XCLAIM: Trustless, Interoperable, Cryptocurrency-Backed Assets

Alexei Zamyatin[†‡], Dominik Harz[†], Joshua Lind[†], Panayiotis Panayiotou[†], Arthur Gervais[†], William Knottenbelt[†]
[†] Imperial College London [‡] SBA Research

***Abstract*—Building trustless cross-blockchain trading protocols is challenging. Centralized exchanges thus remain the preferred route to execute transfers across blockchains. However, these services require trust and therefore undermine the very nature of the blockchains on which they operate. To overcome this, several decentralized exchanges have recently emerged which offer support for atomic cross-chain swaps (ACCS). ACCS enable the trustless exchange of cryptocurrencies across blockchains, and are the only known mechanism to do so. However, ACCS suffer significant limitations; they are slow, inefficient and costly, meaning that they are rarely used in practice.**

We present XCLAIM: the first generic framework for achieving trustless and efficient cross-chain exchanges using *cryptocurrency-backed assets* (CBAs). XCLAIM offers protocols for issuing, transferring, swapping and redeeming CBAs securely in a non-interactive manner on existing blockchains. We instantiate XCLAIM between Bitcoin and Ethereum and evaluate our implementation; it costs less than USD 0.50 to issue an arbitrary amount of Bitcoin-backed tokens on Ethereum. We show XCLAIM is not only faster, but also significantly cheaper than atomic cross-chain swaps. Finally, XCLAIM is compatible with the majority of existing blockchains without modification, and enables several novel cryptocurrency applications, such as cross-chain payment channels and efficient multi-party swaps.

***Index Terms*—blockchain, interoperability, CBA, Bitcoin, Ethereum**

I. INTRODUCTION

Blockchain-based cryptocurrencies enable secure and trustless transactions between parties. As a result, they have gained widespread adoption and popularity in recent years; there are currently over 2 000 different cryptocurrencies in operation, with a total market cap of USD 135bn [48]. However, despite a growing and thriving ecosystem, cryptocurrencies continue to operate in complete isolation from one another: blockchain protocols provide no means by which to communicate or exchange data with external systems. Hence, achieving interoperability between blockchains remains an open challenge.

Centralized exchanges thus remain the preferred route to execute fund transfers and exchanges across blockchains. However, these services require trust and therefore undermine the very nature of the cryptocurrencies on which they operate, making them vulnerable to attacks [32], [35], [89], [95]. To overcome this, *decentralized exchanges* [1], [2], [13], [16], [18] (*DEXs*) have recently emerged, removing the need to trust centralized intermediaries for blockchain transfers. However, the vast majority of DEXs only enable the exchange of *cryptocurrency-assets* within a single blockchain, i.e., they do not operate across blockchains (*cross-chain*). As such, it is only a handful of platforms [17], [30] that actually support cross-chain exchanges through the use of *atomic cross-chain swaps* (ACCS) [4], [33], [69], [105].

ACCS enable secure cross-chain exchanges, e.g. using *hashed timelock contracts* (HTLCs) [5], [47]. At present, they are the only mechanism to do this without necessitating trust. Unfortunately, they require several strong assumptions to maintain security, thus limiting their practicality: they are interactive, requiring all parties to be online and actively monitor all involved blockchains during execution; they require synchronizing clocks between blockchains and rely on pre-established secure out-of-band communication channels. In addition, they also incur long waiting periods between transfers and suffer the limitation that for every cross-chain swap, four transactions need to occur, two on each blockchain. This makes them expensive, slow and inefficient.

We therefore present XCLAIM (pronounced *cross-claim*): the first generic framework for achieving trustless cross-chain exchanges using *cryptocurrency-backed assets*. In XCLAIM, blockchain-based assets can be securely constructed and one-to-one backed by other cryptocurrencies, for example, Bitcoin-backed tokens on Ethereum. Through the secure issuance, swapping, and redemption of these assets, users can perform cross-chain exchanges in a trustless and non-interactive manner, overcoming the limitations of existing solutions.

To achieve this, XCLAIM exploits publicly verifiable *smart contracts* to remove the need for trusted intermediaries and leverages *chain relays* [6], [33], [76], [106] for cross-chain state verification. Using these building blocks, XCLAIM constructs a publicly verifiable audit log of user actions on both blockchains and employs *collateralization* and *punishments* to enforce the correct behavior of participants. Thereby, XCLAIM follows a *proof-or-punishment* approach, i.e., participants must proactively prove adherence to system rules.

Due to its simple and efficient design, XCLAIM enables several novel applications, such as: (i) *cross-chain payment channels*, where users can exchange payments *off-chain* across different blockchains in a trustless manner; (ii) *temporary transaction offloading*, where users temporarily tokenize their cryptocurrency on other blockchains to overcome network congestion and high fees; and (iii) *N-way and multi-party atomic swaps* allowing efficient and complex atomic swaps. Finally, as XCLAIM maintains compatibility with existing standardized asset interfaces [10], [11], the issued assets are tradeable via existing decentralized exchanges, enabling these

exchanges to operate cross-blockchain.

This paper makes the following contributions:

- We define the notion of cryptocurrency-backed assets for blockchains and formulate goals for security and functionality (Section III). We then present XCLAIM, a practical and secure system to construct cryptocurrency-backed assets without trusted intermediaries (Section IV-V).
- We provide a formal protocol specification for XCLAIM and analyze in detail the requirements for the underlying blockchains (Section VI). While the blockchain used to issue cryptocurrency-backed assets must support smart contracts, XCLAIM requires only base-ledger functionality on the backing side, supporting practically all cryptocurrencies.
- We implement XCLAIM(BTC,ETH), to the best of our knowledge, the first system for trustlessly issuing, transferring, swapping and redeeming Bitcoin-backed tokens on Ethereum (Section VIII). In our prototype, it costs USD 0.47 to issue, USD 0.04 to transfer, USD 0.19 to atomically swap and USD 0.49 to redeem an arbitrary amount of cross-chain tokens[1]. We compare performance and costs to HTLC atomic swaps and show XCLAIM is 95.7% faster and 65.4% cheaper for 1000 swaps.
- Finally, we present and describe several novel applications enabled exclusively by XCLAIM, such as cross-chain payment channels and efficient N-way and multi-party atomic swaps (Section IX).

II. BACKGROUND

A. Blockchains and Decentralized Ledgers

Bitcoin [90] allows users to hold and exchange funds in a decentralized manner, without trusting a third party. It achieves this through a peer-to-peer replicated state machine that maintains a global append-only ledger. This ledger maintains the history of all transactions in the network, and is constructed through a sequence of blocks, chained together via the hashes of their predecessors; thus forming the *blockchain*. To append to the blockchain, Bitcoin operates a random leader election protocol using *Proof-of-Work* (PoW) [34], [54], where peers compete to solve a computationally expensive puzzle. This approach, known as *Nakamoto consensus*, achieves agreement in permissionless settings, i.e., where the set of network peers is dynamically changing. Bitcoin, and other permissionless blockchains, therefore provide only *weak identities*, i.e., multiple peers may be controlled by the same entity.

Inspired by Bitcoin, numerous other blockchain-based cryptocurrencies have recently emerged. For example, Ethereum [46], which extends the basic ledger functionality of Bitcoin by adding a Turing-complete programming language to transactions. This allows users to express and create *smart contracts*: programs that operate as independent and automatic entities on the blockchain. Other systems offer alternative consensus mechanisms, such as *Proof-of-Stake* (PoS), where the rate-limiting resource is the currency itself, rather than computational power [44], [75], [77]. For brevity, we use *blockchain* and *chain* as synonyms in the rest of this paper.

[1]According to exchange rates as of 30 November 2018.

B. Overlay Protocols, Colored Coins and Tokens

Overlay protocols and colored coins leverage the infrastructure of existing blockchains by extending blockchains with additional features. Such protocols [8], [21], [97], are supported by most existing blockchains today, as they only require: (i) the inclusion of data in blockchain transactions and (ii) eventual agreement on their ordering [114]. As such, overlay protocols enable the creation of cryptocurrencies without bootstrapping a new dedicated blockchain. These are referred to as *cryptocurrency tokens* and can be used to quantify both fungible and non-fungible assets [10], [12]. Profiting from the expressiveness of Ethereum's scripting language [63], a plethora of such tokens has emerged, rivaling cryptocurrencies both in number and market capitalization [48], [112].

III. SYSTEM OVERVIEW

In this section we first define cryptocurrency-backed assets. We then present the system model and actors in XCLAIM, as well as the network and threat models. Finally, we present XCLAIM's system goals.

A. Cryptocurrency-backed Assets (CBA)

Definition. We define *cryptocurrency-backed assets* (CBAs) as assets deployed on top of a blockchain I that are backed by a cryptocurrency on blockchain B. We denote assets in I as i, and cryptocurrency on B as b. We use $i(b)$ to further denote when an asset on I is backed by b. We extend the definition of assets by Androulaki et al. [31] and describe a CBA through the following fields:

- *issuing blockchain*, the blockchain I on which the CBA $i(b)$ is issued.
- *backing blockchain*, the blockchain B that backs $i(b)$ using cryptocurrency b.
- *asset value*, the units of the backing cryptocurrency b used to generate the asset $i(b)$.
- *asset redeemability*, whether or not $i(b)$ can be redeemed on B for b.
- *asset owner*, the current owner of $i(b)$ on I.
- *asset fungibility*, whether or not units of $i(b)$ are interchangeable.

We define a CBA as *symmetric* if the total amount of backing units b is equivalent to the total amount of issued units $i(b)$, i.e., $|b| = |i(b)|$, and as *asymmetric* if the CBA exhibits an alternate backing rate, i.e., $|b| \neq |i(b)|$. In XCLAIM, we restrict CBAs to be symmetric cryptocurrency-backed assets. Moreover, CBAs can be divided and merged back together as necessary. We defer the analysis of alternate CBAs, such as asymmetric and non-fungible CBAs, to future work.

B. System Model and Actors

XCLAIM operates between a backing blockchain B of cryptocurrency b and an issuing blockchain I with underlying

CBA $i(b)$. To operate CBAs, XCLAIM further differentiates between the following actors in the system:

- **CBA Requester.** Locks b on B to request $i(b)$ on I.
- **CBA Sender.** Owns $i(b)$ and transfers ownership to another user on I.
- **CBA Receiver.** Receives and is assigned ownership over $i(b)$ on I.
- **CBA Redeemer.** Destroys $i(b)$ on I to request the corresponding amount of b on B.
- **CBA Backing Vault ($vault$).** A (non-trusted) intermediary liable for fulfilling redeem requests of $i(b)$ for b on B.
- **Issuing Smart Contract (iSC).** A public smart contract responsible for managing the correct issuing and exchange of $i(b)$ on I. The iSC ensures correct behaviour of the $vault$.

To perform these roles in XCLAIM, actors are identified on a blockchain using their public/private key pairs. As a result, the $requester$, $redeemer$ and $vault$ must maintain key pairs for both blockchains B and I. The $sender$ and $receiver$ only need to maintain key pairs for the issuing blockchain I. iSC exists as a publicly verifiable smart contract on I.

C. Blockchain Model and Assumptions

XCLAIM establishes communication between two independent blockchains with likely varying consensus mechanisms and trust models. Therefore, should either blockchain B or I be compromised by an adversary, the correct functionality of XCLAIM cannot be guaranteed. As such, we assume that the proportion of consensus participants f (or computational power α in the case of Nakamoto consensus) corrupted by an adversary for both B and I is bounded by the threshold necessary to ensure safety and liveness for the underlying blockchains. For example, in Nakamoto consensus based blockchains, e.g. Bitcoin [90] and Ethereum [46], we assume $\alpha \leq 33\%$ [61], [66], [98]. In Byzantine fault tolerant settings using e.g. Proof-of-Stake, such as [44], [75], we assume $f < n/3$ where n is the total number of consensus participants.

These assumptions guarantee that the number of maliciously generated blocks is upper-bounded by $\frac{f}{n-f}$ (*chain quality property*) [62]. As such, we assume the probability of blockchain reorganizations therefore drops exponentially with a security parameter $k \in \mathbb{N}$ (*common-prefix property*) [62], [90], [96]. We measure k in blocks and denote k^B for blockchain B and k^I for blockchain I. Specifically, we say a transaction is *securely*[2] included in the underlying blockchain if, given the current blockchain head at position $h \in \mathbb{N}$, the transaction is included in a block at position $j \in \mathbb{N}$, such that $h - j \geq k$.

