XCLAIM: Interoperability with Cryptocurrency-Backed Tokens[⋆]

Alexei Zamyatin[*†], Dominik Harz[*], Joshua Lind[*], Panayiotis Panayiotou[*], Arthur Gervais[*], William J. Knottenbelt[*]
[*] Department of Computing, Imperial College London, United Kingdom
[†] SBA Research, Austria

***Abstract*—Building trustless cross-blockchain trading protocols is challenging. Therefore, centralized liquidity providers remain the preferred route to execute transfers across chains — which fundamentally contradicts the purpose of permissionless ledgers to replace trusted intermediaries. Enabling cross-blockchain trades could not only enable currently competing blockchain projects to better collaborate, but seems of particular importance to decentralized exchanges as those are currently limited to the trade of digital assets within their respective blockchain ecosystem.**

In this paper we systematize the notion of cryptocurrency-backed tokens, an approach towards trustless cross-chain communication. We propose XCLAIM, a protocol for issuing, trading, and redeeming e.g. Bitcoin-backed tokens on Ethereum. We provide implementations for three possible protocol versions and evaluate their security and on-chain costs. With XCLAIM, it costs at most USD 1.17 to issue an arbitrary amount of Bitcoin-backed tokens on Ethereum, given current blockchain transaction fees. Our protocol requires no modifications to Bitcoin's and Ethereum's consensus rules and is general enough to support other cryptocurrencies.

I. INTRODUCTION

Blockchain protocols can be thought of conceptually as isolated databases, with no dedicated input or output operations. As such, achieving interoperability between blockchains is challenging. Enabling seamless cross-blockchain trades and interaction could thus potentially pave the way for more interoperability and synergies among currently competing blockchain projects.

While the promise of removing trusted intermediaries is an appealing thought, we observe that in practice, many supposedly decentralized trading services rely on traditional custodian architectures. The increasing number of security breaches in centralized exchanges [34], [36], [79], [85] have triggered the development of *decentralized* exchange protocols [1], [2], [14], [17], [19]. These protocols are however limited to the trustless exchange of *cryptocurrency-tokens*, built *within* a specific blockchain infrastructure [46]. As such, they do not offer trades across blockchains (*cross-chain*). Due to the complexities of constructing trustless cross-chain trading protocols, existing techniques are therefore dominated by centralized liquidity providers – contradicting the very purpose of replacing trusted third parties with a decentralized ledger.

Currently, atomic swaps based on time locks and hash preimage revealing techniques [4], [5], [35], [61], [94] are one possible trustless mechanism to perform cross-chain trades. Atomic cross-chain swaps, however, are interactive, require all involved parties to be online, and moreover, rely on the existence of an order matching mechanism and an off-chain channel to exchange metadata [38], [93].

In this paper we present XCLAIM, a protocol for cross-chain cryptocurrency-backed tokens. XCLAIM allows to create tokens on a cryptocurrency A (e.g. Ethereum) backed by units of another cryptocurrency B (e.g. Bitcoin). To achieve this, XCLAIM leverages collateralization and cross-chain state verification to guarantee these tokens can be redeemed for their monetary value. The cryptocurrency-backed tokens issued in our protocol are tradeable via nowadays decentralized exchange protocols, thus enabling trustless cross-chain communication.

Contributions: As a summary, our contributions are as follows:

- To the best of our knowledge, we are the first to present a protocol for issuing, trading, and redeeming cryptocurrency-backed tokens, without necessitating full trust in a centralized entity. Our protocol thereby requires no modifications to the underlying cryptocurrencies. We develop Bitcoin-backed tokens on Ethereum as an example use case, however, our scheme is general enough to be applied to other cryptocurrency pairs, to e.g. issue Litecoin [73] or Zcash [37] tokens on Ethereum Classic [15].
- We present an implementation of our protocol for Bitcoin-backed tokens on Ethereum and evaluate its performance. In our (trustless) prototype, it costs USD 0.67-1.17 to issue, USD 0.41 to trade and USD 0.72 to redeem an arbitrary amount of tokens (given the current blockchain transaction fees)[1]. We further derive two cost optimizations, reducing incurred costs by up to 90%, by making a trade off between performance and security, through the use of trusted hardware.
- We systematize the requirements to the underlying blockchains/cryptocurrencies and analyse the secu-

[⋆] This is a draft (revision 7 August 2018)

[1] According to exchange rates as of 7 August 2018.

rity challenges for cryptocurrency-backed tokens, discussing possible mitigations to preserve security and privacy of our protocol.

Outline: The paper is structured as follows. We provide the necessary background in Section II. We discuss our system model and requirements in Section III and proceed to give a detailed step-by-step description of the protocol design in Section IV. We formulate XCLAIM protocol in Section V and discuss system requirements. In Section VI we present our implementations and provide evaluations of execution and on-chain storage costs. In Section VII we outline security challenges faced and discuss possible mitigations. Section VIII gives an overview of related work. Finally, we conclude the paper in Section IX.

II. Background

We first provide background information on cryptocurrencies and blockchains. Next, we provide an overview of cryptocurrency tokens and cross-chain communication protocols. Finally, we provide a summary of trusted execution environments.

A. *Cryptocurrencies and Blockchains*

Bitcoin [80] is a digital cryptocurrency that allows users to hold and exchange funds in a decentralized manner. In comparison to previous works, Bitcoin was the first digital cryptocurrency to operate securely without a central trusted entity. To achieve such security properties, Bitcoin executes a peer-to-peer replicated state machine that maintains a global append-only ledger. The ledger contains the entire history of all transactions in the network, and is constructed through a sequence of blocks, with each block storing one or more transactions. The blocks in the ledger are chained together using a cryptographic hash function; thus forming a chain of blocks, or *blockchain*.

Every node in the Bitcoin network keeps and maintains a replicated copy of the blockchain. The generation and chaining of new blocks to the blockchain is associated with a cryptographic puzzle, and the nodes in the network compete for block generation. This approach makes it difficult for attackers to modify the blockchain and any transactions already in the blockchain, but easy for nodes to verify the validity of existing transactions. At any time, there may be uncertainty about the current state of the blockchain, i.e. multiple valid new blocks may exist, termed a blockchain *fork*. Due to this, transactions are only guaranteed with a finite probability to be placed on the blockchain, and thus users must wait for a sufficient number of new blocks (confirmations) to be generated before they may consider their transactions final.

B. *Cryptocurrency Tokens*

To extend the functionality offered by cryptocurrencies such as Bitcoin, overlay protocols have been proposed. These protocols sit on top of an underlying blockchain and provide additional functionality to the blockchain while maintaining similar security guarantees. The first approach to creating an overlay protocol was coloured coins [87] which stored additional meta-information in the Bitcoin blockchain to create tradeable Bitcoins with associated attributes. This was followed by other works, such as [8], [23], which created more complex and feature-rich overlay protocols for Bitcoin. These types of overlay protocols fall into the category of *velvet forks* of Bitcoin [67], [100].

Subsequent overlay protocols have since been proposed that create completely new cryptocurrencies of their own, termed *cryptocurrency tokens*, built on top of existing blockchains that offer Turing-complete programming languages, such as Ethereum [52]. Using the expressibility of these blockchains, *smart contracts* can be programmed to execute these protocols directly within the blockchain, without the need to bootstrap a dedicated and separate blockchain. Tokens built using this approach can be used to quantify both fungible and non-fungible assets. Ethereum, for example, specifies standards for the simple creation of fungible (e.g. ERC20 [10], ERC223 [11]) and non-fungible tokens (e.g. ERC721 [12], ERC994 [13]).

C. *Cross-Chain Communication*

There are thousands of different cryptocurrencies and altcoins in existence, each with their own properties and security guarantees. Despite a large and growing ecosystem, it is not currently possible to exchange or trade cryptocurrencies in a trustless manner using their native protocols. Similarly, direct communication between two separate cryptocurrencies and blockchains is also not possible using their native protocols.

As such, additional protocols have been proposed, built on top of existing blockchains, to enable direct bridges between cryptocurrencies. One such approach is chain relays; programs executed on one blockchain capable of interpreting the state of another. A chain relay can verify if transactions or blocks of one blockchain have been included in the underlying data structure. For example, BTCRelay [6] which creates a bridge between Bitcoin and Ethereum [52]. Other approaches to cross-chain communication include atomic swaps via hash time-locked contracts (HTLCs) allowing to perform a single cryptocurrency exchange atomically, i.e., either both parties receive the agreed upon cryptocurrencies, or no trade is executed at all.

Moreover, *Layer-2* systems propose cross-chain communication protocols to provide a base synchronization layer between heterogeneous blockchains, abstracting the underlying details of the interconnected systems. These approaches usually rely on existing consensus protocols, require a permissioned setup and introduce incentive structures to promote honest behaviour. Examples include Polkadot [97], Cosmos [70], AION [91] and COMIT [62].

D. *Trusted Execution Environments*

Recent commodity CPUs offer hardware support for *trusted execution environments* (TEEs). TEEs provide a hardware root-of-trust, offering confidentiality and integrity guarantees to code and data in an untrusted system [33], [63]. Even when the hardware and all privileged software (e.g. OS, hypervisor and BIOS), are controlled by an untrusted entity, confidentiality and integrity are maintained as long as the physical CPU is not breached. This allows new software deployment models: by only trusting the CPU, software can be securely deployed in an otherwise untrusted system. Existing TEE implementations include Intel SGX [63], ARM TrustZone [33], and AMD SEV [66].

TEEs divide computing resources into two distinct environments: a *trusted* environment and an *untrusted* environment. Trusted code and memory is cryptographically secured, while the CPU manages the isolation between the two environments, ensuring that only trusted code accesses trusted memory. The trusted environment has no direct I/O capabilities, but instead communicates with the untrusted environment; a dedicated interface allows trusted code to interact with untrusted code.

In addition, TEEs typically support remote attestation [64], [65], allowing parties to verify the authenticity of program instances in a remote deployment. More specifically, remote attestation provides the ability to ascertain that a certain piece of software is running within a genuine TEE. For this, the CPU (i) hashes the trusted code and data in the TEE, producing a *measurement*; (ii) cryptographically signs the measurement; and (iii) provides the measurements and signatures to the remote attestor. The attestor then verifies the provided attestation, i.e., whether the signature is valid and whether the provided measurements correspond to a set of known values. This allows to establish trust in a remote system.

III. System Overview

In this section, we provide an overview of the system and threat models for cryptocurrency-backed tokens, and formulate the main goals of our approach.

