the servers.

Therefore,  $\Pi_{\text{grad}}$  is more secure when the timeout parameter  $\tau$  is smaller, when the amount of dummy secrets d is larger, and when the batching size m is smaller. In particular, if  $m=\lambda$  then  $\Pi_{\text{grad}}$  is completely insecure: A will be able to corrupt the last server  $S_N$  and verify for each  $\ell$  whether  $\sum_{j=1}^N x_1^{\ell,j} = K$ , without revealing  $x_1^{\ell,N}$  to the other servers.

If m=1 and d>2 then  $\Pi_{\rm grad}$  is strictly more secure than the basic gradual release protocol. Another advantage over the basic protocol is that  $\Pi_{\rm grad}$  requires only one round of communication between Tesseract and the servers  $S_1, S_2, \ldots, S_N$ . However, the number of rounds of communication among  $S_1, S_2, \ldots, S_N$  themselves is  $\Omega(\frac{\lambda}{m} \cdot d)$ , hence larger d or smaller m make  $\Pi_{\rm grad}$  less efficient.

The major disadvantage of  $\Pi_{grad}$  is that the computational power of  $\mathcal A$  must not be significantly greater than that of the honest servers. By contrast,  $\Pi_{prac}$  does not require such an assumption.

# 6 Full Protocol

The Tesseract exchange can be initialized with the SGX server  $S_1$  as its current leader, and execute the Paxos [40] consensus protocol together with the other SGX servers  $S_2, S_3, \ldots, S_N$ . See Figure 16 for an illustration.

The requirements of the Paxos Synod protocol are satisfied in our setting, due to the following reasons:

- 1. Authenticated channels exist, as the messages that each SGX server sends are signed using keys bound to its Tesseract instance via remote attestation.
- Byzantine faults may not occur (unless the SGX signing key is compromised), since the servers are running correct code.

The complete Tesseract protocol  $\Pi_{\text{RTExch}}$  is outlined in Figure 17. To accomplish all-or-nothing settlements,  $\Pi_{\text{RTExch}}$  uses  $\Pi_{\text{prac}}$  as a subroutine. As with  $\Pi_{\text{prac}}$ , the SGX servers  $S_1, S_2, \ldots, S_N$  need to share a symmetric secret key sk that is known only to their enclaves. The exemplary parameters  $d_0 = 5, n_0 = 288$  mean that the all-or-nothing settlements are done once every 24 hours  $(288 \cdot 5 \text{ minutes})$ . In the case of a DoS attack on  $\Pi_{\text{RTExch}}$ ,  $d_0 = 5$  implies that trades in the last 5 minutes (or less) will be lost when the newly elected leader resumes the trading service for the users.

 $\Pi_{RTExch}$  can be regarded as a sequential composition of two components. One is Paxos, which guarantees consistency among the servers, and the other is the all-or-nothing protocol that interacts with cryptocurrencies.

All-or-nothing fairness holds if more than  $\frac{1}{H}$  of the servers are not under adversarial control, even if the network is asynchronous. For example, parameterizing  $\Pi_{\text{RTExch}}$  according to H = 4 would imply that the adversary A must corrupt more than 75% of the servers to violate all-or-nothing fairness (by corrupting > 25% of the servers A can mount a DoS attack). This is because  $\Pi_{RTExch}$  ensures that there will never be two servers that act as leaders of the same epoch: a majority is required to elect a new leader (via the Synod algorithm) in any non-settlement epoch, and the leader  $S_L$  needs acknowledgements from  $\lceil n(1-\frac{1}{H}) \rceil > \frac{n}{2}$  servers before proceeding to Step 3 of  $\Pi_{prac}$ . However, if  ${\cal A}$  controls the communication traffic of  $\left\lceil n(1-\frac{1}{H})\right\rceil$  servers, then  $\mathcal{A}$  can let  $S_L$  receive  $\left\lceil n(1-\frac{1}{H})\right\rceil$  acknowledgments and release  $\mathsf{TX}_1$  to  $\mathcal{C}_1$ , without ever releasing  $\mathsf{TX}_2$  to  $\mathcal{C}_2$  $(\Pi_{\text{prac}}$  can be attacked in this way only if all servers are under adversarial control).

In non-settlement epochs, the first component of  $\Pi_{RTExch}$  ensures liveness if the network is synchronous and there is a majority of non-faulty servers — just because Paxos guarantees liveness when synchrony holds

It is also critical to protect against DoS during the all-or-nothing settlement procedure, since the "nothing" outcome implies that Tesseract has to shut down and start afresh. To minimize the shutdown probability,  $\Pi_{RTExch}$  attempts to restart an all-or-nothing epoch with a new leader, immediately after the last all-or-nothing

#### Protocol $\Pi_{RTExch}$

Let  $S_1, S_2, ..., S_N$  be SGX-enabled servers, and let H > 2. Exemplary parameters:  $d_0 = 5, n_0 = 288$ .

