Issue, Trade, Redeem: Crossing Systems Bounds with Cryptocurrency-Backed Tokens*

Alexei Zamyatin[1,2], Dominik Harz[1], and William J. Knottenbelt[1]

[1] IC3RE, Imperial College London
[2] SBA Research

Abstract. The ecosystem of cryptocurrencies has been steadily growing since the introduction of Bitcoin, the first decentralised digital currency. While the notion of trustless asset exchange lies at the core of most blockchain-based systems, existing cross-chain communication techniques expose limitations regarding security, performance, and usability. As a result, centralised liquidity providers remain the preferred way for cross-chain transactions.
We systematise the notion of cryptocurrency-backed tokens, an approach towards trustless blockchain interoperability. We then propose a protocol for issuing, trading, and redeeming Bitcoin-backed tokens on Ethereum. Consequently, we provide an overview of system requirements, discuss open challenges regarding performance and security, and give an outlook on possible extensions. Our protocol, which requires no modifications to Bitcoin's consensus rules, can thereby be generalised to also support other cryptocurrencies.

1 Introduction

Scalability is perhaps the most commonly cited challenge of decentralised systems and has received a substantial amount of attention from the research community throughout the past years. Apart from parameter optimisations and research into novel and more scalable consensus mechanisms, there have recently been attempts to move away from the concept of isolated chains towards an ecosystem of interconnected and potentially heterogeneous systems [14,80,45]. However, this presumes efficient communication between systems that differ in purpose and structure, perhaps an even more complex problem than achieving scalability in isolated systems.

Today, there exist over 1600 different cryptocurrencies[1] of different design and purpose. Centralized exchanges have taken the leading role of trading cryptocurrencies, both against fiat and other digital assets. Hence, the primary way to obtain a trustless currency is through trusted intermediaries. While the first concept of trustless atomic swaps was introduced as early as 2013 [3], the push towards decentralised exchanges occurred only recently. Thereby, however, the focus mostly lies on facilitating trades within a single blockchain offering multiple tokens, as in the case of the 0xProtocol [1]. As such, the only way to achieve cross-chain trades today is by using atomic swaps, which are slow, interactive, and moreover, rely on censorship resistant channel detection.

Tokens in the field of cryptocurrencies refer to digital assets built on top of an existing blockchain, such as Ethereum [26]. Often, they quantify value and represent trust in a particular system or asset. As such, there exist tokens backed by "real-world assets" including fiat currency and natural resources. Thereby, in contrast to the initial idea of cryptocurrencies, the value stems from external, mostly physical, sources and requires verification by some trusted oracle. However, we can use this concept to introduce tokens issued on distributed ledgers backed by other cryptocurrencies, i.e. infer trust from an already existing decentralised asset. This in turn potentially allows executing trustless cross-chain transfers using existing decentralised token exchange protocols.

Our contribution is three-fold: We formulate three base protocols for issuing, exchanging, and redeeming of Bitcoin-backed tokens on Ethereum, leveraging on threshold signatures, cross-chain state validation and incentive mechanisms. Thereby, we require no modifications to Bitcoin or Ethereum for our base protocol. Furthermore, our scheme can be extended to other cryptocurrencies, e.g., to issue Litecoin [52] or Zcash [15] tokens on Ethereum Classic [7]. We provide an overview of system requirements and discuss challenges regarding performance and security. Finally, we outline possible extensions to our protocol improving liveness and safety properties, also discussing such that would require alterations to Bitcoin.

* This is a draft (revision July 6, 2018)
[1] As of June 2018 according to http://coinmarketcap.com.

The rest of this paper is structured as follows. In Section 2 we provide the necessary background information, while Section 3 gives an overview of related work. We discuss our system model and requirements in Section 4 and proceed to give a detailed step-by-step description of the protocol's design in Section 5. The final protocol is formulated in Section 6 and extensions are outlined in Section 7. We discuss challenges and give an outlook on future work in Section 8, while concluding our paper in Section 9.

2 Background

In the following, we provide the necessary background information on cryptocurrency tokens and cross-chain communication protocols. While we do not offer an introduction to Bitcoin and permissionless blockchains, we direct the interested reader to [59,60,78].

2.1 Overlay Protocols and Tokens

The first approach to store meta-information in cryptocurrencies was by using coloured coins [68], followed by overlay protocols, i.e. velvet forks in Bitcoin [42,82]. Such protocols can be used to add functionality to the existing chain while ideally providing the same or similar security guarantees. A further increase in protocols built on top of existing chains occurred after the introduction of Ethereum [26]. Ethereum provides a Turing-complete programming language, which allows creating so-called *smart contracts* executed within its replicated state machine, the Ethereum Virtual Machine (EVM). Specifically, new cryptocurrencies or *tokens* can be created directly within Ethereum, without the need to bootstrap a dedicated and separate blockchain.

Tokens can be used to quantify both fungible and non-fungible assets. Ethereum thereby specifies multiple standards for the simple creation of fungible (e.g. ERC20 [32], ERC223 [33]) and non-fungible tokens (e.g. ERC721 [34], ERC994 [35]). Of the top 100 cryptocurrencies according to their market capitalisation, more than 90% are rooted as tokens in the Ethereum blockchain[1].

2.2 Cross-Chain Communication

An overview of cross-chain communication can be found in a technical report by Buterin [27], which outlines three bases strategies approaches namely notary schemes, chain relays and atomic swaps via hashed time-lock contracts.

Base Techniques In the following, we provide a brief overview of the current state of research in the three base techniques used to enable cross-chain communication.

Hashed Time-Lock Contracts Hashed Time-Lock Contracts (HTLC) [4] allows payments based on a hashed input and a time constraint additional to the required signatures. The receiver must provide a pre-image to the hash function within a specified time-frame to claim the payment. If the receiver fails to make a claim, the sender of the payment can return the funds after the expiry of the time-lock. This technique is used for example by the Lightning Network and can also be applied to achieve atomic cross-chain swaps [77,3,14,40,13,76,54]. Atomic swaps allow to conduct a single token exchange atomically, i.e., either both parties receive the agreed upon tokens of value, or no trade is executed at all. A formalisation of the concept is provided in [40]. While HTCLs provide a simple mechanism to facilitate cross-chain communication, the timing constraints require both users to be online throughout the trade and expose the scheme to race conditions. Furthermore, the necessity to exchange data off-chain requires an out of band channel to be established between users in a censorship-resistant manner.

Chain Relays A chain relay can informally be defined as a program executed on one chain capable of interpreting the state of another (permissionless) chain. Thereby, similar to a SPV client [2], the chain relay can verify if transactions or blocks of the other chain have been included in the underlying data structure. For example, BTC Relay [5] allows to prove to Ethereum clients that a transaction has been included in Bitcoin. However, chain relays require a sufficient set of operations to be available on the chain hosting the program. As such, we cannot yet implement such programs using Bitcoin's limited Script. Moreover, these programs must be kept up to date with the state of the verified blockchain, i.e., users must continuously submit updates, accounting for the related computation costs. While active use and a working incentive

scheme could motivate participants to maintain a chain relay, we observe that this is currently not the case in practice: BTC Relay has fallen 25.000 blocks behind Bitcoin. Recent work is trying to cope and eliminate these shortcomings by providing more efficient proofing mechanisms, e.g., NiPoPoWs [42], or executing the verification off-chain [75,6]. Other notable chain relay projects include PeaceRelay [9], Project Alchemy [11], Cosmos "Peggy" [28] and Parity Bridge [8].