For existing blockchains, the delay Δ from transaction broadcast to *secure* inclusion depends on the block generation rate τ, i.e., the number of blocks created per round (cf. *chain growth property* [62], [96]). Block generation rates are however non-deterministic for most blockchains[3] and differ from system to system. To ensure secure communication between B and I, we thus formulate assumptions for both Δ^B and Δ^I, i.e., argue on a ratio $r_\tau = \mathbb{E}(\tau^B)/\mathbb{E}(\tau^I)$ between the expected generation rates of B and I. We assume a known upper bound $\overline{r_\tau}$, i.e., increases and decreases of the rate at which blocks are found in B remain within a known bound in relation to the rate of I (and vice-versa).

D. Threat Model and Network Assumptions

We assume that the cryptographic primitives of B and I are secure. We further assume that adversaries are computationally bounded and economically rational, and we model economic incentives as a *zero-sum-game* [108] across B and I. As such, adversaries may perform arbitrary actions to maximize their economic value, such as delay or censor transactions, read unconfirmed transactions in the network or in mining pool memory, and perform Sybil attacks [53].

To manage exchange rate fluctuations between B and I, we assume an oracle $\mathcal{O}$ provides the iSC with the exchange rate $\varepsilon_{(i,b)} \in \mathbb{R}_{\geq 0}$ for cryptocurrencies i to b. We further assume that there exists a lower bound $\min(\varepsilon_{(i,b)})$ for the value of i with respect to b, below which adhering to protocol rules no longer represents the equilibrium strategy of rational adversaries [92], [93]. Therefore, in case of extreme devaluation of i, we assume a delay $\Delta_{\min(\varepsilon)} < \Delta^I$, i.e., that honest users can include a transaction in I before $\varepsilon_{(i,b)} < \min(\varepsilon_{(i,b)})$.

For the underlying network, we make the same assumptions as in prior work [60], [75], [79], i.e., we assume (i) honest nodes are well connected and (ii) communication channels between these nodes are (semi-)synchronous. Specifically, transactions broadcast by users are received by (honest) consensus participants[4] within a known maximum delay Δ_{tx} [96], i.e., Δ^B_{tx} for chain B and Δ^I_{tx} for chain I. Note: $\Delta^I = \frac{k^I}{\tau^I} + \Delta^I_{tx}$ (analogous for B).

E. System Goals

Under the blockchain, network and threat models specified above, in Sections III-B-III-D, we derive the following desirable *security properties* for XCLAIM with regards to CBAs:

- **Auditability.** Any user with read access to blockchains B and I can audit the operation of XCLAIM and detect protocol failures.
- **Consistency.** No CBA units $i(b)$ can be issued without the equivalent amount of backing currency b being locked, i.e., that $|b| = |i(b)|$.
- **Redeemability.** Any user can redeem CBAs $i(b)$ for backing currency b on B, or be reimbursed with equivalent economic value on I.
- **Liveness.** Any user in XCLAIM can issue, transfer and swap CBAs without requiring a third party, i.e., liveness relies only on the secure operation of B and I.
- **Atomic Swaps.** Users can atomically swap XCLAIM CBAs against other assets on I or the native currency i.

Furthermore, we derive the following desirable *functional properties* for XCLAIM:

[2]Note: k is recommended to be set as a function of transferred value [100].

[3]E.g., for Nakamoto consensus the time between generated blocks is exponentially distributed [51], [80], [90].

[4]E.g. miners in the case of Nakamoto consensus [90].

- **Scale-out.** The total amount of CBAs available for circulation increases with the total amount[5] of backing currency locked up in blockchain B. Any user can contribute to this amount by assuming the role of the *vault*.
- **Compatibility.** XCLAIM does not rely on a single cryptocurrency implementation with a set of specific features. Instead, it allows to issue assets $i(b)$ on any blockchain I that supports smart contracts[6], backed by any blockchain B that supports only basic fund transfers between parties. This enables XCLAIM to maintain backward compatibility with existing blockchains that do not provide smart contract support, such as Bitcoin.

IV. STRAWMAN SOLUTION AND DESIGN ROADMAP

In this section we present a strawman solution, CENTRALCLAIM, that outlines how a CBA-based system with the actors defined in Section III-B might operate. We use CENTRALCLAIM to highlight the challenges faced by XCLAIM in achieving the goals from Section III-E. Finally, we lay out a design roadmap for the secure design of XCLAIM. We present XCLAIM in Section V.

A. Strawman Solution

CENTRALCLAIM proposes the use of a single trusted intermediary on the backing blockchain B that takes the role of the *vault*. The iSC is a smart contract deployed on the issuing blockchain I. The *vault* is registered with the iSC, i.e., the iSC can verify the *vault*'s digital signature and knows the *vault*'s public key. As defined in Section III-B, I is responsible for managing the correct issuing and exchange of $i(b)$ on I.

We assume a user Alice controls units of b on a blockchain B, while a user Dave controls units of i on a blockchain I. Alice wishes to create B-backed assets $i(b)$ and transfer them to Dave on I. Dave, at some later point in time, wishes to redeem his units of $i(b)$ for the corresponding amount of b.

To achieve this, CENTRALCLAIM offers four protocols: *Issue*, *Transfer*, *Swap* and *Redeem*. For simplicity, we omit any processing fees charged by the *vault* or the iSC for the use of the service. We also omit the cost of transaction fees on the underlying blockchains B and I.

Protocol: Issue. Alice (*requester*) locks units of b with the *vault* on B to create $i(b)$ on I:

1) *Setup.* First, Alice verifies the iSC smart contract is available on chain I, i.e., the issuing blockchain, and identifies the single backing intermediary on B, i.e., the *vault*.
2) *Lock.* Alice generates a new public/private key pair on I and locks funds b with the *vault* on B in a publicly verifiable manner, i.e., by sending b to the *vault*. As part of locking these funds with the *vault*, Alice also specifies where the to-be-generated $i(b)$ should be sent, i.e., Alice associates her public key on I with the transfer of b to the *vault*.
3) *Create.* The *vault* confirms to the issuing smart contract iSC via a signed message that Alice has correctly locked her funds and forwards Alice's public key on I to the iSC. The iSC verifies the *vault*'s signature, then creates and sends $i(b)$ to Alice, such that $|i(b)| = |b|$.

Protocol: Transfer. Alice (*sender*) transfers $i(b)$ to Dave (*receiver*) on I:

1) *Transfer.* Alice notifies the iSC that she wishes to transfer her $i(b)$ to Dave (public key) on I. The state of the iSC is updated and Dave becomes the new owner of $i(b)$.
2) *Witness.* The *vault* witnesses the change of ownership on I through iSC, and no longer allows Alice to withdraw the associated amount of locked b on B. The process for any further transfers from Dave to other users is analogous.

Protocol: Swap. Alice (*sender*) atomically swaps $i(b)$ against Dave's (*receiver*) i on I:

1) *Lock.* Alice locks $i(b)$ with the iSC.
2) *Swap.* If Dave locks the agreed upon units of i (or any other asset on I) with the iSC within delay Δ_{swap}, the iSC updates the balance of Dave, making him the new owner of $i(b)$, and assigns Alice ownership over i.
3) *Revoke.* If Dave does not correctly lock i with the iSC within Δ_{swap}, the iSC releases locked $i(b)$ to Alice.
4) *Witness.* If the swap is successful, the *vault* witnesses the change of ownership of $i(b)$ and no longer allows Alice to redeem the associated amount.

Protocol: Redeem. Dave (*redeemer*) locks $i(b)$ with the iSC on I to receive b from the *vault* on B; $i(b)$ is then destroyed:

1) *Setup.* Dave creates a new public/private key pair on B.
2) *Lock.* Next, Dave locks $i(b)$ with the iSC on I and requests the redemption of $i(b)$. Thereby, Dave also specifies his new public key on B as the target for the redeem.
3) *Release.* The *vault* witnesses the locking and redemption request of $i(b)$ on I and releases funds b to Dave's specified public key on B, such that $|b| = |i(b)|$.
4) *Burn.* Finally, the *vault* confirms with the iSC that b was redeemed on B, and the iSC destroys, or *burns*, the locked $i(b)$ on I.

B. Strawman Limitations and Properties

While CENTRALCLAIM, as presented in Section IV-A, already provides sufficient functionality for issuing, transferring, swapping and redeeming CBAs, it does not achieve all the goals defined in Section III-E. Namely, it does not achieve **Consistency**, **Redeemability** and **Liveness**. This is because CENTRALCLAIM is inherently centralized around a single *vault*, and trusts the *vault* to behave correctly. This is fundamentally insecure, however, as the *vault* is economically rational and therefore incentivized to misbehave.

For example, the *vault* is trusted to monitor the backing chain B for newly created locks of b and notify the iSC via a signed transaction on I. Should the *vault* fail to do this, it can steal the locked funds and violate **Consistency**. Similarly, the *vault* is trusted to release the correct amount of b on B when a *redeemer* requests the redemption of

[5] Specifically, locked collateral. To become a *vault*, a user must provide at a pre-defined minimal amount of collateral in i; cf. Section V-E.

[6] Turing completeness is not required, as discussed in Section VI-B.

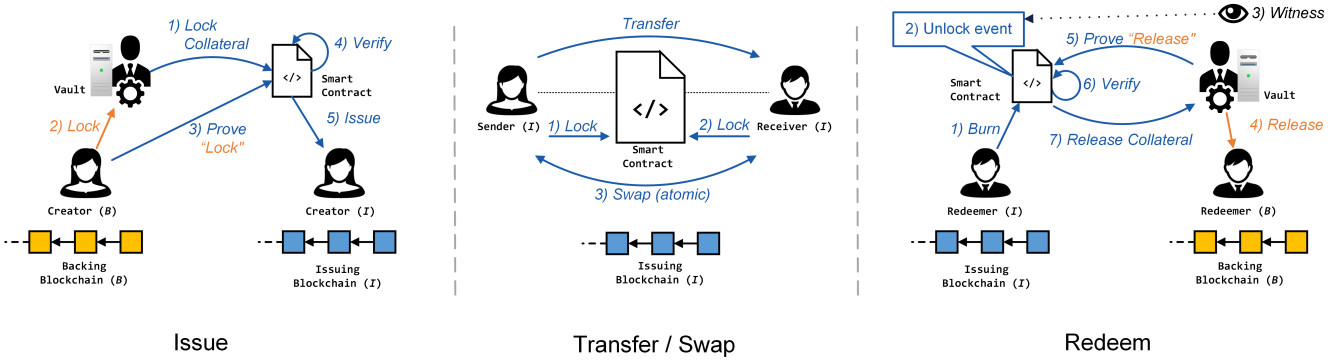

Fig. 1. High-level overview of the *Issue*, *Swap* and *Redeem* protocols in XCLAIM's (under successful execution). All parties interact with the iSC, creating a publicly verifiable audit log. Correct behavior is enforced by (i) over-collateralizing the *vault* and (ii) cross-chain transaction inclusion proofs. When issuing, the *requester* proves correctness of the lock making *Issue non-interactive*. Safety is ensured by forcing the *vault* to *proactively* prove correctness of the *Redeem* process. As a result, XCLAIM enforces *Transfer* and *Swap* occur consistently on the backing (B) and issuing (I) blockchains.

$i(b)$. Failing to do this allows the *vault* to steal the locked b and break **Redeemability**. Finally, CENTRALCLAIM also inherently violates **Liveness**; it exhibits a single point of failure, as backing-funds are locked with a single intermediary, the *vault*. The *vault* is therefore assumed to be interactive, i.e., always online. As such, even in the case that the *vault* behaves honestly, CENTRALCLAIM can fail to achieve **Liveness**, e.g. due to denial-of-service and eclipse attacks [68] on the *vault*.

Surprisingly however, CENTRALCLAIM already exhibits significant advantages over centralized systems offering digital tokens backed by real-world assets, e.g. the US dollar [29]. Specifically, CENTRALCLAIM achieves **Auditability**, allowing users to detect if any actors misbehave, and **Atomic Swaps**, enabling secure swaps of assets and cryptocurrency.

It is easy to see how CENTRALCLAIM achieves **Auditability**: for successful execution, *Issue*, *Transfer*, *Swap* and *Redeem* all require secure transaction inclusion in blockchains B and I with security parameters k^B and k^I. Users can therefore detect both crash and Byzantine failures if incorrect transactions are published or transactions are missing from each blockchain. As such, an adversary could only interfere with this by: (i) preventing transaction inclusion in B and I; or (ii) stopping a user from receiving messages broadcast by other nodes on B and I. Both attack vectors are not possible under the blockchain and (semi-)synchronous network models.