- For every  $i \in [N]$ :
  - The server S<sub>i</sub> initializes L<sub>i</sub> := 1 as the leader index and J<sub>i</sub> := 0 as the first epoch.
- Let L denote the index of the server with  $L_i = i$ .
- Communication with traders.
  - The server S<sub>L</sub> accepts trade requests from new and existing users, and updates their account balances in the data structures that are inside its SGX enclave.
- Synchronization with the other servers:  $J_L \mod n_0 \neq 0$ .
  - After each epoch of  $d_0$  minutes:
    - \*  $S_L$  sets  $J_L := J_L + 1$ .
    - \*  $S_L$  creates  $m = (J_L, \mathsf{encrypt}_{sk}(dat))$ , where dat is its entire enclave data.
    - \*  $S_L$  sends m to the servers  $\{S_i\}_{i\neq L}$ .
    - \* Any server  $S_i$  that received m will set  $J_i := J_L$ .
    - \* Servers that did not receive *m* will invoke the Synod algorithm to update *L* to a new leader.
      - If a new leader was elected, aware servers will inform the users by publishing the index of the new leader (with remote attestation).
- All-or-nothing settlement:  $J_L \mod n_0 = 0$ .
  - $S_L$  invokes  $\Pi_{\text{prac}}$  with the following modifications:
    - \* In Step 1 of  $\Pi_{\text{prac}}$ ,  $S_L$  sends  $m = (J_L, ct)$ , where  $ct = \text{encrypt}_{sk}(dat, \mathsf{TX}_1, \mathsf{TX}_2)$ .
    - \* In Step 2 of  $\Pi_{\text{prac}}$ ,  $S_L$  waits for acknowledgements from  $\left\lceil n(1-\frac{1}{H}) \right\rceil$  or more servers.
  - **-** For every i ∈ [N]:
    - \*  $S_i$  starts the timer  $T_1$  at the beginning of the epoch  $J_L$ , and constructs  $\mathsf{TX}_1^\mathsf{c}, \mathsf{TX}_2^\mathsf{c}$  on its own.
    - \* If  $S_i$  sees that  $\mathsf{TX}_1$  was confirmed on  $\mathcal{C}_1$  and  $\mathsf{TX}_2$  was confirmed on  $\mathcal{C}_2$ , it updates  $J_i := J_L + 1$  and proceeds to the next epoch.
    - \* If  $S_i$  sees that  $\mathsf{TX}_1^c$  was confirmed on  $\mathcal{C}_1$  and  $\mathsf{TX}_2^c$  was confirmed on  $\mathcal{C}_2$ , it invokes Synod to elect a new leader, and then updates  $J_i := J_L + n_0$  to attempt another all-or-nothing settlement.

Figure 17: Outline of the Tesseract protocol.

settlement epoch failed. The enclave of each  $S_i$  will use a random perturbation before proposing itself as the leader, to make it difficult for an adversary to mount DoS attacks on consecutive leaders. Each enclave should also copy dat from servers that already received the latest

trade data that the last leader sent. Thus,  $\Pi_{RTExch}$  has to ensure that a failed all-or-nothing epoch terminates as quickly as possible, so that the following epoch will have enough time to succeed before the expiration of the timeouts that allow users to claim their refunds. This is done by letting each  $S_i$  construct and broadcast the cancellation transactions  $TX_1^c, TX_2^c$  on its own — for example by spending the reserve output (cf. Section 7) into a new output with the same amount (cancellation of an Ethereum settlement transaction is accomplished even more easily by using the current nonce with a noop transaction). This way, each  $S_i$  can start its  $T_1$ timer at the beginning of the epoch, and therefore the adversary cannot target the first server that receives ct by intercepting TX<sub>1</sub><sup>c</sup> and releasing it after an arbitrarily long delay.

Let us note that if the leader or any other server  $S_i$  crashes and does not recover quickly enough, another server  $S_j$  will be the leader in the case that  $S_i$  comes back online later (without any saved data except for the hardware keys that the other servers expect, cf. Section 6.1). Then,  $S_i$  will synchronize with the enclave data m and the clock of the current leader  $S_j$ , and will be able to continue its participation in the execution of  $\Pi_{\text{RTExch}}$ .

The setup procedure for  $S_1, S_2, ..., S_N$  also involves certain security concerns, see Section 6.1.

## **6.1** Setup of the Servers

To achieve maximum security, we design the initialization procedure for  $\Pi_{RTExch}$  as follows. Our enclave program code  $\mathcal{P}_{RTExch}$  contains a hardcoded list of N endorsement public keys RPK<sub>1</sub>, RPK<sub>2</sub>,..., RPK<sub>N</sub>, corresponding to the reputable owners of the N servers (e.g.,  $S_1$  is located at Bank A,  $S_2$  is located at University B, and so on). When the enclave of  $S_i$  is loaded with  $\mathcal{P}_{\text{RTExch}}$ , the code first acquires entropy (cf. Section 4) and generates a fresh keypair  $(tpk_i, tsk_i)$ , and then output  $tpk_i$  together with an encryption encrypt $(tsk_i)$ that is created using the symmetric hardware key of the SGX CPU of  $S_i$ . The owner keeps a backup of encrypt $(tsk_i)$ , sends  $tpk_i$  to  $\{S_j\}_{j\neq i}$ , waits to receive  $\{tpk_j\}_{j\neq i}$ , signs  $m=(tpk_1,tpk_2,\ldots,tpk_N)$  with RSK<sub>i</sub>, and sends the signature  $es_i$  to  $\{S_i\}_{i\neq i}$ . The enclave of  $S_i$  waits to receive the endorsed list of fresh keys  $(m, es_1, es_2, \dots, es_N)$ , and stores this list as immutable data. Following that, the enclave of  $S_i$  establishes secure channels (TLS) with each other server  $S_i$  via the identities  $tpk_i$  and  $tpk_i$ .

If the enclave of  $S_i$  is re-initialized to create a different identity  $tpk_i'$ , it will not be able to communicate with the enclaves of  $\{S_j\}_{j\neq i}$  that are still running. However,  $S_i$  can recover from a crash failure by restarting the enclave

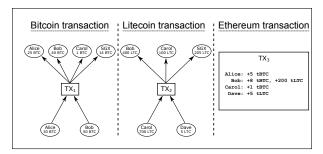


Figure 18: Atomic issuance of tokenized coins.

program  $\mathcal{P}_{\text{RTExch}}$  with m, encrypt $(tsk_i)$  and otherwise a blank slate, then re-establish the TLS channels  $\{S_j\}_{j\neq i}$  and wait to receive the latest data (including the trusted clock offset since the start of the round) from the current leader.