Notary Schemes Notary schemes replace trust in a single entity by trust in a set of different entities, i.e., a committee also referred to as *validators*. Validators employ a consensus algorithm such as Tendermint [25] or HoneyBadger [57] to reach agreement over a set of transactions which transfer tokens of value between chains. Safety and liveness thereby depend on the availability and honest behavior of the majority of validators (exact requirements regarding the trust models may differ). A range of PoS algorithms use such notary schemes to achieve consensus with an additional layer of incentives to promote honest behaviour [56,43,30,19]. This principle can, however, also be used for cross-chain communication. Thereby, the validators can be responsible to actively signing cross-chain transactions, as in *Liquid* [31], or attest to exchange partners that the trading conditions have been met, as in the case of *Interledger* [76]. Notary schemes, however, typically make the assumption that a pre-defined set of validators is available. Without pre-defined validators, the election of leaders can be subject to a range of attacks as described in [18].

Second-Layer Protocols Second-layer protocols for cross-chain communication aim at providing a base synchronization layer between heterogeneous blockchains, abstracting the technical details of the interconnected systems. Thereby, these approaches usually rely on existing consensus protocols and introduce incentive structures to promote honest behavior. Connections between different chains are typically established using the techniques described in the previous section and are often referred to as *bridges*. Thereby, liveness and security are to be maintained through the second layer protocol. Examples include Polkadot [80], Cosmos [45], and AION [74], which require a permissioned set-up and rely on two-phase commit protocols for cross-chain transactions.

3 Related Work

The notion of cryptocurrencies and tokens of value backed by real word assets, such as gold and fiat currency, can be described as an attempt to minimise the price fluctuations and stabilise the market. Thereby, the underlying principle is simple: each cryptocurrency unit is backed by a corresponding unit of a real-world asset, notarized and held in custody by some trusted third party. However, the centralised approach and lack of transparency of such schemes are arguably in conflict with the principles of decentralised systems and potentially allow for manipulations, as suspected in the case of Tether, a cryptocurrency pegged to the US Dollar [39].

A similar principle can, however, be utilised to create tokens backed by cryptocurrencies in a publicly verifiable manner. Instead of relying on a central party, the process of locking funds and releasing the corresponding amount of asset-backed tokens can be handled by publicly verifiable programs, or *smart contracts*, built on top of decentralised ledgers such as Ethereum. This concept is already being utilised on Ethereum to "wrap" the native cryptocurrency ETH in a standardised token format [32,67], which allows trading on decentralised on-chain token exchange platforms [1,12].

PeaceRelay represents one of the first projects discussing how cryptocurrency-backed tokens can be exchanged between two distributed ledgers supporting Turing complete programming languages, namely Ethereum and Ethereum Classic [9]. Bentov et al. describe the how Bitcoin-backed tokens can be issued on Ethereum by an exchange built on top of trusted execution environments (IntelSGX) [17]. Other projects attempt to use cryptocurrency-backed tokens for value transfers between Ethereum and permissioned systems [28,10]. However, in contrast to our scheme, the latter proposals assume Turing complete programming capabilities on both source and received chains, or rely on external validation through some trusted third party.

4 Setting and Notation

In this section, we provide an overview of the system, network and threat models for our protocol for Bitcoin-backed tokens on Ethereum and formulate the main goals of this approach. Note, while we specifically use Bitcoin and Ethereum to describe our protocol, it is generally applicable to other cryptocurrency

pairs with similar system properties, such as Litecoin [52] and Ethereum Classic [7]. The main requirement is that the chain on which the tokens are created must provide a sufficient set of operations[2].

4.1 System Model

We assume a user Alice owns units of Bitcoin, denoted as btc, linked to her public/private key pair (pk_A^{btc}, sk_A^{btc}) and wishes to create the corresponding amount of Bitcoin-backed tokens on Ethereum btc_{eth}. Bob owns units of Ethereum, denoted as eth, linked to his public/private key pair (pk_B^{eth}, sk_B^{eth}) and wishes to acquire Bitcoin, without necessarily creating a wallet, i.e., public/private key pair, on Bitcoin. Transactions created in Bitcoin are denoted as T^{btc}, while transactions on Ethereum are referred to as T^{eth}. We recall, while Bitcoin's Script provides only limited expressiveness, Ethereum offers Turing complete programming capabilities, allowing the creation of publicly verifiable programs or *smart contracts*.

As such, we assume a smart contract used to manage the issuing, trading and redeeming of the tokens is deployed on Ethereum. This contract, which we shall refer to as `treasury`, provides the following functionality:

1) **Create Tokens.** Given proof that some amount of btc has been correctly locked in Bitcoin by a user controlling an Ethereum public key pk^{eth}, the contract creates and after some pre-defined contestation period $t_{contest}$ allocates the corresponding amount of btc_{eth} to the Ethereum account associated with pk^{eth}.
2) **Cancel Creation.** Provided with a proof that a previously claimed lock of some btc is no longer part of the global state / transaction history of Bitcoin, e.g. due to a chain reorganization, within a pre-defined contestation period $t_{contest}$, the contract aborts the issuing process.
3) **Transfer Tokens.** Provided with the digital signature $sig(sk^{eth})$ of the current token owner and a receiver identified by $pk^{eth'}$, the contract reassigns the ownership of the tokens to the new Ethereum account associated with $pk^{eth'}$.
4) **Redeem Tokens.** If a user controlling units of btc_{eth} signals the wish to redeem the corresponding amount of btc on Bitcoin, the contract emits a publicly visible "unlock" event, signaling that the lock-in Bitcoin is to be lifted, and *burns* the returned tokens.

4.2 Network and Threat Model

Garay et al. formalised security properties of Bitcoin in synchronous and partially synchronous settings [37]. Within their analysis, they refer to three properties of a blockchain: common prefix, chain quality, and chain growth. Based on these properties they analysed Bitcoin as an example of a Byzantine consensus protocol.

We make some restricting assumptions with regards to the networks underlying Bitcoin and Ethereum. Where not stated otherwise, we assume the trust models of Bitcoin and Ethereum hold, i.e., the portion of the overall computing power controlled by a computationally bounded adversary is less than 50%. Moreover, we assume that the network synchronises faster than the PoW rate, and the current blocks including transactions are available to honest minders. We further assume the cryptographic primitives used in Bitcoin and Ethereum are secure. Under these assumptions, an honest majority suffices to allow for consensus under Byzantine conditions.

We assume communication between participants in both networks is asynchronous. While honest participants adhere to protocol rules, an adversary can behave arbitrarily. Thereby, we make the assumption that the adversary is economically rational and possesses bounded resources. Our protocol has to take into account that an adversary might censor transactions and/or delay delivery of such. Hence, the protocol needs to take atomicity of trades into account to prevent the partial execution of trades. As we assume that the trust model of the underlying chains holds, tampering with the smart contract issuing tokens is not possible. We have three different actors in our protocol: a sender of a token, a receiver, and an entity moderating the protocol being active on both chains. Ideally, no entity in the system should need to trust any other entity even under the assumption that they will not become malicious.

[2] As the EVM offers a Turing-complete set of operations, it is possible there. We note that Turing-completeness, however, is not necessarily a requirement, as we could also implement the system in other non-Turing-complete languages such as Scilla [71]. The instruction set currently available in Bitcoin, on the other hand, is not sufficient to issue and track new tokens. We leave a thorough analysis of the required operations to future work.