Likewise, it is easy to see how CENTRALCLAIM achieves **Atomic Swaps**: the *Swap* protocol is exclusively executed by the iSC on I. Specifically, to initiate *Swap*, the *sender* locks $i(b)$ in the iSC via a transaction on I. By construction, the iSC will only release $i(b)$ to the receiver if the *receiver* locks the correct amount of i with the iSC within Δ_{swap}. Otherwise, $i(b)$ is released back to the *sender*. An adversary cannot therefore prevent the atomicity of *Swap*: this would require tampering with the iSC, which is not possible under the assumptions of the blockchain and threat models.

Finally, CENTRALCLAIM also provides **Compatibility**, as the only operation executed on the backing chain B is a simple transfer of funds to the *vault*. A detailed overview of operational requirements is provided in Section VI-B.

C. XCLAIM *Design Roadmap*

To address the security challenges and limitations of CENTRALCLAIM, we outline the design roadmap for XCLAIM and introduce the building blocks used in its construction:

1) In Section V-B, we remove the trust required by the *vault* during the issuing of CBAs, and make the issuing process non-interactive, thus achieving **Consistency** and **Liveness.** For this, we use *chain relays* to allow programmatic verification of transaction inclusion proofs for B on I and require all parties to *proactively* prove correct behavior.
2) In Section V-C, we show how to incentivize the correct behaviour of the *vault* during CBA redemption through the introduction of *collateralization* and *punishments*, enforcing a *proof-or-punishment* model. We highlight race conditions during *Issue* due to collateralization, and present two effective mitigations: (i) *deferred collateral withdrawal* and (ii) *collateralized issue commitments*. Hence, we achieve **Redeemability** under a *fixed* exchange rate $\varepsilon_{(i,b)}$.
3) In Section V-D, we show how to prevent collateral deterioration due to exchange rate fluctuations by introducing (i) *over-collateralization*, (ii) *collateral adjustment* and (iii) *automatic liquidation*. As a result, XCLAIM achieves **Redeemability** under *non-constant* exchange rates.
4) Finally, in Section V-E, we achieve **Scale-Out** by removing single points of failure in CENTRALCLAIM. We do this by making XCLAIM a multi-*vault* system where *any user* can assume the role of the *vault*.

V. XCLAIM SECURE DESIGN

This section presents the secure design of XCLAIM. We first provide the high-level overview and intuition behind XCLAIM, and then follow the technical roadmap outlined in Section IV to provide a detailed system description.

A. XCLAIM *Overview*

XCLAIM overcomes the limitations of CENTRALCLAIM through three primary techniques: (i) constructing secure audit logs on both the backing blockchain B and the issuing blockchain I to trace all actions in the system; (ii) transaction inclusion proofs via chain relays to prove correct behaviour on the backing blockchain B to the iSC; and (iii) collateralization to incentivize correct behaviour through *proof-or-punishment*. We provide a brief overview and intuition for XCLAIM below. Figure 1 illustrates the *Issue*, *Transfer*, *Swap* and *Redeem* protocols in XCLAIM, while the design of the issuing smart contract iSC is shown in Figure 2.

Similar to CENTRALCLAIM, in XCLAIM, funds on the backing blockchain B are secured by backing intermediaries, *vaults*. The *vaults* store locked coins b on blockchain B and handle issue and redeem requests. To avoid necessitating trust in the *vault* however, XCLAIM uses collateral to incentivize behaviour; XCLAIM requires actors, such as the *vault*, to deposit collateral on blockchain I, owned by the iSC. Every action in XCLAIM is then logged securely via the iSC and misbehaving actors, such as the *vault*, are punished by *slashing* collateral belonging to them, and reimbursing wronged actors. XCLAIM ensures deposited collateral is always sufficient, even in case of exchange rate fluctionations between b and i.

For the iSC to ensure that correct behaviours have taken place on blockchain B, where the iSC does not have direct visibility, XCLAIM uses *chain relays*. Chain relays provide external blockchain data, such as the transactions in blockchain B, to the iSC executing on I. As such, the iSC can trace every action by every actor in the system across blockchains. Actors in XCLAIM therefore proactively prove their honest behaviour to the iSC via the chain relay; failure to do so results in punishment. By combining secure audit logs, chain relays, and collateralization in this way, XCLAIM can overcome the limitations of CENTRALCLAIM, and achieve the properties defined in Section III-E.

B. *Chain Relays: Cross-Chain State Verification*

As outlined in Section V-A, XCLAIM employs chain relays [6], [76], [106] to provide data from the backing blockchain B to the iSC on the issuing blockchain I. We use chain relays to make the issuing of assets $i(b)$ on I non-interactive. For this, XCLAIM introduces a chainRelay component to the smart contract iSC (cf. Figure 2). The chainRelay is capable of interpreting the state of the backing blockchain B and provides functionality comparable to an SPV or light client [3], [19], [23], [33]. That is, a chainRelay stores and maintains block headers from blocks in B on I, and provides two functionalities to the iSC: *Transaction inclusion verification* and *Consensus verification*:

- **Transaction inclusion verification.** The chainRelay stores every block header in the backing blockchain B on I. Each block header in chainRelay contains the root of the Merkle tree [87] containing all transactions (or their identifiers) for that block. To verify the correct inclusion of a transaction in a block in B, it is sufficient to provide the Merkle tree path from the root to the leaf containing the transaction (identifier) and the transaction data itself. This verification can be then be performed in a non-interactive manner by the chainRelay as part of the iSC.
- **Consensus verification.** The chainRelay can also verify that any given block header is part of the backing blockchain B, i.e., has been agreed upon by the majority of consensus participants. In XCLAIM, consensus verification depends on the consensus mechanism used by the backing blockchain B. For Nakamoto consensus [90], the chainRelay must (i) know the difficulty adjustment policy and (ii) verify that the received headers are on the chain with the most accumulated Proof-of-Work [6], [76]. For Proof-of-Stake blockchains, e.g Ouroboros [75], the chainRelay must (i) be aware of the protocol/staking epochs and (ii) verify the signature membership of elected leader(s) for the threshold/multi-signatures of block headers [64]. For permissioned (Proof-of-Authority) systems, the verification is analogous, or simpler, if the consensus participants are pre-defined [109]. We provide a formal definition for the necessary functionality of Proof-of-Work chain relays in Appendix D.

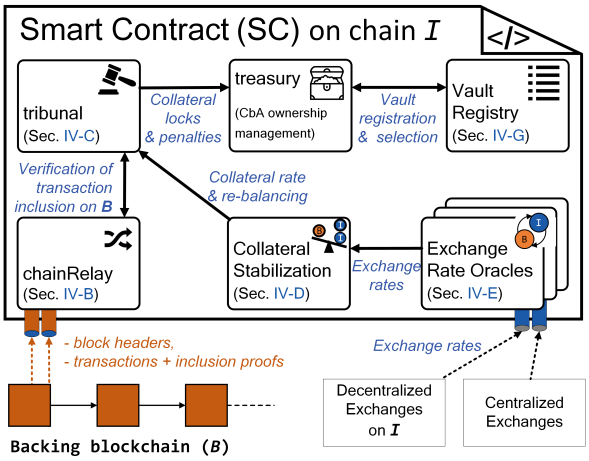

Fig. 2. High level overview of the architecture of the XCLAIM smart contract (iSC) and the interactions between its components. References to sections introducing each component are provided. The treasury refers to the basic ledger functionality of I.

XCLAIM uses the chainRelay to modify the *Issue* protocol presented in CENTRALCLAIM (Section IV-A): after locking funds b with the *vault*, the *requester* must prove to the chainRelay that funds were locked correctly by presenting the transaction generated when sending b to the *vault* in B. The chainRelay can then verify that the given transaction has been securely included in B and the funds were locked correctly. If successful, the chainRelay triggers the automatic issuing of the corresponding amount of $i(b)$ in the iSC.

Similarly, XCLAIM also modifies the *Redeem* protocol: upon a redeem request being made by the user, the *vault* is required to prove that (i) the funds b were released to the *redeemer* and (ii) the released amount corresponds to the burnt CBA, i.e.,

$|b| = |i(b)|$. This is done by presenting the chainRelay with the transaction that sends b to the *redeemer* within a maximum delay Δ^I_{redeem}. Should the *vault* fail to comply, it incurs a financial penalty and the iSC guarantees reimbursement to the *redeemer*; we discuss this in Sections V-C and V-D.

We note that to correctly verify inclusion proofs, the chainRelay must be up to date with the block headers of B. As XCLAIM makes timing assumptions on inclusion proof submission during *Redeem*, we must define an upper bound Δ_{relay} for the delay between generation of a block containing transaction T^B on B and the submission of (i) the block header and (ii) the inclusion proof for T^B to the iSC via the chainRelay. Hence, we define $\Delta_{relay} = \Delta^B + \Delta_{submit} + 2\Delta^I$, where Δ_{submit} is the delay before a transaction submitting a B block header to the chainRelay is broadcast. If batched submission of n block headers within a single transaction on I is possible, the upper bound for the delay is increased to $\Delta_{submit} + n(\Delta^B + 2\Delta^I)$. This also applies for compact proofing techniques, e.g. NiPoPoWs [74] or FlyClient [85].

Security Arguments for Liveness: The chainRelay makes the *Issue* protocol non-interactive: instead of trusting the *vault* to confirm the lock of b, the iSC accepts a transaction inclusion proof provided by the *requester*. To prevent the *requester* from executing *Issue*, an adversary hence must control all funds i on I and/or prevent inclusion of transactions in B or I. As *Transfer* and *Swap* only require interaction with the iSC, to interfere, an adversary must modify the behavior of the iSC or prevent transaction inclusion in I. This, however, is not possible under the assumptions of the blockchain and threat models. Hence, XCLAIM achieves **Liveness**.

Security Arguments for Consistency: By construction, the iSC only issues $i(b)$ if the provided transaction inclusion proof shows that the correct amount on b was locked on B, i.e., $|b| = |i(b)|$. From the blockchain and threat models we know an adversary cannot tamper with the iSC. Hence, XCLAIM achieves **Consistency.**

We note that for the *vault* to have a realistic time window to provide a proof, we must consider the security parameters for B and I, as well as the block generation rates when parameterizing Δ_{redeem}, i.e., it must hold that $\Delta_{redeem} > \Delta^B + \Delta_{relay}$.

C. Tribunal: Incentives via Collateralization

We next modify CENTRALCLAIM by introducing *collateral* as a means to incentivize honest behavior in XCLAIM and impose punishment on misbehaving parties through the iSC. We refer to this component of the iSC as the tribunal (cf. Figure 2). Specifically, we modify CENTRALCLAIM by requiring the *vault* to lock up units of i as collateral when registering with the iSC, which we denote as i_{col}. If the *vault* fails to prove correct execution of the *Redeem* protocol, the collateral is automatically used by the iSC to compensate the *redeemer* and to pay an additional punishment fee.

For collateralization of the *vault* to be effective in terms of maintaining incentives to behave honestly, the collateral must be at least equal to the funds locked on backing chain B. One challenge faced by this approach in XCLAIM is that the *vault*'s collateral is locked in currency i, while the value it is balanced against is measured in currency b. To this end, we must ensure $i_{col} \geq b_{lock} \cdot \varepsilon_{(i,b)}$ holds, where b_{lock} refers to the units of b locked with the *vault* on B and $\varepsilon_{(i,b)}$ is the exchange rate provided by oracle $\mathcal{O}$. For ease of explanation, at this point we assume the exchange rate $\varepsilon_{(i,b)}$ is constant. We discuss challenges of non-constant exchange rates and mitigation thereof in Section V-D.