A. System Model and Actors

We assume a user Alice owns funds in cryptocurrency A and wishes to create the corresponding amount of A-backed tokens on cryptocurrency B. We further assume a user Bob owns cryptocurrency B and wishes to acquire A-backed tokens on cryptocurrency B, which may at a later point be redeemed for A. We denote funds held in cryptocurrencies A and B as a and b, respectively. We also denote A-backed tokens on cryptocurrency B as a_b.

We differentiate between the following types of actors in our protocol:

- **Creator.** Locks a on A to create a_b tokens on B.
- **Sender.** Owns a_b and transfers ownership to another user on B (e.g. Alice transfers ownership of a_b to Bob).
- **Receiver.** Receives a_b as part of an exchange transaction on B (e.g. Bob receives a_b from Alice).
- **Redeemer.** Destroys or *burns* a_b on B to unlock the corresponding amount of a on A .
- **Issuer.** An intermediary overseeing the correct issuing and redeeming of tokens on A.
- **Treasury.** An intermediary responsible for issuing, trading and redeeming a_b tokens on B.

Sender (Alice) and receiver (Bob) interact during the exchange and mutually distrust each other. Both creator and redeemer interact with the issuer. We discuss the trust requirements for the issuer in detail in Section IV. All parties interact with the Treasury.

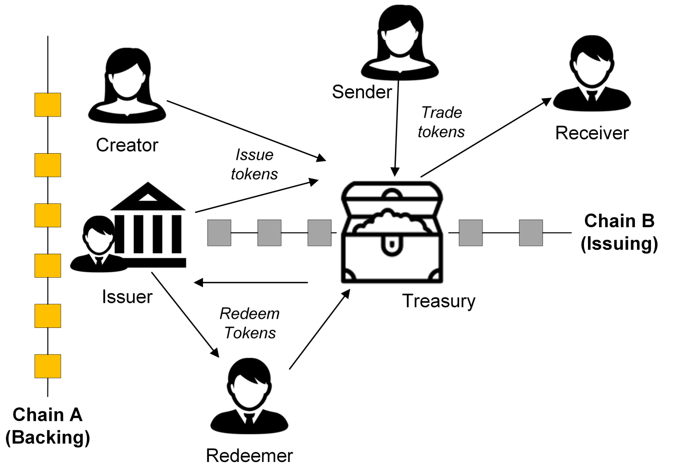


Fig. 1. Simplified visualization of the actors involved in the XCLAIM protocol, along with the interactions between them.

B. Network and Threat Model

We make several assumptions about the networks underlying the cryptocurrencies on which our protocol operates. Unless stated otherwise, we assume the trust models of cryptocurrencies A and B to hold, i.e., the portion of the overall mining power controlled by a computationally-bounded adversary is less than 50%. We further assume the cryptographic primitives used in cryptocurrencies A and B to be secure. Under these assumptions, an honest majority suffices to allow for consensus under Byzantine conditions, making tampering with on-chain smart contracts and state not possible. However, since cryptocurrencies A and B may only provide eventual consistency guarantees [54], accidental chain reorganizations must be considered possible.

While honest participants adhere to protocol rules, an adversary can behave arbitrarily. Thereby, we make the assumption that the adversary is economically rational. Our protocol has to take into account that an adversary might censor transactions and/or delay delivery of such transactions if it benefits them. As such, our approach must prevent partial execution of trades to avoid inconsistencies.

C. Protocol Goals

The goal of our protocol is to enable the issuing, exchange and redeeming of cryptocurrency-backed tokens, while minimizing trust between any of the actors in our system.

As such, we ideally do not want to require the *safety* of the protocol to depend on the availability and honest behaviour of a (trusted) third party. If a third party is necessary, however, then deviations from the protocol must be penalised, with potential financial damage to users reimbursed, so as to minimise the incentive for malicious behaviour by economically-rational adversaries.

The desirable properties for cryptocurrency-backed tokens can be formulated as follows:

1) **Generality.** The scheme for issuing, trading and redeeming tokens should be general enough to allow backing by many different cryptocurrencies, i.e., our protocols should not introduce dependencies on a specific cryptocurrency or implementation, such as Bitcoin, or Ethereum.
2) **Fungibility.** Alice must be able to lock any portion of her funds a to create the corresponding amount of a_b tokens on cryptocurrency B. In turn, she must be able to trade these tokens to Bob against b or some other tokens on B.
3) **Divisibility.** Alice must be able to trade any fraction of a_b, as long as it exceeds the minimal possible unit in A, e.g., in Bitcoin, the minimal possible unit would be 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.
4) **Value Redeemability.** Any user on B, e.g., both Alice or Bob, when in possession of a_b must be able to redeem the equivalent amount of a (less potential transaction fees) on A, or the corresponding (monetary) value in b, by destroying or *burning* the tokens on B.
5) **Transfer Atomicity.** Once Alice has transferred a_b she can no longer spend the corresponding units of a on A, as this would constitute a *double spend.*
6) **Consistency.** At any point in time, the existence of a_b tokens and the availability of the corresponding units of a are mutually exclusive. That is, a_b tokens can only be generated on B if the respective amount of a is locked in A, while the lock can be released only if the corresponding tokens have been destroyed on B.

IV. Design Roadmap

We propose a protocol for creating cryptocurrency-backed tokens. We start by describing a naïve centralised approach to outline the intuition behind our design and then reduce the trust requirements step by step. We settle on a final solution, made up of several sub-protocols, in Section V.

For ease of explanation, we present our protocols in the context of constructing Bitcoin-backed tokens on Ethereum. We note, however, that our approach and protocols are not limited to these two cryptocurrencies; they are general enough to support many other cryptocurrency-backed tokens. We discuss the generality of our approach in Section V-C.

A. Centralised Issuing: A Strawman Scheme

We first present a strawman scheme outlining the general idea of Bitcoin-backed tokens on Ethereum. We refer to the actors outlined in III-A throughout our descriptions.

The intuition behind the strawman scheme is simple; create a publicly verifiable log of actions by all actors. Should an actor misbehave, their misbehaviour can be seen, proven and action can be taken against them. To achieve this scheme, we: (i) make use of a single trusted entity to act as the *Issuer*, trusted to process the issuing and redeeming of tokens on Bitcoin correctly; and (ii) deploy a publicly verifiable smart contract on Ethereum to act as the *Treasury*, referred to as the `treasury` contract. By forcing all actors to interact with the `treasury` contract, their actions can be logged in a secure, publicly verifiable way, thus creating a secure audit-trail that can be used alongside public transactions in Bitcoin and Ethereum to prove any misbehaviour.

To show how this solution works, we present three sub-protocols, *Issue*, *Trade* and *Redeem* below. We assume a user Alice owns Bitcoin (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 Ethereum (eth), associated with his public/private key pair (pk_B^{eth}, sk_B^{eth}) and wishes to acquire btc_{eth}, and at some later point redeem the tokens for units of btc on Bitcoin. In this scenario, Alice takes the roles of the *Creator* and the *Sender*, while Bob takes the roles of the *Receiver* and *Redeemer* (see Section III-A). We denote transactions created in Bitcoin T^{btc}, while transactions on Ethereum are denoted T^{eth}. For simplicity we omit fees charged by the Issuer in the following sections.

Sub-protocol: Issue (see Figure 2)

1) Alice as the initiator of the protocol verifies the `treasury` contract is correct, 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 be verified by any Bitcoin client. That is, 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[2].
3) Once T_{lock}^{btc} has been included in the Bitcoin blockchain and has received sufficient confirmations (cf. Section VII-C), the Issuer, providing his digital signature $sig(sk_I^{eth})$, instructs the `treasury` contract to issue btc_{eth} to Alice on Ethereum, such that $|btc_{eth}| = |btc|$.

Sub-protocol: 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.

Sub-protocol: Redeem (see Figure 3)

1) Once Bob decides to redeem his btc_{eth} for the corresponding amount btc, he first creates a new public/private key pair (pk_B^{btc}, sk_B^{btc}) on Bitcoin.

[2]This can be achieved by using the `OP_RETURN` opcode in Bitcoin [41], which allows to push up to 80 bytes of arbitrary data onto the Script stack.

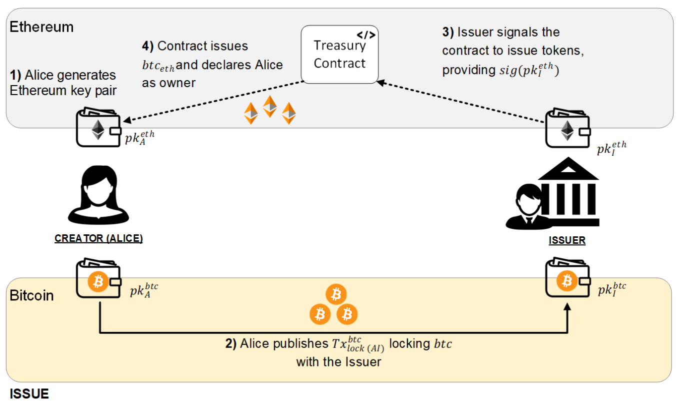

Ethereum
4) Contract issues btc_{eth} and declares Alice as owner
Treasury Contract
3) Issuer signals the contract to issue tokens, providing $sig(pk_I^{eth})$
1) Alice generates Ethereum key pair
pk_A^{eth}
pk_I^{eth}
CREATOR (ALICE)
ISSUER
Bitcoin
pk_A^{btc}
pk_I^{btc}
2) Alice publishes $Tx_{lock\ (AI)}^{btc}$ locking btc with the Issuer
ISSUE

Fig. 2. Visualization of the strawman*Issue* protocol, on the example of Bitcoin and Ethereum.

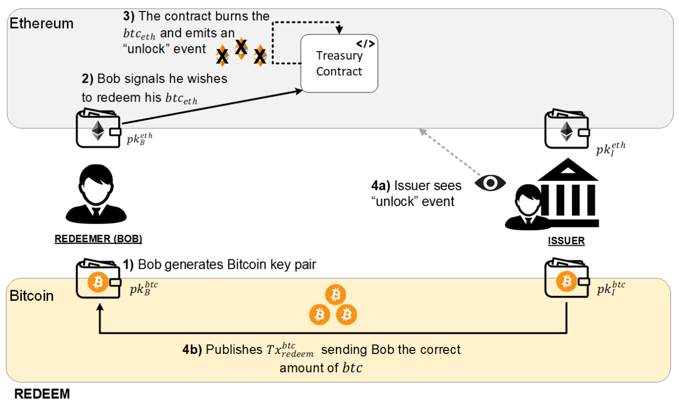

Ethereum
3) The contract burns the btc_{eth} and emits an "unlock" event
Treasury Contract
2) Bob signals he wishes to redeem his btc_{eth}
pk_B^{eth}
pk_I^{eth}
4a) Issuer sees "unlock" event
REDEEMER (BOB)
ISSUER
1) Bob generates Bitcoin key pair
Bitcoin
pk_B^{btc}
pk_I^{btc}
4b) Publishes Tx_{redeem}^{btc} sending Bob the correct amount of btc
REDEEM

Fig. 3. Visualization of the strawman*Redeem* protocol, on the example of Bitcoin and Ethereum.