4.3 Protocol Goals

The goal of our protocol is to enable the issuing, exchange and redeeming of Bitcoin-backed tokens on Ethereum, without necessitating trust between involved parties. As such, ideally, we do not want to require the *safety* of the protocol to depend on the availability and honest behaviour of a (trusted) third party. However, if a third party is necessary, then deviations from the protocol must be penalised, while potential financial damage to users reimbursed, so as to minimise the incentive for malicious behaviour by economically rational adversaries.

The desirable properties for Bitcoin-backed tokens on Ethereum can, in turn, be formulated as follows:

- **Fungibility.** Alice must be able to lock any portion of her btc to create the corresponding amount of btc_{eth} tokens on Ethereum. In turn, she must be able to trade these tokens to Bob against eth or some other ERC20 or ERC223 token on Ethereum.
- **Divisibility.** Alice must be able to trade any fraction of btc_{eth}, as long as it exceeds the minimal possible unit in Bitcoin, i.e., a Satoshi (10^{-8} BTC). We note this requirement must only hold for the token itself - services used to execute token exchanges may impose restrictions with regards to the minimal amount or value of transferred tokens.
- **Redeemability.** Any user on Ethereum, i.e., both Alice herself or Bob, when in possession of btc_{eth} must be able to redeem the equivalent amount of btc (less potential transaction fees) on Bitcoin by destroying or *burning* the tokens on Ethereum.
- **Atomicity.** By transferring btc_{eth} Alice implicitly transfers ownership of the corresponding amount of btc to Bob, i.e., Alice is no longer able to access the locked units of btc. As such, transfer of ownership occurs atomically both on Bitcoin and Ethereum, or not at all.
- **Consistency.** At any given point in time, the existence of btc_{eth} tokens and the availability of the corresponding units of btc are mutually exclusive. That is, btc_{eth} tokens can only be generated on Ethereum if the respective amount of btc is locked in Bitcoin, while the lock can be released only if the corresponding tokens have been destroyed on Ethereum.

5 Protocol Design

We propose how Bitcoin-backed tokens can be created in Ethereum. Thereby, we define three main protocols: *Issue*, *Trade* and *Redeem*. We start by describing a naive centralised approach to describe the intuition behind the protocols and then reduce the trust requirements step by step to achieve the desirable properties described in Section 4. The final protocols are presented in Section 6. Note: while none of the mechanisms described here requires changes to the underlying base protocols, we outline more intrusive but potentially more secure and efficient approaches in Section 7.

5.1 Centralised Issuing: A Strawman Scheme

We now present a strawman scheme outlining the general idea of Bitcoin-backed tokens on Ethereum. Thereby, we make use of a central entity to process the issuing and redeeming of tokens, i.e., the locking and unlocking of btc and btc_{eth} tokens respectively. We refer to this entity as "Issuer" and assume it controls the public/private key pair (pk_I^{btc}, sk_I^{btc}) on Bitcoin and (pk_I^{eth}, sk_I^{eth}) on Ethereum. Initially, we assume the Issuer is the owner of the `treasury` contract and it is publicly verifiable that he is responsible for the issuing and redeeming of tokens.

Issue

1. Alice as the initiator of the protocol verifies the `treasury` contract is available and creates a new account on Ethereum, i.e., a public/private key pair (pk_A^{eth}, sk_A^{eth}).
2. Next, she locks her funds on Bitcoin in a publicly verifiable manner, such that this event can, in theory, be checked by any Bitcoin client. That is, in the naive centralised scenario, Alice creates a transaction T_{lock}^{btc} signed with sk_A^{btc} by which she transfers the to-be-locked btc to the Issuer. In this transaction she also includes pk_A^{eth} (or a hash-based "address" thereof) so as to inform the Issuer, where the tokens shall be issued to[3].

[3] This can be achieved by using the `OP_RETURN` opcode in Bitcoin [22], which allows to push up to 80 bytes of arbitrary data onto the Script stack.

3. Once T^{btc}_{lock} has been included in the underlying blockchain and has received sufficient confirmations (e.g. six), the Issuer, providing his digital signature $sig(sk^{eth}_I)$, instructs the `treasury` contract to issue btc_{eth} to Alice on Ethereum, such that $|btc_{eth}| = |btc|$.

Trade

1. When Alice trades some amount of btc_{eth} tokens to Bob, she appoints him as the new owner via the `treasury` contract.
2. From this moment on, the Issuer will no longer allow Alice to withdraw the associated amount of locked btc in Bitcoin. That is, the transfer of ownership occurs atomically on both chains. The process for any further transfers is analogous.

Redeem

1. Once Bob decides to redeem his btc_{eth} for the corresponding amount btc, he first creates a new public/private key pair (pk^{btc}_B, sk^{btc}_B) on Bitcoin.
2. Next, he signals to the `treasury` contract that he wishes to initiate the redeem procedure (e.g., with a function call).
3. In turn, the contract burns the btc_{eth} tokens and emits an "unlock" event verifiable by any Ethereum client.
4. The Issuer becomes aware of this and sends the corresponding amount of btc to Bob on Bitcoin by publishing a transaction T^{btc}_{redeem}.

While the naive centralised case is simple to implement and it is easy to see that the Issuer can enforce the correct behaviour of the protocol, full trust in the availability and honest behaviour of the Issuer is required. As such, neither safety nor liveness are guaranteed if an economically rational Issuer becomes malicious and decides to steal Alice's btc, generate fake btc_{eth} or not send btc to Bob despite btc_{eth} having been burnt. However, this approach is nevertheless arguably more transparent than tokens backed by real-world assets, since any user on Bitcoin and Ethereum can at least trace the actions of the Issuer. As such, the users of both systems would quickly become aware of the malicious behaviour and cease to use the protocol. While the presence of a well-defined fee model may theoretically be sufficient to incentivise honest behaviour of the Issuer, this approach is no different from relying on centralised liquidity providers, i.e., exchanges.

5.2 2-of-2 Multisig Escrows: Improving Safety

We now proceed to improve the safety properties of our scheme by making use of Bitcoin multisignatures [21], short *multisigs*, to reduce trust put in the Issuer. During *Issue*, instead of directly sending btc to the Issuer, Alice creates a transaction with a 2-of-2 multisig output in Bitcoin, which requires both her and the Issuer's digital signatures for spending. This prevents the Issuer from withdrawing the locked btc without Alice's consent (i.e., stealing), while it is easy to see Alice still cannot withdraw the funds before locking btc_{eth} in the `treasury` contract, as enforced by the Issuer. We denote the corresponding transaction as $T^{btc}_{lock(AI)}$.

This modification, however, in turn, requires a change to the *Trade* protocol: when transferring ownership of btc_{eth} to Bob, Alice must now also replace herself by Bob in the multisig lock. To this end, she creates and signs a new Bitcoin transaction $T^{btc}_{lock(BI)}$, which spends the output(s) of $T^{btc}_{lock(AI)}$ and creates a new 2-of-2 multisig output, conditioned on the existence of Bob's and the Issuer's signatures. Note: $T^{btc}_{lock(BI)}$ can only be included in the underlying blockchain after $T^{btc}_{lock(AI)}$, that it is the complete *transaction chain* must be published.

Reducing Waiting Times In a naive and slow implementation, Alice and Bob would wait for the Issuer's signature before finalising the transfer. However, since the transaction malleability fix introduced in Bitcoin with BIP141 (Segregated Witness) [53], we can get rid of this requirement. Specifically, since the signature data is no longer part of the transaction identification, we can require the Issuer to sign *both* transactions only when Bob decides to redeem his btc_{eth}, i.e., at the end of the token's lifespan after all transfers have been completed. To this end, `P2WSH` outputs [49] must be used instead of `P2SH` [23], when creating the multisig.