This way, when the Tesseract platform is launched, the sensitive reputation key  $RSK_i$  is used only once to endorse the physical machine that hosts the *i*th enclave in order to avoid man-in-the-middle attacks, and  $S_i$  can continue to be part of the platform as long as its SGX CPU is undamaged.

# 7 Fungible Tokenized Coins

The Tesseract platform also allows its users to withdraw and circulate tokenized coins that are pegged to some specific cryptocurrency, with no need to trust a human element and no exposure to markets fluctuations. Essentially, this is done by maintaining a reserve of the pegged cryptocurrency within the SGX enclave, and employing the all-or-nothing fairness protocol (cf. Sections 5 and 6) to ensure that the enclave remains solvent.

Thus, for example, Carol can deposit 600 LTC to the Tesseract exchange, trade the 600 LTC for 2 BTC, and withdraw 2 tokenized BTC (tBTC) into the Ethereum blockchain. Then, Carol could deposit her 2 tBTC to any smart contract that recognizes the assets that Tesseract issues. For instance, Carol may wish to play a trust-free poker game in which the pot is denominated in tBTC instead of ETH (it is impractical to play poker directly on the Bitcoin blockchain and instead Ethereum's stateful contracts need to utilized, see [14]). Another example is a crowdfunding contract that raises money denominated in both tBTC and ETH, but returns all the funds to the investors if the target amount was not reached before a deadline.

The issuance of tokenized coins is illustrated in Figure 18. When a user requests to withdraw tokenized coins, the enclave will move the coins to a *reserve* address, and mint the same amount of new tokens (using ERC20 contract, see next). In the illustration:

- Alice withdraws 5 tBTC out of her 30 BTC,
- Bob trades 2 BTC in exchange for Carol's 600 LTC,
- Bob withdraws 8 tBTC and 200 tLTC,
- Carol keeps 1 BTC and withdraws 1 tBTC,
- Dave uses all of his 5 LTC to withdraw 5 tLTC.

The enclave updates its reserve outputs (14 BTC and 205 LTC in the illustration) by adding coin amounts that match the amounts of tokenized coins that the users withdrew.

Unlike the native coin deposits, reserve outputs and the tokenized coins are not constrained by a timeout, and therefore the tokenized coins are fungible. Any holder of tokenized coins (e.g., tBTC) can later deposit her tokens into the enclave (she can create an account on the Tesseract exchange if she does not have one yet), and receive native coins (e.g., BTC) upon doing so. The enclave will simply discard the tokenized coins that were deposited. Hence, the tokenized coins can circulate freely on the blockchain in which they are issued (the Ethereum blockchain in our implementation), without the involvement of the Tesseract exchange.

For the exchange to remain solvent, we must guarantee all-or-nothing fairness with respect to Definition 1 for the transaction that moves native coins (from the users to the reserve output) and the transaction that mints tokenized coins. In Figure 18 for example, if TX<sub>1</sub> is not committed to the Bitcoin blockchain but TX<sub>3</sub> is committed to the Ethereum blockchain, then the eventual holders of the 14 tBTC will not be able to deposit their tokens in order to convert them to native BTC, because the reserve output (of 14 BTC) does not exist. Likewise, if TX<sub>3</sub> is not committed to the Ethereum blockchain but TX<sub>1</sub> is committed to the Bitcoin blockchain, then the Bitcoin holders will be damaged (e.g., Alice will lose 5 BTC).

As described in Sections 4 and 5, the all-or-nothing settlement should occur after an interval that is longer than the time that it takes for the all-or-nothing protocol execution to complete (e.g., an interval of 24 hours can be sensible). This means that when a user requests to withdraw tokenized coins, there will be a waiting period (say, somewhere between 1 hour and 25 hours) before she receives the tokens. This also implies good scalability, since all the native coins (that are kept in reserve) are accumulated into a single output that is updated on-chain only after a lengthy time interval.

In our implementation, the tokenized coins are issued on the Ethereum blockchain in the form of an ERC20 contract [74]. It is also possible to mint the tokenized coins as colored coins [67] on the Bitcoin blockchain, though that is problematic for two reasons. First, tagging-based colored coins have not been implemented

yet in cryptocurrencies such as Bitcoin and Litecoin (cf. Section 8). More importantly, the principal reason for having tokenized coins is to use them in smart contracts, and Ethereum is better suited for this purpose.

Since the tokenized coins are issued by the Tesseract exchange and are fungible, the holders of these tokens will be unable to convert them to native coins in the case that the Tesseract platform is destroyed. In Section 6 we give the full version of Tesseract, which is distributed and hence highly unlikely to fail. It is also possible to incorporate a timeout to the reserve outputs that specifies that the coins will be controlled by (say) a multisig of several reputable parties if Tesseract stops updating the reserve outputs and thus the time expiration is reached. However, this gives an incentive to these several parties to destroy the Tesseract platform and collect the reserve coins.

## 8 Fiat Currencies

For fiat currency transactions that are done via the traditional banking system, it is problematic to offer integration with a protocol that is based on cryptographic assumptions. One reason for this is that fiat transactions can be reversed as a result of human intervention (e.g., in the course of investigating a complaint by a customer of a bank).