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_{redeem}^{btc}.

While this approach is easy to implement and it is easy to see how 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.

Furthermore, to achieve this scheme, the following functionalities are required to be supported by the `treasury` contract deployed on Ethereum:

Functionality 1 (`treasury`: issueTokens)
Given proof that an amount of btc has been locked in Bitcoin for some pre-defined contestation period $t_{contest}$, i.e., received sufficient confirmations, create and allocate the corresponding amount of btc_{eth} to the Ethereum account associated with a given public key pk^{eth}.

Functionality 2 (`treasury`: transferTokens)
Provided with the 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 $pk^{eth'}$.

Functionality 3 (`treasury`: redeemTokens)
If a user controlling units of btc_{eth} signals to redeem the corresponding amount of btc on Bitcoin, emit a publicly visible "unlock" event, signalling that the lock-in Bitcoin is to be lifted, and burn *the returned btc_{eth} tokens.*

Although this approach requires trust in the *Issuer*, it is still arguably more transparent than tokens backed by real-world assets, since any user with access to the Bitcoin and Ethereum blockchains can at least observe an audited trace of the actions of the Issuer. As such, users would quickly become aware of malicious behaviour and cease to trust the misbehaving parties in 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 similar to relying on centralised liquidity providers, i.e., exchanges.

B. Chain Relays: Non-interactive Token Issuance

The strawman solution presented in IV-A requires the Issuer to monitor the Bitcoin blockchain to notify the `treasury` of confirmed Bitcoin transactions in order to issue tokens (see sub-protocol *Issue* in IV-A). In addition, the Issuer is required to monitor the Ethereum blockchain and `treasury` contract to release Bitcoin when it is redeemed by burning btc_{eth} (see sub-protocol *Redeem* in IV-A).

We remove these requirements by adding Bitcoin transaction verification logic to the `treasury` contract. This makes the `treasury` capable of verifying the inclusion of transactions in the Bitcoin blockchain, comparable to a Bitcoin SPV-Client [3]. We achieve this by deploying a *chain relay* [96] contract for Bitcoin on Ethereum. We note that such a contract is already available in the form of BTC Relay [6]. We therefore assume the chain relay is directly incorporated in the `treasury` contract, extending it by the following functionality:

Functionality 4 (`treasury`: verifyBtcTx)
Given a chain of Bitcoin block headers starting with the genesis block, a transaction and a Merkle Tree proof [75], verify that the transaction has been included in the Bitcoin blockchain for at least $t_{contest}$.

By enabling verification of Bitcoin transactions in the `treasury` contract, we achieve two improvements:

- During the *Issue* sub-protocol, Alice can directly prove to the `treasury` that she has correctly locked up her funds with T_{lock}^{btc}. This eliminates the requirement for the Issuer to be involved in token issuance, making this part of the protocol *non-interactive*.

- The `treasury` contract requires the Issuer to prove correct behaviour during the *Redeem* protocol, i.e., submit a transaction inclusion proof showing he released btc to Bob within some grace period t_{redeem}. Should the Issuer fail to comply by not providing the required proof, he is in turn financially penalised. We discuss the penalties in the next sections. Note: just as in the case of the *Issue* protocol, the proof must consider a sufficient contestation period $t_{contest}$ of T^{btc}_{redeem}.

C. Collateral: Introducing Incentives

Using a chain relay, users can prove misbehaviour by the Issuer to the `treasury`. Although this does not prevent the Issuer from misbehaving, it allows the `treasury` to impose penalties upon such misbehaviour; if the Issuer decides to ignore a redeem request and does not release Bitcoin when btc_{eth} is burnt (see Section IV-B), the `treasury` will financially penalise the Issuer.

To this end, we introduce the notion of *collateral*, where the Issuer is required to lock up funds on Ethereum as collateral to enable users to be reimbursed in the case of deviation from the protocol. To incentivize the Issuer to lock up collateral and partake in the scheme, they can earn fees on token issuance, trade and redemption.

As such, the Issuer is required to lock up sufficient collateral in eth to match the value of issued btc_{eth} tokens, plus an additional amount to cover any potential penalties. Now, with the means to penalise crash and Byzantine failures of the Issuer, we instruct the `treasury` to issue btc_{eth} tokens if and only if sufficient eth collateral has been locked by the Issuer. We note that the Issuer should be able to incrementally add and remove collateral from the `treasury` as needed; to handle changing demand in the number of tokens to be issued. A discussion on the correct parametrization of collateral requirements is provided in Section VII-F.

As such, we add the following required functionalities to the `treasury` contract :

Functionality 5 (`treasury`: lock/releaseCollateral)
Accept collateral deposits from the Issuer and hold these funds until all associated btc_{eth} are redeemed.

Functionality 6 (`treasury`: penalizeIssuer)
If the Issuer fails to submit a transaction inclusion proof showing he released btc to a user, within some grace period t_{redeem}, penalize the Issuer by taking his collateral and reimbursing the user.

D. Preventing Race Conditions during Issue

As defined in Section IV-C, the Issuer is required to first provide sufficient collateral before any corresponding tokens can be issued. Despite this requirement, the *Issue* protocol exhibits potential race conditions that expose the Sender's locked funds to theft during token creation.

When attempting to issue btc_{eth}, Alice (the Sender) must first lock in the corresponding amount of btc by transferring it to the Issuer. The T^{btc}_{lock} must be included in the Bitcoin blockchain for at least $t_{contest}$ before the `treasury` contract considers the transaction valid, i.e., the transaction must receive a sufficient number of confirmations. Between the time that Alice publishes T^{btc}_{lock} on the blockchain and the time it receives sufficient confirmations, Alice's funds are vulnerable to theft as the Issuer may withdraw not yet locked collateral or another user may lock up the maximum amount of collateral for new tokens before Alice can finalize the issue process. This prevents the `treasury` from being able to reimburse Alice if the Issuer misbehaves, as there will be insufficient collateral deposited by the Issuer. To avoid these race conditions, we propose two possible solutions:

1) Collateralized Issue Commitments: One solution is to require Alice to register an issue request with the `treasury` before initiating the *Issue* protocol. This locks the corresponding collateral of the Issuer for a predefined amount of time t_{commit}. Once the commitment has been registered, Alice must then create T^{btc}_{lock} and prove its existence to the chain relay within period t_{commit}. During this time, the Issuer cannot withdraw the specified amount of T^{btc}_{lock} from the contract, nor can it be assigned to anyone else's commitment.

To avoid *griefing*, Alice must also temporarily lock up some amount of eth as collateral committing herself to lock up the corresponding amount of btc in T^{btc}_{lock}, i.e., create a *collateralized commitment*. If Alice fails to transfer btc to the Issuer before the time expires, her collateral is confiscated and (optionally) transferred to the Issuer. Alice's collateral therefore must be dependent on the total btc_{eth} she is requesting.

As such, we require the following functionality from the `treasury` contract:

Functionality 9 a (`treasury`: registerIssueCommit)
Given a specified amount of btc_{eth}, and a corresponding amount of eth as user collateral, lock the appropriate amount of Issuer collateral for time t_{commit}.

One major advantage of this scheme is that it does not require any interaction from the Issuer during the *Issue* protocol, and therefore maintains non-interactivity as originally achieved in Section IV-B. However, requiring users to provide collateral is less than ideal and may present an obstacle to users, impeding adoption.

2) Hashed Time-Lock Contracts: An alternative solution to preventing race conditions in the *Issue* protocol is through the use of Hashed Time-Lock Contracts (HTLCs). Instead of directly transferring btc to the Issuer, the Sender (Alice) generates a random secret s and creates a deposit with the following spending conditions: (i) either the Issuer provides the hash pre-image to $H(s)$ alongside his digital signature $sig(sk^{btc}_I)$, where H is a cryptographically secure hash function[3], or (ii) Alice revokes the lock by providing her digital signature $sig(sk^{btc}_A)$ after a pre-defined period t_{abort}.

Once Alice has published the respective $T^{btc}_{lock(HTLC)}$ (`P2SH` [42]) transaction, she can prove to the chain relay that the correct amount of btc has been locked in Bitcoin. The `treasury` will then require Alice to publicly reveal the hash

[3]For example, SHA-256 or Keccak-256

pre-image s on Ethereum via $T^{eth}_{reveal(HTLC)}$, before issuing the btc_{eth} tokens. To prevent Alice from publishing s only shortly before the expiry of t_{abort} and attempting to spend from $T^{btc}_{lock(HTLC)}$ before the Issuer, essentially double spending the locked btc, the treasury contract requires that s be revealed before t_{reveal} ($t_{reveal} < t_{abort}$!).

As a result, there are two possible outcomes for the *Issue* protocol:

- **Success.** The Issuer has locked sufficient collateral in the treasury contract and Alice reveals s before t_{reveal}. The contract issues btc_{eth} to Alice and the Issuer, now in possession of s, spends from $T^{btc}_{lock(HTLC)}$.
- **Abort.** Alice proves the inclusion of $T^{btc}_{lock(HTLC)}$ in Bitcoin to the treasury contract but the Issuer possesses insufficient collateral to process the issue request. After t_{abort}, Alice revokes the lock and spends $T^{btc}_{lock(HTLC)}$ by providing her digital signature.

The implementation of HTLCs requires the following functionality to be added to the treasury contract:

Functionality 9 b (treasury: verifyHTLC)
Given $T^{btc}_{lock(HTLC)}$ and a secret s, verify s is the pre-image of $H(s)$ in $T^{btc}_{lock(HTLC)}$ and that s was revealed within the time period $t_{reveal} < t_{abort}$.

While HTLCs do not require users to lock in collateral (in contrast to IV-D1), the Issuer is now required to be online throughout the *Issue* protocol. This breaks the non-interactive property previously established in Section IV-B. If the Issuer is not online, Alice could attempt to double spend the locked btc In addition, another drawback to this approach occurs in the case of an aborted issue: while Alice is able to reclaim the locked btc, she must cover the incurred transaction costs both in Bitcoin and Ethereum. However, a possible mitigation to this is to simply introduce a fund in the treasury contract to reimburse users in such cases, instantiated during contract deployment.

E. Issuer Replacement

Until now, for simplicity, we have assumed the Issuer remains part of the protocol indefinitely. However, in a real world scenario, the Issuer may wish to leave the scheme and transfer their role to another party safely, without loss of collateral. We hence present a straightforward *Replace* sub-protocol allowing the Issuer to find replacement for his role the treasury contract.