The improved performance, however, comes at the cost of increased storage requirements: the creation and signing of $T^{btc}_{lock(BI)}$ by Alice must be handled such that it is publicly verifiable, most importantly by the Issuer. Otherwise, if the transaction were to be given to Bob via an out of band channel, Alice could attempt to double spend, e.g., by creating a conflicting transaction $T^{btc}_{lock(CI)}$ moving funds to Carol. Neither Bob, nor Carol, nor the Issuer would know which transaction was created first, while Alice would receive funds from both trading partners. Hence, we require Alice to store $T^{btc}_{lock(BI)}$ in the `treasury` contract[4] and Bob only accepts the trade once he is sure the data is included in the Ethereum blockchain, i.e., that the Issuer can be expected to have become aware of the trade. This is essential when introducing fraud proofs and penalising of service level agreement failures (crash or Byzantine failures) of the Issuer in Section 5.4.

Privacy Privacy can be improved by requiring Alice to encrypt the uploaded transaction data with a new key k (symmetric or asymmetric, depending on further use cases) and provide k to the Issuer and Bob. The latter can be achieved by encrypting k with pk^{eth}_I and pk^{eth}_B respectively and attaching the resulting outputs to the data included in the contract.

UTXO Grouping A further disadvantage of this scheme is that in the worst case, each token trade on Ethereum results in a transaction in Bitcoin, even if published at a later point in time. An improvement can be achieved by compressing the multisig outputs into a single transaction, which is the optimal scenario will only be published at the end of the token's lifespan, i.e., when it is redeemed for units of Bitcoin.

To this end, a UTXO grouping scheme can be implemented, whereby transaction chains are replaced with a single transaction with multiple outputs. If Alice sends half of her tokens to Bob and later sends a part of her remaining tokens to Carol, this would result in two separate transactions being created in Bitcoin during the *Redeem* protocol. The resulting outputs can, however, be grouped into a single *grouping* transaction, sending the corresponding amount of Bitcoin directly to Bob and Carol (and the remainders to Alice or back to the multisig lock).

However, this would require the active participation of Alice in creating or at least signing the new transaction. The contract on Ethereum can automate the creation of the UTXO-grouping transactions. An incentive scheme can be introduced, whereby Alice receives some fees for being online and signing grouping transactions, e.g., a fraction of the fees which would be charged by miners for including the transaction chain in Bitcoin.

Optimistically, this approach allows to compress n trades into a single Bitcoin transaction. In the worst case, however, the number of Bitcoin transactions is equal to the number of token trades, i.e., worst-case complexity remains $\mathcal{O}(n)$. We note that the provided high-level description does not cover many edge cases and possible optimisations. However, while a more detailed and formal analysis goes beyond the scope of this first proposal, we shall extend upon this in future work.

Liveness and Safety While the 2-of-2 multisig prevents the escrow from stealing the user's funds in Bitcoin, it does not prevent (or even incentivise) the Issuer to abstain from signing transactions at all, locking up funds indefinitely. Furthermore, since the tokens are still issued centrally in Ethereum, there is no guarantee that the escrow will release the locked Bitcoin once the users burn Bitcoin-backed tokens in the contract during the redeem phase. In the following sections, we proceed to loosen these constraints.

5.3 Chain Relays: Non-interactive/Automated Token Issuance

Until now, the `treasury` contract only issued tokens upon presentation of the Issuer's digital signature. We now proceed to remove this requirement by adding Bitcoin transaction verification logic to the contract, that is, we make the contract capable of verifying the inclusion of transactions in the Bitcoin main chain, comparable to a Bitcoin SPV-Client [2]. We achieve this functionality by deploying a *chain relay* [27] contract for Bitcoin on Ethereum. In fact, such contract is already available in the form of BTC Relay [5]. For simplicity, in the following we assume the chain relay is directly incorporated in the `treasury` contract, extending it by the following functionality:

[4] Storage costs in Ethereum can be reduced by only persisting the transaction hash in the contract, while storing the full transaction data externally, e.g., on IPFS [16].

5) **Verify Bitcoin Transactions.** Given a chain of Bitcoin block headers starting with the genesis block, a transaction and a Merkle Tree proof [55], the contract is capable of verifying that the transaction has been included in the Bitcoin blockchain.

In the rest of this paper, we make the simplified assumption that the contract always receives and stores the most recent Bitcoin block headers. In reality, we observe BTC Relay is currently non-functional due to its high operating costs and lack of incentive for users to submit Bitcoin block header.

By enabling verification of Bitcoin transactions by the `treasury` contract, we achieve the following improvements to our scheme:

- During the *Issue* protocol, Alice can now directly prove to the contract that she has correctly locked up her funds with $T^{btc}_{lock(AI)}$. This eliminates the requirement for the Issuer to be online during the issuance process, making this part of the scheme *non-interactive*.
- Users can now prove a Byzantine failure of the Issuer to the contract, i.e. if he signed conflicting transactions redeeming the same btc. Furthermore, they can accuse the Issuer of not releasing locked btc within some grace period t_{grace} despite the corresponding btc_{eth} having been burnt. In the latter case, the Issuer can, however, object to the accusations by proving to the contract that he had published T^{btc}_{redeem} on time. We discuss the details of such accusations, fraud proofs and resulting penalties in the next sections.

5.4 Collateral: Introducing Incentives

Now that we have enabled users to prove misbehaviour of the Issuer, or accuse him thereof, we consider penalties for scenarios where the Issuer indeed fails to adhere to the protocol rules. To this end, we introduce the notion of *collateral* to our scheme.

The Issuer acts as service provider, even if by now in a non-custodial manner, and will be earning fees on token issuance/trade/redeeming. As such, we argue the Issuer can be required to lock up funds on Bitcoin and/or Ethereum as collateral, to ensure users are reimbursed in case of human-made (by the Issuer) protocol failures. We thereby identify two possible ways to implement collateral payments in our scheme:

- **Collateral on Bitcoin.** A simple approach is to require the Issuer to match the amount of btc locked by Alice in the multisig, i.e., $btc^{col} = btc$. While it does not allow for penalising malicious behavior, this measure provides a disincentive for the Issuer to permanently freeze Alice's funds, as this would prevent him from accessing his funds as well.
- **Collateral on Ethereum.** The more expressive programming capabilities of Ethereum allow defining more complex collateral schemes, than on Bitcoin. As such, the Issuer can be required to lock up sufficient collateral in eth to match the value of issued btc_{eth} tokens plus an amount to cover potential penalty payments. A significant difficulty of this approach is the often volatile exchange rate between cryptocurrencies. As such, the collateral locked in the `treasury` contract must account for potential price drops/surges.

For the first version of our scheme, we opt to introduce only collateral on Ethereum. Thereby, we assume for simplicity that the Bitcoin-Ethereum exchange rate can be retrieved from some oracle and all participants agree on it. As such, we add the following functionality to the `treasury` contract:

6) **Lock/Release Collateral.** The contract can accept collateral payments from the Issuer and will hold these funds locked until all associated btc_{eth} are redeemed.
7) **Slash Collateral.** If provided with proof that the Issuer causes financial damage to users by failing to adhere to protocol rules, the contract slashes the Issuer's collateral fund and reimburses users accordingly.