The problem can be outsourced by relying on a counterparty that provides recognizable tokens that can be transferred via the underlying cryptocurrency system, and are supposed to represent an equivalent amount of fiat currency. This approach enables fiat currency transfers that become irreversible just like the cryptocurrency payments, and depends on the reputation of the counterparty to redeem the token for the actual fiat currency. See for example [19, Section 5.2] and [67] in this regard. One instantiation of this idea that enjoys a relatively high degree of popularity is Tether [21, 18], which circulates tokens that are pegged to the U.S. dollar (involving proof-of-reserve) using the Omni layer [56] on top of Bitcoin. The Tesseract service provider may even issue its own fiat tokens (by accepting traditional wire transfers of fiat currency), and other platforms and users may assign value to the tokens if they consider this Tesseract service provider to be trustworthy.

The SGX enclave of Tesseract can thus support assets that are redeemable for fiat currencies, by recognizing certain predefined types of tokens in the deposit transactions. In the case that the cryptocurrency (in which such an asset circulates) supports tagging-based colored coins [67, 41], the validation predicate for the deposit is easy to implement. This is because the predicate would inspect only current deposit transaction,

rather than also inspecting prior transfers of ownership that ended up as this deposit. For non-tagging-based colored coin, Tesseract would need to run a full node inside an SGX enclave, which is far more demanding than running an SPV [54] client (the Tesseract enclave operation that we specify in Section 4 is essentially an SPV client). Tagging-based colored coins require miners and full nodes to perform a moderate amount of extra work (only for colored transactions), which is not supported on cryptocurrencies such as Bitcoin and Litecoin yet (see [41] for a proposed implementation and [60] regarding future ideas). However, Ethereum already supports the equivalent of tagging-based colored coins, in the form of an ERC20 smart contract (cf. [74]). Our reference implementation of Tesseract already supports ERC20 assets as well, see Appendix B.

Hence, other than just allowing cryptocurrencies to be traded for one another, Tesseract can also let the users trade cryptocurrencies for traditional assets that have a digital representation (in particular flat currencies), though this capability involves trust in the reputation of the issuers of the assets.

## 9 Related Work

Trusted hardware has been proposed as an effective tool for different kinds of cryptocurrency use-cases, such as off-chain payment channels [45, 44], reputable data feed services [86], and a mixing service [79]. These schemes offer better efficiency and features by placing more trust in the hardware manufacturer: in particular, off-chain channels and mixers can also be accomplished without secure processors (see, e.g., [50, 14, 69, 32]). By contrast, Tesseract reduces the amount of trust that needs to be placed in the exchange service relative to all other real-time exchange schemes (to the best of our knowledge). In Section 3 we provide a comparison between Tesseract and various other cryptocurrency exchange schemes.

Trusted hardware can also be used to achieve significant efficiency gains for well-known cryptographic primitives such as functional encryption [28], secure MPC [62], and NIZK in the presence of side-channels [78]. Pass, Shi, and Tramèr give a formal model of trusted hardware and remote attestation [59].

Several works achieve fair exchange and secure cash distribution via interaction with a cryptocurrency system, cf. [4, 3, 13, 14, 37, 39]. However, these works enable fair exchange (with penalties) by using a single cryptocurrency system, while Tesseract has to provide all-or-nothing fairness among multiple cryptocurrency systems.

Outside of academic work, a wide range of industry

and community efforts have attempted to realize various aspects of cross-chain distributed exchange. One strategy is the use of payment channels to achieve a hub-and-spoke exchange, work which we examine in Section 3. These efforts are primarily based on the work in [61], which introduces HTLCs (hashed timelocked contracts) for payment channels in cryptocurrencies. A practical example of such applications is available at [65].

Several exchanges aim at using raw atomic swaps, as in Figures 1 and 2. Unfortunately as discussed in Section 2, due to their need for on-chain mediation, there is no known solution for extending these efforts to real-time trades. Also, requiring the user to sign HTLC transactions presents a DoS vector whereby a user can choose to abort post-matching. A further comparison of such swap-based exchanges to the channel-based Lightning network approach is available in [73].

Further alternatives to an atomic swap model for decentralized, non-custodial exchanges are also explored in Section 3, with notable early efforts making use of both reputation [27] and IOUs [70] to achieve economic security for their trades. Notably, these exchanges still require on-chain per-trade mediation, and cannot be real-time. Nonetheless, a wide range of efforts have focused on developing these swaps, the building blocks of a potential exchange. Such primitive swaps from, for example, Bitcoin to ZCash have been demonstrated in [66].

Lastly, a wide range of decentralized exchanges run inside a single blockchain and allow users to exchange assets on that chain, as mentioned in Section 1. Some of these exchanges use custody in smart contracts to trustlessly hold user assets (including EtherDelta [64]), whereas others rely on atomic swaps to exchange assets that never leave user custody (including AirSwap [57]). Because of their on-chain settlement, these exchanges are not real time, and suffer from a number of manipulation vectors across a design space explored in [12]. Also, they do not support cross chain exchange.

#### **Acknowledgements**

We thank Fred Schneider, Elaine Shi, Ittay Eyal, and Aniket Kate for useful discussion.

This work is funded in part by NSF grants CNS-1330599, CNS-1514163, CNS-1564102, and CNS-1704615, as well as ARO grant W911NF-16-1-0145.

#### References

 ANATI, I., GUERON, S., JOHNSON, S., AND SCARLATA, V. Innovative Technology for CPU Based Attestation and Sealing. In *HASP'13* (2013), pp. 1–7.