Sub-protocol: Replace

1) The Issuer to submits a *replacement* request to the treasury contract ($T^{eth}_{replace}$).
2) In turn, one user (the first to respond) may answer the request by locking the necessary eth collateral in the treasury, providing their Bitcoin public key (or hash-based address).
3) The active Issuer must then migrate the locked btc to the new Issuer by creating a transaction $T^{btc}_{migrate}$ on Bitcoin and proving its inclusion in the Bitcoin blockchain to the treasury, within a period $t_{migrate}$.
4) After a pre-defined contestation period (cf. Section VII-C), the treasury releases the current Issuer's collateral, finalizing the replacement through the new candidate.

Should the Issuer not execute the migration within the specified period, the new candidate's funds are released, while transaction fees incurred by the locking/unlocking process are reimbursed from the Issuer's collateral.

We thus add one final required functionality to the treasury contract:

Functionality 10 (treasury: replaceIssuer)
Upon receipt of a replacement request by the Issuer, one user (the first to respond) may become an Issuer by locking the corresponding amount of eth as collateral. Given proof the original Issuer generated and included $T^{btc}_{migrate}$ in the Bitcoin blockchain, unlock the original Issuer's collateral and return it. Otherwise release the new Issuer's collateral.

V. XCLAIM PROTOCOL

In this section we provide a formal description of the XCLAIM protocol, followed by an overview of its requirements and a discussion on how XCLAIM satisfies the goals formulated in Section III-C.

A. Formal Protocol Description

We present the XCLAIM protocol. For ease of explanation, we present XCLAIM in the context of constructing Bitcoin-backed tokens on Ethereum (as in Section IV). We note, however, that XCLAIM is not limited to these two cryptocurrencies; we discuss the generality of our approach in V-C.

XCLAIM follows the roadmap outlined in Section IV. It makes use of all incremental design improvements and consists of sub-protocols: *Issue*, *Trade*, *Redeem* and *Replace*. XCLAIM offers two variants in the case of *Issue*; the first is based on HTLCs as discussed Section IV-D2, and the second on collateralized issue commitments, as discussed in Section IV-D1. Figure 4 shows the life-cycle of a Bitcoin-backed token on Ethereum in XCLAIM. We refer to this life-cycle throughout our algorithms. Furthermore, where relevant, we refer directly to the functionalities required by the treasury contract (as defined in section IV).

We write $btc \rightarrow pk^{btc}_{A}$ to denote that btc is controlled by Alice's Bitcoin public key pk^{btc}_{A}. Similar, $eth \rightarrow$ treasury expresses that eth is controlled by the treasury contract. We use $pk^{btc}_{A} \xrightarrow{btc} pk^{btc}_{I}$ to describe a transfer/allocation of btc from Alice to the Issuer. If a transaction has a complex spending condition, the conditions are listed after a vertical bar "|". Collateral provided, e.g. by the Issuer, is denoted as eth^{col}_{I} (analogous for other users). We further use $|btc|$ to refer to the economic value of the specified cryptocurrency/token units.

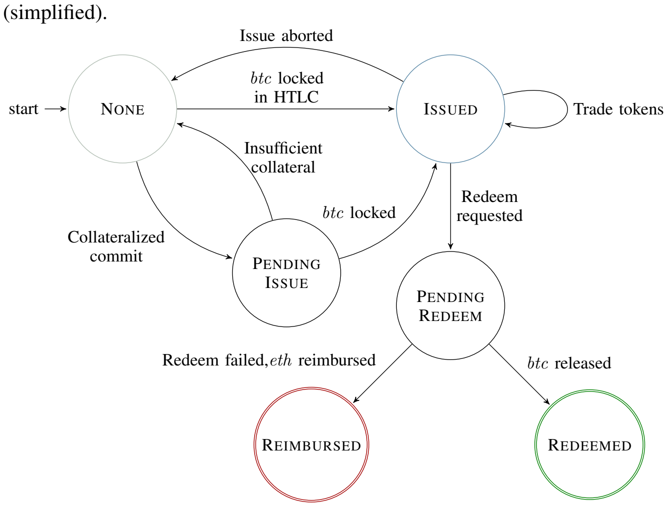
Algorithms 1 and 2 show the two variants of the *Issue* sub-protocols. The purpose of *Issue* is to issue Bitcoin-backed tokens on Ethereum securely, through the subsequent locking up of Bitcoin. To avoid the vulnerabilities outlined in IV-D, either HTLCs are required, or collateralized issue commitments. Upon successful execution, *Issue* moves from life-cycle state `NONE` to `ISSUED` (algorithm 1, line 11 and algorithm 2, line 9).

Algorithm 3 shows the *Trade* sub-protocol. The purpose of *Trade* is to enable Bitcoin-backed tokens to be transferred from one user (sender) to another (receiver). This happens securely through the `treasury` contract. Upon successful execution, XCLAIM tokens are transferred between users; tokens remain in the `ISSUED` state (algorithm 3, lines 7 and 10).

Algorithm 4 shows the *Redeem* sub-protocol. The purpose of *Redeem* is to enable Bitcoin-backed tokens on Ethereum to be redeemed for Bitcoin. This happens securely through the `treasury` contract and the *Issuer*. Upon successful execution, Bitcoin-backed tokens on Ethereum are burnt, and a corresponding amount of Bitcoin is issued to the redeemer, moving tokens from state `ISSUED` to `REDEEMED` (algorithm 4, line 10). Otherwise, if the *Issuer* misbehaves, collateral is confiscated and used to reimburse the user, moving from state `ISSUED` to `REIMBURSED` (algorithm 4, line 14). The use of collateral in this way is discussed in Section IV-D1.

Algorithm 5 shows the *Replace* sub-protocol. The purpose of *Replace* is to enable the Issuer to leave the protocol and transfer their role to another party safely, without losing collateral. This happens through interaction with the `treasury` contract, who assigns the role of the Issuer from one user to another securely, before releasing collateral. See the discussion in IV-E.

Fig. 4. State machine visualizing the token lifecycle throughout the protocol (simplified).



Sub-protocol 1 Issue - Hashed Time-Lock Contracts

```
Actors: Creator (Alice): $(pk_A^{btc}, sk_A^{btc}), (pk_A^{eth}, sk_A^{eth})$; Issuer:
        $(pk_I^{btc}, sk_I^{btc}), (pk_I^{eth}, sk_I^{eth})$; treasury.
Require: $btc \rightarrow pk_A^{btc}$, $eth_I^{col} \rightarrow$ treasury.
 1: procedure ISSUE_HTLC( )
 2:     Alice generates $s \leftarrow$ random()
 3:     Alice publishes $T^{btc}_{lock(HTLC)}$: $pk_A^{btc} \xrightarrow{btc} pk_I^{btc}$
        $| \left(\exists sig(sk_I^{btc}) \wedge H(<\texttt{input}>) = H(s)\right) \vee$
        $\left(\exists sig(sk_A^{btc}) \wedge t_{current} \geq t_{abort}\right)$, including $pk_A^{eth}$ as data
 4:     Alice publishes $T^{eth}_{proof}$, calling verifyBtcTx($T^{btc}_{lock}$)
        in treasury
 5:     if $\left(\text{verifyBtcTx}(T^{btc}_{lock}) = \top \wedge \exists(eth_I^{col}) \wedge |eth_I^{col}| \approx |btc|\right)$
    then
 6:         Alice publishes $T^{eth}_{reveal}$, calling verifyHTLC($s$) in
            treasury
 7:         if $(t_{current} \leq t_{reveal} \wedge \text{verifyHTLC}(s) = \top)$ then
 8:             treasury executes issueTokens($btc_{eth}, pk_A^{eth}$)
 9:             do
10:                 Issuer publishes $T^{btc}_{spend(HTLC)}$: $pk_A^{btc} \xrightarrow{btc} pk_I^{btc}$
                    spending $T^{btc}_{lock(HTLC)}$ by providing $s$
11:                 return SUCCESS
                    // Token state: ISSUED
12:             while $t_{current} < t_{abort}$
13:         else
14:             return ABORT_ISSUE()
15:         end if
16:     else
17:         return ABORT_ISSUE()
18:     end if
19: end procedure
20: procedure ABORT_ISSUE( )
21:     while $t_{current} < t_{abort}$ do
22:         wait
23:     end while
24:     Alice spends from $T^{btc}_{lock(HTLC)}$ by publishing $T^{btc}_{spend(HTLC)}$ :
        $pk_A^{btc} \xrightarrow{btc} pk_I^{btc}$ and providing $s$
25:     return FAIL
        // Token state: NONE
26: end procedure
Result if Success: $btc \rightarrow pk_I^{btc}$, $btc_{eth} \rightarrow pk_A^{eth}$;
```

B. System Requirements

To create XCLAIM cryptocurrency-backed tokens, the backing cryptocurrency A and issuing cryptocurrency B must satisfy several properties. We outline these requirements below:

- A, as the backing cryptocurrency, must support the following operations on the underlying blockchain: (i) transfer of cryptocurrency units; (ii) verification of digital signatures; (iii) a cryptographically secure hash function; and (iv) the ability to store additional metadata for transactions.
- B, as the issuing cryptocurrency, must provide a programming language expressive enough to implement all the required functionalities (1-9) outlined for the `treasury` contract in Section IV. The only functionality that is not required by default, is verifyBtcTx (functionality 4, section IV-B) for implementing chain relays on the underlying blockchain. This functionality is recommended, but is not fundamentally required.[4].

While Bitcoin meets the requirements for the backing cryptocurrency A, the instruction set available in its current implementation is not sufficient for Bitcoin to act as the issuing cryptocurrency B [40]. Ethereum, on the other hand, can successfully act as the backing cryptocurrency A as well as the issuing cryptocurrency B, as the Ethereum Virtual Machine offers a Turing-complete set of operations [55]. We note that

[4]We present a more generic approach in Section VI that does not require support for chain relays, but this comes at the cost of additional trust assumptions.

