Tesseract: Real-Time Cryptocurrency Exchange using Trusted Hardware

Iddo Bentov
Cornell University

Yan Ji
Cornell University

Fan Zhang
Cornell University

Yunqi Li
SJTU

Xueyuan Zhao
SJTU

Lorenz Breidenbach
ETH Zürich and Cornell Tech

Philip Daian
Cornell Tech

Ari Juels
Cornell Tech

ABSTRACT

We propose Tesseract, a secure real-time cryptocurrency exchange service. Centralized exchange designs are vulnerable to theft of funds, while decentralized exchanges cannot offer real-time cross-chain trades. All the existing exchanges are also vulnerable frontrunning attacks. Tesseract overcomes these flaws by using a trusted execution environment, specifically Intel SGX.

The task of committing the recent trades data to independent cryptocurrency systems presents an all-or-nothing fairness problem, that can be solved by means of SPV proofs or multiple SGX-enabled servers. Tesseract also mitigates denial-of-service attacks by running a consensus protocol among SGX-enabled servers.

Tesseract supports not only real-time cross-chain cryptocurrency trading, but also a secure method to tokenize assets pegged to various 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

Bitcoin [63, 35, 67] and similar cryptocurrencies derive their security from two assumptions:

1. The majority of the participants follow the consensus protocol. The relative power of each participant is determined according to certain scarce resources that she possesses (cf. Section 1.1).

2. Standard cryptographic assumptions regarding the security of hash functions and digital signatures.

Since these kinds of *permissionless* consensus protocols do not rely on privileged entities and do not have any single point of failure, it is conceivable that an exchange between various cryptocurrencies can operate without the need to trust one entity (or a fixed-size set of entities). For example, one could imagine a trust-free exchange between the Bitcoin (BTC), Ethereum (ETH) [86, 21], and Litecoin (LTC) [52] cryptocurrencies, that is at least as secure as the weakest of these three systems.

Indeed, a trust-free cryptocurrency exchange can be realized in the form of atomic cross-chain swaps (ACCS) [23], though it would require the users to wait for many minutes (in fact, hours) for each trade to execute. Since it serves as a useful reference point, we elaborate on the concept and limitations of ACCS in Section 2.

A challenge that is much more far reaching is to build a trust-free cryptocurrency exchange where the participants can respond to price fluctuations in a rapid manner, i.e., by altering their trading positions within seconds. In fact, some traders may choose to utilize automated programs for high frequency trading and arbitrage (cf. [6]), and would therefore prefer to modify their trading positions within milliseconds. Such a *real-time* exchange enables *price discovery*: traders 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., [89]), then modify their trading positions, and in this process the price converges so that the gap (a.k.a. "spread") between the bids and asks is small.

As discussed in Section 3.1, centralized cryptocurrency exchanges can facilitate real-time trades and price discovery, but they also entail a systemic risk as all of the traders' funds can be stolen. Since ACCS do not provide price discovery, traders who wish to engage in ACCS would have to resort to other means in order to set their desired price for each swap. This implies that the trust-free nature of ACCS cannot in itself be a fully secure solution: many traders will continue to use centralized exchanges for real-time trades, and thus the systemic risk remains.

In this work we remedy the risks of cryptocurrency trading by presenting Tesseract, a real-time cryptocurrency exchange that utilizes Intel SGX in order to ensure that the traders' funds can never be stolen. To this end, our assumptions will be quite conservative: we allow a potential thief to gain complete physical access to the machine in which the funds are stored, and assume that the constant-time and constant-memory code that we run inside the SGX enclave is secure against side-channel attacks. Our design also provides mitigation to denial-of-service (DoS) attacks.

In a sense, the Tesseract exchange still relies on a trusted party in the form of the hardware manufacturer, because the asymmetric secret key that resides inside CPU (and generates signatures for remote attestation) must be known to 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 (see also Section 4.3). Thus, Tesseract still requires trust, but to a significantly lesser degree than centralized exchanges and other possible real-time exchange schemes (cf. Section 3).

Additionally, different types of valuable assets can circulate within a single cryptocurrency system (cf. [73, 22, 69, 24, 17]), hence a secure exchange service can be useful even if it interacts with only one cryptocurrency. Since real-time response to price fluctuations can be highly important for asset trading, a variant of Tesseract (that is in fact significantly simpler) can be deployed in this case too.

1.1 Cryptocurrency Systems

Bitcoin is the first decentralized cryptocurrency that has gained popularity, and it remains the most popular cryptocurrency in terms of value transfers per timeframe. In proof-of-work (PoW) based cryptocurrencies such as Bitcoin, each participant possesses machines (that perform the PoW computations) as the scarce resources that determine the participant's relative power in the consensus protocol.

Cryptocurrency protocols can also be based on other kinds of scarce resources. In particular, in a proof-of-stake [16, 33, 12, 45, 37, 27, 40] based cryptocurrency the scarce resources are the coins that circulate in the system, and in a proof-of-space [66] based cryptocurrency the scarce resource is storage space.

While our reference implementation of Tesseract (cf. Appendix B) currently supports only PoW based cryptocurrencies, we note that blockchain based proof-of-stake cryptocurrencies can be supported in a similar manner. Typically, the blocks of a PoW blockchain are validated by inspecting a hash digest, and blocks in a proof-of-stake blockchain are validated by inspecting the UTXO set (i.e., the current unspent outputs) and verifying digital signatures. Hence, the enclave code can maintain the UTXO set and verify the needed signatures for the new blocks. In fact, if the proof-of-stake protocol requires blocks to contain a commitment to the UTXO set, then the complexity of the enclave code will be quite minimal.

1.2 Related Works

Trusted hardware has been proposed as an effective tool for different kinds of cryptocurrency use-cases, such as off-chain payment channels [56, 55], reputable data feed service [90], and a mixing service [84]. 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., [60, 15, 75, 42]). 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 [34], secure MPC [71], and NIZK in the presence of side-channel attacks [83]. Pass, Shi, and Tramèr give a formal modeling of trusted hardware and remote attestation [68].

Several works achieve fair exchange and secure cash distribution via interaction with a cryptocurrency system, cf. [3, 2, 14, 15, 47, 49]. 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.

2. ATOMIC CROSS-CHAIN SWAPS

A secure protocol for ACCS was given in [23]. We specify an intuitive description of the protocol in Fig. 1, demonstrating a swap of bitcoins for litecoins as an example. The main thrust of the protocol Π_{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 Π_{accs} in Appendix A.

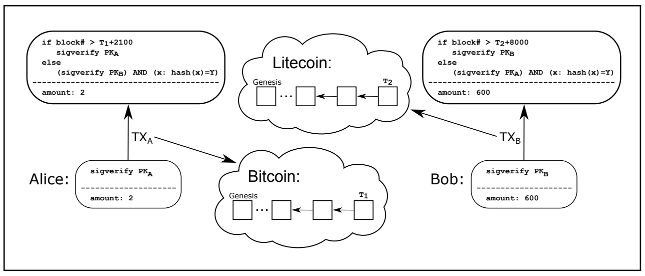
Protocol Π_{accs}

1. Alice samples a random $x \in \{0,1\}^\lambda$, computes a hash commitment $Y = \mathsf{hash}(x)$, and broadcasts a transaction $\mathsf{TX_A}$ 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_1 BTC by supplying a preimage of Y and signing with his secret key.
2. After $\mathsf{TX_A}$ is buried under c_0 extra blocks and therefore becomes irreversible w.h.p., Bob broadcasts a transaction $\mathsf{TX_B}$ that spends his n_2 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_2 LTC by supplying a preimage of Y and signing with her secret key.
3. After $\mathsf{TX_B}$ 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.

The first two steps of Π_{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. Per the related details that we discuss in Section 3.1, a reasonable choice for the function $f(\cdot)$ 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 Π_{accs} protocol completes.

In the accompanying illustration (Fig. 2), Alice trades $n_1 = 2$ BTC for Bob's $n_2 = 600$ LTC. The last block of the Bitcoin blockschain 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 ≈ 16 hours on average. Note that both Bitcoin and Litecoin allow to specify 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.



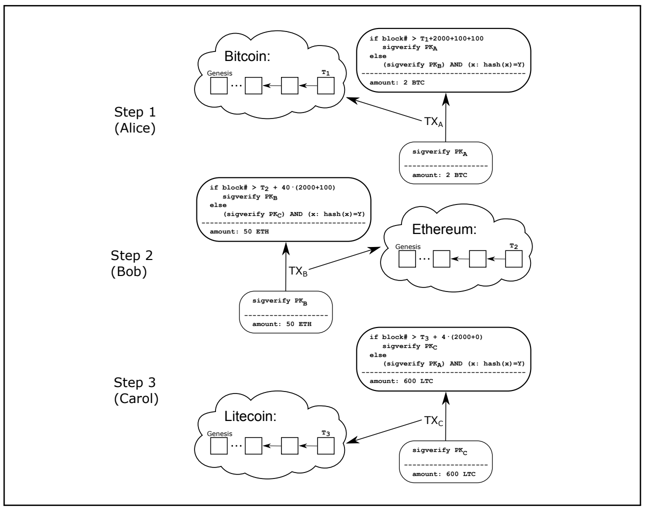
```
if block# > T1+2100
   sigverify PKA
else
   (sigverify PKB) AND (x: hash(x)=Y)
----------------------------------------
amount: 2
```

```
if block# > T2+8000
   sigverify PKB
else
   (sigverify PKA) AND (x: hash(x)=Y)
----------------------------------------
amount: 600
```

```
sigverify PKA
--------------------
amount: 2
```

```
sigverify PKB
--------------------
amount: 600
```

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



```
if block# > T1+2000+100+100
   sigverify PKA
else
   (sigverify PKB) AND (x: hash(x)=Y)
----------------------------------------
amount: 2 BTC
```

```
sigverify PKA
--------------------
amount: 2 BTC
```

```
if block# > T2 + 40·(2000+100)
   sigverify PKB
else
   (sigverify PKC) AND (x: hash(x)=Y)
----------------------------------------
amount: 50 ETH
```

```
sigverify PKB
--------------------
amount: 50 ETH
```

```
if block# > T3 + 4·(2000+0)
   sigverify PKC
else
   (sigverify PKA) AND (x: hash(x)=Y)
----------------------------------------
amount: 600 LTC
```

```
sigverify PKC
--------------------
amount: 600 LTC
```

Figure 3: Atomic cross-chain swap among 3 parties.