Note: while not discussed further in the protocol description, Issuers can leave the scheme by finding a replacement ready to purchase the their locked funds, and passing on the private keys. More reliable pass-over schemes will be discussed in future work.

5.5 The Tribunal: Fraud Proofs and Accusations

We now proceed to describe how our scheme handles fraud proofs and accusations by users. As mentioned, while the Issuer cannot directly steal from the multisig, he can ignore redeem requests from Bob. In the following we assume the Issuer did not send btc to Bob within grace period t_{grace} after the burning of btc_{eth}.

1. **Accusation.** As a reaction, Bob publicly accuses the Issuer via the contract. Thereby, Bob must lock some pre-defined amount of eth in the contract, which serves as collateral in case Bob made a false accusation. The provided collateral must thereby exceed the costs faced by the Issuer to disprove the accusation, to prevent *griefing*.
2. **Rebuttal Period.** The accused Issuer can react to the accusation within some rebuttal period $t_{rebuttal}$. That is, the Issuer is given a chance to prove he did transfer the correct amount of eth within the grace period.
3. **Verdict.** This accusation procedure has two possible outcomes:
   (a) *Correct Accusation*. The Issuer fails to provide the necessary proof of correct behaviour within $t_{rebuttal}$. As a result, the Issuer is penalised and the user reimbursed for the incurred financial loss. Alternatively, Bob can decide to retry the redeem process, without incurring penalties yet.
   (b) *False Accusation*. The Issuer provides a transaction inclusion proof to the `treasury` contract using the chain relay functionality and shows that Bob has made a false accusation. As a result, Bob's collateral is confiscated and paid to the Issuer as reimbursement for the costs of the provided proof.

6 Detailed Protocol Description

In this section we now provide a detailed formulation of the three discussed protocols *Issue*, *Trade* and *Redeem*. Accompanying visualizations are given in Figures 1 - 3. Note: for simplification, we do not consider the fees incurred by the Issuer.

6.1 Issue

Protocol 1 Issue

Precondition. Alice controls units of btc on Bitcoin: $btc \rightarrow pk_A^{btc}$

The protocol:

(1) Alice generates (pk_A^{eth}, sk_A^{eth})
(2) Alice publishes $T_{lock(AI)}^{btc}$: $pk_A^{btc} \xrightarrow{btc} \langle pk_A^{btc} \wedge pk_I^{btc} \rangle$ | cond.: $\exists sig(sk_A^{btc} \wedge sk_I^{btc})$
(3) Alice publishes T_{proof}^{eth}, calling $\mathsf{verifyBtcTx}(T_{lock(AI)}^{btc})$ in `treasury`
(4) IF $\mathsf{verifyBtcTx}(T_{lock(AI)}^{btc}) = \top$
   (a) `treasury` contract waits until $t_{current} \geq t_{contest}$
   (b) The contract declares $btc_{eth} \rightarrow pk_A^{eth}$
(4') ELSE: `treasury` contract aborts the process

Result. Alice and the Issuer have locked btc in a multisig output: $btc \rightarrow \langle pk_A^{btc} \wedge pk_I^{btc} \rangle$, Alice controls units of btc_{eth} on Ethereum: $btc_{eth} \rightarrow pk_A^{eth}$

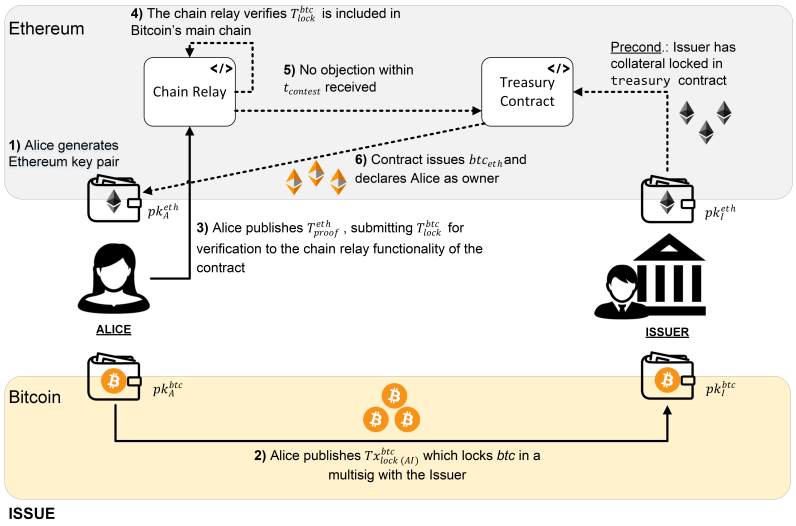

Fig. 1. Visualization of the *Issue* protocol. The chain relay and the rest of the contract functions are separated for better readability.

6.2 Trade

Protocol 2 Trade

Precondition. $btc \rightarrow \langle pk^{btc}_A \wedge pk^{btc}_I \rangle$, $btc_{eth} \rightarrow pk^{eth}_A$, $eth \rightarrow pk^{eth}_B$

The protocol:

(1) Alice creates and signs $T^{btc}_{lock(BI)}$: $\langle pk^{btc}_A \wedge pk^{btc}_I \rangle \xrightarrow{btc} \langle pk^{btc}_B \wedge pk^{btc}_I \rangle$ | cond.: $\exists sig(sk^{btc}_B \wedge sk^{btc}_I)$
(2) Alice publishes T^{eth}_{offer}, calling transferTokens(pk^{eth}_B, $T^{btc}_{lock(BI)}$) in `treasury`
(3) Bob publishes T^{eth}_{trade}: $pk^{eth}_B \xrightarrow{eth}$ `treasury`
(4) `treasury` contract declares $pk^{eth}_A \xrightarrow{btc_{eth}} pk^{eth}_B$
(5) Alice publishes $T^{eth}_{withdraw}$: `treasury` $\xrightarrow{eth} pk^{eth}_A$

Result. $btc \rightarrow \langle pk^{btc}_B \wedge pk^{btc}_I \rangle$, $btc_{eth} \rightarrow pk^{eth}_B$, $eth \rightarrow pk^{eth}_A$

6.3 Redeem

Protocol 3 Redeem

Precondition. $btc \rightarrow \langle pk^{btc}_B \wedge pk^{btc}_I \rangle$, $btc_{eth} \rightarrow pk^{eth}_B$

The protocol:

(1) Bob generates (pk^{btc}_B, sk^{btc}_B)
(2) Bob creates and signs $T^{btc}_{redeem(BI)}$
(3) Bob publishes T^{eth}_{burn}, calling redeemTokens($T^{btc}_{redeem(BI)}$, btc_{eth}) in `treasury`
(4) `treasury` contract burns tokens: $pk^{eth}_B \xrightarrow{btc_{eth}} \mathsf{X}$
(5) `treasury` contract emits Event("unlock btc to pk^{btc}_B")
(6) Before $t_{current} \geq t_{grace}$ the Issuer:
  (a) Signs $T^{btc}_{lock(BI)}$
  (b) Signs $T^{btc}_{redeem(BI)}$
  (c) Publishes $T^{btc}_{lock(BI)} \xleftarrow{\text{spends}} T^{btc}_{redeem(BI)}$