Sub-protocol 2 Issue - Collateralized Issue Commitments

```
Actors: Creator (Alice): $(pk_A^{btc}, sk_A^{btc}), (pk_A^{eth}, sk_A^{eth})$; treasury.
Require: $btc \rightarrow pk_A^{btc}$, $eth_I^{col} \rightarrow$ treasury.
 1: procedure ISSUE_COMMIT( )
 2:     Alice publishes $T_{commit}^{eth}: pk_A^{eth} \xrightarrow{eth_A^{col}}$ treasury , calling
        registerIssueCommit($btc^{issue}, pk_A^{btc}$) in treasury
 3:     treasury executes lockCollateral($eth_I^{col}, t_{commit}$), where
        $|eth_I^{col}| \approx |btc^{issue}|$
        // Token state: PENDING ISSUE
 4:     Alice publishes $T_{lock}^{btc}: pk_A^{btc} \xrightarrow{btc} pk_I^{btc}$
 5:     Alice publishes $T_{proof}^{eth}$, calling verifyBtcTx($T_{lock}^{btc}$) in
        treasury
 6:     if $\left(\text{verifyBtcTx}(T_{lock}^{btc}) = \top \wedge t_{current} \leq t_{commit}\right)$ then
 7:         treasury executes issueTokens($btc_{eth}, pk_A^{eth}$)
 8:         Alice publishes $T_{withdraw}^{eth}$: treasury $\xrightarrow{eth_A^{col}} pk_A^{eth}$
 9:         return SUCCESS
            // Token state: ISSUED
10:     else
11:         treasury executed unlockCollateral($eth_I^{col}$)
12:         [Optional]: treasury reimburses the Issuer with Alice's
            collateral: $eth_A^{col} \rightarrow pk_I^{eth}$
13:         return FAIL
            // Token state: NONE
14:     end if
15: end procedure
Result if Success: $btc \rightarrow pk_I^{btc}$, $btc_{eth} \rightarrow pk_A^{eth}$;
```

Sub-protocol 3 Trade

```
Actors: Sender (Alice):$(pk_A^{eth}, sk_A^{eth})$; Receiver (Bob): $(pk_B^{eth}, sk_B^{eth})$;
        treasury.
Require: $btc \rightarrow pk_I^{btc}$, $btc_{eth} \rightarrow pk_A^{eth}$, $eth \rightarrow pk_B^{eth}$
 1: procedure TRADE( )
 2:     Alice publishes $T_{offer}^{eth}$ declaring $btc_{eth} \rightarrow$ treasury
 3:     do
 4:         Bob publishes $T_{match}^{eth}: pk_B^{eth} \xrightarrow{eth}$ treasury
 5:         treasury declares $btc_{eth} \rightarrow pk_B^{eth}$
 6:         Alice publishes $T_{withdraw}^{eth}$: treasury $\xrightarrow{eth} pk_A^{eth}$
 7:         return SUCCESS
            // Token state: ISSUED
 8:     while $t_{current} < t_{offer}$
        // Bob does not match the offer, trade aborted
 9:     treasury declares $btc_{eth} \rightarrow pk_A^{eth}$
10:     return FAIL
        // Token state: ISSUED
11: end procedure
Result if Success: $btc \rightarrow pk_I^{btc}$, $btc_{eth} \rightarrow pk_B^{eth}$, $eth \rightarrow pk_A^{eth}$
```

Sub-protocol 4 Redeem

```
Actors: Redeemer (Bob): $(pk_B^{btc}, sk_B^{btc}), (pk_B^{eth}, sk_B^{eth})$; Issuer:
        $(pk_I^{btc}, sk_I^{btc}), (pk_I^{eth}, sk_I^{eth})$; treasury.
Require: $btc \rightarrow pk_I^{btc}$, $btc_{eth} \rightarrow pk_B^{eth}$
 1: procedure REDEEM( )
 2:     Bob publishes $T_{redeem}^{eth}$, calling redeemTokens($btc_{eth}$) in
        treasury
 3:     treasury contract emits Event("unlock $btc$ to $pk_B^{btc}$")
 4:     do
 5:         Issuer publishes $T_{redeem}^{btc}: pk_I^{btc} \xrightarrow{btc} pk_B^{btc}$
 6:         Issuer publishes $T_{proof}^{eth}$, calling verifyBtcTx($T_{redeem}^{btc}$) in
            treasury
 7:         if $\left(\text{verifyBtcTx}(T_{redeem}^{btc}) = \top\right)$ then
 8:             treasury contract destroys tokens: $btc_{eth} \rightarrow$ X
 9:             treasury executed unlockCollateral($eth_I^{col}$)
10:             return SUCCESS
            // Token state: REDEEMED
11:         end if
12:     while $t_{current} < t_{grace}$
        // Issuer provides no proof
13:     treasury executes penalizeIssuer($eth_I^{col}, pk_B^{eth}$), reimbursing
        Bob the value of $btc_{eth}$
14:     return FAIL
        // Token state: REIMBURSED
15: end procedure
Result if Success: $btc \rightarrow pk_B^{btc}$, $btc_{eth} \rightarrow$ X
```

Sub-protocol 5 Replace

```
Actors: Issuer: $(pk_I^{btc}, sk_I^{btc}), (pk_I^{eth}, sk_I^{eth})$; New issue candidate
        (Carol): $(pk_C^{btc}, sk_C^{btc}), (pk_C^{eth}, sk_C^{eth})$; treasury.
Require: $btc \rightarrow pk_I^{btc}$, $eth_I^{col} \rightarrow$ treasury, $eth \rightarrow pk_C^{eth}$
 1: procedure REPLACE( )
 2:     The Issuer publishes $T_{replace}^{eth}$, calling replaceIssuer()
 3:     Carol publishes $T_{lock}^{eth}$: calling lockCollateral($eth_C^{col}$) in
        treasury
 4:     if $\left(|eth_C^{col}| = |eth_I^{col}|\right)$ then
 5:         do
 6:             Issuer publishes $T_{migrate}^{btc}: pk_I^{btc} \xrightarrow{btc} pk_C^{btc}$
 7:             Issuer publishes $T_{proof}^{eth}$, calling verifyBtcTx($T_{migrate}^{btc}$)
                in treasury
 8:             if $\left(\text{verifyBtcTx}(T_{migrate}^{btc}) = \top\right)$ then
 9:                 treasury executes lockCollateral($eth_C^{col}$) and
                    unlockCollateral($eth_I^{col}$)
10:                 Issuer published $T_{withdraw}^{eth}$: treasury $\xrightarrow{eth_I^{col}} pk_I^{eth}$
11:                 return SUCCESS
12:             end if
13:         while $t_{current} < t_{migrate}$
14:     end if
        // Issuer ignored migration
15:     treasury executes unlockCollateral($eth_C^{col}$)
16:     Carol publishes $T_{abort}^{eth}$: treasury $\xrightarrow{eth_C^{col}} pk_C^{eth}$
17:     return FAIL
18: end procedure
Result if Success: $btc \rightarrow pk_B^{btc}$, $btc_{eth} \rightarrow$ X
```

Turing-completeness, however, is not necessarily a requirement for the issuing cryptocurrency B, since XCLAIM is also supported by non-Turing-complete languages such as Scilla [89]. Examples of backing cryptocurrencies that are supported by XCLAIM are Bitcoin [83], Namecoin [81], Zcash [37], Litecoin [73] and Monero [20]. Examples of issuing cryptocurrencies supported by XCLAIM are Ethereum [52], Ethereum Classic [15], Cardano [7], NEO [22] and Rootstock [72].

C. Satisfaction of Protocol Goals

We now discuss how XCLAIM satisfies the goals formulated in Section III-C for cryptocurrency-backed tokens.

1) **Generality.** As discussed in V-B above, XCLAIM requires only several properties from the underlying blockchains on which it operates. As such, XCLAIM is not limited to a single blockchain implementation, such as Bitcoin, or Ethereum, but can operate in its current form on many different cryptocurrencies. Furthermore, the implementation already presented in Section VI can be used to generate Namecoin-backed [81] tokens on Ethereum Classic [15], straight out the box, with little to no modification.

2) **Fungibility.** Once issued, XCLAIM does not allow to distinguish between two tokens a_b and a'_b issued on B backed by the same cryptocurrency A. As such, the value of one token is substantially equivalent to the value of any other token (backed by the same cryptocurrency), at any given point in time.

3) **Divisibility.** XCLAIM imposes no restrictions regarding the divisibility of the issued tokens, and hence this property is dependent on the functionality of the issuing blockchain. In the case of Ethereum, the smallest currency unit is 10^{-18}, exceeding the divisibility limits of Bitcoin (10^{-8}).

4) **Value Redeemability.** XCLAIM is trustless in the sense that users are guaranteed to receive the (monetary) value of the issued and collateralized cryptocurrency-backed tokens a_b, even in case of a crash or Byzantine failure of the Issuer. That is, either a_b can be redeemed for corresponding units of b, or the `treasury` contract reimburses the user with units of a, equivalent in value to a_b.

5/6) **Transfer Atomicity and Consistency.** In XCLAIM, the Issuer ensures locked units b of the backed cryptocurrency B cannot be moved while the corresponding units of tokens a_b are in circulation. Should the Issuer fail to prevent atomicity, the chain relay functionality allows to prove the failure to the contract, which will destroy the corresponding tokens a_b, ensuring atomicity, and reimburse the victim with the equivalent value in a using the Issuer's collateral.

VI. Implementation and Evaluation

We implement the XCLAIM protocol as described in Section V to create Bitcoin-backed tokens on Ethereum. Our implementation supports both the collateralized commitment and the HTCL-based schemes of the *Issue* sub-protocol. Moreover, our `treasury` contract is ERC20 compatible allowing tokens issued through XCLAIM to be traded at various decentralized exchanges [1], [17], [19]. We also present two optimizations to our implementation through the use of trusted execution environments (TEEs). Here, we use Intel SGX; our optimizations trade-off performance and efficiency against trust. These are: (i) using TEEs for the chain relay functionality; and (ii) using TEEs for the Issuer. Finally, we provide an evaluation of the three implementations.

A. Implementation

For our implementation of the `treasury` contract on Ethereum we use the Solidity smart contract programming language v0.4.24 [29]. Our implementation consists of around 820 lines of Solidity code. Besides the constructor, the `treasury` contract exposes API calls for issuing (both versions), trading, and redeeming Bitcoin-backed tokens, as well as gracefully replacing the issuer. All additional functions necessary to be compliant with the ERC20 standard [10] are also implemented. The contract provides a simple access control system, as access to certain functions must be restricted to either Issuer or the chain relay, depending on the implementation.

The existing version of BTC Relay is implemented in the outdated Serpent [30] programming language[5] and is based on an old version of the EVM, exhibiting significant performance issues. Hence, we implement a subset of the chain relay functionality necessary for the evaluation of XCLAIM in Solidity v0.4.24 [29]. Specifically, we implement the `verifyTx` functionality which, given a transaction, a Merkle Tree proof, the index of the transaction in its block, and the hash of the block as input, checks if the transaction was included in the block. Our implementation consist of 140 lines of code and improves upon the existing BTC Relay version (Serpent) by using native functions for the SHA-256 implementation and the bitwise negation functions in Solidity.