Since the long confirmation time in decentralized networks makes Π_{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 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 Π_{accs} by itself is not a complete trading solution, because real-time exchanges are still needed for price discovery.

Can ACCS be generalized? Suppose that Alice wants to trade her 2 BTC for 600 LTC, Bob wants to trade his 50 ETH for 2 BTC, and Carol wants to trade her 600 LTC for 50 ETH. In Fig. 3, we show how an atomic swap among Alice, Bob, and Carol can be accomplished. Similarly to the 2-party swap protocol Π_{accs}, Alice picks a random secret x that would allow both Bob and Carol to redeem their desired cryptocurrency amounts. It is detrimental for Alice to reveal x before the third step: Bob will claim her 2 BTC, and neither Bob nor Carol will be damaged. Since the timeouts give each party enough time to redeem her desired output (after her input was claimed by another party), the 3-party atomic swap is secure. Note that the 15 seconds block interval of Ethereum implies a multiplicative factor of 40, relative to the 600 seconds interval of Bitcoin. As with Π_{accs}, the 3-party protocol is slow, because each of the first three steps requires blockchain confirmations. By contrast, a liquid real-time exchange only needs to support 2-party swaps, since the traded amounts are determined according to the current market price.

A matching service for ACCS was established in 2015, but it became defunct due to lack of popularity [57].

3. CRYPTOCURRENCY EXCHANGES

We describe several alternative designs for a real-time cryptocurrency exchange, and also survey the 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), which implies that an attacker with $1/10$ of the total computational power can double-spend her deposit with less than 0.1% probability (cf. [74, Table 1]). To take another example, most exchanges accept a Litecoin deposit after 12 PoW confirmations that take 30 minutes on average (see [13, Appendix A] and [53, Section 4] regarding the disadvantages of a faster confirmation time in blockchain protocols). Once a deposit is confirmed, the exchange credits the user with the extra balance, and allows the user to engage in real-time trades. This done by letting the user connect to the exchange server and place bid and ask orders, in accord with the current balance that the user has in the exchange server's database. When the user wishes to take ownership of her funds, she issue a withdrawal request to the exchange server, and waits for an on-chain transfer to be confirmed.

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 that users deposited to centralized exchanges (cf. [31]). In particular, about 650000 bitcoins were lost when the MtGox exchange shut down in February 2014 [48], and the users of the Bitfinex exchange lost 120000 bitcoins in August 2016 [5].

Let us note that many centralized exchanges also allow users to trade fiat currencies (such as U.S. dollar and Chinese yuan) in exchange for cryptocurrencies, by accepting fiat deposits through the traditional banking system. In Section 7 we describe how fiat currencies can be supported by Tesseract.

3.2 Exchange Based on Multisig with TTP

An exchange design under which the traders' funds cannot be stolen is described in [17, Appendix A]. The idea is that each trader will deposit her assets into a script that is controlled both by her and by a semi trusted third party (TTP). Traders will then communicate their trades in real-time to the TTP, and the TTP is supposed to keep honest accounting off-chain. Periodically, the traders and the TTP will cooperate to sign the new state after all the trades that have been made, and broadcast the result to the blockchain.

This process is highly susceptible to DoS by malicious traders who would abort instead of signing the new state. Therefore, the exchange may require splitting the traders into smaller factions, or impose penalties on misbehaving traders who refuse to sign a new state. However, such penalties would require additional collateral from traders who wish to trade with a relatively modest amount of funds (since a malicious trader can perform an abort attack by sacrificing her funds), which makes the exchange service less attractive.

As with all of the TTP-based designs, this exchange is susceptible to frontrunning attacks by a dishonest TTP (cf. Section 4.2).

3.3 Exchange with Off-chain Channels and TTP

In this design, each user establishes off-chain bi-directional payment channels [70, 28, 60] 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 Fig. 4 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 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 [62].

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.

3.4 Non-real-time Exchanges

Hallex [41] is a trust-free exchange that relies on a smart contract to ensure that users' funds cannot be stolen. However, Hallex does not offer real-time trades, since its centralized exchange server should broadcast successful trades to the blockchain as they occur. Moreover, Hallex does not support cross-chain trades, and is exposed to frontrunning attacks by the operator of its centralized server (cf. Section 4.2).

Bitsquare [19] is a non-real-time exchange with a decentralized order book, that relies on escrows (cf. [38]), security deposits, and arbitration. BitBay [18] is a decentralized non-real-time exchange in which each of trading parties provides a security deposit, and the deposits will be destroyed unless the parties come to an agreement (hence one party may attempt to extort the other by demanding a side payment).

EtherDelta is a non-real-time non-cross-chain decentralized exchange that has been operational since July 2016, with quite a significant amount of popularity. However, EtherDelta is vulnerable to frontrunning attacks, see [11].

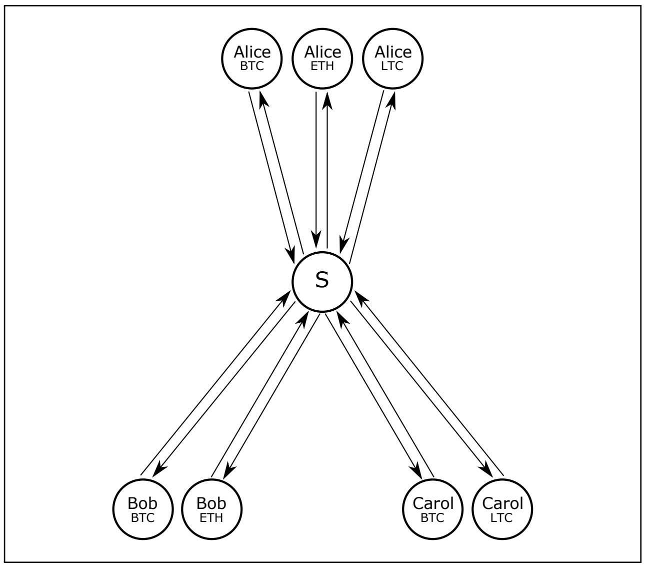


Figure 4: Exchange via off-chain channels.

3.4.1 Exchange Based on Mutual Distrust

Instead of relying on trusted hardware, it is possible 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 [29] 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 [36].

While an honest majority among the physical servers can guarantee protection from theft, attempting to provide resiliency against a dishonest majority is problematic because all-or-nothing fairness is crucial for security (as explained in Section 5). For instance, a naive protocol that forces 80% of the physical servers to sign the settlement transactions will allow a malicious coalition of 21% of the servers to commit only one of the settlements. Another possibility is to emulate the Π_{prac} protocol of Section 5.2 by using multiple logical servers, but this approach is likely to be detrimental because a hostile takeover of just one of the logical servers will violate the all-or-nothing fairness requirement.

Since the physical servers better reside in different geographical locations, 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 should probably be measured in seconds (depending on the number of physical servers). By contrast, the responsiveness of Tesseract can be measured in milliseconds.

3.4.2 ShapeShift

ShapeShift [78] 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

Table 1: Comparison of Cryptocurrency Exchanges

	Trust	DoS	Collateral	Front-running	Price Discovery
Centralized	yes	minor	no	yes	yes
TTP/multisig (Section 3.2)	minor	yes	from users	yes	yes
TTP/channels (Section 3.3)	semi	minor	from TTP	yes	yes
ShapeShift	semi	minor	no	yes	no
Tesseract	SGX	minor	no	no	yes

(i.e., via a transaction on the blockchain). Since ShapeShift is rather popular, the aggregated amount of funds that can be stolen is likely to be substantial.

Moreover, since ShapeShift does not support real-time trades and price discovery, it fetches the current prices from centralized exchanges. If ShapeShift allowed traders to play with their deposits for longer periods of time, it could perhaps support price discovery, but it would then bear a resemblance to a standard centralized exchange. In this sense, ShapeShift does not offer a solution to the systemic risk that centralized exchanges entail.

3.4.3 BitShares

BitShares [76] offers a cryptocurrency exchange that does not rely on trusted parties. It is not real-time, 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. The representatives determine the BTS price by observing centralized exchanges that are external to the BitShares system. The IOUs are required to lock extra BTS collateral that should be high enough to cover a short squeeze. See [76, Section 2] and [81] regarding the potential for market manipulations with this IOU-based 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.

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, 44, 43, 61].