Result. $btc \rightarrow pk^{btc}_B$, $btc_{eth} \rightarrow \mathsf{X}$

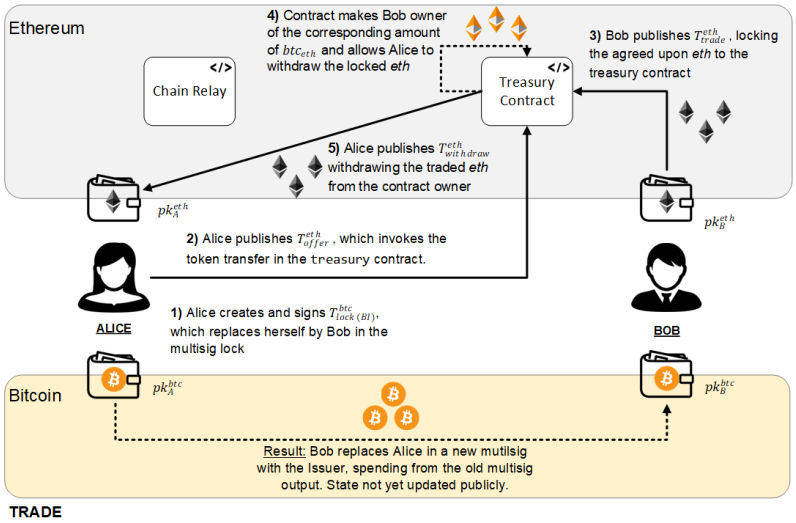

Fig. 2. Visualization of the *Trade* protocol. The chain relay and the rest of the contract functions are separated for better readability.

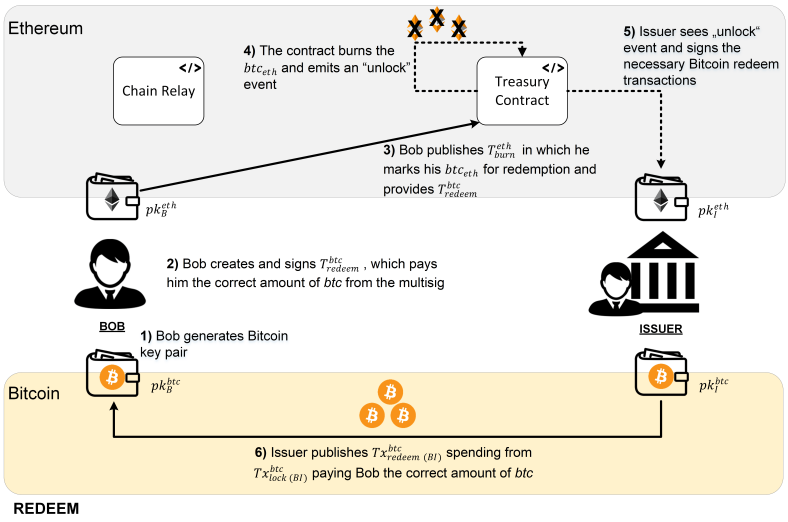

Fig. 3. Visualization of the *Redeem* protocol. The chain relay and the rest of the contract functions are separated for better readability.

7 Extensions

We discuss possible extensions to improve further the liveness and safety properties of the scheme, which either require modifications to Bitcoin or rely on external resources, i.e. trusted hardware.

7.1 Issuer Committee: Disseminating Availability (and Trust)

A final improvement to the scheme can be made by introducing issuer committees, instead of relying on a single entity. The assumption here is simple: the probability of a single issuer facing a crash failure is lower than n issuers crashing simultaneously for $n > 1$. Furthermore, if the multisig were to require the signatures of $\frac{n}{2}$ or $\frac{2n}{3}$ issuers, the safety properties can be improved as well.

Committee election schemes have been described for PoS protocols [56,43,30,19]; however it is also possible to facilitate election protocols on top of PoW blockchains. Concepts including Byzcoin [44] and HybridConsensus [64] describe how a committee can be sampled from PoW solutions over some period. Thereby, the committee leverages on the same trust model, as the underlying consensus mechanism. How-

ever, Pass and Shi emphasise the need for selfish mining attack prevention [64], as otherwise, the committee constellation could be biased to benefit an attacker. To this end, it is possible to utilise schemes like Fruitchains [63], Bobtail [20] and potentially the DAG-based proposal PHANTOM [73].

However, some of these protocols require substantial modifications to Bitcoin can only be deployed as soft or hard forks. Furthermore, multisig outputs in Bitcoin currently can only include up to 20 signatures. While this constraint could potentially be resolved by the concept of Merkelized Abstract Syntax Trees (MAST) [69,46,36], it is unclear if and when this mechanism will be deployed in Bitcoin.

Velvet Committee Election In theory, if the committee participation were to be made optional, it could be implemented as a velvet fork in Bitcoin, avoiding the risk of chain splits. However, as the rules in such case are non-enforceable for the majority of the consensus participants, we can no longer assume the same trust model as Bitcoin. That is, if only 10% of consensus participants employ a committee election scheme via PoW, we cannot be sure that a single entity is not controlling the committee. Hence, while both liveness and safety are only improved *optimistically* for voluntary committee schemes. With adoption rates $> 50\%$, however, the improvements are notable. A more detailed analysis of possible committee-based protocols will be studied in future work.

Merge-Mined Election The `treasury` contract on Ethereum must be informed about the committee elected in Bitcoin. This can be achieved by submitting the necessary data to the chain relay and mirroring the committee election algorithm to the smart contract.

An alternative solution has been discussed in Rootstock [48], a sidechain proposal for Bitcoin equipped with a version of the Ethereum Virtual Machine. Rootstock is merge-mined with Bitcoin, that is, it uses the same PoW algorithm and accepts PoW solution found by Bitcoin miners. Thereby, solutions are only accepted if they exceed Rootstock's required difficulty target and contain a reference to the current Rootstock block header. Merged mining can, however, also be used to make clients on Rootstock aware of the committee election in Bitcoin [66,65]. As such, by natively supporting the propagation of the committee election process, a committee issuer scheme could potentially be implemented with fewer costs compared to Ethereum. Furthermore, while classical merged mining only verifies the PoW of Bitcoin, it can be extended to allow transaction verification. That is, the chain relay functionality can also be implemented directly in the Rootstock client, reducing the costs associated with the chain relay functionality in our protocol. Once Bitcoin-backed tokens are issued on Rootstock, they are transferred to Ethereum using approaches described by, for example, PeaceRelay [9]. We note, however, the deployment of Rootstock requires a soft fork in Bitcoin.

7.2 Trusted Hardware

An approach which would significantly simplify our scheme for Bitcoin-backed tokens on Ethereum is the use of trusted execution environments (TEEs). The idea of this concept is that a secure hardware enclave can attest it executed precisely the code which was deployed by a user - a mechanism referred to as *remote attestation*. In theory, users can hence trust TEEs, even if they are located outside of their reach, e.g., in a data centre rather than on their local machine. The most popular TEE implementation today is arguable Intel SGX [41].

Existing research work already describes how Intel SGX can act as a "semi-trusted" third party in blockchain protocols, including payment channels [50,51], useful mining [83] and cryptocurrency exchanges [17]. The latter research work by Bentov et al. is specifically relevant for our scheme, as it mentions how cryptocurrency-backed could be implemented using Intel SGX. Applied to our protocol, the Issuer would be replaced by an Intel SGX enclave, which could then, in theory, be trusted. We note, however, that recent attacks [81,72,29,79,38,70,24,47,58] have shown that more research into the security of trusted hardware may be necessary before it becomes suitable for large-scale deployment in cryptocurrencies.

8 Discussion and Future Work

We now proceed to discuss challenges faced by our approach and perspectives for future work.