For Bitcoin, we implement the hashed time-lock contract used in the HTLC-based version of the *Issue* sub-protocol using Bitcoin's Script according to the `P2SH` [42] transaction format.

B. Optimization 1: XCLAIM *with SGX Relay*

The XCLAIM protocol requires no trust between the actors in the system. However, the need to perform on-chain transaction verification through a chain relay incurs high costs. To avoid this, one could trade-off performance, efficiency and practicality against trust; operating a chain relay inside a TEE, such as Intel SGX, to remove the cost of having to operate a chain-relay on-chain.

For this, the TEE would operate a full cryptocurrency node and verify the state of the blockchain in a secure manner. Using the confidentiality and integrity guarantees of TEEs, the `treasury` could first attest the TEE chain relay before trusting it to verify the inclusion of transactions in the underlying blockchain and notify it of misbehaviours by the actors. However, this optimization comes at the cost of having to trust the TEE to operate securely. Otherwise, if the TEE is compromised by an adversary, the Issuer and the adversary could collude to present invalid transaction inclusion proofs to the `treasury` to steal funds. We discuss mitigations against compromise in VII.

C. Optimization 2: XCLAIM *with SGX Issuer*

A second optimization that XCLAIM could employ would be to execute the role of the Issuer inside a TEE. The benefit of this approach is that the Issuer would be trusted to operate correctly, issuing tokens and redeeming funds on request. This would significantly simplify the XCLAIM protocol, removing the need for chain relays and Issuer collateral. In comparison to VI-B, this approach further trades-off simplicity and practicality for trust. The TEE who operates the Issuer would be trusted entirely; if it were to be compromised or suffered a loss of availability, funds could be stolen. We discuss mitigations to these in VII.

D. Execution and Storage Evaluation

To evaluate the on-chain execution and storage costs of XCLAIM, we deploy the `treasury` contract on the Ethereum Ropsten test network [16]. We deploy XCLAIM with and without optimizations. The three implementations we deploy are thus: XCLAIM[6], XCLAIM with an SGX chain relay[7] and

[5]Last commit on 1 October 2017.

[6]Contract address online: 0xcfdb6fcb7f3c2b5acefb0121d7aa68aa4690dab9

[7]Contract address online: 0xefd81db13797cb010ed67b0abfc9c78e228a8fae

TABLE I. Detailed evaluation of execution and on-chain storage costs of our XCLAIM prototype implementation, including the optimizations using TEEs. Measurements are provided for individual sub-protocols. Presented costs include both Bitcoin transaction fees and Ethereum gas costs, calculated according to BTC/USD and ETH/USD exchange rates as of 7 August 2018.

		XCLAIM † (Trustless)						XCLAIM with SGX Relay						XCLAIM with SGX Issuer					
		Success			Failure			Success			Failure			Success			Failure		
		TX		Cost [USD]	TX		Cost [USD]	TX		Cost [USD]	TX		Cost [USD]	TX		Cost [USD]	TX		Cost [USD]
		ETH	BTC		ETH	BTC		ETH	BTC		ETH	BTC		ETH	BTC		ETH	BTC	
Issue	**HTLC**	2	2	1.17	2	2	1.10	2	2	0.76	2	2	0.69	2	2	0.76	2	2	0.69
	Collateralized Commit	2	1	0.67	2	-	0.45	2	1	0.45	2	-	0.23	2	1	0.45	2	-	0.23
Trade		2	-	0.41	1	-	0.25	2	-	0.41	1	-	0.25	2	-	0.41	1	-	0.25
Redeem		2	1	0.72	2	-	0.61	2	1	0.49	2	-	0.38	1	1	0.19	-	-	-
Replace‡		3+	2+	0.69+	3+	-	0.27+	3+	2+	0.47+	3+	-	0.27+	-	-	-	-	-	-

†Costs for verification of Bitcoin transaction inclusion proofs calculated for the average number of transaction per block in 2018, i.e., ≈ 1349 [43].
‡The execution costs of the *Replace* protocol depend on the number of Bitcoin UTXOs which need to be migrated.

XCLAIM with an SGX Issuer[8].

We define on-chain execution costs as the amount of Bitcoin transaction fees and Ethereum gas costs to execute each of the sub-protocols: *Issue*, *Trade*, *Redeem* and *Replace*[9]. We define on-chain storage costs as the number of transactions to be placed on the underlying blockchains for each sub-protocol. Table I shows a detailed breakdown of the on-chain execution costs (in USD), and on-chain transaction storage costs (in number of transactions) of each of the three implementations, for both successful execution and faulty termination.

Issue. We observe a difference in the number of Bitcoin transactions to be placed on the Bitcoin blockchain for the two versions of the *Issue* sub-protocol (2 vs 1). Furthermore, the HTLC scheme places larger transactions on the blockchain due to more complex conditions (`P2SH`). This increases the corresponding cost of protocol execution. While using an on-chain relay allows to check Bitcoin transaction inclusion proofs in a publicly verifiable manner, the fact that the verification logic is executed on Ethereum results in higher gas costs. As such, it costs ≈398.8 thousand gas (USD 0.82) to successfully issue (HTLC-based) cryptocurrency-backed tokens. The Ethereum gas costs are reduced by 50% (to USD 0.41) if the chain relay functionality is outsourced to TEEs. For the version using collateralized commitments the SGX relay achieves an improvement from USD 0.67 to USD 0.45 (32.8%).

Trade. The *Trade* sub-protocol does not require any interactions with the Issuer or the chain relay and thus the execution and storage costs are contained within Ethereum; these costs are therefore identical across all three implementations.

Redeem. Similar to *Issue*, the redeem sub-protocol requires verification of Bitcoin transactions. As a result, the execution costs of ≈289.1 thousand gas (USD 0.59) on Ethereum are reduced by 39% (to USD 0.36) if the chain relay is implemented using TEEs. Furthermore, in case the Issuer is completely implemented via TEEs, the gas costs for redeeming tokens are minimized to USD 0.06, i.e., an improvement of 89.8%, due to less transactions being published on Ethereum.

Replace. The costs of the *Replace* protocol depend on the number of Bitcoin deposits the original Issuer must forward to the new Issuer. In the optimal case, the Issuer could first group the deposits before executing the protocol, reducing the number of Bitcoin transaction costs substantially. The measurements presented in Table I do this, and are thus lower bounds.

[8]Contract address online: 0xb300630d2e658cdb0a691cba48581a5da942ed36
[9]Conversion rates as of 7 August 2018: BTC/USD 6949.17; ETH/USD 409.00

VII. Security Challenges

We discuss the security challenges of XCLAIM; we outline possible attack vectors and their impact on the system.

A. Infrastructure Denial-of-Service

The Issuer, as an individual, may be exposed to out of band Denial-of-Service (DoS) attacks. Crash failures only ever benefit the Issuer in the case of a significant exchange rate fluctuation during the *Redeem* sub-protocol. To overcome this, multiple Issuers can be introduced, forming a committee, and using threshold signatures for token issuance and redemption. This makes the cost of DoS attacks significantly more expensive. Committee election schemes are generally well known and have been proposed by many works [39], [48], [68], [76]. As such, it is potentially possible to utilise schemes like Byzcoin [69], Hybrid Consensus [86] and PeerCensus [49] for XCLAIM to sample a committee of issuers from PoW blockchains.

Moreover, attempting to perform a DoS attack against the `treasury` contract in XCLAIM is difficult; such an attack is equivalent to a DoS attack against either cryptocurrency network that underpins the tokens. If however trusted hardware is used alongside XCLAIM instead of a chain relay, a single point of failure is introduced. To avoid centralization, multiple trusted hardware devices can operate in unison, sharing state and operating a consensus protocol such as PAXOS [71] to agree on the state of the underlying blockchain. This avoids centralization through the use of trusted hardware chain relays.

B. Network Layer Eclipse Attacks

An adversary may attempt to perform network layer *eclipse attacks* [57], [60]. In XCLAIM, eclipse attacks pose a threat for the Issuer; a successful attack on the Issuer has the same effect as an infrastructure DoS attack. This is because blockchain transactions trigger no actions by the Issuer, but rather by the chain relay functionality of the `treasury`. Similarly, if the chain relay is operated in trusted hardware as an optimization, it can also be isolated from the network, allowing an adversary to present it with an invalid state of the blockchain.

The mitigation of eclipse attacks against the Issuer or the chain relay functionality of the contract, as in the case of DoS attacks, is replication: rely on the availability and honest behaviour of multiple Issuers/chain relays, instead of a single entity. We note here that vulnerability to eclipse-attacks and network partitioning is not a result of the XCLAIM protocol, but rather a general problem faced by all peer to peer cryptocurrencies [56], [60], [98].

C. Chain Reorganizations and Forking Attacks

Most cryptocurrencies, such as Bitcoin and Ethereum, provide only eventual consistency guarantees for transactions [54] i.e., chain reorganizations and *forks* can occur. Furthermore, Byzantine consensus participants can perform *selfish mining* attacks, where an adversary attempts to create a secret chain with more accumulated work to force the blockchain to be reorganized [53], [56], [88].

In cases of accidental or malicious reorganizations, all cryptocurrency applications are vulnerable. Likewise for XCLAIM; an unsafe or too optimistic parametrization of the contestation periods for the underlying cryptocurrencies, e.g. $t^{btc}_{contest}$ and $t^{eth}_{contest}$, can lead to inconsistencies between the amount of locked btc and issued btc_{eth}. As such, to address this, Sompolinsky and Zohar propose that the period to wait until a transaction is considered secure must be set dynamically, depending on the transferred value [90]. A detailed overview of possible forking attacks against Bitcoin-backed tokens in XCLAIM, their impact on security, as well as possible mitigations, is presented in Table II.

D. Compromise of Trusted Execution Environments

Should TEEs be employed by XCLAIM as an optimization, compromise of the hardware is a concern. To defend against compromise, XCLAIM offers two approaches: First, to protect against side-channel attacks [99], which may compromise the confidentiality and integrity of the TEE, side-channel resistant libraries and practical mitigation strategies can be employed inside the TEE (e.g. [44], [58], [84]). This makes side-channel attacks more expensive and less practical to exploit by an adversary. Second, should compromise of a single TEE still remain a concern, XCLAIM can use the replication strategies outlined for the Issuer and the chain relay as described in VII-B and VII-A, respectively. Replication in this manner would now require that a subset of TEEs must be compromised successfully, and in unison, in order to violate the security properties offered by XCLAIM. This makes attacks against the hardware much more unlikely, and difficult.