The operation of Tesseract is illustrated in Fig. 5. 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 maximal time window that the enclave maintains. For instance, 8064 Bitcoin block headers would correspond to a 2 months 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 Ethereum block header is ≈ 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 [44] that the public key pk is its deposit address, and the attestation of pk should be published through multiple services (such as websites, IPFS [10], and even Bitcoin and other blockchains). As an example, Fig. 5 shows two such deposits addresses $\mathrm{PK}_{\mathrm{SGXBTC}}$, $\mathrm{PK}_{\mathrm{SGXLTC}}$, for Bitcoin and Litecoin. The anti-DoS component that we describe in Section 8 is also useful for making sure that the attested deposit addresses will be publicly known.

In fact, the deposit address better be a hash of the public key, as this increases the security and reduces the size of unspent outputs on the public ledger. For example, 257-bit compressed ECDSA public key gives 128 bits of security at the 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). Note that there is no point to mount a collision attack on a scriptless address [4]. 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. After deposit transaction was 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 (see Fig. 20), 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,

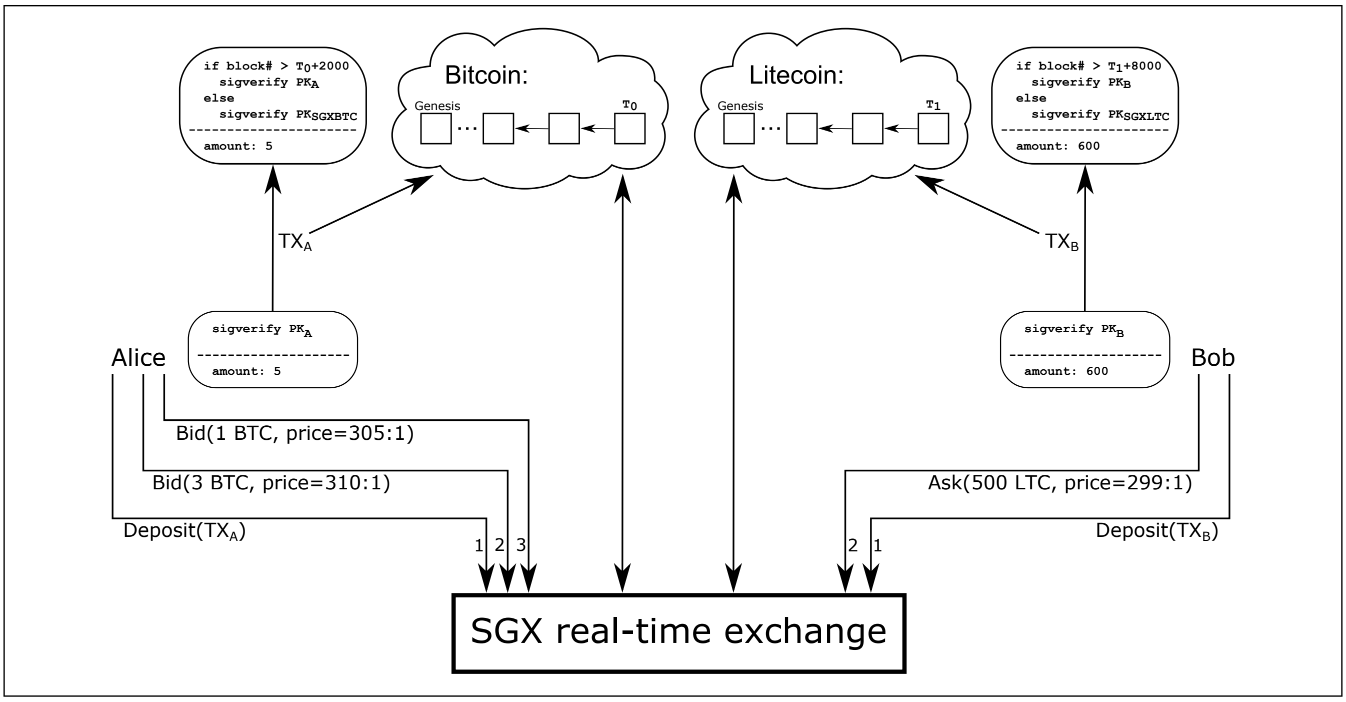


Figure 5: Illustration of deposits that are followed by bidding/asking.

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 Fig. 5, the evidence that Alice provides is $\mathsf{Deposit}(\mathsf{TX}_\mathsf{A})$.

As shown in Fig. 5, 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 $\mathrm{PK}_{\mathrm{SGXBTC}}$ is hardcoded in the output and the spending is done by creating a signature with the corresponding secret key $\mathrm{SK}_{\mathrm{SGXBTC}}$). 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 [82]. Technically, $\mathrm{SK}_{\mathrm{SGXBTC}}$ can still spend the output after the time limit (since Bitcoin transactions should be *reorg consistent* [64, 82]), but 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 in the case that 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 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 less 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 to accept 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 Fig. 5, Bob opens an account with $\mathsf{Deposit}(\mathsf{TX}_\mathsf{B})$, and then asks to sell 500 LTC for the price of 299 LTC per BTC. Since Alice's bid 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 into the order book that is kept inside the enclave. 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. Each user can request her recent trades history via the secure channel, and cancel her pending orders.

The 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 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 Fig. 7 for an illustration, and Section 5 regarding a secure settlement protocol.

The user can 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 transaction. The enclave will extend the time limit of each user's output in the settlement transactions that it constructs, thereby allowing 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 disallowed 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 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 the miners for the settlement transactions.

A forthcoming Bitcoin support for aggregated Schnorr signatures [87] 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 can be more than twice smaller, 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 hardfork of the kind of Ethereum Classic or Bitcoin Cash, users should secure themselves against replay attacks (cf. [59, 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 hardfork (or completely new cryptocurrencies), that 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

Suppose that a malicious user P_i succeeds to deceive the enclave into accepting a fake Bitcoin deposit, and the account of P_i inside the enclave is credited with the extra money. Suppose that P_i then sends an order bid to trade the BTC that she deposited for LTC, and an honest user P_j matches that bid. The next Bitcoin settlement transaction that the enclave constructs will not be confirmed, since the Bitcoin network will regard the fake deposit input as invalid. However, the next Litecoin settlement transaction should be valid, hence P_i will profit at the expense of P_j.

Let us assume that an adversary $\mathcal{A}$ has p fraction of the computational power of the Bitcoin network, and also has physical access to the Tesseract server. Thus, $\mathcal{A}$ can cut the communication between the enclave and the Bitcoin network, feed the enclave with dummy blocks that that include her fake deposit, and wait for the enclave to construct and release the Litecoin settlement transaction.

Since $p < \frac{1}{2}$, the rate at which $\mathcal{A}$ feeds blocks to the enclave is at least twice slower than in the case that there is no attack. By relying on the SGX 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 [80, 58] by design (cf. Section 8 regarding our approach to resiliency).

The time between every two consecutive Bitcoin blocks is an exponential 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. We define n to be the number of blocks that a deposit needs to be buried under, before it is credited by the enclave. We define δ as 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 δ 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 δ 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.

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

- An adversary with $p \leq \frac{1}{5}$ of the computational power can mount a successful eclipse attack on the enclave with probability 2^{-92} or smaller.
- In 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

Table 2: Deposit confidence vs false positives

p	δ	n	$\Pr[\text{Erlang}(n,p) \leq \delta n]$	$\Pr[\text{Erlang}(n,1) > \delta n]$
$\frac{1}{10}$	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}

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 refuse to trade 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 5.2.

4.2 Secure Communication

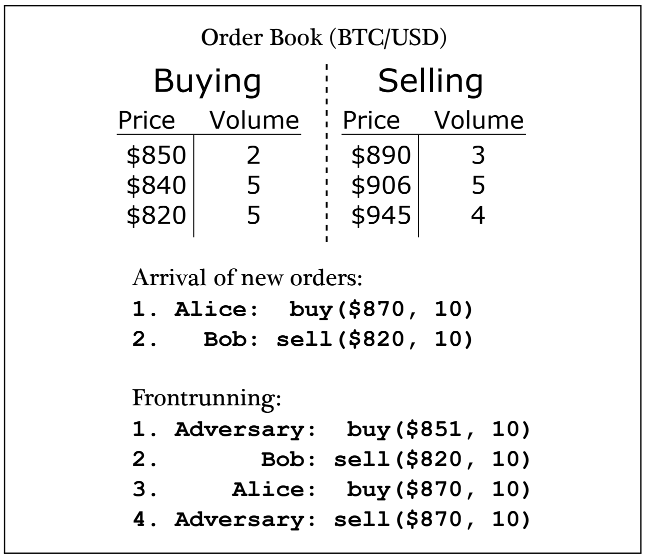
For each user who has already opened an account with Tesseract, we establish a secure channel (TLS) when the user wishes to communicate with the enclave. The reasons for a channel with authenticated encryption are the following:

- Fast identification: the authenticated messages are computed via symmetric-key operations, after the initial key exchange (done via public-key operations) that established the channel. This form of communication is suitable for real-time trades, since symmetric-key operations are an order of magnitude faster than public-key operations.

- 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 perform trades that would be advantageous to her. The encrypted communication avoid such attacks.

An example of a frontrunning attack is shown in Fig. 6. 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 will be 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 new orders with her account. In this scenario, the adversary would be guaranteed to gain $\$10 \cdot (870 - 851) = \$190$, by buying Bob's 10 BTC for cheap and then selling it to Alice.

Since all the 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


Order Book (BTC/USD)

Buying Price	Buying Volume	Selling Price	Selling Volume
\$850	2	\$890	3
\$840	5	\$906	5
\$820	5	\$945	4

Arrival of new orders:

```
1.  Alice:  buy($870, 10)
2.    Bob: sell($820, 10)
```

Frontrunning:

```
1. Adversary:  buy($851, 10)
2.       Bob: sell($820, 10)
3.     Alice:  buy($870, 10)
4. Adversary: sell($870, 10)
```

Figure 6: Example of frontrunning.