To ensure **Redeemability**, users must only initiate the *Issue* protocol, if sufficient collateral is provided by the *vault* in the iSC. However, the naive *Issue* protocol of CENTRALCLAIM exhibits vulnerabilities to *race conditions*: (i) the *vault* can attempt to withdraw collateral before the *requester* can finalize the issuing process, i.e., provide the transaction inclusion proof to the chainRelay, and (ii) multiple *requesters* can attempt to simultaneously issue for the same amount of the *vault*'s collateral, triggering a race where the loser's locked funds b_{lock} are not secured by collateral. We present mitigations for the above attacks in XCLAIM:

- **Deferred Collateral Withdrawal**. The *vault* may exploit race conditions due to network latency, delays Δ^B and Δ^I or DoS attacks against the *requester* to attempt collateral withdrawal *after* a lock on B is executed, committing unpunished theft. We derive a simple announce-delay-withdraw scheme to prevent such attacks. Specifically, we require the *vault* to announce collateral withdrawal publicly via the iSC. The iSC allows users to finalize (in theory also to initiate new) issue processes within a delay $\Delta_{withdraw}$, after which the *vault* may withdraw the remaining unused collateral. Thereby, the lower bound for $\Delta_{withdraw}$ is the upper bound on transaction inclusion proofs Δ_{relay} defined in Section V-B, i.e., $\Delta_{withdraw} > \Delta_{relay}$.
- **Collateralized Issue Commitments**. To prevent multiple *requesters* from concurrently locking funds b using the same amount of the *vault*'s collateral, we introduce a registration step to the setup phase of the *Issue* protocol. Specifically, a *requester* must register an issue request for $i(b)$ with the iSC, which temporarily locks the corresponding amount of the *vault*'s collateral. Within the following delay $\Delta_{commit} > \Delta^B + \Delta_{relay}$ the *requester* can then safely execute the remaining steps of the *Issue* protocol. The iSC therefore only accepts pre-registered issuing attempts. To avoid *griefing* attacks by malicious *requesters*, i.e., continuous locks of the *vault*'s collateral, we require the *requester* to commit to issuing by providing collateral herself. The latter is used to reimburse the *vault* in case of failure. We note that multiple collateralized commitments can be created in parallel, of which only a single one will be accepted by the iSC, on a first-come-first-served basis. In this worse-case scenario, the losses faced by *requesters* are hereby limited to a transaction fee on I.

Security Arguments for Redeemability under constant $\varepsilon_{(i,b)}$: By introducing collateralization in XCLAIM we ensure that an economically rational *vault* has no incentive to misbehave. Specifically, by construction, the iSC only accept issue re-

quests if collateral $i_{col} \geq b \cdot \varepsilon_{(i,b)}$ is locked by the *vault*. During the *Redeem* protocol, the *vault* is required to include a transaction in B, sending b to the *redeemer* such that $|b| = |i(b)|$ and provide an inclusion proof to the iSC. If the *vault* misbehaves, it will lose collateral i_{col}, which the iSC uses to reimburse the *redeemer*, and miss out on fees for honest behavior. That is, a *vault* gains negative utility from misbehaving and does not execute its equilibrium strategy.

Deferred collateral withdrawal and collateralized issue commitments therefore prevent the *vault* from exploiting network related race conditions to defraud users. It is also easy to see that collusion of malicious *vault* and *redeemer* yields no benefit, as issuing and redeeming in XCLAIM is a zero-sum game. In fact, transaction fees on B and I lead to negative utility in such scenarios. Hence, under the economically rational adversaries as per our threat model, XCLAIM achieves **Redeemability** under constant exchange rates.

D. Mitigating Exchange Rate Fluctuations

Until now, we have assumed that both the exchange rate $\varepsilon_{(i,b)}$ and the collateral i_{col} provided by the *vault* remain unchanged. However, real world observations show that the exchange rate $\varepsilon_{(i,b)}$ between the two cryptocurrencies may be susceptible to strong fluctuations. To ensure **Redeemability** under *non-constant* exchange rates, we hence (i) over-collateralize the *vault*, (ii) enable adjustment of the *vault*'s locked collateral and (iii) introduce automatic liquidation to prevent financial loss in case of extreme devaluation of i.

Over-collateralization helps mitigate failures due to sudden drops of $\varepsilon_{(i,b)}$. We over-collateralize the *vault* by a factor $r_{col} \in \mathbb{R}_{\geq 1}$, creating a buffer to account for possible exchange rate fluctuations. For secure operation, the following must hold for the lifecycle of XCLAIM:

$$i_{col} \geq b_{lock} \cdot \left(r_{col} \cdot \varepsilon_{(i,b)}\right) \geq b_{lock} \tag{1}$$

As a result, the over-collateralization factor r_{col} becomes a security parameter in XCLAIM. The combination of r_{col} with the exchange rate $\varepsilon_{(i,b)}$ then defines how many units of the backing cryptocurrency b a *requester* can safely lock with the *vault*, i.e., the maximum amount of safely issuable $i(b)$:

$$\max(i(b)) = \frac{i_{col}}{r_{col} \cdot \varepsilon_{(i,b)}} \tag{2}$$

For clarity, we denote *blocked* collateral, i.e., collateral already used to securely issue $i(b)$, as $i^{-}_{col} = i(b) \cdot r_{col} \cdot \varepsilon_{(i,b)}$ and *free* collateral as $i^{+}_{col} = i_{col} - i^{-}_{col}$.

While over-collateralization helps mitigate extreme fluctuations in the short term, it may be insufficient to securely handle long-term issuing. To this end, we enable the adjustment of the *vault*'s collateral and introduce the notion of automatic liquidation of $i(b)$ by the iSC. We derive a simple multi-stage system for collateral i_{col}. The latter defines the behavior of the iSC, based on the observed collateral rate $r^{*}_{col} = \frac{i^{-}_{col} + i^{+}_{col}}{b_{lock} \cdot \varepsilon_{(i,b)}}$ and the (parameterized) ideal rate r_{col}. Specifically, we introduce thresholds $r_{col} > r^{liq}_{col} > 1.0$. For ease of explanation, we assume an exemplary collateral rate $r_{col} = 2.0$. We define the multi-stage system for collateral as follows:

- **Secure Operation** : The *vault* has locked more collateral than necessary to ensure **Redeemability** in XCLAIM, i.e., new $i(b)$ can be issued correctly. Similarly, the available free collateral i^{+}_{col} can be withdrawn by the *vault*, as long as $r^{*}_{col} \geq r_{col}$ holds.
- **Buffered Collateral**: The collateral rate r^{*}_{col} has dropped below ideal rate r_{col}, however there is sufficient buffer to ensure secure operation of XCLAIM. However, as defined in Eq. 2, no new $i(b)$ can be issued.
- **Liquidation**: The collateral rate is critically close to the lower bound of 1.0 (e.g. $r^{liq}_{col} = 1.05$). If the *vault* does not re-balance r^{*}_{col} by increasing i_{col}, the iSC automatically initiates *Redeem* for all existing $i(b)$. The remaining collateral buffer $r^{*}_{col} - 1.0 > \varepsilon_{(i,b)}$ is thereby used to cover transaction fees. This measure is necessary to prevent users from facing financial loss, should r^{*}_{col} drop below 1.0.

Security Arguments for Redeemability under non-constant $\varepsilon_{(i,b)}$: Through over-collateralization of the *vault*, we use a buffer to tolerate sudden exchange rate drops. As the *vault* can update collateral, it can, in the optimistic case, maintain secure operation of XCLAIM even if the buffer is depleted. Should the latter fail, the iSC ensures users do not face financial losses via automatic liquidation. Specifically, an economically rational *vault* will only misbehave if $i_{col} < b \cdot \varepsilon_{(i,b)}$. By construction, the iSC automatically reimburses i_{col} to a *redeemer* if $i_{col} < b \cdot \varepsilon_{(i,b)}$. Hence, misbehaving only becomes the equilibrium strategy of the *vault*, if it can alter the behaviour of the iSC. As this is not possible under the assumptions of the blockchain and threat model, it follows that XCLAIM achieves **Redeemability** under non-constant exchange rates $\varepsilon_{(i,b)}$.

Note: from $\Delta_{\min(\varepsilon)} < \Delta^{I}$ it follows the *redeemer* can either initiate the *Redeem* protocol or, in case of automatic liquidation, withdraw i from the iSC before $\varepsilon_{(i,b)} < \min(\varepsilon_{(i,b)})$.

E. Multi-vault System: Removing Single Points of Failure

Until now, we have assumed a single *vault*. However, the design of XCLAIM allows it to be easily extended to a multi-*vault* system. Hence, we allow *any* user to become a *vault* by registering with the iSC and providing collateral. The list of *vaults* is maintained in a public registry in the iSC. By allowing both *requesters* and *redeemers* to freely choose which *vault* they wish to use for issuing and redeeming, we create a free market driven by charged fees and the observed collateral rate $r_{col}*$ of each *vault*. The availability of multiple *vaults* further allows a *redeemer*, upon a failed *Redeem* caused by a *vault*, to choose between: (i) being reimbursed from the slashed collateral i_{col} or (ii) retrying the *Redeem* using a different *vault*.

One challenge that arises from a multi-*vault* system is ensuring correct automatic liquidation. Deterioration of collateral of a single *vault* does not affect the entire system, but only the corresponding fraction of issued $i(b)$. In a first step, the iSC offers *beneficial* liquidation, i.e., redemption of $i(b)$ against the corresponding amount of b_{lock} and an additional small

Operations:

Backing Blockchain:
For the backing blockchain B, operations are executed by the $requester$ and the $vault$. We differentiate between the two:

Operations performed by the $requester$:

- $\mathsf{lockB}(b, cond) \rightarrow T^B_{lock}$ which locks coins b on chain B under conditions $cond$.

Operations performed by the $vault$:

- $\mathsf{verifyIOp}(\mathsf{operation}, T^I_{op}[, \Delta_{op}]) \rightarrow \top|\bot$ which verifies that $\mathsf{operation}$ was executed on I, i.e., that T^I_{op} is securely included in I according to k^I and within optional delay Δ_{op} as per XCLAIM parameters.
- $\mathsf{redeemB}(b, user^B) \rightarrow T^B_{redeem}$ that releases locked coins b to $user^B$.
- $\mathsf{transferB}(b, user^B) \rightarrow T^B_{transfer}$ which transfers ownership of b to user $user^B$.

Issuing Blockchain:
For the issuing blockchain I, operations are executed by the iSC:

- $\mathsf{lock}(x, cond) \rightarrow T^I_{lock}$ which locks x under conditions $cond$ on I, where x can be $i(b)$ or i.
- $\mathsf{release}(x, user^I) \rightarrow T^I_{release}$ that releases asset or coin x to $user^I$, where x can be $i(b)$ or i.
- $\mathsf{slash}(x, user^I, user^I_*) \rightarrow T^I_{slash}$ that destroys or slashes collateral funds x of $user^I$ and reimburses them to $user^I_*$, where x can be $i(b)$ or i.
- $\mathsf{commit}(b, user^B, vault, i) \rightarrow T^I_{commit}$ which commits $user^B$ to calling $\mathsf{lockB}(b, \sigma^B_{vault})$ within Δ_{commit}. Locks i as collateral conditioned on the iSC's signature via $\mathsf{lock}(i, \sigma^I_{SC})$.
- $\mathsf{verifyBOp}(\mathsf{operation}, T^B_{op}[, \Delta_{op}]) \rightarrow \top|\bot$ which verifies $\mathsf{operation}$ was executed on B, i.e., securely included in B via T^B_{op} according to k^B and within optional delay Δ_{op} as per XCLAIM parameters.
- $\mathsf{issue}(b, user^I) \rightarrow T^I_{issue}$ which creates and allocates $i(b)$ to $user^I$, such that $|i(b)| = |b|$.
- $\mathsf{transfer}(x, user^I) \rightarrow T^I_{transfer}$ which transfers ownership of x to user $user^I$, where x can be $i(b)$ or i.
- $\mathsf{swap}(x, user^I_x, y, user^I_y) \rightarrow T^I_{swap}$ which transfers ownership of x to $user^I_y$ and y to user $user^I_x$ atomically; x and y can be $i(b)$ or i.
- $\mathsf{burn}(i(b)) \rightarrow T^I_{burn}$ that destroys $i(b)$.