- [2] ANDRESEN, G. P2SH. https://github.com/bitcoin/ bips/blob/master/bip-0016.mediawiki.
- [3] ANDRYCHOWICZ, M., DZIEMBOWSKI, S., MALINOWSKI, D., AND MAZUREK, L. Fair two-party computations via Bitcoin deposits. In First Bitcoin Workshop, FC (2014).
- [4] ANDRYCHOWICZ, M., DZIEMBOWSKI, S., MALINOWSKI, D., AND MAZUREK, L. Secure multiparty computations on Bitcoin. In *IEEE S&P* (2014).
- [5] BACK, A. O(2<sup>80</sup>) theoretical attack on p2sh. https://bitcointalk.org/index.php?topic=323443.0, 2013.
- [6] BALDWIN, C. http://www.reuters.com/article/usbitfinex-hacked-hongkong-idUSKCN10EOKP.
- [7] BARISSER, A. High frequency trading on the Coinbase exchange, 2015. https://medium.com/on-banking/ high-frequency-trading-on-the-coinbase-exchangef804c80f507b.
- [8] BARTOLETTI, M., AND POMPIANU, L. An analysis of Bitcoin OP\_RETURN metadata. In Financial Cryptography 4th Bitcoin Workshop (2017). https://arxiv.org/abs/1702.01024.
- [9] BEAVER, D., AND GOLDWASSER, S. Multiparty computation with faulty majority. In 30th Annual Symposium on Foundations of Computer Science (FOCS) (1989).
- [10] BEEKMAN, J. A Denial of Service Attack against Fair Computations using Bitcoin Deposits, 2014. https:// eprint.iacr.org/2014/911.
- [11] BENET, J. https://ipfs.io/.
- [12] BENTOV, I., BREIDENBACH, L., DAIAN, P., JUELS, A., LI, Y., AND ZHAO, X. The cost of decentralization in 0x and EtherDelta. http://hackingdistributed.com/2017/08/ 13/cost-of-decent/, 2017.
- [13] BENTOV, I., AND KUMARESAN, R. How to use Bitcoin to design fair protocols. In *Crypto* (2) (2014), pp. 421–439.
- [14] BENTOV, I., KUMARESAN, R., AND MILLER, A. Instantaneous decentralized poker. In *Asiacrypt* (2017).
- [15] BENTOV, I., MIZRAHI, A., AND ROSENFELD, M. Decentralized prediction market without arbiters. In *Financial Cryptography 4th Bitcoin Workshop* (2017).
- [16] BENTOV, I., TIERNOLAN, ET AL. Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic= 193281.msg2224949#msg2224949, 2013.
- [17] BROWN, D. G. How i wasted too long finding a concentration inequality for sums of geometric variables. <a href="https://cs.uwaterloo.ca/~browndg/negbin.pdf">https://cs.uwaterloo.ca/~browndg/negbin.pdf</a>.
- [18] CAPITALIZATIONS, C. M. https://coinmarketcap.com/
- [19] CLARK, J., BONNEAU, J., FELTEN, E. W., KROLL, J. A., MILLER, A., AND NARAYANAN, A. On decentralizing prediction markets and order books. In WEIS (2014).
- [20] CROMAN, K., DECKER, C., EYAL, I., GENCER, A. E., JUELS, A., KOSBA, A., MILLER, A., SAXENA, P., SHI, E., SIRER, E. G., SONG, D., AND WATTENHOFER, R. On scaling decentralized blockchains. In *Financial Cryptography* 3rd Bitcoin Workshop (2016).
- [21] CURRENCIES ON THE BITCOIN BLOCKCHAIN, T. F. https://tether.to/wp-content/uploads/2016/06/ TetherWhitePaper.pdf, 2016.
- [22] DECKER, C., AND WATTENHOFER, R. A fast and scalable payment network with Bitcoin duplex micropayment channels. In 17th Stabilization, Safety, and Security of Distributed Systems (SSS) (2015).

- [23] DESMEDT, AND FRANKEL. Threshold cryptosystems. In *CRYPTO: Proceedings of Crypto* (1989).
- [24] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. F. Tor: The second-generation onion router. In 13th USENIX Security (2004).
- [25] DREE12 (PSEUDONYM). List of major Bitcoin heists, thefts, hacks, scams, and losses. https://bitcointalk.org/ index.php?topic=576337.
- [26] DUBHASHI, D. P., AND PANCONESI, A. Concentration of Measure for the Analysis of Randomized Algorithms. Cambridge University Press, 2009.
- [27] ENGLISH, S. M., ORLANDI, F., AND AUER, S. Disintermediation of inter-blockchain transactions. *CoRR* abs/1609.02598 (2016).
- [28] FISCH, B. A., VINAYAGAMURTHY, D., BONEH, D., AND GORBUNOV, S. Iron: Functional encryption using Intel SGX, 2017.
- [29] GARAY, J., KIAYIAS, A., AND LEONARDOS, N. The Bitcoin backbone protocol: Analysis and applications. In *Eurocrypt* (2015).
- [30] GENNARO, R., GOLDFEDER, S., AND NARAYANAN, A. Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security. In 14th ACNS (2016).
- [31] GOLDWASSER, S., AND LEVIN, L. Fair computation of general functions in presence of immoral majority. In *Proceedings of Advances in Cryptologie (CRYPTO '90)* (Berlin, Germany, Aug. 1991), A. J. Menezes and S. A. Vanstone, Eds., vol. 537 of *LNCS*, Springer, pp. 77–93.
- [32] HEILMAN, E., ALSHENIBR, L., BALDIMTSI, F., SCAFURO, A., AND GOLDBERG, S. TumbleBit: An untrusted Bitcoin-compatible anonymous payment hub. https:// eprint.iacr.org/2016/575, 2016.
- [33] HEILMAN, E., KENDLER, A., ZOHAR, A., AND GOLDBERG, S. Eclipse attacks on Bitcoin's peer-to-peer network. In 24th USENIX Security Symposium (USENIX Security 15) (Washington, D.C., 2015), USENIX Association, pp. 129–144.
- [34] HOEKSTRA, M., LAL, R., PAPPACHAN, P., PHEGADE, V., AND DEL CUVILLO, J. Using innovative instructions to create trustworthy software solutions. In *HASP* (2013).
- [35] JOHNSON, S., SCARLATA, V., ROZAS, C., BRICKELL, E., AND MCKEEN, F. Intel Software Guard Extensions: EPID provisioning and attestation services, 2016. https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services.
- [36] KIAYIAS, A., MILLER, A., AND ZINDROS, D. Non-interactive proofs of proof-of-work. https://eprint.iacr.org/2017/ 963, 2017.
- [37] KIAYIAS, A., ZHOU, H.-S., AND ZIKAS, V. Fair and robust multi-party computation using a global transaction ledger. In *Eurocrypt* (2015).
- [38] KNIGHT, S. http://www.reuters.com/article/us-bitcoin-mtgox-wallet-idUSBREA2K05N20140321.
- [39] KOSBA, A. E., MILLER, A., SHI, E., WEN, Z., AND PAPAMANTHOU, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE* S&P (2016).
- [40] LAMPORT, L. The part-time parliament. ACM Trans. Comput. Syst 16, 2 (1998), 133–169.
- [41] LAU, J. bip-color. https://github.com/j12012/bips/blob/color/bip-color.mediawiki, 2017.