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 [30]) 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. [6]), but these automated traders do communicate each of their orders to the server.

4.3 Double Attestation

It may be the case that several reputable providers would offer different variants of the Tesseract service (perhaps with their own tokenized coins and fiat assets, see Sections 6 and 7). This raises the following question: does a single entity (i.e., the hardware manufacturer) has the power to compromise the security of all the Tesseract-based platforms, simultaneously?

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 the security is by attesting to fraudulent data. In our context, suppose for example that there are two Tesseract-based exchanges X_1, X_2 that invite users to deposit their funds to $\text{PK}_{\text{SGXBTC1}}$ and $\text{PK}_{\text{SGXBTC2}}$, 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 $\text{PK}'_{\text{SGXBTC1}}, \text{PK}'_{\text{SGXBTC2}}$ that Intel would generate together with the corresponding secret keys $\mathrm{SK}'_{\mathrm{SGXBTC1}}, \mathrm{SK}'_{\mathrm{SGXBTC2}}$. Thus, Intel will be able deceive users into sending their deposits to $\mathrm{PK}'_{\mathrm{SGXBTC1}}, \mathrm{PK}'_{\mathrm{SGXBTC2}}$, and then steal funds that users wished to deposit to X_1, X_2.

The manufacturer may also break the 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 was one assembly opcode that generates a random key (using 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 (as in the case of SGX and Tesseract).

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 [72]), but it is not an inherent defect of the trusted hardware model (and hopefully the Management Engine will have an opt-out option in the future).

Similarly to [77, Section V.A], 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 the 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 8.1), 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 the real-time updates to the public order book, the secondary signature is already implied in case 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 7) 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 the adversary to steal funds from honest users. We provide an illustration of the risk in Fig. 7. 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 broadcasted 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.

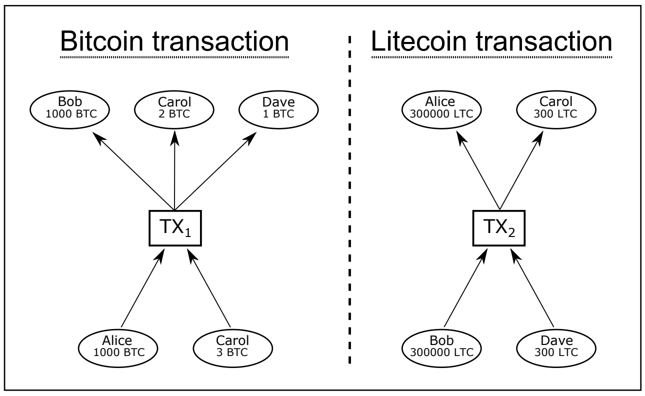


Figure 7: The cross-chain settlement problem.

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

Definition 1 (All-or-nothing settlement). *Given the transaction tx_1 for system $\mathcal{C}_A$ and the transaction tx_2 for system $\mathcal{C}_B$, an all-or-nothing cross-chain settlement is a protocol that guarantees that*

1. *Both tx_1 will become confirmed on system $\mathcal{C}_A$ and tx_2 will become confirmed on system $\mathcal{C}_B$, or*
2. *Neither tx_1 will become confirmed on system $\mathcal{C}_A$ nor will tx_2 become confirmed on system $\mathcal{C}_B$.*

In our context, $\mathcal{C}_A$ and $\mathcal{C}_B$ are cryptocurrencies. We stress that the parties that execute the consensus protocol for $\mathcal{C}_A$ may be unaware of the existence of $\mathcal{C}_B$, 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 in comparison to the popular 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 $U_1^{\mathrm{in}}, U_2^{\mathrm{in}}$ denote the sets of users in the inputs of the transactions*

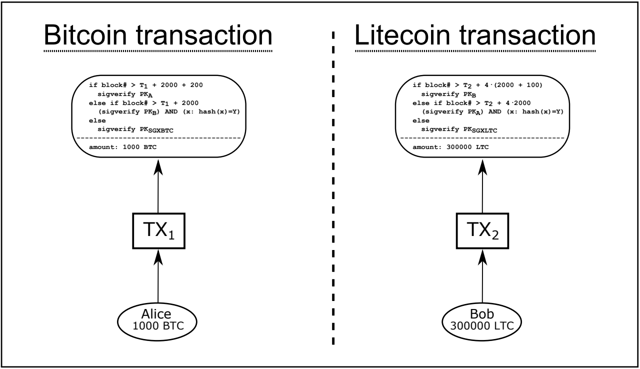
tx_1, tx_2, and let $U_1^{\text{out}}, U_2^{\text{out}}$ denote the sets of users in the outputs of tx_1, tx_2. Let $U = U_1^{\text{in}} \cup U_2^{\text{in}} \cup U_1^{\text{out}} \cup U_2^{\text{out}}$. An unprivileged cross-chain settlement is a protocol that satisfies Definition 1 in the presence of an adversary $\mathcal{A}$ 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 the leaves the CPU, before honest users have an opportunity to make use of this data in a secure fashion. Notice 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 $\mathcal{A}$ more power than a real-world 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, $\mathcal{A}$ will not be able to obtain the data that Carol accesses; the only action available to $\mathcal{A}$ 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 self-interest), 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 from having 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 Fig. 8, 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 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 TX_1 and TX_2 are both confirmed, it sends x in encrypted form *only* to Alice, by using 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 TX_1 is longer, so that Bob will have enough time to redeem the 1000 BTC after Alice reveals x and thereby spends the 300000 LTC.

Let us note that Definition 2 does not give $\mathcal{A}$ the power to observe secret information that is inside the enclave. In the Tesseract implementation, this is justified because we use a constant-time constant-memory library for the cryptographic operations [88], hence the potential for side-channel attacks is greatly reduced.



```
if block# > T1 + 2000 + 200
  sigverify PKA
else if block# > T1 + 2000
  (sigverify PKB) AND (x: hash(x)=Y)
else
  sigverify PKSGXBTC
------------------------------------
amount: 1000 BTC
```

```
if block# > T2 + 4·(2000 + 100)
  sigverify PKB
else if block# > T2 + 4·2000
  (sigverify PKA) AND (x: hash(x)=Y)
else
  sigverify PKSGXLTC
------------------------------------
amount: 300000 LTC
```

Figure 8: Settlement with two parties.

Protocol Π_{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 $S_1, S_2, \ldots, S_N$.
4. The enclave waits for acknowledgements from $S_1, S_2, \ldots, S_N$.
5. The enclave broadcasts TX_1 to $\mathcal{C}_1$ and TX_2 to $\mathcal{C}_2$.
6. Each S_i that sees TX_i but not TX_{3-i} will fetch K from TX_i, decrypt ct, and broadcast TX_{3-i} to $\mathcal{C}_{3-i}$.

Figure 9: Naive protocol for fair settlement.

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

5.1 Theoretical Solution

To clarify why an intricate protocol is needed, we first describe a simple protocol Π_{simp} that relies on N extra servers $S_1, S_2, \ldots, S_N$ that are supposedly reputable. See Fig. 9.

The cryptocurrency systems $\mathcal{C}_1$ and $\mathcal{C}_2$ can be for example Bitcoin and Litecoin as in Fig. 7. The embedding of K into TX_1 and TX_2 can be done with the `OP_RETURN` script instruction [7], that 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 TX_1 or TX_2, because the signatures for TX_1 and TX_2 are done on the entire transaction data (i.e., data that includes the `OP_RETURN` output).

Since information that is published on a blockchain becomes publicly available, the idea behind Π_{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, Π_{simp} is insecure, due to a race condition. The adversary $\mathcal{A}$ can intercept both TX_1 and 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 TX_i is about

```
Functionality RMIT (refundable multi-input transaction)

Notation: let $\mathcal{C}$ be a cryptocurrency system.

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

Figure 10: The ideal functionality RMIT.

to expire, and then broadcast TX_{3-i}. Then, $\mathcal{A}$ will instruct P_j to spend that input, thereby invalidating 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 TX_{3-i} and broadcast their decrypted TX_i.

If the cryptocurrency systems $\mathcal{C}_1$,$\mathcal{C}_2$ allow 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 TX_i will embed the TX_{3-i} data in a designated output, the enclave will broadcast both TX_1 and TX_2, and any user would then have the opportunity to enforce fairness. This would bloat $\mathcal{C}_i$ with the entire 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 Π_{simp} exhibits.

Let us therefore present a theoretical solution that avoids the race condition. Our strategy is to condition the second settlement transaction TX_2 on the result of the first settlement transaction TX_1, by constraining TX_2 with PoW-based predicates that verify whether certain events occurred 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 Fig. 10.

The description of $\mathsf{TX}_1, \mathsf{TX}_2$ is outlined is Fig. 11. We use the notation $\mathsf{TX}_{i,j}$ to denote that TX_i was updated by supplying w that satisfied ϕ_j. The secrets $x_1 \in \{0,1\}^\lambda, x_2 \in \{0,1\}^\lambda$ are generated inside the enclave. The predicates ϕ'_1, ϕ'_2 are specified in Fig. 12. To elaborate, the hardcoded parameter D_0 specifies a difficulty level for PoW mining, ℓ_1 is an upper bound on the length of an authentication path of a Merkle tree, and ℓ_2 is a PoW confidence parameter. The input witness w for ϕ'_1 consists of up to ℓ_1 sibling hash values v_j for the authentication path (with direction $d_j \in \{\text{'L'},\text{'R'}\}$), together with exactly ℓ_2 header values. The predicate ϕ'_1 will verify that $\mathsf{TX}_{1,1}$ is in a leaf that reaches some root value r, and that r is extended by valid proofs of

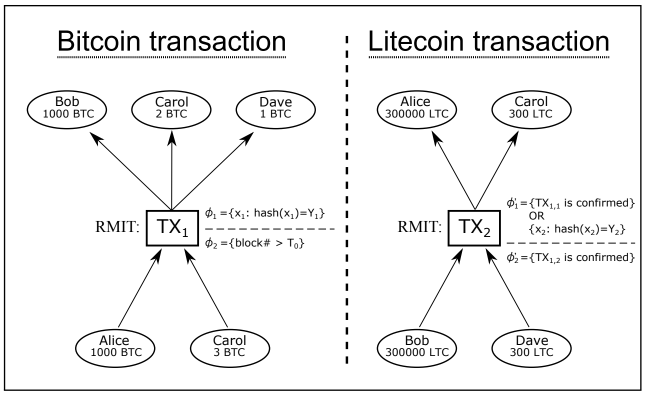


Figure 11: Theoretical fair settlement transactions.