Algorithms:

protocol *Issue*
  $vault$.lock(i_{col})
  $requester$.commit(b_{lock}, $\mathsf{pk}^B_{requester}$, pk^I_{vault}, i^{commit}_{col})
  $requester$.lockB(b_{lock}, σ_{vault}) /* $\rightarrow T^B_{lock}$ */
  $requester$ submits T^B_{lock} to iSC calling verifyBOp
  if iSC.verifyBOp(lockB, T^B_{lock}, Δ_{commit}) $= \top$ **then**
    iSC.issue(b_{lock}, $\mathsf{pk}^I_{requester}$)
    iSC.release(i^{commit}_{col}, $\mathsf{pk}^I_{requester}$)
  else
    iSC.slash(i^{commit}_{col}, $\mathsf{pk}^I_{requester}$, pk^I_{vault})

protocol *Swap*
  $sender$.lock($i(b)$, σ^I_{SC})
  $receiver$.lock(i, σ^I_{SC}) /* $\rightarrow T^I_{lock}$ */
  if iSC.verifyIOp(lock, T^I_{lock}, Δ_{swap}) $= \top$ **then**
    iSC.swap($i(b)$, $\mathsf{pk}^I_{receiver}$, i, pk^I_{sender})
  else
    iSC.release($i(b)$, pk^I_{sender})

protocol *Transfer*
  $sender$ calls iSC.transfer($i(b)$, $\mathsf{pk}^I_{receiver}$)

protocol *Redeem*
  $redeemer$ calls iSC.lock($i(b)$, $\mathsf{pk}^B_{redeemer}$) /* $\rightarrow T^I_{lock}$ */
  iSC signals $|b| = |i(b)|$ is to be released to $redeemer$ on B
  if $vault$.verifyIOp(lock, T^I_{lock}) $= \top$ **then**
    $vault$.redeemB(b_{lock}, $\mathsf{pk}^B_{redeemer}$) /* $\rightarrow T^B_{redeem}$ */
    $vault$ submits T^B_{redeem} to iSC calling verifyBOp
    if iSC.verifyBOp(redeemB, T^B_{redeem}, Δ_{redeem}) $= \top$ **then**
      iSC.release(i_{col}, pk^I_{vault})
    else
      slash(i_{col}, pk^I_{vault}, $\mathsf{pk}^I_{redeemer}$)
    iSC.burn($i(b)$)

Fig. 3. Overview of operations exhibited by XCLAIM on the backing blockchain B and issuing blockchain I (left), and the algorithms specifying XCLAIM's *Issue*, *Transfer*, *Swap* and *Redeem* protocols for cryptocurrency-backed assets (right).

premium in i, deducted from the $vault$'s available collateral ($r^*_{col} - 1.0$). Should insufficient users wish to execute *Redeem*, the iSC, as a final fallback, equally distributes the liquidation among all users of XCLAIM. Note: tracing CBAs back to the $vault$ they were issued makes CBAs non-fungible, which is contrary to the desired functional properties.

Arguments for Scale-Out: By construction, any user can register as a $vault$ with the iSC on I by locking collateral i_{col}, i.e., the set of $vaults$ can change dynamically and is not pre-defined. It is easy to see any user can hence increase the total amount of safely issuable CBAs, $\max(i(b))$. We note that to prevent registration of new $vaults$, an adversary must: (i) control all funds i on I and/or (ii) prevent inclusion of transactions in I. Both scenarios are not possible under the assumptions of the blockchain and threat models. Hence, under secure operation of B and I, XCLAIM achieves **Scale-Out**.

VI. Formal Protocol Specification

This section presents a formal specification XCLAIM's *Issue*, *Transfer*, *Swap* and *Redeem* protocols, as well as the requirements imposed on the backing and issuing blockchains.

A. XCLAIM Operations and Protocols

Notation. We differentiate between *state changing* and *non-state changing* operations in XCLAIM; state changing operations result in new transactions (T) in the underlying blockchain, while non-state changing operations, such as verifying operations, are "read-only", returning boolean values ($\top|\bot$). We use T^B_{id} to refer to a transaction created on chain B with identifier id and T^I_{id} for transactions on I respectively. The execution of an operation on input in that produces an output out is denoted as $\mathsf{operation}(in) \rightarrow out$. To indicate that an operation is executed by a user $user$, we write $user$.operation. We identify a user $user$ in XCLAIM by her public key pk^X_u (with corresponding private key sk^X_u), where X can be either blockchain B or I. For readability, we often write $user^X$ when referring to pk^X_u. We use $cond$ to refer to locking and unlocking conditions for funds on both B and I, e.g. a condition for a transaction may be the digital signature of user u on chain X with private key sk^X_u, denoted as σ^X_{user} A summary of symbols is provided in Appendix B.

We parameterize XCLAIM, using: (i) the blockchain security parameters k^B and k^I; (ii) block generation rates τ^B and τ^I; (iii) collateral rate r_{col} and the automatic liquidation threshold r^{liq}_{col}; and (iv) the delays Δ_{redeem}, $\Delta_{withdraw}$, Δ_{commit}, Δ_{swap} used in the *Issue*, *Redeem* and *Swap* protocols. The algorithms specifying XCLAIM's *Issue*, *Transfer*, *Swap* and *Redeem* protocols, as well as the necessary operations exhibited by XCLAIM on the underlying blockchains B and I are provided in Figure 3.

B. Blockchain Requirements

Using the system operations performed by XCLAIM as defined in section VI-A, we derive the requirements for the

underlying blockchains B and I. We summarize our findings in Table I and provide examples for backing and issuing chains currently supported by XCLAIM. We note that neither B nor I require a Turing-complete instruction set.

Backing Blockchain (B): On the backing blockchain B we need to lock and redeem funds based on conditions $cond$, i.e., a user's digital signature. *Boolean* operations are required to verify $cond$ are either true or false. Moreover, conditions for locking and redeeming from different users require (i) a *stack* to store intermediary values, (ii) *read* and *write* operations for the current stack, and (iii) public-key encryption and signature verification. *Flow control* operations do not have to be available to users and can be expressed by stack states (empty / not empty) [41]. In addition, while public-key encryption and signatures can be implemented using basic *arithmetic* and *bitwise* operations, both script complexity and execution cost can be reduced if *cryptographic* operations, including hash and signature verification functions, are supported by the scripting language as dedicated operations. Finally, B requires a method to store data, necessary to e.g. include the target public key of the $requester$ for issuing on I.

Issuing Blockchain (I): To support issuing of CBAs in a smart contract, the issuing chain I requires a method to create custom assets, which are part of the consensus protocol. This can be realized by *permanent storage*, i.e., *storage read* and *storage write* operations. In accordance to our definitions (cf. Section III-A), the CBA attributes, *issuing chain*, *backing chain*, and *asset value* are represented as integers; integer balances are assigned to the *asset owner*'s public keys (or digests thereof). Modification of asset balances can be realized using *arithmetic* operations on integers, and the authorization of changes via *boolean* operations.

For transaction verification, the chainRelay requires (i) *permanent storage* to store block header, transaction, and proof data and (ii) *arithmetic*, *bitwise*, and *boolean* operations for proof verification. Verifying Merkle tree inclusion requires to traverse the data structure (both of block headers and transaction lists), i.e., I must support *finite loops or recursion*. If I supports the same (or super-) set of *cryptographic* operations used on B (specifically hash functions and signature schemes), the verification may be executed at lesser cost. Next, I requires more complex conditional locks than B, i.e., *flow control* must be supported (in addition to *arithmetic* and *boolean* operations). Since *Issue* and *Redeem* require checking delays (blocks or based on time stamps), I must allow access to XCLAIM parameters via *global parameters* (integers).

Discussion: One of the main challenges overcome by the design of XCLAIM is backward compatibility: any blockchain supporting the minimum requirement of transferring funds between users can act as backing blockchain B. However, there are cases where B may support a similar set of operations to I, i.e., B may also allow the deployment of a smart contract bSC. More specifically, programmatic verification of transaction inclusion proofs via chainRelay components is supported bilaterally, i.e., the verifyIOp operation in *Issue* and *Redeem* can be executed directly by bSC rather than by an intermediate $vault$. The rest of the system remains unchanged. The use of collateralized intermediaries in this scenario, while no longer necessary for secure operation, can act as a performance and cost improvement. An example are Ethereum and Ethereum Classic, where inclusion proofs can be verified via PeaceRelay [24].

TABLE I
Required operations on backing (B) and issuing (I) chains including application candidates.

Requirements	*Operations*									*Examples for supported blockchains*
	Arithmetic	*Boolean*	*Bitwise*	*Cryptographic*[†]	*Flow control*	*Local stack*	*Finite loops or recursion*	*Permanent storage*	*Global parameters*	
Backing chain B	✓	✓	(✓[‡])	(✓)	✗[⋆]	✓	✗	✓	✗	Bitcoin [94], ZCash [37], Namecoin [91], Litecoin [84], Ethereum [58]
Issuing chain I	✓	✓	✓	(✓)	✓	✓	✓	✓	✓	Ethereum [58], Zilliqa [99], Cardano [7], Neo [20], Ethereum Classic [14], RSK [82]

[†] Not strictly required, but reduces script complexity.
[‡] Not necessary if native support for cryptographic operations (hash and signature verification) is available.
[⋆] Can be represented as stack states (empty / not empty).

VII. SECURITY ANALYSIS

In this section we provide an informal security analysis to supplement the design choices and security propositions presented in Sections IV and V. We discuss attack vectors, potential impacts and their mitigations.

A. Chain Relay Poisoning

An adversary may attempt to *poison* the chainRelay with false information regarding blockchain B. Such attacks are equivalent to *selfish mining* [61] attacks, as the adversary must trigger a chain reorganization according to the underlying consensus rules. Even though in our model we assume $f < n/3$ (or $\alpha < 33\%$), if the assumptions regarding data availability of the chainRelay do not hold (cf. Section V-B), a poisoning attack can be successful well below this threshold. While an adversary cannot prevent inclusion of transactions in I under our model, the lack of honest block headers submitted to the chainRelay may have alternate reasons, e.g. high cost. To mitigate relay poisoning due to temporary lack of block header data, we can introduce a *maturity period* $\Delta_{maturity}$ for newly generated CBAs [33], similar to that of newly minted coins in PoW blockchains. If a (correct) conflicting chain is submitted within $\Delta_{maturity}$, the pending CBAs are not issued.

B. Replay Attacks on Inclusion Proofs

Without adequate protection, inclusion proofs for transactions on B can be *replayed* by: (i) the $requester$ to trick the iSC into issuing duplicate $i(b)$ and (ii) the $vault$ to reuse a single transaction on B to falsely prove multiple redeem requests. A simple and practical mitigation is to introduce unique identifiers for each execution of *Issue* and *Redeem* and require transactions on B submitted to the chainRelay of these protocols to contain the corresponding identifier.

C. Counterfeiting

A *vault* which receives b_{lock} from a *requester* during *Issue* could use these coins to re-execute *Issue* itself, creating counterfeit $i(b)$, i.e., $|b_{lock}| < |i(b)|$. To this end, the iSC forbids *vaults* to move locked funds b_{lock} received during *Issue*. From **Auditability** we know any user with read access to B can detect misbehavior and can submit a transaction inclusion proof to the iSC showing the *vault* spent locked funds b_{lock}. To restore **Consistency**, the iSC slashes the *vault*'s entire collateral and executes automatic liquidation, following the steps described in Sections V-D and V-E, yielding negative utility for the *vault*. To allow economically rational *vaults* to move funds on B we describe *Replace*, a non-interactive atomic cross-chain swap (ACCS) protocol based on cross-chain state verification, in Appendix C.

D. Permanent Blockchain Splits

Permanent chain splits or *hard forks* occur where consensus rules are loosened or conflicting rules are introduced [114], resulting in multiple instances of the same blockchain. Thereby, a mechanism to differentiate between the two resulting chains (*replay protection*) is necessary for secure operation [86].

Backing Chain: If replay protection is provided after a permanent split of B, the chainRelay must be updated to verify the latter for B (or B' respectively). If no replay protection is implemented, the chainRelay will behave according to the protocol rules of B for selecting the "main" chain. For example, it will follow the chain with most accumulated PoW under Nakamoto consensus.

Issuing Chain: A permanent fork on the issuing blockchain results in two chains I and I' with two instances of the iSC identified by the same public key. To prevent an adversary exploiting this to execute replay attacks, both *requester* and *vault* must be required to include the public key of the iSC (or a digest thereof) in the transactions published on B as part of *Issue* and *Redeem* (in addition to the identifiers introduces in VII-B). Next, we identify two possibilities to synchronize $i(b)$ balances on I and I': (i) deploy a chain relay for I on I' and vice-versa to continuously synchronize iSC [24] and iSC$'$ states or (ii) redeploy the iSC on both chains and require users and *vaults* to re-issue $i(b)$, explicitly selecting I or I'.

E. Denial-of-Service Attacks

XCLAIM is decentralized by design, thus making denial-of-service (DoS) attacks difficult. Given that any user with access to B and I can become a *vault*, an adversary would have to target all *vaults* simultaneously. Where there are a large number of *vaults*, this attack would be impractical and expensive to perform. Alternatively, an attacker may try to target the iSC. However, performing a DoS attack against the iSC is equivalent to a DoS attack against the entire issuing blockchain or network, which conflicts with our assumptions of a resource bounded adversary and the security models of B and I. Moreover, should an adversary perform a Sybil attack and register as a large number of *vaults* and ignore service requests to perform a DoS attack, the adversary would be required to lock up a large amount of collateral to be effective. This would lead to the collateral being slashed by the iSC, making this attack expensive and irrational.