E. User Privacy

XCLAIM allows users to transfer cryptocurrencies in the form of tokens, without being visible on the backing cryptocurrency blockchain (e.g. Bitcoin). As such, only creator and redeemer must disclose their addresses/public keys, while all intermediates remain anonymous. However, since all trades are executed via the `treasury` contract, a publicly auditable log of all token transfers is created on the issuing blockchain (e.g. Ethereum). Furthermore, since *Issue* and *Redeem* require information on the source/target addresses, in a naïve implementation the creator and redeemer of tokens essentially link their accounts on the issuing and backing cryptocurrencies (e.g. link their Ethereum accounts to their Bitcoin addresses).

This can be partly avoided by allowing the redeemer to encrypt the public key, to which the units of the backing cryptocurrency are to be released to, with the Issuer public key before submitting the data to the `treasury` contract. As a result, optimistically only the Issuer will be able to link the redeemer's public keys across chains. A further mitigation for such privacy leaks is to introduce mixing services to the `treasury` contract, which make transaction linking difficult (e.g. [59], [74], [95]). Finally, ongoing research on zero knowledge protocols and ring signatures for permissionless cryptocurrencies (e.g. [37], [51], [82], [92]) may introduce new mechanisms for preserving privacy during token trading.

F. Collateral Deterioration due to Exchange Rate Volatility

One difficulty of our approach is needing to account for volatile exchange rates between cryptocurrencies. To overcome this, the collateral locked in the `treasury` contract must be sufficient enough to account for any potential price drops/surges. We therefore require *over-collateralization* by the Issuer. Furthermore, we also assume for simplicity that the Bitcoin–Ethereum exchange rate can be retrieved from a trusted oracle and all participants agree on it, e.g., Oraclize [24]. Although relying on a trusted third party for the exchange rate is problematic, as this introduces a single point of failure, we defer alternative, less centralized, solutions to future work.

VIII. RELATED WORK

A. Centralized and decentralized exchanges

While the core idea behind decentralized cryptocurrencies is to move away from centralized payment providers, centralized exchanges remain the preferred way to exchange cryptocurrencies. However, with increasing popularity and more money moving into the market, centralized service providers become a lucrative target for attacks, leading to a number of high-profile thefts throughout the past years [34], [36], [79], [85]. The most famous of these was Mt. Gox [21], [78].

As a response, decentralized exchanges, i.e., exchanges where no trust is required by the liquidity provider, have begun to operate [1], [2], [17]–[19], [32]. However, these protocols are mostly limited to facilitating trades of cryptocurrency tokens within a single cryptocurrency, e.g., ERC-20 tokens on Ethereum [46], and do not offer cross-chain exchanges.

B. Cross-Chain Communication Techniques

There are three fundamental approaches to achieving cross-chain communication [96]: (i) atomic swaps via hashed time-lock contracts; (ii) chain relays; and (iii) notary schemes.

Hashed Time-Lock Contracts (HTLC) can be used to achieve atomic cross-chain swaps [4], [31], [35], [61], [93], [94]. A formalisation of the concept is provided in [61]. 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.

TABLE II. Overview of possible forking attacks, their impact on the security of XCLAIM and possible mitigations.

		Bitcoin Fork (Relay Poisoning†)			Ethereum Fork		
		Attack	**Impact**	**Mitigation**	**Attack**	**Impact**	**Mitigation**
Issue	**HTLC**	Creator: Reverse $T^{btc}_{lock(HTLC)}$; Reverse $T^{btc}_{redeem(HTLC)}$	fake btc_{eth}	increase $t^{btc}_{contest}$	Issuer: reverse T^{eth}_{proof} or T^{eth}_{reveal}	btc theft	increase $t^{eth}_{contest}$; resubmit T^{eth}_{proof} + T^{eth}_{redeem}
	Collateralized Commit	Creator: Reverse T^{btc}_{lock}	fake btc_{eth}	increase $t^{btc}_{contest}$	Issuer: reverse T^{eth}_{commit} or T^{eth}_{proof}	btc theft	increase $t^{eth}_{contest}$; resubmit T^{eth}_{proof} + T^{eth}_{commit}
Trade		–	–	–	Sender: reverse T^{eth}_{offer}; Receiver: reverse T^{eth}_{match}	Tx fee loss	–
Redeem		Issuer: Reverse T^{btc}_{redeem}; (Redeemer: prevent T^{eth}_{proof} †)	btc theft; (eth col. theft†)	increase $t^{btc}_{contest}$	Redeemer: reverse T^{eth}_{redeem}	btc theft	increase $t^{eth}_{contest}$; resubmit T^{eth}_{redeem} & T^{eth}_{proof} ‡
Replace		Issuer: Reverse T^{btc}_{redeem}; (new candidate: prevent T^{eth}_{proof} †)	btc theft; (eth col. theft†)	increase $t^{btc}_{contest}$	New candidate: Reverse T^{eth}_{lock}	btc theft	increase $t^{eth}_{contest}$

†Relay poisoning is a special case of selfish mining; an adversary attempts to submit a conflicting version of the block chain to the chain relay. If successful, this allows to present arbitrary Bitcoin states to the `treasury` contract.

‡Note: cannot be effectively mitigated if the Redeemer is quick enough to spend the, now non-backed, btc_{eth} before the Issuer can resubmit the transaction inclusion proof for T^{btc}_{redeem}.

Chain relays can verify if transactions or blocks of the one blockchain have been included in the underlying data structure. However, chain relays require a sufficient set of operations to be supported by the underlying blockchain. 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. Notable chain relay projects include BTC Relay [6], PeaceRelay [26], Project Alchemy [28], Dogethereum [9], Cosmos "Peggy" [47] and Parity Bridge [25].

Notary schemes replace trust in a single entity by trust in a set of entities, i.e., a committee also referred to as a set of *validators*. Validators employ a consensus algorithm such as Tendermint [45] or HoneyBadger [77] 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 behaviour of the majority of validators. Notary schemes can be used for cross-chain communication; validators can be responsible for actively signing cross-chain transactions, as in *Liquid* [50], or attest to exchange partners that the trading conditions have been met, as in the case of *Interledger* [93]. Notary schemes however suffer from high overheads, due to the need to include a sufficient number of validators, and for those validators to constantly remain online. Furthermore, electing an honest validator committee under dynamically changing pseudonymous participants and not fully synchronous network assumptions, is a non-trivial problem.

C. Cryptocurrency-backed tokens

PeaceRelay [26] first proposed how cryptocurrency-backed tokens can be exchanged between two distributed ledgers supporting Turing complete programming languages, namely Ethereum and Ethereum Classic. Further works use cryptocurrency-backed tokens for value transfers between Ethereum and permissioned systems [27], [47]. In contrast to our scheme, however, all of these proposals require Turing complete programming capabilities on both source and receiving chains. Bentov et al. describe how cryptocurrency-backed tokens can be issued within a centralized exchange platform built on top of trusted execution environments (TEEs) [38]. In contrast, XCLAIM uses TEEs as an optional optimization only; their use is not required by default. Without TEEs, XCLAIM still offers trustless cryptocurrency-backed tokens.

IX. Conclusion

We presented XCLAIM, the first trustless protocol for constructing cryptocurrency-backed tokens on blockchains. XCLAIM is general in design and supports many existing blockchain implementations. It offers four sub-protocols: *Issue*, *Trade*, *Redeem* and *Replace*, including two variants for *Issue*; each providing different benefits and costs. We implemented XCLAIM to construct Bitcoin-backed tokens on Ethereum. We also presented two optional optimizations to XCLAIM that use TEEs to trade-off practicality, performance and efficiency against trust. Finally, we evaluated XCLAIM and compared the execution and storage costs of each design.

References

[1] "0xproject whitepaper," https://0xproject.com/pdfs/0x_white_paper.pdf, accessed: 2018-05-23.

[2] "Airswap," https://www.airswap.io/, accessed: 2018-07-30.

[3] "Bitcoin Developer Guide: Simplified Payment Verification (SPV)," https://bitcoin.org/en/developer-guide\#simplified-payment-verification-spv, accessed: 2018-05-16.

[4] "Bitcoin Wiki: Atomic cross-chain trading," https://en.bitcoin.it/wiki/Atomic_cross-chain_trading, accessed: 2018-05-16.

[5] "Bitcoin Wiki: Hashed Time-Lock Contracts," https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts, accessed: 2018-05-16.

[6] "Btc relay," https://github.com/ethereum/btcrelay, accessed 2018-04-17.

[7] "Cardano sl," https://github.com/input-output-hk/cardano-sl, accessed: 2018-07-30.

[8] "Counterparty," https://counterparty.io/, accessed: 2018-05-03.

[9] "Dogethereum," https://github.com/dogethereum/dogerelay, accessed 2018-04-17.

[10] "Erc20: Token standard," https://github.com/ethereum/EIPs/issues/20, accessed 2018-06-27.

[11] "Erc223: Token standard," https://github.com/ethereum/EIPs/issues/223, accessed 2018-06-27.

[12] "Erc721: Non-fungible token standard," https://github.com/namecoin/namecoin, accessed 2018-06-27.

[13] "Erc994: Delegated non-fungible token standard," https://github.com/ethereum/EIPs/issues/994, accessed 2018-06-27.

[14] "Etherdelta," https://etherdelta.com/, accessed: 2018-07-30.

[15] “Ethereum Classic,” https://github.com/ethereumproject, accessed: 2018-05-23.

[16] “Etherscan - ropsten testnet explorer,” https://ropsten.etherscan.io/, accessed: 2018-05-23.

[17] “Idex whitepaper,” https://idex.market/static/IDEX-Whitepaper-V0.7.5.pdf, accessed: 2018-05-23.

[18] “Komodo barterdex,” https://komodoplatform.com/decentralized-exchange/, accessed: 2018-05-23.

[19] “Kyber network whitepaper,” https://home.kyber.network/assets/KyberNetworkWhitepaper.pdf, accessed: 2018-05-23.

[20] “Monero reference implementation,” https://github.com/monero-project/monero, accessed: 2018-07-30.

[21] “Mt. gox,” https://www.mtgox.com/, accessed: 2018-05-23.

[22] “Neo whitepaper,” http://docs.neo.org/en-us/, accessed: 2018-07-30.

[23] “Omnilayer specification,” https://github.com/OmniLayer/spec, accessed: 2018-05-03.

[24] “Oraclize,” http://www.oraclize.it/, accessed: 2018-07-30.

[25] “Parity-Bridge,” https://github.com/paritytech/parity-bridge, accessed: 2018-05-21.

[26] “Peace relay,” https://github.com/loiluu/peacerelay, accessed 2018-04-17.

[27] “Poa bridge,” https://github.com/poanetwork/poa-bridge, accessed: 2018-05-23.