```
Predicate $\phi'_1$

Hardcoded parameters: $\mathsf{TX}_1, D_0, \ell_1, \ell_2$
Input: $w = ((v_1, d_1), (v_2, d_2), \ldots, (v_k, d_k), H_1, H_2, \ldots, H_{\ell_2})$
  1. Embed $\mathsf{hash}(\mathsf{TX}_{1,1})$ into $y$
  2. For $j = 1$ to $\mathsf{min}(k, \ell_1)$
       • If $d_j$ = 'L' then $y := \mathsf{hash}(y, v_j)$ else $y := \mathsf{hash}(v_j, y)$
  3. For $j = 1$ to $\ell_2$
       • $y := \mathsf{hash}(y, H_j)$
       • 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: $w = (G_1, \ldots, G_n, (v_1, d_1), \ldots, (v_k, d_k), H_1, \ldots, H_{\ell_2})$
  1. $z := b_1$
  2. For $j = 1$ to $\mathsf{max}(n, \ell_3)$
       • $z := \mathsf{hash}(z, G_j)$
       • If $z > D_0$ then return false
  3. Embed $\mathsf{hash}(\mathsf{TX}_{1,2})$ into $y$
  4. For $j = 1$ to $\mathsf{min}(k, \ell_1)$
       • If $d_j$ = 'L' then $y := \mathsf{hash}(y, v_j)$ else $y := \mathsf{hash}(v_j, y)$
  5. If $y \neq z$ then return false
  6. For $j = 1$ to $\ell_2$
       • $y := \mathsf{hash}(y, H_j)$
       • If $y > D_0$ then return false
  7. return true
```

Figure 12: The cryptocurrency scripts ϕ'_1, ϕ'_2.

work $H_1, H_2, \ldots, H_{\ell_2}$ that meet the difficulty level D_0. The input witness w for ϕ'_2 does the same, but also verifies that there is a chain of at least ℓ_3 blocks between the hardcoded b_1 and $\mathsf{TX}_{1,2}$.

We describe the theoretical protocol Π_{theo} for all-or-nothing settlement in Fig. 13. Note that the enclave constructs TX_2 only after it receives the evidence that TX_1 was confirmed in the end of Step 1, by hardcoding b_1 as the hash of the

Protocol Π_{theo}

1. The enclave releases TX_1 and waits for evidence that it was confirmed on the system $\mathcal{C}_1$.
2. The enclave releases TX_2 and waits for evidence that it was confirmed on the system $\mathcal{C}_2$.
3. The enclave releases x_1 and waits for evidence that $\mathsf{TX}_{1,1}$ was confirmed on the system $\mathcal{C}_1$.
4. The enclave releases x_2.

Figure 13: Theoretical protocol for fair settlement.

Table 3: Breaking the security of Π_{theo}

p	T_1	ℓ_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}

block in which TX_1 resides.

Essentially, Π_{theo} avoids the race condition by first making sure that TX_1 was resolved on the cryptocurrency system $\mathcal{C}_1$ either by committing the output or by committing the inputs, and then allowing 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 Fig. 11), which would be followed by the miners of $\mathcal{C}_1$ helping 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 help to resolve 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 $\mathcal{C}_2$ 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 [74], because Π_{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 TX_1 into $\mathsf{TX}_{1,1}$. The miners of $\mathcal{C}_1$ will create the witness w_2 in which 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

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

The first negative binomial variable counts the number 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 ϕ_2' requires. The second negative binomial variable count the number of blocks that $\mathcal{A}$ creates while the honest miners are creating ℓ_2 blocks. If $\mathcal{A}$ can extend her head start to reach ℓ_2 or more blocks before the honest miners, then the attack succeeds.

In Table 3, we give exemplary figures for the attack on Π_{theo}. For easy comparison, we also include the success probability without a head start (i.e., $T_1 = 0$), which is simply the probability $\Pr[\text{NegBin}(\ell_2, p) \geq \ell_2]$.

For the opposite attack, $\mathcal{A}$ may intercept x_1 in Step 3 and then create a hidden chain w_2 that excludes x_1. With this attack strategy, $\mathcal{A}$ will broadcast x_1 to $\mathcal{C}_1$ right before the timeout T_0 is reached, in hope that her hidden chain w_2 will outcompete the chain that the miners of $\mathcal{C}_1$ begin to create. This attack vector is mitigated by disallowing a precomputation of w_2. Specifically, the enclave hardcodes b_1 into TX_2, and the predicate ϕ_2' verifies that b_1 is buried under at least ℓ_3 PoW blocks.

The parameter ℓ_3 should be set to $2\ell_2 + T_1$. This gives a time span of T_1 blocks to update 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 $\mathcal{C}_2$ relative to $\mathcal{C}_1$.

In the current Bitcoin network, $\ell_1 = 12$ suffices, hence the predicates ϕ_1', ϕ_2' require $\leq 12 + \ell_2 + \ell_3$ hash invocations for confidence level ℓ_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 ϕ_1', ϕ_2' for a mild fee. These parameters for PoW-based SPV proofs can be even better if the cryptocurrency system supports NIPoPoW [46].

It is unlikely that Π_{theo} will be vulnerable to an attack that embeds a transaction that spends 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 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 TX_1) as its hash.

The main obstacle to an implementation of Π_{theo} in Bitcoin is the RMIT functionality. It is possible to implement the specific RMIT that Π_{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_{refund} that has `locktime` [85] 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 Π_{theo} reaches Step 3 and the enclaves 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

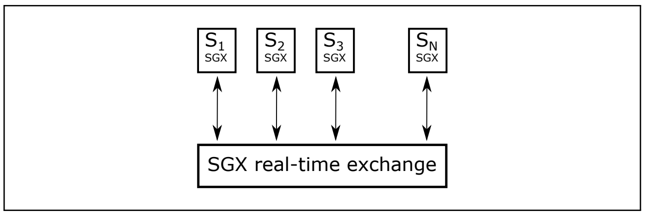


Figure 14: Practical fair settlement.

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 ϕ_1 or ϕ_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 Π_{theo} (and its generalization to more than two systems) requires RMIT support by all the cryptocurrency systems that are involved in the settlement.

5.2 Practical Solution

The theoretical protocol Π_{theo} of Section 5.1 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 Π_{prac} for the all-or-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 Fig. 14), and ensure that Π_{prac} satisfies Definition 2 if there exists at least one server $S_i \in \{S_1, S_2, \ldots, S_N\}$ that is beyond the physical reach of the adversary $\mathcal{A}$. That is to say, we assume that S_i can communicate with the cryptocurrency systems $\mathcal{C}_1$,$\mathcal{C}_2$ without interference.

The main idea of Π_{prac} is to emulate the essential characteristic of the theoretical protocol Π_{theo}, which is to wait for a proof that the settlement transaction TX_1 was either committed to $\mathcal{C}_1$ or cancelled, and then do the same for the settlement transaction TX_2.

The settlement protocol Π_{prac} that Tesseract and the servers $S_1, S_2, \ldots, S_N$ execute is specified in Fig. 15. 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 $\mathsf{TX}_1^{\mathsf{c}}, \mathsf{TX}_2^{\mathsf{c}}$ are "cancellation" transactions that invalidate the settlement transactions $\mathsf{TX}_1, \mathsf{TX}_2$, respectively. In Bitcoin, $\mathsf{TX}_i^{\mathsf{c}}$ can be implemented simply by spending one of the inputs of TX_i into a new output that is identical to that input (this will cause $\mathsf{TX}_i, \mathsf{TX}_i^{\mathsf{c}}$ to conflict with each other).

Thus, the protocol Π_{prac} seeks to preserve the property that TX_2 remains confidential inside the enclaves for as long as TX_1 is not yet confirmed. This property avoids the risk that $\mathsf{TX}_i, \mathsf{TX}_{3-i}^{\mathsf{c}}$ will compete for confirmations at the same time, as that can easily violate to the all-or-nothing requirement.

In the case that at least one server S_i is not under physical

Protocol Π_{prac}

1. Tesseract sends $ct = \mathsf{encrypt}_K(\mathsf{TX}_1, \mathsf{TX}_2, \mathsf{TX}_1^{\mathsf{c}}, \mathsf{TX}_2^{\mathsf{c}})$ 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 the system $\mathcal{C}_1$.
4. Starting from the time at which it received ct in Step 1, each server $S_i \in \{S_1, S_2, \ldots, S_N\}$ inspects the next blocks of the system $\mathcal{C}_1$
   - If S_i does not see 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 TX_1 is confirmed on $\mathcal{C}_1$, then it broadcasts TX_2 to the system $\mathcal{C}_2$.
   - If S_i sees that $\mathsf{TX}_1^{\mathsf{c}}$ is confirmed on $\mathcal{C}_1$, then it broadcasts $\mathsf{TX}_2^{\mathsf{c}}$ to the system $\mathcal{C}_2$.

Figure 15: Practical protocol for fair settlement.

attack, we have that either TX_1 or $\mathsf{TX}_1^{\mathsf{c}}$ will be broadcasted to $\mathcal{C}_1$ within T_1 blocks. As a consequence, either TX_1 or $\mathsf{TX}_1^{\mathsf{c}}$ will be confirmed after $T_1 + \ell_2$ blocks. This would allow S_i or one of the other non-adversarial servers to broadcast the appropriate transaction (i.e., TX_2 or $\mathsf{TX}_2^{\mathsf{c}}$) to the cryptocurrency system $\mathcal{C}_2$, causing it to be confirmed too.

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

1. $\mathcal{A}$ will intercept the data TX_1 that Tesseract reveals in Step 3 of Π_{prac}, and deactivate the Tesseract server.
2. $\mathcal{A}$ will eclipse the server S_j, and feed it with a fake blockchain (generated by $\mathcal{A}$ herself) that contains TX_1.
3. When the enclave of S_j becomes convinced that TX_1 was confirmed, it will release TX_2.
4. $\mathcal{A}$ will wait until $\mathsf{TX}_1^{\mathsf{c}}$ is confirmed on $\mathcal{C}_1$, and then broadcast TX_2 to $\mathcal{C}_2$.

As in Section 5.1, 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 TX_1, whereas $\mathcal{A}$ begins to create her fake chain immediately. Note that the purpose of the cancellation transaction $\mathsf{TX}_2^{\mathsf{c}}$ is to defeat this race attack, in the case that $\mathcal{A}$ fails to generate ℓ_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 Π_{prac}, than it is to attack Π_{theo}. This is because Π_{prac} can specify the precise duration T_1, and Π_{theo} has to estimate T_1 by setting T_0 in the predicate ϕ_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 ϕ_2 to be triggered and thus abort the settlement.

Notice that $\mathcal{A}$ cannot mount an eclipse attack before Step 3 of Π_{prac} is reached. The reason is that only the Tesseract enclave can produce the data 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-or-nothing 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 $S_i \in \{S_1, S_2, \ldots, S_N\}$, 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 8.

Another advantage of Π_{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 ϕ_1', ϕ_2' lack the power to express the irreversibility condition of a more complex cryptocurrency system. See Section 1.1 for further details.

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.

5.3 Solution with One Secure Processor

Is it possible to devise a workable protocol for all-or-nothing 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 [8, 39] to reveal $\mathsf{TX}_1, \mathsf{TX}_2$ simultaneously. The Tesseract enclave can generate a fresh symmetric key $K \in \{0,1\}^\lambda$, send the ciphertext $ct = \mathsf{encrypt}_K(\mathsf{TX}_1, \mathsf{TX}_2)$ 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 λ 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 [54]. The combined protocol Π_{grad} is parameterized according to a decoys amount $d \geq 2$, batching value $m \geq 1$, and a timeout τ (for example $\tau = 10$ minutes). See Fig. 16 for the description of Π_{grad}.