F. Fee Model Security: Sybil Attacks and Extortion

While the exact design of the fee model lies beyond the scope of this paper, we outline the following two restrictions, necessary to protect against attacks by malicious *vaults*.

Sybil Attacks: To prevent financial gains from Sybil attacks, where a single adversary creates multiple low collateralized *vaults*, the iSC can enforce (i) a minimum necessary collateral amount and (ii) a fee model based on issued volume, rather than "pay-per-issue". In practice, users can in principle easily filter out low-collateral *vaults*.

Extortion: Without adequate restrictions, *vaults* could set extreme fees for executing *Redeem*, making redeeming of $i(b)$ unfeasible. To this end, the iSC must enforce that either (i) no fees can be charged for executing *Redeem* or (ii) fees for redeeming must be pre-agreed upon during *Issue*.

VIII. XCLAIM (BTC,ETH) IMPLEMENTATION AND EVALUATION

We instantiate XCLAIM as XCLAIM (BTC,ETH) to issue Bitcoin-backed tokens on Ethereum. Ethereum's virtual machine provides Turing completeness [63], fulfilling the requirements (Section VI-B) for the issuing chain I. Bitcoin, due to the limited operation set of its Script language [41] only qualifies as a backing blockchain (Section VI-B). Both Bitcoin and Ethereum use ECDSA with the secp256k1 Koblitz curve [42], [78], providing native support for the corresponding cryptographic operations. Ethereum further makes the SHA-256 and RIPEMD-160 hash functions, which are used in Bitcoin, available as pre-compiled contracts [63].

We implement the iSC smart contract on Ethereum in Solidity v0.4.24 [27] in approximately 820 lines of code. On Bitcoin we use regular P2PKH [40] transactions. We use the existing Serpent [28] implementation of BTCRelay [6] for the chainRelay component of the iSC. Bitcoin-backed tokens per our implementation are compliant with the ERC20 token standard [10], and hence usable in most services on Ethereum, including *decentralized exchanges*. To evaluate cost and performance of XCLAIM (BTC,ETH), we deploy the iSC on the Ethereum Ropsten test network [15]. For evaluations, we assume a *vault* is registered with the iSC and has locked in collateral on Ethereum.

A. Protocol Execution Costs

We define on-chain execution costs measured in USD as the amount of Bitcoin and Ethereum transaction fees required to execute each of the protocols: *Issue*, *Transfer*, *Swap*, *Redeem* and *Replace*. The costs are calculated using current conversion rates[7]. In Bitcoin, transaction fees are calculated based on the transaction size, i.e., the number of consumed inputs and generated outputs. To ensure transactions are included in the

[7]Storage and execution costs are in USD as per exchange rates of 30 Nov. 2018: BTC/USD 3717.38 and ETH/USD 105.71

TABLE II
Overview of execution costs[7] and performance[9] for the *Issue*, *Transfer*, *Swap* and *Redeem* protocols in XCLAIM (BTC,ETH).

Protocols	Transactions	Cost (USD) Ethereum	Bitcoin	Total	Duration (minutes)
Issue	2^{Eth} 1^{Btc}	0.16	0.31	0.47	75.98
Swap	2^{Eth}	0.19		0.19	5.98
Redeem	2^{Eth} 1^{Btc}	0.18	0.31	0.49	75.98
Total	6^{Eth} 2^{Btc}	0.53 (46.1%)	0.62 (53,9%)	1.15	157.94
Transfer	1^{Eth}	0.04		0.04	2.99

next generated block without delays, we calculate fees with 40 Satoshis (10^{-8} BTC) per byte [56]. In Ethereum, transaction fees are measured in *gas*, depending both on the transaction size and the cost of executed smart contract operations [63]. We assume a gas cost of 9 Gwei based on current network fees for fast transaction inclusion [57].

We summarize our measurement results in Table II. In our measurements, we refer to the complete process of issuing (*Issue*), executing a trade (*Swap*) and then redeeming (*Redeem*) an arbitrary amount of Bitcoin-backed tokens on Ethereum as a *round*. Our experiments show a full protocol execution round only costs USD 1.15. The main cost factor thereby are Bitcoin transaction fees (53.9%). As such, transferring ownership over units of Bitcoin via XCLAIM Bitcoin-backed tokens on Ethereum costs only USD 0.04, in contrast to USD 0.31 if the transfer is executed natively on Bitcoin (87.1% cheaper). We note, additional costs are incurred for keeping BTCRelay up to date with the current state of the Bitcoin blockchain, amounting to approximately USD 27 per day (not included in the table). These costs are fixed and shared among all users of XCLAIM; each user's share decreases with higher adoption of XCLAIM. We further note this number constitutes an *upper bound* given current prices: (i) the existing implementation of BTCRelay is non-optimal[8] and (ii) our measurements consider the worst case scenario where batched block header submissions are not available (cf. Section V-B).

B. Performance

We evaluate the performance of XCLAIM (BTC,ETH) with respect to the duration of the *Issue*, *Transfer*, *Swap*, *Redeem* and *Replace* protocols (measurements provided in Table II). Thereby, we adhere to the recommended security parameters regarding transaction confirmations based on our threat model ($\alpha \leq 33\%$): $k^B = k^{BTC} = 6$ and $k^I = k^{ETH} = 12$. Recall, we consider a transaction in block at position j as securely included when the blockchain tip reaches position h, with $h - j \geq k$. At the time of writing, the block time in Bitcoin amounts to 10 minutes, and in Ethereum to 14 seconds.

[8]The Serpent language is not actively maintained (last commit on 1 October 2017) and is not optimized to the current version of the EVM, resulting in higher execution costs. More efficient proofing techniques can further reduce costs [72], [74], [85], [102].

[9]Performance is measured in minutes and includes security parameters: 6 conf. a 10 min for Bitcoin (k^B); 12 conf. a 14 sec for Ethereum (k^I).

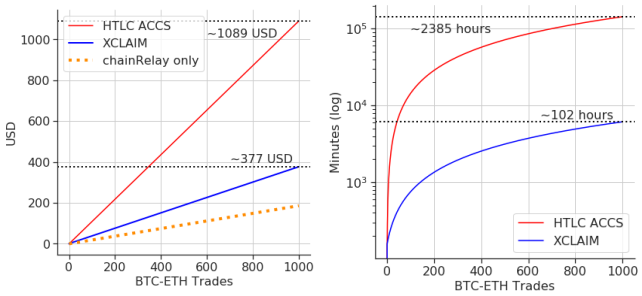

Fig. 4. Comparison of BTC-ETH atomic swaps via XCLAIM and via HTLC ACCS for 1000 individual swaps. Storage and execution costs (Left) are in USD[7]; performance (Right, logarithmic y-axis) is measured in minutes[9]. We observe XCLAIM is 95.7% faster and 65.4% cheaper for 1000 swaps.

One execution round of issuing, atomically swapping and redeeming of Bitcoin-backed token on Ethereum requires 2 Bitcoin transactions (1 user transaction and 1 *vault* transaction) and 6 Ethereum transactions (5 user transactions and 1 *vault* transaction). The end-to-end process is securely completed after 158 minutes; a *Swap* only takes 6 minutes, while *Issue* and *Redeem* account for the greater part of the delay due to Bitcoin transaction processing (96,2%). Note: we can use off-chain payment channels [73], [88] on the issuing chain (Ethereum) to significantly reduce execution cost and make *Swap* real time (see Section IX).

C. Comparison to HTLC Atomic Swaps

We compare the performance and execution costs of XCLAIM (BTC,ETH) to that of atomic cross-chain swaps (ACCS) based on hashed-timelock contracts (HTLCs) [4], [5], [105]. Both implementations are tested under identical conditions, including fee calculation and security parameters. We visualize the results of our experiments in Figure 4.

Each interactive atomic swap requires users to create two transactions on both Bitcoin and Ethereum (4 in total). Including a minimum necessary delay to prevent race conditions, an atomic swap takes approximately 146.5 minutes to execute securely. Note: we omit the additional necessary time to establish out-of-band channels and exchange revocation transactions in ACCS; hence the ACCS measurements are *lower bounds*.

In XCLAIM, after an initial *Issue* process, each additional *Swap* requires only 2 Ethereum transactions. As such, using XCLAIM to atomically exchange BTC against ETH is already more efficient than ACCS after the second swap; for 1000 swaps XCLAIM is 95.7% faster. Similarly, XCLAIM (including BTCRelay fees) outperforms ACCS cost-wise after the second trade; for 1000 trades XCLAIM is 65.4% cheaper than ACCS. Note, that a significant cost factor in XCLAIM (BTC,ETH) are the non-optimized BTCRelay maintenance fees, which e.g. account for 49.3% of the incurred cost for 1000 swaps.

IX. APPLICATIONS

This section provides a brief overview of several new and novel applications enabled by XCLAIM cryptocurrency-backed tokens. These applications illustrate how XCLAIM paves the way for usable and scalable cross-chain communication.

Cross-Chain Payment Channels: Cryptographic payment channels [47], [55], [73], [83], [88] address the performance limitations of blockchain protocols [50], [65] by avoiding the need to publish every transaction on the blockchain. Instead, transactions are executed directly between participants *off-chain*, and only the final balances of the participants are published on the blockchain. Despite improving transaction throughput and latency considerably, payment channels cannot execute payments across different blockchains. As such, users are required to setup and instantiate multiple channels, one for every blockchain they wish to participate in. With XCLAIM, however, payment channels deployed on a blockchain capable of issuing XCLAIM CBAs become cross-blockchain compatible automatically; users can transfer CBAs as per normal in a payment channel network, and later redeem those CBAs for native coins. As such, XCLAIM allows existing issuing blockchains, such as Ethereum, to process transactions of any backing cryptocurrency off-chain, without requiring changes to the underlying code. XCLAIM can therefore be used to provide novel contributions to state of the art payment channels.

Temporary Transaction Offloading: The design of XCLAIM allows for both long-term and short-term issuance of CBAs. As such, during temporary periods of high network congestion [43] or transaction fee spikes [59], XCLAIM CBAs can be used to temporarily switch to another blockchain for secure payment processing. Moreover, users can temporarily leverage this technique in XCLAIM to exploit features or benefits that may be present in the issuing chain, but not on the backing chain. For example, Bitcoin users may temporarily switch to Bitcoin-backed tokens on Ethereum to avoid long transaction processing times, or to leverage more complex transaction scripts and smart contracts in Ethereum. Once such periods have passed, users may exchange their CBA back to coins on the native blockchain securely, without requiring trust.

N-Way and Multi-Party Atomic Swaps: XCLAIM is more efficient than atomic cross-chain swaps (ACCS), both in terms of performance and cost (see Section VIII). In addition, XCLAIM can be leveraged to perform more complex and intricate swap constructions. For example, XCLAIM enables *N-way atomic swaps*: by extending the *Swap* protocol in XCLAIM, users can swap multiple different units of cryptocurrencies for others, e.g. trading units of cryptocurrency x and y against units of w and z atomically within a single swap. Comparing this to ACCS, N-way swaps are impractical, as they would require the creation of N locking and spending transactions, while monitoring all involved chains for failures.

In addition, XCLAIM also enables *multi-party atomic swaps*. Assume Alice owns coin x and wants to acquire coin y owned by Bob. Bob, however, will only trade y for coin z owned by Carol. Attempting to resolve this situation with ACCS would require Alice to separately swap with Carol and then with Bob, i.e., this process would not be atomic, resulting in 8 transactions on 3 different chains [4]. However, with XCLAIM, if x, y and z are CBAs, Alice can construct a non-interactive multi-party swap via the iSC, where a single transaction can change the ownership of all 3 coins in iSC, atomically.

X. RELATED WORK

The only existing mechanism to perform a trustless cross-chain transfer today is *atomic cross-chain swaps* (ACCS) based on hashed timelocks [4], [5], [69], [105]. Although ACCS enable trustless exchanges, they are *interactive*, i.e., they rely on all parties being online and monitoring the blockchain throughout the exchange to ensure security. Each swap thereby incurs long waiting periods to prevent fraud through exploiting blockchain reorganizations, which substantially hinders performance and involves synchronizing clocks between independent blockchains. Moreover, ACCS are vulnerable to packet and transaction memory-pool sniffing, allowing an adversary to exploit blockchain race conditions to steal funds. Finally, ACCS rely on a pre-established out-of-band communication channel between parties, required to exchange security-critical *revocation transactions* [38], [104]. XCLAIM not only avoids these problems, as discussed throughout the paper, but is also more efficient in terms of execution costs and time.