- [42] LAU, J. P2wpk. https://github.com/j12012/bips/blob/ vault/bip-0VVV.mediawiki, 2017.
- [43] LIN, H.-Y., AND HARN, L. Fair reconstruction of a secret. *Information Processing Letters* 55, 1 (7 July 1995), 45–47.
- [44] LIND, J., EYAL, I., KELBERT, F., NAOR, O., PIETZUCH, P. R., AND SIRER, E. G. Teechain: Scalable blockchain payments using trusted execution environments. *CoRR abs/1707.05454* (2017).
- [45] LIND, J., EYAL, I., PIETZUCH, P. R., AND SIRER, E. G. Teechan: Payment channels using trusted execution environments. In Financial Cryptography 4th Bitcoin Workshop (2017).
- [46] LUU, L., AND VELNER, Y. KyberNetwork white paper, 2017. https://kyber.network/assets/ KyberNetworkWhitepaper.pdf.
- [47] MAPPUM (PSEUDONYM). Mercury fully trustless cryptocurrency exchange, 2015. https://bitcointalk.org/ index.php?topic=946174.0.
- [48] MATETIC, S., AHMED, M., KOSTIAINEN, K., DHAR, A., SOMMER, D., GERVAIS, A., JUELS, A., AND CAPKUN, S. ROTE: Rollback protection for trusted execution, 2017. http://eprint.iacr.org/2017/048.
- [49] MCCORRY, P., HEILMAN, E., AND MILLER, A. Atomically trading with Roger: Gambling on the success of a hardfork. https://eprint.iacr.org/2017/694, 2017.
- [50] MCCORRY, P., MÖSER, M., SHAHANDASHTI, S. F., AND HAO, F. Towards Bitcoin payment networks. In *Information Security and Privacy - 21st Australasian Conference, ACISP* (2016).
- [51] MCKEEN, F., ALEXANDROVICH, I., BERENZON, A., ROZAS, C., SHAFI, H., SHANBHOGUE, V., AND SAVAGAONKAR, U. Innovative instructions and software model for isolated execution. In *HASP* (2013).
- [52] MCMILLAN, R. \$1.2m hack shows why you should never store Bitcoins on the internet. https://www.wired.com/2013/11/ inputs/, 2013.
- [53] MILLER, A. Provable Security for Cryptocurrencies. PhD thesis, University of Maryland, College Park, 2016.
- [54] NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. Bitcoin.org (2008).
- [55] NAKAMOTO, S. https://bitcointalk.org/ index.php?topic=1786.msg22119#msg22119, 2010.
- [56] OMNI. http://www.omnilayer.org/.
- [57] OVED, M., AND MOSITES, D. Swap: A peer-to-peer protocol for trading Ethereum tokens, 2017. https://swap.tech/ whitepaper/.
- [58] PASS, R., SEEMAN, L., AND ABHI SHELAT. Analysis of the Blockchain protocol in asynchronous networks. In *Eurocrypt* (2017).
- [59] PASS, R., SHI, E., AND TRAMER, F. Formal abstractions for attested execution secure processors. In *Eurocrypt* (2017).
- [60] POELSTRA, A., BACK, A., FRIEDENBACH, M., MAXWELL, G., AND WUILLE, P. Confidential assets. In Financial Cryptography Bitcoin Workshop. https://blockstream.com/ bitcoin17-final41.pdf.
- [61] POON, J., AND DRYJA, T., 2016. https://lightning.network/lightning-network-paper.pdf.
- [62] PORTELA, B., BARBOSA, M., SCERRI, G., WARINSCHI, B., BAHMANI, R., BRASSER, F., AND SADEGHI, A.-R. Secure multiparty computation from SGX. In *Financial Cryptography* (2017).