It is inherently the case that the adversary $\mathcal{A}$ can 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 Π_{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 α 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 α correctly is $\frac{1}{d-1}$. Furthermore, under the assumption that $\mathcal{A}$ cannot breaks K within τ 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.

Protocol Π_{grad}

1. The enclave picks a random symmetric key $K \in \{0,1\}^\lambda$ such that $K = (w_1, w_2, \ldots, w_q), w_i \in \{0,1\}^m, \lambda = q \cdot m$.
2. The enclave creates $ct = \mathsf{encrypt}_K(\mathsf{TX}_1, \mathsf{TX}_2)$.
3. For $w_i \in \{w_1, w_2, \ldots, w_q\}$, the enclave creates N-out-of-N additive secret shares of w_i, 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 τ time, then the honest servers begin an exhaustive search for the key K.
7. Any server that decrypts ct will broadcast TX_1 to $\mathcal{C}_1$ and TX_2 to $\mathcal{C}_2$.

Figure 16: Gradual protocol for fair settlement.

Therefore, Π_{grad} is more secure when the timeout parameter τ 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 Π_{grad} is completely insecure: $\mathcal{A}$ will be able to corrupt the last server S_N and verify for each ℓ 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 Π_{grad} is strictly more secure than the basic gradual release protocol. Another advantage over the basic protocol is that Π_{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 Π_{grad} less efficient.

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

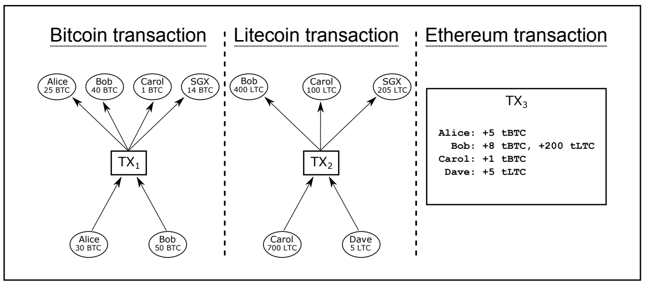


Figure 17: Atomic issuance of tokenized coins.

6. 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 8) 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 [15]). 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 Fig. 17. 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 Fig. 17 for example, if TX_1 is not committed to the Bitcoin blockchain but TX_3 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_3 is not committed to the Ethereum blockchain but TX_1 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 [79]. It is also possible to mint the tokenized coins as colored coins [73] 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 7). 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 8 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.

7. 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 [24, Sec-

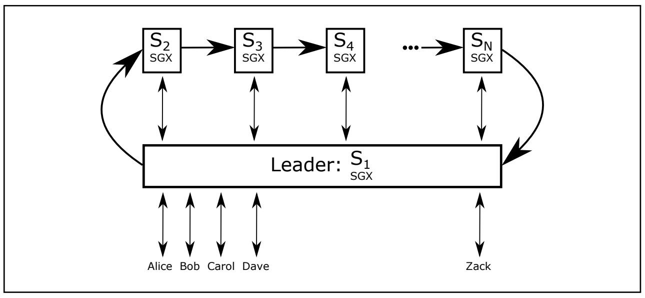


Figure 18: Fairness with anti-DoS protection.

tion 5.2] and [73] in this regard. One instantiation of this idea that enjoys a relatively high degree of popularity is Tether [26, 22], which circulates tokens that are pegged to the U.S. dollar (involving proof-of-reserve) using the Omni layer [65] 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 [73, 51], 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 [63] 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 [51] for a proposed implementation and [69] regarding future ideas). However, Ethereum already supports the equivalent of tagging-based colored coins, in the form of an ERC20 smart contract (cf. [79]). 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 fiat currencies), though this capability involves trust in the reputation of the issuers of the assets.

8. AVAILABILITY

The Tesseract exchange service can be initialized with the SGX server S_1 as its current leader, and execute the Paxos [50] consensus protocol together with the other SGX servers $S_2, S_3, \ldots, S_N$. See Fig. 18 for an illustration.

The requirements that the Paxos Synod protocol relies upon are satisfied in our setting, because of the following reasons:

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

The complete Tesseract protocol Π_{RTExch} is outlined in Fig. 19. To accomplish all-or-nothing settlements, Π_{RTExch} uses Π_{prac} as a subroutine. As with Π_{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 ($288 \cdot 5$ minutes). In the case of a DoS attack on Π_{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.

Π_{RTExch} can be regarded as a composition of two components, one is the Paxos protocol that guarantees consistency among the servers, and the other is the all-or-nothing fairness protocol that interacts with cryptocurrency systems.

Since $H > 2$ and only non-Byzantine faults are possible (due to SGX), all-or-nothing fairness holds if at least $\lceil \frac{1}{H} \rceil$ of the servers not under adversarial control, even if the network is asynchronous. For example, parametrizing Π_{RTExch} according to $H = 4$ would imply that the adversary $\mathcal{A}$ must corrupt more than 75% of the servers to violate all-or-nothing fairness (and by corrupting 25% of the servers $\mathcal{A}$ can mount a DoS attack). This follows because Π_{RTExch} ensures that there will never be two servers that act as leaders of different epochs at the same time: 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 Π_{prac}. However, if $\mathcal{A}$ controls the communication traffic of $\lceil n(1 - \frac{1}{H}) \rceil$ servers, then $\mathcal{A}$ can let S_L receive $\lceil n(1 - \frac{1}{H}) \rceil$ acknowledgments and release TX_1 to $\mathcal{C}_1$, without ever releasing TX_2 to $\mathcal{C}_2$ (Π_{prac} can be attacked in the same manner only if all the servers are under adversarial control).

In non-settlement epochs, the first component of Π_{RTExch} ensures liveness if the network is synchronous and there is a majority of non-faulty servers. This is simply because Paxos guarantees liveness in a synchronous network.

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, Π_{RTExch} attempts to restart an all-or-nothing epoch with a new leader, immediately after the last all-or-nothing 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, Π_{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 $\mathsf{TX}_1^c, \mathsf{TX}_2^c$ on its own — for example by spending the reserve output (cf. Section 6) 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 starts 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_1^c and releasing it after an

Protocol Π_{RTExch}

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

- For every $i \in [N]$:
  - The server S_i initializes $L_i := 1$ as the leader index and $J_i := 0$ as the first epoch.
- Let L denote the index of the server with $L_i = i$.
- *Communication with traders.*
  - The server S_L 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 Π_{prac} with the following modifications:
    * In Step 1 of Π_{prac}, S_L sends $m = (J_L, ct)$, where $ct = \mathsf{encrypt}_{sk}(dat, \mathsf{TX}_1, \mathsf{TX}_2)$.
    * In Step 2 of Π_{prac}, S_L waits for acknowledgements from $\lceil n(1 - \frac{1}{H}) \rceil$ or more servers.
  - For every $i \in [N]$:
    * 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 TX_1 was confirmed on $\mathcal{C}_1$ and 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 $\mathsf{TX}_1^{\mathsf{c}}$ was confirmed on $\mathcal{C}_1$ and $\mathsf{TX}_2^{\mathsf{c}}$ 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 19: Outline of the Tesseract protocol.

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 8.1). 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 the Π_{RTExch} protocol.

8.1 Setup of the Servers

To achieve maximum security, we design the initialization procedure for Π_{RTExch} as follows. Our enclave program code $\mathcal{P}_{\text{RTExch}}$ contains a hardcoded list of N *endorsement* public keys $\mathsf{RPK_1}, \mathsf{RPK_2}, \dots, \mathsf{RPK_N}$, corresponding to the reputable owners of the N servers (e.g., S_1 is located at Cornell University, S_2 is located at MIT, 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 $\mathsf{encrypt}(tsk_i)$ that is created using the symmetric hardware key of the SGX CPU of S_i. The owner keeps a backup of $\mathsf{encrypt}(tsk_i)$, sends tpk_i to $\{S_j\}_{j \neq i}$, waits to receive $\{tpk_j\}_{j \neq i}$, signs $m = (tpk_1, tpk_2, \dots, tpk_N)$ with $\mathsf{RSK_i}$, and sends the signature es_i to $\{S_j\}_{j \neq i}$. 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_j via the identities tpk_i and tpk_j.

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 program $\mathcal{P}_{\text{RTExch}}$ with $m, \mathsf{encrypt}(tsk_i)$ 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 ith 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.

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

9. REFERENCES

[1] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative Technology for CPU Based Attestation and Sealing. In *HASP'13*, 2013.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Fair two-party computations via the bitcoin deposits. In *First Bitcoin Workshop, FC*, 2014.

[3] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *IEEE Security and Privacy*, 2014.

[4] Adam Back. $O(2^{80})$ theoretical attack on p2sh. https://bitcointalk.org/index.php?topic=323443.0, 2013.

[5] Clare Baldwin. http://www.reuters.com/article/us-bitfinex-hacked-hongkong-idUSKCN10E0KP.

[6] Andrew Barisser. High frequency trading on the coinbase exchange, 2015. https://medium.com/on-banking/high-frequency-trading-on-the-coinbase-exchange-f804c80f507b.

[7] Massimo Bartoletti and Livio Pompianu. An analysis of bitcoin op_return metadata. In *Financial Cryptography 4th Bitcoin Workshop*, 2017. https://arxiv.org/abs/1702.01024.

[8] D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, 1989.

[9] Jethro Beekman. A Denial of Service Attack against Fair Computations using Bitcoin Deposits, 2014. https://eprint.iacr.org/2014/911.

[10] Juan Benet. https://ipfs.io/.

[11] Iddo Bentov, Lorenz Breidenbach, Phil Daian, Ari Juels, Yunqi Li, and Xueyuan Zhao. The cost of decentralization in 0x and EtherDelta. http://hackingdistributed.com/2017/08/13/cost-of-decent/, 2017.

[12] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *Financial Cryptography Bitcoin Workshop*, 2016.

[13] Iddo Bentov, Pavel Hubáček, Tal Moran, and Asaf Nadler. Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies. http://eprint.iacr.org/2017/300.

[14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Crypto*, 2014.

[15] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In *Asiacrypt*, 2017.

[16] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: extending bitcoin's proof of work via proof of stake. In *SIGMETRICS Workshop on Economics of Networked Systems, NetEcon*, 2014.

[17] Iddo Bentov, Alex Mizrahi, and Meni Rosenfeld. Decentralized prediction market without arbiters. In *Financial Cryptography 4th Bitcoin Workshop*, 2017.

[18] BitBay. http://bitbay.market/.

[19] Bitsquare. https://bitsquare.io/whitepaper.pdf.

[20] Daniel G. Brown. How i wasted too long finding a concentration inequality for sums of geometric variables. https://cs.uwaterloo.ca/~browndg/negbin.pdf.

[21] Vitalik Buterin. Ethereum white paper. http://github.com/ethereum/wiki/wiki/White-Paper.

[22] CryptoAsset Market Capitalizations. https://coinmarketcap.com/assets/.

[23] Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949, 2013.

[24] Jeremy Clark, Joseph Bonneau, Edward W. Felten, Joshua A. Kroll, Andrew Miller, and Arvind Narayanan. On Decentralizing Prediction Markets and Order Books. In *WEIS*, 2014.