Most existing approaches towards CBA systems require trust in intermediaries. Liquid [52], RSK [82] and PoA Network [25] use permissioned blockchains on the issuing side, where a pre-defined set of validators is trusted with control over assets on the backing chain. As such, these systems resemble CENTRALCLAIM. In contrast, XCLAIM makes no such trust assumptions and, in fact, can be applied to improve security of CBAs issued on or from permissioned blockchains, just like with CENTRALCLAIM. Dogethereum, a trustless CBA approach similar to XCLAIM was described in a report released online subsequently [103]. Bentov et al. describe how to tokenize exiting cryptocurrencies via trusted execution environments (TEEs) [38]. TEEs however, are known to be vulnerable to a wide range of side-channel attacks [67], [110], [113] and require trust in the hardware manufacturer.

Other approaches attempting to achieve blockchain interoperability, such as Polkadot [22], [111], Cosmos [81], AION [101] and COMIT [70], to this date, only achieve communication between instances of their own permissioned blockchains, i.e., they support sharding [49] rather than cross-chain communication. XCLAIM can connect these systems to permissionless blockchains such as Bitcoin and Ethereum.

XI. CONCLUSION

In this paper, we formalized the notion of cryptocurrency-backed assets. We presented XCLAIM, a system for issuing, transferring, swapping and redeeming cryptocurrency-backed assets between blockchains, without necessitating trust. We provided a detailed analysis of XCLAIM's design, a formal protocol specification and identified requirements for the underlying blockchains. XCLAIM is general in design and supports many existing cryptocurrencies without modification. We implemented XCLAIM (BTC,ETH) to construct Bitcoin-backed tokens on Ethereum and evaluated the performance and execution costs; XCLAIM achieves a significant improvement over atomic cross-chain swaps. Finally, we outlined several novel and interesting applications enabled by XCLAIM.

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] "Btcrelay," 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] "Dogerelay," 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] "Etherdelta," https://etherdelta.com/, accessed: 2018-07-30.

[14] "Ethereum Classic," https://github.com/ethereumproject, accessed: 2018-05-23.

[15] "Etherscan - ropsten testnet explorer," https://ropsten.etherscan.io/, accessed: 2018-05-23.

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

[17] "Komodo barterdex," https://github.com/KomodoPlatform/KomodoPlatform/wiki/BarterDEX-%E2%80%93-A-Practical-Native-DEX, accessed: 2018-11-28.

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

[19] "Light client protocol," https://github.com/ethereum/wiki/wiki/Light-client-protocol, accessed: 2018-11-20.

[20] "Neo whitepaper," http://docs.neo.org/en-us/, accessed: 2018-07-30.

[21] "Omnilayer specification," https://github.com/OmniLayer/spec, accessed: 2018-05-03.

[22] "Parity-Bridge," https://github.com/paritytech/parity-bridge, accessed: 2018-05-21.

[23] "The parity light protocol - wiki," https://wiki.parity.io/The-Parity-Light-Protocol-(PIP), accessed: 2018-10-30.

[24] "Peace relay," https://github.com/loiluu/peacerelay, accessed 2018-04-17.

[25] "Poa bridge," https://github.com/poanetwork/poa-bridge, accessed: 2018-05-23.

[26] "Project alchemy," https://github.com/ConsenSys/Project-Alchemy, accessed 2018-04-17.

[27] "Solidity progamming language," https://github.com/ethereum/solidity, accessed: 2018-07-30.

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

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

[30] "Decred cross-chain atomic swapping," https://github.com/decred/antomicswap, 2017, accessed: 2018-05-16.

[31] E. Androulaki, C. Cachin, A. De Caro, and E. Kokoris-Kogias, "Channels: Horizontal scaling and confidentiality on permissioned blockchains," in *European Symposium on Research in Computer Security*. Springer, 2018, pp. 111–131.

[32] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *International Conference on Principles of Security and Trust*. Springer, 2017, pp. 164–186.

[33] 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.

[34] A. Back *et al.*, "Hashcash-a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002, accessed: 2016-03-09.

[35] C. Baldwin, "Bitcoin worth $72 million stolen from bitfinex exchange in hong kong," *Reuters*, accessed: 2018-05-23. [Online]. Available: https://www.reuters.com/article/us-bitfinex-hacked-hongkong-idUSKCN10E0KP

[36] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity." *IACR Cryptology ePrint Archive*, vol. 2018, p. 46, 2018.

[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.

[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." *IACR Cryptology ePrint Archive*, vol. 2017, p. 1153, 2017.

[39] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ACM, 2012, pp. 326–349.

[40] "Pay-to-Pubkey Hash," https://en.bitcoinwiki.org/wiki/Pay-to-Pubkey_Hash, bitcoin.it, accessed: 2018-11-28.

[41] "Script," https://en.bitcoin.it/wiki/Script, bitcoin.it, accessed: 2018-11-28.

[42] S. Blake-Wilson and M. Qu, "Standards for efficient cryptography (sec) 2: Recommended elliptic curve domain parameters," *Certicom Research, Oct*, 1999.

[43] "Bitcoin transaction fees," https://www.blockchain.com/en/charts/transaction-fees?timespan=all, Blockchain.com, accessed: 2018-11-28.

[44] 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.

[45] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *Bulletproofs: Short Proofs for Confidential Transactions and More*. IEEE, 2018.

[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.

[47] Christian Decker and Roger Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[48] "Coinmarketcap," http://coinmarketcap.com/, CoinMarketCap, accessed 2016-09-10.

[49] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Googles globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, p. 8, 2013.

[50] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. Gün, "On scaling decentralized blockchains," in *3rd Workshop on Bitcoin and Blockchain Research, Financial Cryptography 16*, 2016.

[51] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–10.

[52] 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.

[53] J. R. Douceur, "The sybil attack," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 251–260.

[54] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Annual International Cryptology Conference*. Springer, 1992, pp. 139–147.

[55] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *To appear in the Proceedings of the IEEE), IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2019.

[56] "Predicting bitcoin fees for transactions," https://bitcoinfees.earn.com/, earn.com, accessed: 2018-11-28.

[57] "Ethereum fee estimates," https://ethgasstation.info/, ETH Gas Station, accessed: 2018-11-28.

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

[59] "Ethereum BlockSize History," https://etherscan.io/chart/blocksize, Etherscan.io, accessed: 2018-11-28.

[60] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX Security Symposium on Networked Systems Design and Implementation (NSDI'16)*. USENIX Association, Mar 2016. [Online]. Available: http://www.usenix.org/system/files/conference/nsdi16/nsdi16-paper-eyal.pdf

[61] 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.

[62] 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.

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

[64] P. Gaži, A. Kiayias, and D. Zindros, "Proof-of-stake sidechains," in *To appear in the Proceedings of the IEEE Symposium on Security & Privacy*. IEEE Computer Society Press, 2019.

[65] A. Gervais, G. Karame, S. Capkun, and V. Capkun, "Is bitcoin a decentralized currency?" in *IEEE Security & Privacy*, vol. 12, no. 3, 2014, pp. 54–60.

[66] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 3–16.

[67] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*. ACM, 2017, p. 2.

[68] 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.

[69] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. ACM, 2018, pp. 245–254.

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

[71] Intel, "Intel software guard extensions (intel sgx) sdk," https://software.intel.com/sgx-sdk, accessed: 2018-05-23.

[72] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *Proceedings of the 27th USENIX Conference on Security Symposium*. USENIX Association, 2018, pp. 1353–1370.

[73] R. Khalil, A. Gervais, and G. Felley, "Nocust–a non-custodial 2 nd-layer financial intermediary," *IACR Cryptology ePrint Archive*, vol. 2018, p. 1642.

[74] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," *IACR Cryptology ePrint Archive*, vol. 2017, p. 963, 2017.

[75] 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.

[76] A. Kiayias and D. Zindros, "Proof-of-work sidechains," 2019.

[77] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," https://peercoin.net/assets/paper/peercoin-paper.pdf, Aug 2012, accessed: 2017-01-07. [Online]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf

[78] N. Koblitz, "Cm-curves with good cryptographic properties," in *Annual international cryptology conference*. Springer, 1991, pp. 279–287.

[79] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[80] D. Kraft, "Difficulty control for blockchain-based consensus systems," in *Peer-to-Peer Networking and Applications*, vol. 9, no. 2, 2016, pp. 397–413.

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

[82] S. D. Lerner, "Rootstock: Bitcoin powered smart contracts," https://docs.rsk.co/RSK_White_Paper-Overview.pdf, 2015.

[83] 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.

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

[85] L. Luu, B. Buenz, and M. Zamani, "Flyclient super light client for cryptocurrencies," accessed 2018-04-17. [Online]. Available: https://stanford2017.scalingbitcoin.org/files/Day1/flyclientscalingbitcoin.pptx.pdf

[86] 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.

[87] 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.

[88] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," in *Financial Cryptography and Data Security*, 2019.

[89] 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.

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

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

[92] J. Nash, "Non-cooperative games," *Annals of mathematics*, pp. 286–295, 1951.

[93] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic game theory*. Cambridge University Press, 2007.

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

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

[96] R. Pass, L. Seeman, and a. shelat, "Analysis of the blockchain protocol in asynchronous networks," *IACR Cryptology ePrint Archive*, vol. 2016, p. 454, 2016.

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

[98] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 515–532.

[99] I. Sergey, A. Kumar, and A. Hobor, "Scilla: a smart contract intermediate-level language," *arXiv preprint arXiv:1801.00687*, 2018.

[100] Y. Sompolinsky and A. Zohar, "Bitcoin's security model revisited," *arXiv preprint arXiv:1605.09193*, 2016.

[101] M. Spoke and Nuco Engineering Team, "Aion: The third-generation blockchain network," https://aion.network/media/2018/03/aion.network_technical-introduction_en.pdf, accessed 2018-04-17.

[102] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," https://truebit.io/, March 2017, accessed:2017-10-06.

[103] J. Teutsch, M. Straka, and D. Boneh, "Retrofitting a two-way peg between blockchains," Tech. Rep., 2018. [Online]. Available: https://people.cs.uchicago.edu/~teutsch/papers/dogethereum.pdf

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

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

[106] V. Buterin, "Chain interoperability," https://static1.squarespace.com/static/55f73743e4b051cfcc0b02cf/t/5886800ecd0f68de303349b1/1485209617040/Chain+Interoperability.pdfi, 2016, accessed: 2018-11-30.

[107] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 991–1008.

[108] J. Von Neumann and O. Morgenstern, "Theory of games and economic behavior," 1944.

[109] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.

[110] 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*. Springer, 2016, pp. 440–457.

[111] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White Paper*, 2015.
[112] K. Wu, S. Wheatley, and D. Sornette, "Classification of cryptocurrency coins and tokens by the dynamics of their market capitalizations," *Royal Society Open Science*, vol. 5, no. 9, p. 180381, 2018.
[113] 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.
[114] 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.

APPENDIX A
ACKNOWLEDGEMENTS

The authors would like to thank Nicholas Stifter, Pedro Moreno-Sanchez, Andrew Miller, Georgia Avarikiot, Peter Smith and Edgar Weippl for helpful comments and insightful discussions. The authors also wish to thank the IEEE S&P reviewers for their valuable comments that significantly improved the presentation of our results. This research was sponsored by Blockchain.com, Outlier Ventures, Bridge 1 858561 SESC, Bridge 1 864738 PR4DLT (all FFG), the Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQI), and the competence center SBA-K1 funded by COMET.

APPENDIX B
SYMBOLS AND NOTATIONS

Table III provides an overview of symbols and variables used in this paper.

TABLE III
Summary of used symbols and notations.