- [63] PORTNOY, E., AND ECKERSLEY, P. Intel's management engine is a security hazard, and users need a way to disable it. https://www.eff.org/deeplinks/2017/05/intelsmanagement-engine-security-hazard-and-usersneed-way-disable-it.
- [64] PROFITGENERATOR. EtherDelta decentralized token exchange, 2017. https://steemit.com/ethereum/ Oprofitgenerator/etherdelta-decentralized-tokenexchange.
- [65] REDMAN, J. Altcoin exchange performs first atomic swap between bitcoin and ethereum, 2017. https: //news.bitcoin.com/altcoin-exchange-performsfirst-atomic-swap-between-bitcoin-and-ethereum/.
- [66] REDMAN, J. Engineers demonstrate zcash/bitcoin atomic swaps, 2017. https://news.bitcoin.com/engineersdemonstrate-zcashbitcoin-atomic-swaps/.
- [67] ROSENFELD, M. Colored coins. https://
  bitcoil.co.il/files/Colored%20Coins.pdf
  https://bitcoil.co.il/BitcoinX.pdf, 2012.
- [68] ROSENFELD, M. Analysis of hashrate-based double spending. http://arxiv.org/abs/1402.2009, 2014.
- [69] RUFFING, T., MORENO-SANCHEZ, P., AND KATE, A. P2P Mixing and Unlinkable Bitcoin Transactions. In NDSS 2017 (2017).
- [70] SCHUH, F., AND LARIMER, D. BitShares. https: //bravenewcoin.com/assets/Whitepapers/bitsharesfinancial-platform.pdf.
- [71] SCHUSTER, F., COSTA, M., FOURNET, C., GKANTSIDIS, C., PEINADO, M., MAINAR-RUIZ, G., AND RUSSINOVICH, M. VC3: Trustworthy data analytics in the cloud using SGX. In IEEE S&P (2015).
- [72] SHAPESHIFT. https://shapeshift.io/.
- [73] SONG, J. Atomic swaps, 2017. https:// bitcointechtalk.com/atomic-swaps-d6ca26b680fe.
- [74] STANDARD #20, E. T. https://github.com/ethereum/ EIPs/issues/20.
- [75] STRACKX, R., AND PIESSENS, F. Ariadne: A minimal approach to state continuity. In 25th USENIX Security (2016).
- [76] SZTORC, P. BitUSD isn't worth the trouble. http://www.truthcoin.info/blog/bitusd/, 2015.
- [77] TODD, P. OP.CHECKLOCKTIMEVERIFY. BIP 65, https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, 2014.
- [78] TRAMER, F., ZHANG, F., LIN, H., HUBAUX, J.-P., JUELS, A., AND SHI, E. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In *Euro S&P* (2017).
- [79] TRAN, M., LUU, L., KANG, M. S., BENTOV, I., AND SAXENA, P. Obscuro: A secure and anonymous Bitcoin mixer using SGX, 2017.
- [80] HTTPS://BITCOIN.ORG/EN/GLOSSARY/LOCKTIME.
- [81] WARREN, W., AND BANDEALI, A. 0x: An open protocol for decentralized exchange on the Ethereum blockchain, 2017. https://oxproject.com/pdfs/0x\_white\_paper.pdf.
- [82] WUILLE, P., ET AL. https://bitcoincore.org/en/2017/ 03/23/schnorr-signature-aggregation/.
- [83] WUILLE, P., MAXWELL, G., ET AL., 2015. https://github.com/bitcoin-core/secp256k1.
- [84] XU, Y., CUI, W., AND PEINADO, M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In Security and Privacy (SP), 2015 IEEE Symposium on (2015), IEEE, pp. 640–656.

- [85] YOUNG, J. China imposes new capital controls; Bitcoin price optimistic, 2016. https://cointelegraph.com/ news/china-imposes-new-capital-controls-bitcoinprice-optimistic.
- [86] ZHANG, F., CECCHETTI, E., CROMAN, K., JUELS, A., AND SHI, E. Town Crier: An authenticated data feed for smart contracts. In CCS (2016).

# A Proof of Security for ACCS

Per Definition 1, let us prove that the all-or-nothing requirement holds for the  $\Pi_{accs}$  protocol that we described in Section 2.

We use TXOUT<sub>A</sub>,TXOUT<sub>B</sub> to denote the outputs of the transactions TX<sub>A</sub>,TX<sub>B</sub>, respectively. We denote by TX<sub>A</sub>,TX<sub>B</sub> the transactions that spend TXOUT<sub>A</sub> and TXOUT<sub>B</sub> in steps 3 and 4 of  $\Pi_{accs}$ , respectively.

**Proposition 1.** Assume that  $s_0 = \Omega(\sqrt{t_0})$ , and that any Bitcoin client that wishes to submit a valid transaction will be able to broadcast the transaction and have it included in one of the next  $s_0$  blocks. Assume that the probability of reversing  $c_0$  Bitcoin blocks or  $4c_0$  Litecoin blocks is negligible. Let  $E_0$  denote the event that the all-or-nothing property holds w.r.t. the transactions  $TX_A^S$  and  $TX_B^S$ . If  $hash(\cdot)$  is preimage-resistant and the signature scheme is existentially unforgeable, then  $\neg E_0$  occurs with negligible probability.

*Proof sketch.* We define the following events:

- $E_1 = \{ \mathsf{TX}_\mathsf{A} \text{ was reversed after Bob broadcasted } \mathsf{TX}_\mathsf{B} \}$
- $E_2 = \{ \mathsf{TX}_\mathsf{B} \text{ was reversed after Alice revealed } x \}$
- $E_3 = \{\text{Bob spent TXOUT}_A \text{ before Alice revealed } x\}$
- E<sub>4</sub> = {Alice spent both TXOUT<sub>A</sub> and TXOUT<sub>B</sub>
   without forging a signature}
- $E_F = \{ \text{The adversary forged a signature} \}$
- $E_A = \{ \mathsf{TX}_\mathsf{A}^\mathsf{S} \text{ was confirmed by the Bitcoin network} \}$
- $E_B = \{ \mathsf{TX}_\mathsf{R}^\mathsf{S} \text{ was confirmed by the Litecoin network} \}$