[25] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gun Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains. In *Financial Cryptography 3rd Bitcoin Workshop*, 2016.

[26] Tether: Fiat currencies on the Bitcoin blockchain. https://tether.to/wp-content/uploads/2016/06/TetherWhitePaper.pdf, 2016.

[27] Phil Daian, Rafael Pass, and Elaine Shi. Snow White: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.

[28] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *17th Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2015.

[29] Desmedt and Frankel. Threshold cryptosystems. In *CRYPTO: Proceedings of Crypto*, 1989.

[30] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium*, 2004.

[31] dree12 (pseudonym). List of major bitcoin heists, thefts, hacks, scams, and losses. https://bitcointalk.org/index.php?topic=576337.

[32] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, 2009.

[33] Tuyet Duong, Lei Fan, Thomas Veale, and Hong-Sheng Zhou. Securing bitcoin-like backbone protocols against a malicious majority of computing power. *IACR Cryptology ePrint Archive*, 2016:716.

[34] Ben A. Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. Iron: Functional encryption using intel sgx, 2017. https://eprint.iacr.org/2016/1071.

[35] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[36] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In *Applied Cryptography and Network Security - 14th ACNS*, volume 9696 of *Lecture Notes in Computer Science*, pages 156–174. Springer, 2016.

[37] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *26th Symposium on Operating Systems Principles*, 2017.

[38] Steven Goldfeder, Joseph Bonneau, Rosario Gennaro, , and Arvind Narayanan. Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin. In *Financial Cryptography*, 2017. https://freedom-to-tinker.com/2017/03/22/how-to-buy-physical-goods-using-bitcoin-with-improved-security-and-privacy/.

[39] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *Proceedings of Advances in Cryptologie (CRYPTO '90)*, volume 537 of *LNCS*, pages 77–93, Berlin, Germany, August 1991. Springer.

[40] BitFury Group, 2015. http://bitfury.com/content/5-white-papers-research/pos-vs-pow-1.0.2.pdf.

[41] Jeppe Hallgren, Malte Hallgren, Sam Fisher, Jakob Hautop, Nicolai Larsen, and Omri Ross. Hallex: A trust-less exchange system for digital assets. https://www.deepdotweb.com/2017/03/10/hallex-

decentralized-cryptocurrency-exchange-via-ethereums-turing-complete-blockchain/, 2017.

[42] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. https://eprint.iacr.org/2016/575, 2016.

[43] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy - HASP '13*, pages 1–1, 2013.

[44] SP Johnson, VR Scarlata, C Rozas, E Brickell, and F Mckeen. 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.

[45] Aggelos Kiayias, Ioannis Konstantinou, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, 2017. http://eprint.iacr.org/2016/889.

[46] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. https://eprint.iacr.org/2017/963, 2017.

[47] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Eurocrypt*, 2015.

[48] Sophie Knight. http://www.reuters.com/article/us-bitcoin-mtgox-wallet-idUSBREA2K05N20140321.

[49] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (S&P)*, 2016.

[50] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst*, 16(2):133–169, 1998.

[51] Johnson Lau. bip-color. https://github.com/jl2012/bips/blob/color/bip-color.mediawiki, 2017.

[52] Charles Lee. Litecoin. http://litecoin.org/, 2011.

[53] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S. Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *International Conference on Autonomous Agents and Multiagent Systems*, 2015.

[54] Hung-Yu Lin and Lein Harn. Fair reconstruction of a secret. *Information Processing Letters*, 55(1):45–47, 7 July 1995.

[55] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter R. Pietzuch, and Emin Gün Sirer. Teechain: Scalable blockchain payments using trusted execution environments. *CoRR*, abs/1707.05454, 2017.

[56] Joshua Lind, Ittay Eyal, Peter R. Pietzuch, and Emin Gün Sirer. Teechan: Payment channels using trusted execution environments. In *Financial Cryptography 4th Bitcoin Workshop*, 2017.

[57] mappum (pseudonym). Mercury – fully trustless cryptocurrency exchange, 2015. https://bitcointalk.org/index.php?topic=946174.0.

[58] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback Protection for Trusted Execution, 2017. http://eprint.iacr.org/2017/048.

[59] Patrick McCorry, Ethan Heilman, and Andrew Miller. Atomically trading with roger: Gambling on the success of a hardfork. https://eprint.iacr.org/2017/694, 2017.

[60] Patrick McCorry, Malte Möser, Siamak Fayyaz Shahandashti, and Feng Hao. Towards bitcoin payment networks. In Joseph K. Liu and Ron Steinfeld, editors, *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part I*, volume 9722 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2016.

[61] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy - HASP '13*, pages 1–1, 2013.

[62] Robert McMillan. $1.2m hack shows why you should never store bitcoins on the internet. https://www.wired.com/2013/11/inputs/, 2013.

[63] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Bitcoin.org*, 2008.

[64] Satoshi Nakamoto. https://bitcointalk.org/index.php?topic=1786.msg22119#msg22119, 2010.

[65] Omni. http://www.omnilayer.org/.

[66] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, Joël Alwen, Georg Fuchsbauer, and Peter Gazi. Spacemint: A cryptocurrency based on proofs of space. *IACR Cryptology ePrint Archive*, 2015:528, 2015.

[67] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt*, 2017. http://eprint.iacr.org/2016/454.

[68] Rafael Pass, Elaine Shi, and Florian Tramer. Formal abstractions for attested execution secure processors. In *Eurocrypt*, 2017.

[69] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *Financial Cryptography Bitcoin Workshop*. https://blockstream.com/bitcoin17-final41.pdf.

[70] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016. https://lightning.network/lightning-network-paper.pdf.

[71] Bernardo Portela, Manuel Barbosa, Guillaume Scerri, Bogdan Warinschi, Raad Bahmani, Ferdinand Brasser, and Ahmad-Reza Sadeghi. Secure multiparty computation from sgx. In *Financial Cryptography*, 2017.

[72] Erica Portnoy and Peter Eckersley. Intel's management engine is a security hazard, and users need a way to disable it. https://www.eff.org/deeplinks/2017/05/intels-management-engine-security-hazard-and-users-need-way-disable-it.

[73] Meni Rosenfeld. Colored Coins. https://bitcoil.co.il/files/Colored%20Coins.pdf and https://bitcoil.co.il/BitcoinX.pdf, 2012.

[74] Meni Rosenfeld. Analysis of hashrate-based double spending. http://arxiv.org/abs/1402.2009, 2014.

[75] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. P2P Mixing and Unlinkable Bitcoin Transactions. In *NDSS 2017*, 2017.

[76] Fabian Schuh and Daniel Larimer. Bitshares. https://bravenewcoin.com/assets/Whitepapers/bitshares-financial-platform.pdf.

[77] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy data analytics in the cloud using sgx. In *IEEE Symposium on Security and Privacy*, 2015.

[78] ShapeShift. https://shapeshift.io/.

[79] ERC: Token standard #20. https://github.com/ethereum/EIPs/issues/20.

[80] Raoul Strackx and Frank Piessens. Ariadne: A minimal approach to state continuity. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 875–892, Austin, TX, 2016.

[81] Paul Sztorc. BitUSD isn't worth the trouble. http://www.truthcoin.info/blog/bitusd/, 2015.

[82] Peter Todd. OP_CHECKLOCKTIMEVERIFY. BIP 65, https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, 2014.

[83] Florian Tramer, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In *Euro S&P*, 2017.

[84] Muoi Tran, Loi Luu, Min Suk Kang, Iddo Bentov, and Prateek Saxena. Obscuro: A secure and anonymous bitcoin mixer using sgx, 2017. https://eprint.iacr.org/2017/974.

[85] https://bitcoin.org/en/glossary/locktime.

[86] Gavin Wood. http://gavwood.com/paper.pdf, 2014.

[87] Pieter Wuille et al. https://bitcoincore.org/en/2017/03/23/schnorr-signature-aggregation/.

[88] Pieter Wuille, Gregory Maxwell, et al. Optimized c library for ec operations on curve secp256k1, 2015. https://github.com/bitcoin-core/secp256k1.

[89] Joseph Young. China imposes new capital controls; bitcoin price optimistic, 2016. https://cointelegraph.com/news/china-imposes-new-capital-controls-bitcoin-price-optimistic.

[90] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *CCS*, 2016.

APPENDIX

A. PROOF OF SECURITY FOR ACCS

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

We use $\mathsf{TXOUT_A}, \mathsf{TXOUT_B}$ to denote the outputs of the transactions $\mathsf{TX_A}, \mathsf{TX_B}$, respectively. We denote by $\mathsf{TX^S_A}, \mathsf{TX^S_B}$ the transactions that spend $\mathsf{TXOUT_A}$ and $\mathsf{TXOUT_B}$ in steps 3 and 4 of Π_{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 $\mathsf{TX^S_A}$ and $\mathsf{TX^S_B}$. If $\mathsf{hash}(\cdot)$ 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:

- $E_1 = \{\mathsf{TX_A}$ was reversed after Bob broadcasted $\mathsf{TX_B}\}$
- $E_2 = \{\mathsf{TX_B}$ was reversed after Alice revealed $x\}$
- $E_3 = \{$Bob spent $\mathsf{TXOUT_A}$ before Alice revealed $x\}$
- $E_4 = \{$Alice spent both $\mathsf{TXOUT_A}$ and $\mathsf{TXOUT_B}$ without forging a signature$\}$
- $E_F = \{$The adversary forged a signature$\}$
- $E_A = \{\mathsf{TX^S_A}$ was confirmed by the Bitcoin network$\}$
- $E_B = \{\mathsf{TX^S_B}$ 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

$$\begin{aligned}\Pr[\neg E_0] &= \Pr[(\neg E_0 \cap E_F) \cup (\neg E_0 \cap \neg E_F)] \\ &\leq \Pr[E_F] + \Pr[\neg E_0 \cap \neg E_F]\,.\end{aligned}$$

Assume that E_F did not occur. If Alice redeems $\mathsf{TXOUT_B}$ then Bob will be able to redeem $\mathsf{TXOUT_A}$ unless either the block that contains $\mathsf{TX_A}$ was reversed on the Bitcoin blockchain (event E_1), or $\mathsf{TXOUT_A}$ 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 $\mathsf{TXOUT_A}$ then Alice will be able to redeem $\mathsf{TXOUT_B}$ unless either the block that contains $\mathsf{TX_B}$ was reversed on the Litecoin blockchain (event E_2), or $\mathsf{TXOUT_B}$ 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

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

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

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 $\mathsf{TXOUT_A}$ first, and still have enough time to redeem $\mathsf{TXOUT_B}$ 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 $\mathrm{E}[Z] = 5(t_0 + s_0)$. By using a standard tail inequality [20, 32] for the *binomial* distribution $B(\mu \cdot \mathrm{E}[Z], \frac{1}{5})$ 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 $\Pr[E_4] < e^{-\lambda^2/3} = \mathsf{negl}(\lambda)$. $\square$

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 [9], 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 [25] 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 $\mathsf{TXOUT}_\mathsf{A}$ correspond to $4s_0$ expected growth that $\mathsf{TXOUT}_\mathsf{B}$ 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 implementation of the Tesseract protocol Π_{RTExch} in Figs. 20 to 22. The full source code of our demo will be made public at a later time.