Symbol	Description
B	Backing blockchain
b	Cryptocurrency unit underlying the backing blockchain B
b_{lock}	Units of b locked by a *requester* with a *vault* during *Issue*
I	Issuing blockchain
i	Cryptocurrency unit underlying the issuing blockchain I
i_{col}	Units of i locked by a user with the iSC
$i(b)$	Unit of a CBA on I backed by units of b on B
k^X	Security parameter of blockchain X. Defines how many confirmations are necessary for secure transaction inclusion
Δ^X	Maximum delay from transaction broadcast to secure inclusion in blockchain X
$\overline{r_\tau}$	Upper bound for deviations of expected and observed ratio between block generation rates of B and I.
Δ_{tx}^X	Maximum delay from transaction broadcast to receipt by honest consensus participants in X
$\varepsilon_{(i,b)}$	Exchange rate of i to b, given by oracle $\mathcal{O}$
$\min \varepsilon_{(i,b)}$	Lower bound for $\varepsilon_{(i,b)}$ below which econom. rational adversaries are incentivized to misbehave
$\Delta_{\min(\varepsilon)}$	Delay before $\varepsilon_{(i,b)}$ falls below lower bound $\min(\varepsilon_{(i,b)})$
τ^X	Block generation rate of blockchain X
$(\mathsf{pk}_u^X, \mathsf{sk}_u^X)$	Public / private key pair of user u on blockchain X
σ_u^X	Digital signature of user u on chain X, i.e., via sk_u^X
T_{id}^X	Transaction created on blockchain X with identifier id
$\mathsf{operation}(in) \rightarrow out$	Operation taking in as input an generating output out
operation	Short for operation with default inputs and outputs
$cond$	Conditions used to lock coins, e.g. presenting a user's digital signature σ_{user}
Δ_{id}	Time delay introduced in XCLAIM's protocols with identifier id
r_{col}	Ideal (parameterized) collateralization rate of *vaults*
r_{col}^*	Observed collateralization rate of a *vault* in XCLAIM
r_{col}^{liq}	Collateralization threshold for automatic liquidation

APPENDIX C
ATOMIC *vault* REPLACEMENT

Until now, we have assumed a *vault* cannot locked funds b_{lock} on B and must remain in XCLAIM until the latter is fully redeemed by users. In a real world scenario, the *vault* may wish to leave and transfer their role to another party earlier, or move funds to a different account on B for practical purposes. To this end, we describe *Replace*, a non-interactive atomic cross-chain swap (ACCS) protocol based on cross-chain state verification, which allows *vaults* to move funds on B without being punished by the iSC.

Protocol: Replace. *vault* migrates locked funds b to $vault'$, who replaces *vault*'s collateral in the iSC.

1) *Setup.* The *vault* submits a *replacement* request to the iSC and locks up collateral $i_{col}^{replace}$, sufficient to cover costs of a transaction on I.
2) *Lock.* A new candidate $vault'$ can lock the corresponding amount of collateral for a pre-defined period $\Delta_{replace}$ with the iSC on I, such that $|i_{col}^{vault}| = |i_{col}^{vault'}|$, providing their public key on backing chain B.
3) *Migrate.* Within $\Delta_{replace} > \Delta_{relay}$, the still active *vault* must migrate the locked b_{lock} to the public key of $vault'$ on B and submit the corresponding transaction inclusion proofs to the chainRelay on I.
4) *Release.* The chainRelay verifies the migration was executed correctly on B and the iSC releases the old *vault*'s collateral, i.e., both i_{col}^{vault} and $i_{col}^{replace}$. If the *vault* does not execute the migration on B within $\Delta_{replace}$, the iSC releases the new candidate's collateral, while using $i_{col}^{replace}$ to reimburse wasted transaction fees.

Performance and Costs: We implement *Replace* in Solidity v0.4.24 and measure the performance and execution costs of *Replace* under the conditions described in Section VIII. The results of our experiments are presented in Table IV. Note: the duration and costs of *Replace* depend on the number of Bitcoin UTXOs which need to be migrated. The provided numbers are hence lower bounds.

Collateral Balancing: We can further use the *Replace* protocol to enable re-balancing of collateral among *vaults*. Assume a *vault*'s observed collateralization rate r_{col}^{*} has dropped significantly below ideal rate r_{col}. To prevent automatic liquidation, the *vault* must contribute additional collateral to the iSC on I. Alternatively, the *vault* can execute *Replace* to migrate a fraction of total locked coins b_{lock} to $vault'$, so as to re-balance her collateralization rate r_{col}^{*}. Should no single *vault* have sufficient free collateral to complete the re-balancing, the *Replace* protocol can be executed iteratively with multiple *vaults*, e.g. until $r_{col}^{*} \geq r_{col}$ holds. This procedure is specifically useful if a *vault* cannot provide additional collateral due to insufficient funds on I but intends to ensure secure operation of iSC.

APPENDIX D
PROOF-OF-WORK CHAIN RELAY MODEL

As discussed in Section V-B, a *chain relay* is a program deployed on a blockchain, capable of reading and verifying the state of some other blockchain, comparable to the notion of SPV-Clients [3]. That is, a chain relay stores and maintains block headers and allows to verify transaction inclusion proofs.

In the following, we provide a formal model for the requirements of a program π to represent a functioning chain relay for a proof-of-work blockchain $\mathcal{C}$. Then, we discuss the practical challenges faced by chain relays today.

protocol *Replace*
- $vault$.lock($i_{col}^{replace}, \sigma_{SC}^{I}$)
- $vault'$.lock($i'_{col}, \sigma_{SC}^{I}$)
- $vault$.transferB(b_{lock}, $\mathsf{pk}_{vault'}^{B}$) /* → $T_{transfer}^{B}$ */
- $vault$ submits $T_{transfer}^{B}$ to iSC calling verifyBOp
- **if** iSC.verifyBOp(transferB, $T_{transfer}^{B}, \Delta_{replace}$) = ⊤ **then**
  - iSC.release($i_{col}, \mathsf{pk}_{vault}^{I}$)
  - iSC.release($i_{col}^{replace}, \mathsf{pk}_{vault}^{I}$)
- **else**
  - iSC.release($i_{col'}, \mathsf{pk}_{vault'}^{I}$)
  - iSC.slash($i_{col}^{replace}, pkIvault'$)

Fig. 5. Algorithm specifying XCLAIM's *Replace* protocol for securely moving / swapping locked funds b_{lock} on B.

TABLE IV
Performance and execution costs of XCLAIM's *Replace* protocol.

Transactions	Cost (USD) Ethereum	Bitcoin	Total	Duration (minutes)
3^{Eth} 2^{Btc}+	0.13	0.62	0.75	148.97+

A. Model

Notation. We denote $H(x)$ as the output of a cryptographic hash function over some input x. Further, $\mathcal{B}_i$ shall denote the block header of the block at position i in the blockchain, represented by the tuple $\langle S_{i-1}, \mathcal{T}_i, M_i, N_i, t_i \rangle$, where

- S_{i-1} is the reference to the PoW hash (i.e., solution) of the predecessor of block i,
- $\mathcal{T}_i$ is the (expected) PoW difficulty target at block i as defined by consensus rules,
- M_i is the root hash of the Merkle tree of the hashes of all transactions $(T_0, T_1, ..., T_n)$ included in i,
- N_i is the random nonce used to generate the PoW solution hash $S_i = H(\mathcal{B}_i)$,
- and t_i is the timestamp specifying when block i was generated.

We refer to the header of the first ($i = 0$) or so called *genesis* block as $\mathcal{G}$. We assume protocol rules of C require that the PoW difficulty target $\mathcal{T}$ is adjusted every r blocks based on the relation of the time between each two adjustments and some pre-defined desired block generation rate τ. Note, while potentially useful for more extensive block validity checks, for simplification we ignore other information usually included in the block headers. Furthermore, as it is not of greater relevance to our model, we assume the same cryptographically secure hash function $H()$ is used to calculate both the hashes of block headers and transactions.

We require a chain relay program π to support the following functionalities with regards to the state of a Proof-of-Work blockchain $\mathcal{C}$:

Functionality 1 (Difficulty Adjustment).
Program π has knowledge of the difficulty adjustment rate r and ideal block generation rate τ and, given $\mathcal{B}_i$ and $\mathcal{B}_{i+r}$, where $i \pmod i = 0$, outputs the new difficulty target $\mathcal{T}_{i+r}$ according to consensus rules of $\mathcal{C}$.

Functionality 2 (Block Validation).
Program π has a function `checkBlock` *which takes as input a block header $\mathcal{B}_i$ and a PoW solution S_i, and returns* `True` *if and only if $\mathcal{B}_i$ is the pre-image of S_i, $\mathcal{T}_i$ is the difficulty target required at block i and it holds that $S_i \leq \mathcal{T}_i$.*

Functionality 3 (Chain Validation).
Program π has a function `checkChain` *which takes as input the genesis block $\mathcal{G}$, a list of consecutive block headers $(\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_n)$ and the list of corresponding PoW solutions $(S_1, S_2, ..., S_n)$, and returns* `True` *if and only if for each tuple $(\mathcal{B}_i, S_i)|i \leq n$, it holds that* `checkBlock`$(\mathcal{B}_i, S_i) =$ `True` *and for each two consecutive block headers $\mathcal{B}_i$ and $\mathcal{B}_{i+1}$ it holds that $S_i \in \mathcal{B}_{i+1}$, i.e., $\mathcal{B}_i$ is the predecessor of $\mathcal{B}_{i+1}$.*

Definition 1 (Valid Chain).
We define the tuple $\langle \mathcal{G}, (\mathcal{B}_1, ..., \mathcal{B}_n), (S_1, ..., S_n) \rangle$ as a valid chain *if* `checkChain` *outputs* `True` *given this tuple as input.*

Functionality 4 (Main Chain Detection).
Program π provides a function denoted `mainChain` *which takes as input two valid chains $\langle \mathcal{G}, (\mathcal{B}_1, ..., \mathcal{B}_i, \mathcal{B}_{i+1}, ..., \mathcal{B}_n), (S_1, ..., S_i, S_{i+1}, ..., S_n) \rangle$ and $\langle \mathcal{G}, (\mathcal{B}_1, ..., \mathcal{B}_i, \mathcal{B}'_{i+1}, ..., \mathcal{B}'_m), (S_1, ..., S_i, S'_{i+1}, ..., S'_m) \rangle$ where $n \neq m$ and for every $j \geq i$ it holds that $\mathcal{B}_j \neq \mathcal{B}'_j$, $S_j \neq S'_j$, and outputs the main chain according to the consensus rules of $\mathcal{C}$, e.g., the longest chain in the case of Nakamoto consenus.*

Functionality 5 (Transaction Inclusion Verification).
Program π provides a function `checkTransaction` *which, if given a valid chain $\langle \mathcal{G}, (\mathcal{B}_1, ..., \mathcal{B}_n), (S_1, ..., S_n) \rangle$, a block header $\mathcal{B}_i$, a transaction T and a Merkle tree path p, outputs* `True` *if and only if $H(T)$ is contained in the Merkle tree with root $M_i \in \mathcal{B}_i$ at the position defined by p, $\mathcal{B}_i \in (\mathcal{B}_1, ..., \mathcal{B}_n)|i \leq n$ and the provided chain is the main chain of $\mathcal{C}$.*

Definition 2 (Chain Relay).
A program π is a chain relay *of a Proof-of-Work blockchain $\mathcal{C}$, if it satisfies Requirements 1-5 with regards to $\mathcal{C}$.*

B. Practicability of Chain Relays

Existing chain relays [6], [24], [26] require to store all block headers of the verified blockchain, which can incur substantial cost. Furthermore, the verification of PoW requires to implement the verification procedure for the respective hash functions. However, if native support for the necessary cryptographic primitives is not provided in the blockchain the relay is deployed on, verification may be infeasible. For example, Ethereum currently only allows feasible chain relays for a subset of existing of blockchains, which: (i) use the same PoW algorithm, e.g. Ethereum Classic [24], or (ii) SHA-256 available as pre-compiled contract, such as Bitcoin [6] and Namecoin [91].

The main performance and cost challenges of chain relays as such lie in (i) efficient PoW / hash pre-image verification and (ii) reduction of necessary input data (block headers). The former can in theory be achieved by using non-interactive zero-knowledge proof (ZKP) systems such as SNARKs [39], STARKs [36] or Bulletproofs [45] for hash pre-image verification (assuming the corresponding arithmetic circuits can be efficiently constructed). Alternatively, the verification of PoW can be outsources to users [9], [102], with disputes handled via interactive games or ZKPs in smart contracts. However, existing schemes have been shown to currently suffer security challenges [72].

New proposals for efficient SPV clients, leveraging concepts such as NiPoPoW [74], [76] or FlyClient [85], can significantly reduce verification costs by reducing the number of stored block headers. At the moment of writing, however, both require additional data to be included in the block headers of the to-be-verified blockchain and do not provide details on how to handle difficulty adjustments. While this can be achieved as a velvet fork [74], [114], the performance improvement and security of these schemes rely on the availability of this data.

Finally, deploying chain relays in trusted execution environments (TEEs) (e.g. Intel SGX [71]) may present a cheap and scalable approach to cross-chain state verification, at the cost of trusting hardware manufacturers. However, recent vulnerabilities detected in well known TEE implementations, in particular to side-channel attacks [67], [107], [110], [113], highlight existing security risks and raise questions about current applicability to financial systems. Furthermore, deploying blockchain clients in trusted execution environments may require modifications to the original implementation, increasing the long term maintenance costs and potentially introducing compatibility issues with protocol upgrades, e.g. hard forks.