8.1 Incentives

Enabling liveness of the protocol requires finding Issuers willing to facilitate the protocol. These parties must have both Bitcoin and Ethereum accounts with sufficient funds to provide the potentially necessary collateral. Hence, the incentives provided to Issuers must outweigh any cost occurring. At the same time, the fees for the protocol must be reasonable otherwise users would not execute such a service. One of the main challenges of our approach is therefore handling a potentially unstable exchange rate between Bitcoin and Ethereum, which could in extreme cases could shift the incentives of economically rational Issuers towards acting maliciously.

Formally, the protocols incentive structure can be described as a mechanism $M = (\Sigma, g)$ [61]. Σ is a set of possible actions of agents and g the resulting function of those actions. Actors in the system are *rational* agents $P = \{1..n\}$ that want to optimize their utility u_i. Sender and receiver wish to exchange currency while likely trying to minimise cost under specific requirements of transaction times and security assumptions. Issuers, on the other hand, wish to optimise their income by facilitating the trades. The idea of the mechanism is to be incentive aligned [62]. The dominant strategy of each agent results in g, which is ideally close or precisely the desired function f, the *social welfare* of the protocol. We leave a thorough analysis of the mechanism and especially an incentive alignment for future work.

8.2 Isolation of Failures

Bitcoin Failure Assuming Bitcoin or any other source chain suffers from a liveness failure, issuing or redeeming tokens will come to a stop. Hence, the tokens currently in circulation on any connected receiver chain might become "untrusted". If the liveness failure is only of short duration, issuing and redeeming can continue as before without affecting any tokens. However, if liveness failures persist, receiving parties might stop accepting such backed tokens.

Should the trust model of Bitcoin be corrupted, i.e., if a single entity controls more that 50% of the overall hash rate, fake cryptocurrency backed tokens could be generated by an adversary. While there is no real defense against a such attack, we believe this would be visible in Ethereum and users would cease to use the token issuing and trading schemes.

Ethereum Failure With our proposed protocol it is possible to lock btc and issue the corresponding amount of btc_{eth} tokens. However, we depend on the assumption that Ethereum or any other receiver chain is secure. When trading the tokens back, malicious parties might be able to receive locked initially btc that were not intended to them. Assuming tokens are issued on a chain where, for example, majority attacks are practical, malicious parties could insert heir public keys as receiver of btc_{erc}. They might even duplicate btc_{erc} and claim back the original btc. This would result in a hard-fork since other nodes would not validate the blocks as correct. We argue that this does not impact the security of Bitcoin.

However, one could imagine an issue with "trust" in cryptocurrency-backed tokens in cases where consensus protocol of a receiver chain breaks or the coin issuing contract is implemented defectively (i.e. does not enforce the mapping btc). Assume an attacker can redeem the initially intended coins, while still having coins that seem valid (i.e. backed by btc) on the receiver chain. In this case, the attacker could use the btc_{erc} without other parties noticing that failed backing.

The current version of the protocol proposes to store collateral only in the receiver chain due to limited scripting possibilities in Bitcoin. However, this relies on a partially stable exchange rate between the source an receiver chain. We leave the design of a collateral scheme on Bitcoin as future work.

8.3 High Cost of Chain relays

Current chain relays like BTC Relay are not actively used due to high fees for storing and validating block headers and transaction inclusion proofs. However, new proposals leveraging concepts like Non-Interactive Proofs of Proof of Works (NiPoPoWs) [42], capable of reducing validation costs, are actively being developed. There also exist alternatives to using a chain relay. TrueBit [75] offers relatively cheap execution of arbitrary computations in eWASM [75]. This enables the building of bridges between Ethereum and cryptocurrencies, where the validation of proof-of-work and consensus rules is infeasible on-chain, e.g., Litecoin or Dogecoin. We leave the integration of such proposals to future work.

8.4 Outlook

The concept could further be applied to permissioned chains. Permissioned chains would not necessarily need to issue their tokens but could use established currencies such as Bitcoin to conduct trades on its platform. This could be a reasonable trade-off for user acceptance: while permissioned chains can allow higher transaction throughput and faster confirmation rates, users have to trust the native currency. If a decentralised currency backs the native currency, users can leave the native chain without worrying about dealing with yet another native currency.

The proposed concept, however, is for now only applicable to currencies. As part of future work, state synchronization of smart contracts is still left open. This would allow deployment of a smart contract for example in RSK and Ethereum with a consistent state.

9 Conclusion

We presented a scheme towards Bitcoin-backed tokens in Ethereum, consisting of three base protocols for issuing, trading and redeeming tokens. The protocols leverage on a 2-of-2 multisig escrow scheme, chain relays, and incentive mechanisms. Our scheme can be easily generalised to allow tokens backed by other cryptocurrencies and requires no changes to the underlying consensus rules. In fact, cryptocurrencies such as Litecoin, which in many aspects resemble the design of Bitcoin, are already supported. We discussed system requirements and open challenges regarding security and performance. We then elaborated on extensions to improve the liveness and safety features of our protocol, including issuer committees, merged mining, and trusted hardware.

References

1. 0xproject whitepaper. https://0xproject.com/pdfs/0x_white_paper.pdf. Accessed: 2018-05-23.
2. Bitcoin Developer Guide: Simplified Payment Verification (SPV). https://bitcoin.org/en/developer-guide#simplified-payment-verification-spv. Accessed: 2018-05-16.
3. Bitcoin Wiki: Atomic cross-chain trading. https://en.bitcoin.it/wiki/Atomic_cross-chain_trading. Accessed: 2018-05-16.
4. Bitcoin Wiki: Hashed Time-Lock Contracts. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts. Accessed: 2018-05-16.
5. Btc relay. https://github.com/ethereum/btcrelay. Accessed 2018-04-17.
6. Dogethereum. https://github.com/dogethereum/dogerelay. Accessed 2018-04-17.
7. Ethereum Classic. https://github.com/ethereumproject. Accessed: 2018-05-23.
8. Parity-Bridge. https://github.com/paritytech/parity-bridge. Accessed: 2018-05-21.
9. Peace relay. https://github.com/loiluu/peacerelay. Accessed 2018-04-17.
10. Poa bridge. https://github.com/poanetwork/poa-bridge. Accessed: 2018-05-23.
11. Project alchemy. https://github.com/ConsenSys/Project-Alchemy. Accessed 2018-04-17.
12. Radar relay. https://radarrelay.com/. Accessed: 2018-05-23.
13. BIP199: Hashed Time-Locked Contract transactions. https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki, 2017. Accessed: 2018-05-16.
14. A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille. Enabling blockchain innovations with pegged sidechains. https://blockstream.com/sidechains.pdf, 2014. Accessed: 2016-07-05.
15. E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.
16. J. Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.
17. I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, and A. Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. Cryptology ePrint Archive, Report 2017/1153, 2017. Accessed:2017-12-04.
18. I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y. *ACM SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.
19. I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. https://eprint.iacr.org/2016/919.pdf, 2016. Accessed: 2016-11-08.
20. G. Bissias and B. N. Levine. Bobtail: A proof-of-work target that minimizes blockchain mining variance. arXiv:1709.08750, 2017. Accessed:2017-11-10.
21. Bitcoin community. Multisignature. https://en.bitcoin.it/wiki/Multisignature. Accessed: 2018-05-23.