It is enough to prove that  $\Pr[\neg E_0 \cap \neg E_F]$  is negligible, because  $\Pr[E_F]$  is negligible by assumption and

$$\Pr[\neg E_0] = \Pr[(\neg E_0 \cap E_F) \cup (\neg E_0 \cap \neg E_F)] \\
\leq \Pr[E_F] + \Pr[\neg E_0 \cap \neg E_F].$$

Assume that  $E_F$  did not occur. If Alice redeems TXOUT<sub>B</sub> then Bob will be able to redeem TXOUT<sub>A</sub> unless either the block that contains TX<sub>A</sub> was reversed on the Bitcoin blockchain (event  $E_1$ ), or TXOUT<sub>A</sub> was spent after the  $c_0 + t_0 + s_0$  timeout expired (event  $E_4$ ). More formally, we have  $E_A \cap \neg E_B \cap \neg E_F \subseteq E_1 \cup E_4$ .

Assume again that  $E_F$  did not occur. If Bob redeems TXOUT<sub>A</sub> then Alice will be able to redeem TXOUT<sub>B</sub> unless either the block that contains TX<sub>B</sub> was reversed on the Litecoin blockchain (event  $E_2$ ), or TXOUT<sub>B</sub> never appeared on the Litecoin blockchain (event  $E_3$ ). More formally, we have  $E_B \cap \neg E_A \cap \neg E_F \subseteq E_2 \cup E_3$ .

Therefore, we obtain

$$\begin{aligned} & & \Pr[\neg E_0 \cap \neg E_F] \\ & = & \Pr[\left((E_A \cap \neg E_B) \cup (E_B \cap \neg E_A)\right) \cap \neg E_F\right] \\ & \leq & \Pr[E_A \cap \neg E_B \cap \neg E_F] + \Pr[E_B \cap \neg E_A \cap \neg E_F] \\ & \leq & \Pr[E_1 \cup E_4] + \Pr[E_2 \cup E_3] \\ & \leq & \Pr[E_1] + \Pr[E_2] + \Pr[E_3] + \Pr[E_4] \,. \end{aligned}$$

By assumption,  $\Pr[E_1]$  and  $\Pr[E_2]$  are negligible since  $c_0$  is large enough. Furthermore,  $\Pr[E_3] = \operatorname{negl}(\lambda)$  because the event  $E_3$  implies that Bob computed a preimage of hash(Y).

To bound  $Pr[E_4]$ , we need to consider the event that the Bitcoin chain grew by  $t_0 + s_0$  blocks before the Litecoin chain grew by  $4t_0$  blocks. If this event occurs, then Alice will be able to redeem TXOUT<sub>A</sub> first, and still have enough time to redeem TXOUT<sub>B</sub> too. Note that the Bitcoin network is expected to generate only  $t_0$  blocks by the time that the Litecoin network generated  $4t_0$  blocks.

Let  $Z = Z(t_0 + s_0, \frac{1}{5})$  be a random variable with negative binomial distribution that counts the total number of blocks that both the Bitcoin and Litecoin networks generated by the time that the Bitcoin network generated  $t_0 + s_0$  blocks, hence  $E[Z] = 5(t_0 + s_0)$ . By using a standard tail inequality [17, 26] for the *binomial* distribution  $B(\mu \cdot E[Z], \frac{1}{5})$  with  $\mu \triangleq \frac{t_0}{t_0 + s_0}$ , we obtain

$$\begin{aligned} \Pr[E_4] &= \Pr[Z < 5t_0] = \Pr[Z < \mu \cdot \mathrm{E}[Z]] \\ &= \Pr\left[B(\mu \cdot \mathrm{E}[Z], \frac{1}{5}) > t_0 + s_0\right] \\ &< e^{-\frac{1}{3}(\frac{1}{\mu} - 1)^2 \mu(t_0 + s_0)} \\ &= e^{-\frac{1}{3}(s_0/t_0)^2 \cdot t_0} = e^{-\frac{1}{3}s_0^2/t_0}. \end{aligned}$$

Thus, 
$$s_0 = \lambda \sqrt{t_0}$$
 implies  $\Pr[E_4] < e^{-\lambda^2/3} = \operatorname{negl}(\lambda)$ .  $\square$ 

Proposition 1 makes the assumption that clients cannot be denied from communicating with the Bitcoin network during a long enough time period. While DoS attack on clients has been suggested as a possible vulnerability of Bitcoin based protocols [10], our assumption is quite reasonable as it is far more difficult to mount a DoS attack on a client (that can connect to the internet from various endpoints) in comparison to a DoS attack on a server. However, in case the Bitcoin blocks approach their full capacity due to a high transaction volume, the client may indeed find it difficult to incorporate the desired transaction in one of the next  $s_0$  blocks (see

for example [20] regarding the scalability prospects of Bitcoin). Still, the client should be able to include her transaction by attaching a high enough fee and thus signal the Bitcoin miners to prioritize the transaction.

Notice that the chain growth ratio between Litecoin and Bitcoin (i.e., the constant 4) does not influence the proof, because the extra  $s_0$  confirmations in TXOUT<sub>A</sub> correspond to  $4s_0$  expected growth that TXOUT<sub>B</sub> precludes.

Let us also note that the above proof makes the implicit supposition that the computational power that is devoted to the Bitcoin and Litecoin networks remains constant. It is possible to generalize Proposition 1 by assuming that the computational power may not fluctuate beyond a certain bound.

# **B** Implementation

We highlight parts of our reference code for the Tesseract protocol  $\Pi_{RTExch}$  in Figures 19 and 20. The full source code of our demo will be made public at a later time.

Figure 19: Verify authentication path of a deposit.

Figure 20: Renewal order.