[28] “Project alchemy,” https://github.com/ConsenSys/Project-Alchemy, accessed 2018-04-17.

[29] “Solidity progamming language,” https://github.com/ethereum/solidity, accessed: 2018-07-30.

[30] “The witness algorithm: Privacy protection in a fully transparent system,” https://gist.github.com/gavofyork/dee1f3b727f691b381dc, accessed: 2018-07-30.

[31] “BIP199: Hashed Time-Locked Contract transactions,” https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki, 2017, accessed: 2018-05-16.

[32] “The Blocknet Design Specification (Whitepaper),” https://www.blocknet.co/wp-content/uploads/2018/04/whitepaper.pdf, 2018, accessed: 2018-05-21.

[33] ARM Ltd., “TrustZone,” https://www.arm.com/products/security-on-arm/trustzone. Accessed May 2017, 2017.

[34] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on ethereum smart contracts,” http://eprint.iacr.org/2016/1007, Oct 2016, accessed: 2016-11-08. [Online]. Available: https://eprint.iacr.org/2016/1007.pdf

[35] 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. [Online]. Available: https://blockstream.com/sidechains.pdf

[36] C. Baldwin, “Bitcoin worth $72 million stolen from bitfinex exchange in hong kong,” *Reuters*, accessed: 2018-05-23.

[37] 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*. IEEE, 2014, pp. 459–474. [Online]. Available: http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf

[38] 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. [Online]. Available: https://eprint.iacr.org/2017/1153.pdf

[39] 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. [Online]. Available: https://eprint.iacr.org/2016/919.pdf

[40] Bitcoin community, “Bitcoin-core source code,” https://github.com/bitcoin/bitcoin, accessed: 2015-06-30.

[41] “OP_RETURN,” https://en.bitcoin.it/wiki/OP_RETURN, bitcoin.it, accessed: 2018-05-23.

[42] “Pay to script hash,” https://en.bitcoin.it/wiki/Pay_to_script_hash, bitcoin.it, accessed: 2018-05-23.

[43] Blockchain.info, “Bitcoin currency statistics,” http://blockchain.info/, Blockchain.info, accessed: 2015-06-30.

[44] F. Brasser, S. Capkun, A. Dmitrienko, T. Frassetto, K. Kostiainen, U. Müller, and A.-R. Sadeghi, “Dr. sgx: Hardening sgx enclaves against cache attacks with data location randomization,” *arXiv preprint arXiv:1709.09917*, 2017.

[45] 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.

[46] 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. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[47] Cosmos Developer Team, “Peggy,” https://github.com/cosmos/peggy, accessed: 2018-05-23.

[48] 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. [Online]. Available: http://eprint.iacr.org/2017/573.pdf

[49] C. Decker and R. Wattenhofer, “Bitcoin transaction malleability and mtgox,” in *Computer Security-ESORICS 2014*. Springer, 2014, pp. 313–326. [Online]. Available: http://www.tik.ee.ethz.ch/file/7e4a7f3f2991784786037285f4876f5c/malleability.pdf

[50] 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.

[51] J. Eberhardt and S. Tai, “Zokrates-scalable privacy-preserving off-chain computations.”

[52] Ethereum community, “Ethereum: A secure decentralised generalised transaction ledger,” https://github.com/ethereum/yellowpaper, accessed: 2016-03-30.

[53] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” in *Financial Cryptography and Data Security*. Springer, 2014, pp. 436–454. [Online]. Available: http://arxiv.org/pdf/1311.0243

[54] J. A. Garay, A. Kiayias, and N. Leonardos, “The bitcoin backbone protocol with chains of variable difficulty,” http://eprint.iacr.org/2016/1048.pdf, 2016, accessed: 2017-02-06. [Online]. Available: http://eprint.iacr.org/2016/1048.pdf

[55] Gavin Wood, “Ethereum: A secure decentralised generalised transaction ledger eip-150 revision (759dccd - 2017-08-07),” 2017, accessed: 2018-01-03. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[56] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, “On the security and performance of proof of work blockchains,” https://eprint.iacr.org/2016/555.pdf, 2016, accessed: 2016-08-10. [Online]. Available: https://eprint.iacr.org/2016/555.pdf

[57] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun, “Tampering with the delivery of blocks and transactions in bitcoin,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 692–705. [Online]. Available: https://eprint.iacr.org/2015/578.pdf

[58] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, “Strong and efficient cache side-channel protection using hardware transactional memory,” in *USENIX Security Symposium*, 2017, pp. 217–233.

[59] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, “Tumblebit: An untrusted bitcoin-compatible anonymous payment hub,” 2016, accessed: 2017-09-29. [Online]. Available: https://eprint.iacr.org/2016/575.pdf

[60] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, “Eclipse attacks on bitcoin’s peer-to-peer network,” in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-heilman.pdf

[61] M. Herlihy, “Atomic cross-chain swaps,” arXiv:1801.09515, 2018, accessed:2018-01-31. [Online]. Available: https://arxiv.org/pdf/1801.09515.pdf

[62] J. Hosp, T. Hoenisch, and P. Kittiwongsunthorn, https://www.comit.network/doc/COMIT%20white%20paper%20v1.0.2.pdf, 2017, accessed: 2018-07-30.

[63] Intel Corp., "Software Guard Extensions Programming Reference, Ref. 329298-002US," https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, 2014. [Online]. Available: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[64] Intel Inc., "Intel Software Guard Extensions Remote Attestation End-to-End Example," 2016, https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-example. Accessed May 2017.

[65] Johnson, Simon et al., "Intel® Software Guard Extensions: EPID Provisioning and Attestation Services," https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services, 2016.

[66] D. Kaplan, J. Powell, and T. Woller, "AMD Memory Encryption," *White paper*, 2016.

[67] 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. [Online]. Available: https://eprint.iacr.org/2017/963.pdf

[68] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," Cryptology ePrint Archive, Report 2016/889, 2016, https://eprint.iacr.org/2016/889.

[69] 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: USENIX Association, Aug. 2016. [Online]. Available: http://arxiv.org/pdf/1602.06997.pdf

[70] J. Kwon and E. Buchman, "Cosmos: A network of distributed ledgers," https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md, 2015.

[71] L. Lamport, "The part-time parliament," vol. 16, no. 2. ACM, 1998, pp. 133–169. [Online]. Available: http://research.microsoft.com/en-us/um/people/lamport/pubs/lamport-paxos.pdf

[72] S. D. Lerner, "Rootstock: Bitcoin powered smart contracts," https://www.rsk.co/, 2015.

[73] Litecoin community, "Litecoin reference implementation," github.com/litecoin-project/litecoin, accessed: 2017-06-30.

[74] S. Meiklejohn and R. Mercer, "Möbius: Trustless tumbling for transaction privacy," Cryptology ePrint Archive, Report 2017/881, 2017, accessed:2017-09-26. [Online]. Available: http://eprint.iacr.org/2017/881.pdf

[75] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1987, pp. 369–378.

[76] S. Micali, "Algorand: The efficient and democratic ledger," http://arxiv.org/abs/1607.01341, 2016, accessed: 2017-02-09. [Online]. Available: https://arxiv.org/pdf/1607.01341.pdf

[77] 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*. ACM, 2016, pp. 31–42.

[78] T. Moore and N. Christin, "Beware the middleman: Empirical analysis of Bitcoin-exchange risk," in *Proceedings of IFCA Financial Cryptography'13*, Okinawa, Japan, April 2013. [Online]. Available: https://www.andrew.cmu.edu/user/nicolasc/publications/MC-FC13.pdf

[79] Moore, Tyler and Christin, Nicolas, "Beware the middleman: Empirical analysis of bitcoin-exchange risk," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 25–33.

[80] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, Dec 2008, accessed: 2015-07-01. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[81] Namecoin community, "Namecoin reference implementation," https://github.com/namecoin/namecoin, accessed 2017-06-30.

[82] S. Noether, "Ring signature confidential transactions for monero." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1098, 2015.

[83] K. Okupski, "Bitcoin protocol specification," https://github.com/minium/Bitcoin-Spec, accessed: 2014-10-14.

[84] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer, "Varys: Protecting {SGX} enclaves from practical side-channel attacks," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 227–240.

[85] J. Pagliery, "Another bitcoin exchange goes down," *CNN Tech*, accessed: 2018-05-23.

[86] R. Pass and E. Shi, "Hybrid consensus: Scalable permissionless consensus," https://eprint.iacr.org/2016/917.pdf, Sep 2016, accessed: 2016-10-17. [Online]. Available: https://eprint.iacr.org/2016/917.pdf

[87] M. Rosenfeld, "Overview of colored coins," https://bitcoil.co.il/BitcoinX.pdf, 2012, accessed: 2016-03-09. [Online]. Available: https://bitcoil.co.il/BitcoinX.pdf

[88] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," http://arxiv.org/pdf/1507.06183.pdf, 2015, accessed: 2016-08-22. [Online]. Available: http://arxiv.org/pdf/1507.06183.pdf

[89] I. Sergey, A. Kumar, and A. Hobor, "Scilla: a smart contract intermediate-level language," arXiv:1801.00687, 2018, accessed:2018-01-08. [Online]. Available: https://arxiv.org/pdf/1801.00687.pdf

[90] Y. Sompolinsky and A. Zohar, "Bitcoin's security model revisited," http://arxiv.org/pdf/1605.09193, 2016, accessed: 2016-07-04. [Online]. Available: http://arxiv.org/pdf/1605.09193.pdf

[91] M. Spoke and Nuco Engineering Team.

[92] S.-F. Sun, M. H. Au, J. K. Liu, T. H. Yuen, and D. Gu, "Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero," Cryptology ePrint Archive, Report 2017/921, 2017, accessed:2017-09-26. [Online]. Available: http://eprint.iacr.org/2017/921.pdf

[93] S. Thomas and E. Schwartz, "A protocol for interledger payments," *URL https://interledger.org/interledger.pdf*, 2015.

[94] TierNolan, "Atomic swaps using cur and choose," https://bitcointalk.org/index.php?topic=1364951, 2016, accessed: 2018-05-16.

[95] M. Tran, L. Luu, M. S. Kang, I. Bentov, and P. Saxena, "Obscuro: A bitcoin mixer using trusted execution environments," Cryptology ePrint Archive, Report 2017/974, 2017, accessed:2017-10-06. [Online]. Available: http://eprint.iacr.org/2017/974.pdf

[96] n. h. Vitalik Buterin, year =2016, "Chain interoperability."

[97] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White Paper*, 2015.

[98] K. Wüst and A. Gervais, "Ethereum eclipse attacks," ETH Zurich, Tech. Rep., 2016.

[99] 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*. IEEE, 2015, pp. 640–656.

[100] 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.