```
typedef unsigned char byte;

bool verifyMerklePath(const byte* root, const byte* leaf,
                      const byte** branch, int dirvec) {
   byte curr[SHA256_DIGEST_LENGTH];

   memcpy(curr, leaf, SHA256_DIGEST_LENGTH);
   byte_swap(curr, SHA256_DIGEST_LENGTH);

   for(int i=0; dirvec>1; ++i,dirvec>>=1) {
      if( (branch[i]).empty() ) {
         sha256double(curr, curr, curr);
         continue;
      }
      if(dirvec & 1)
         sha256double(curr, branch[i], curr);
      else
         sha256double(branch[i], curr, curr);
   }

   byte_swap(curr, SHA256_DIGEST_LENGTH);
   return memcmp(curr, root, SHA256_DIGEST_LENGTH);
}
```

Figure 20: Verify authentication path of a deposit.

```
typedef byte digest[SHA256_DIGEST_LENGTH];

void recursiveMerk(const digest* level, int size, int path) {
   int k = (size + (size & 1))/2;
   digest * next = new digest[k];

   for(int i=0; i<k; ++i){
      const byte * left_node = level[2*i];
      const byte * right_node =
         ((2*i + 1) == size ? left_node : level[2*i+1]);
      sha256double(left_node, right_node, next[i]);
      if(path == (2*i+1)) {
         cout << "L:␣";
         hexdump(left_node, SHA256_DIGEST_LENGTH);
         continue;
      }
      if(path == (2*i)) {
         cout << "R:␣";
         if(left_node != right_node)
            hexdump(right_node, SHA256_DIGEST_LENGTH);
         else
            cout << endl;
      }
   }
   if (k>1)
      recursiveMerk(next,k,path/2);
   else {
      byte_swap(next[0], SHA256_DIGEST_LENGTH);
      hexdump(next[0], SHA256_DIGEST_LENGTH);
   }
   delete[] next;
}
```

Figure 21: Build authentication path of a deposit.

```
typedef unsigned long long cointype;

time_t renew(time_t timestamp, long user_id, cointype fee) {
   if (book.find(user_id) == book.end()
      || timestamp + RENEW_PERIOD > book[user_id].timeout
      || book[user_id].left < fee) {
      return -1;
   } else {
      book[user_id].volume -= fee;
      book[user_id].left -= fee;

      timestamp = max(book[user_id].timeout,
                      timestamp + DEPOSIT_PERIOD);
      book[user_id].timeout = timestamp;
      return timestamp;
   }
}
```

Figure 22: Renewal order.