22. Bitcoin community. OP_RETURN. https://en.bitcoin.it/wiki/OP_RETURN. Accessed: 2018-05-23.
23. Bitcoin community. Pay to script hash. https://en.bitcoin.it/wiki/Pay_to_script_hash. Accessed: 2018-05-23.
24. F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi. Software grand exposure: Sgx cache attacks are practical. *arXiv preprint arXiv:1702.07521*, page 33, 2017.
25. E. Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. http://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf, Jun 2016. Accessed: 2017-02-06.
26. V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2014. Accessed: 2016-08-22.
27. V. Buterin. Chain interoperability. https://static1.squarespace.com/static/55f73743e4b051cfcc0b02cf/t/5886800ecd0f68de303349b1/1485209617040/Chain+Interoperability.pdfi, 2016. Accessed: 2017-03-25.
28. Cosmos Developer Team. Peggy. https://github.com/cosmos/peggy. Accessed: 2018-05-23.
29. V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.
30. B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573, 2017. Accessed: 2017-06-29.
31. J. Dilley, A. Poelstra, J. Wilkins, M. Piekarska, B. Gorlick, and M. Friedenbach. Strong federations: An interoperable blockchain solution to centralized third party risks. *arXiv preprint arXiv:1612.05491*, 2016.
32. Ethereum community. Erc20: Token standard. https://github.com/ethereum/EIPs/issues/20. Accessed 2018-06-27.
33. Ethereum community. Erc223: Token standard. https://github.com/ethereum/EIPs/issues/223. Accessed 2018-06-27.
34. Ethereum community. Erc721: Non-fungible token standard. https://github.com/namecoin/namecoin. Accessed 2018-06-27.
35. Ethereum community. Erc994: Delegated non-fungible token standard. https://github.com/ethereum/EIPs/issues/994. Accessed 2018-06-27.
36. M. Friedenbach, K. Alm, and B. (Pseudonym).
37. J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015*, pages 281–310. Springer, 2015.
38. J. Götzfried, M. Eckert, S. Schinzel, and T. Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, page 2. ACM, 2017.
39. J. M. Griffin and A. Shams. Is bitcoin really un-tethered? 2018.
40. M. Herlihy. Atomic cross-chain swaps. arXiv:1801.09515, 2018. Accessed:2018-01-31.
41. Intel. Intel software guard extensions (intel sgx) sdk. https://software.intel.com/sgx-sdk. Accessed: 2018-05-23.
42. A. Kiayias, A. Miller, and D. Zindros. Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963, 2017. Accessed:2017-10-03.
43. A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. https://pdfs.semanticscholar.org/1c14/549f7ba7d6a000d79a7d12255eb11113e6fa.pdf, 2016. Accessed: 2017-02-20.
44. E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, Aug. 2016. USENIX Association.
45. J. Kwon and E. Buchman. Cosmos: A network of distributed ledgers. https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md, 2015.
46. J. Lau.
47. S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. Inferring fine-grained control flow inside sgx enclaves with branch shadowing. In *26th USENIX Security Symposium, USENIX Security*, pages 16–18, 2017.
48. S. D. Lerner. Rootstock: Bitcoin powered smart contracts. https://www.rsk.co/, 2015.
49. Libbitcoin developers. P2WSH Transactions. https://github.com/libbitcoin/libbitcoin/wiki/P2WSH-Transactions. Accessed: 2018-05-23.
50. J. Lind, I. Eyal, F. Kelbert, O. Naor, P. Pietzuch, and E. G. Sirer. Teechain: Scalable blockchain payments using trusted execution environments. *arXiv preprint arXiv:1707.05454*, 2017.
51. J. Lind, I. Eyal, P. R. Pietzuch, and E. G. Sirer. Teechan: Payment channels using trusted execution environments. https://arxiv.org/abs/1612.07766, 2016. Accessed: 2017-03-09.
52. Litecoin community. Litecoin reference implementation. github.com/litecoin-project/litecoin. Accessed: 2017-06-30.
53. E. Lombrozo, J. Lau, and P. Wuille.
54. P. McCorry, E. Heilman, and A. Miller. Atomically trading with roger: Gambling on the success of a hardfork. In *CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology*, Sep 2017.
55. R. C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.
56. S. Micali. Algorand: The efficient and democratic ledger. http://arxiv.org/abs/1607.01341, 2016. Accessed: 2017-02-09.
57. A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.

58. A. Moghimi, G. Irazoqui, and T. Eisenbarth. Cachezoom: How sgx amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 69–90. Springer, 2017.
59. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, Dec 2008. Accessed: 2015-07-01.
60. A. Narayanan and J. Clark. Bitcoin's academic pedigree. volume 15, pages 20:20–20:49, New York, NY, USA, aug 2017. ACM.
61. N. Nisan and A. Ronen. Algorithmic Mechanism Design. *Games and Economic Behavior*, 35(1-2):166–196, apr 2001.
62. N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*, volume 1. Cambridge University Press, Cambridge, jul 2007.
63. R. Pass and E. Shi. Fruitchains: A fair blockchain. http://eprint.iacr.org/2016/916.pdf, 2016. Accessed: 2016-11-08.
64. R. Pass and E. Shi. Hybrid consensus: Scalable permissionless consensus. https://eprint.iacr.org/2016/917.pdf, Sep 2016. Accessed: 2016-10-17.
65. C. Paul Sztorc. Drivechains BIP1: hashrate-escrow. https://github.com/drivechain-project/docs/blob/master/bip1-hashrate-escrow.md. Accessed: 2018-05-21.
66. C. S. Paul Sztorc, CryptAxe (Pseudonym). Drivechains BIP2: blind-merged-mining. https://github.com/drivechain-project/docs/blob/master/bip1-hashrate-escrow.md. Accessed: 2018-05-21.
67. Radar Relay Inc. W-eth: Wrapped eth. https://weth.io/. Accessed: 2018-05-23.
68. M. Rosenfeld. Overview of colored coins. https://bitcoil.co.il/BitcoinX.pdf, 2012. Accessed: 2016-03-09.
69. J. Rubin, M. Naik, and N. Subramanian. Merkelized abstract syntax trees. http://www.mit.edu/~jlrubin/public/pdfs/858report.pdf, 2014.
70. M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. Malware guard extension: Using sgx to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24. Springer, 2017.
71. I. Sergey, A. Kumar, and A. Hobor. Scilla: a smart contract intermediate-level language. arXiv:1801.00687, 2018. Accessed:2018-01-08.
72. S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena. Preventing your faults from telling your secrets: Defenses against pigeonhole attacks. *arXiv preprint arXiv:1506.04832*, 2015.
73. Y. Sompolinsky and A. Zohar. Phantom: A scalable blockdag protocol. Cryptology ePrint Archive, Report 2018/104, 2018. Accessed:2018-01-31.
74. M. Spoke and Nuco Engineering Team.
75. J. Teutsch and C. Reitwießner. A scalable verification solution for blockchains. https://truebit.io/, March 2017. Accessed:2017-10-06.
76. S. Thomas and E. Schwartz. A protocol for interledger payments. *URL https://interledger.org/interledger.pdf*, 2015.
77. TierNolan. Atomic swaps using cur and choose. https://bitcointalk.org/index.php?topic=1364951, 2016. Accessed: 2018-05-16.
78. F. Tschorsch and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials*, 18(3):2084–2123, 2016.
79. N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza. Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves. In *European Symposium on Research in Computer Security*, pages 440–457. Springer, 2016.
80. G. Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2015.
81. Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 640–656. IEEE, 2015.
82. A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, and W. J. Knottebelt. (Short Paper) A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer*, 2018.
83. F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. van Renesse. Rem: Resource-efficient mining for blockchains. http://eprint.iacr.org/2017/179, 2017. Accessed: 2017-03-24.