XCC: Theft-Resilient and Collateral-Optimized Cryptocurrency-Backed Assets

Theodore Bugnet
Interlay
Imperial College London

Alexei Zamyatin
Interlay
Imperial College London

***Abstract*—The need for cross-blockchain interoperability is higher than ever. Today, there exists a plethora of blockchain-based cryptocurrencies, with varying levels of adoption and diverse niche use cases, and yet communication across blockchains is still in its infancy. Despite the vast potential for novel applications in an interoperable ecosystem, cross-chain tools and protocols are few and often limited.**

Cross-chain communication requires a trusted third party, as the Fair Exchange problem is reducible to it. However, the decentralised consensus of blockchains can be used as a source of trust, and financial incentives can achieve security. XCLAIM uses these principles to enable collateralised cryptocurrency-backed assets to be created and used. However, full collateralization is inefficient, and to protect against exchange rate fluctuations overcollateralization is necessary. This is a significant barrier to scaling, and as a result, in practice, most systems still employ a centralised architecture.

In this work, we introduce XCC, an extension to the XCLAIM framework which allows for a significant reduction in collateral required. By making use of periodic, timelocked commitments on the backing blockchain, XCC decouples locked collateral from issued CBAS, allowing fractional collateralization without loss of security. We instantiate XCC between Bitcoin and Ethereum to showcase practical feasibility. XCC is compatible with the majority of existing blockchains without modification.

I. INTRODUCTION

Ever since the success of Bitcoin [29], introduced in 2008 by Satoshi Nakamoto, a vast market of cryptocurrencies has emerged. Over 8000 individual currencies, the majority of them blockchain-based, with a total market cap exceeding $2.47 trillion, are currently in circulation [7]. However, the majority of blockchains operate independently and do not natively provide methods to interoperate with other chains. Indeed, distributed cross-chain applications are non-trivial to implement. Often, a central trusted intermediary is used, as is the case for instance with traditional centralised cryptocurrency exchanges. But this can be seen as running counter to the vision for a distributed ecosystem with no trusted authorities and no central points of failure. In fact, over the last decade, over $1.8 billion worth of cryptocurrency has been lost due to compromises in centralised exchanges [9].

XCLAIM [37] aims to offer a solution to this problem, by introducing cryptocurrency-backed assets (CBAS). CBAS involve locking funds on one chain (the *backing chain*) and issuing new corresponding tokens — the backed assets — on another chain (the *issuing chain*). The CBAS can then be redeemed for the original funds at any time, which ensures their value remains pegged to the value of the backing funds. While distinct from two-way swaps as facilitated by traditional exchanges, CBAS, thanks to their pegged value, can be used to effectively trade on one blockchain using funds from another.

XCLAIM employs *vaults* to ensure backing assets are correctly locked while CBAS exist, and are unlocked when CBAS are redeemed. Any participant can become a vault and a smart contract is employed on the issuing chain to enforce correct protocol execution, tracking the backing chain via a cross-chain light client, a so-called *chain relay* [36]. To achieve economic trustlessness, XCLAIM requires vaults to bond collateral in the smart contract. This ensures rational vaults are disincentivised from breaching the protocol, while allowing users to be reimbursed should a vault misbehave.

This approach has two drawbacks. Firstly, this means that full collateralization of all CBAS existing in the system is required, which is arguably not capital efficient. Secondly, since the collateral is locked in the smart contract on the issuing chain, its value relative to the backing funds is vulnerable to exchange rate fluctuations. Due to this, XCLAIM requires *overcollateralization*. As cryptocurrency exchange rates can be quite volatile, the amount of overcollateralization required to ensure secure operation is significant; the original specification suggests an ideal collateral factor of at least 2x for secure operation. This is a significant barrier to large-scale use of XCLAIM, as vaults must be willing to put down significant capital amounts for any CBAS issued. Furthermore, since CBAS can exist indefinitely, vaults cannot reliably predict for how long their collateral is locked.

A. Contributions

In this paper, we extend XCLAIM to greatly reduce the collateralization required in the protocol. We present XCC, a novel, round-based CBA framework which combines XCLAIM with commit chains [26], a technique used for scalable off-chain transaction processing. XCC ensures backing assets are cryptographically locked on the backing chain for pre-defined periods, avoiding the need for continuous collateral lockup, and introducing the concept of *on-demand collateralization*. Vaults broadcast regular on-chain checkpoint transactions to update the on-chain state of the CBA ledger and renew the cryptographic locks at the user's request. By construction, XCC ensures users can always exit the system, even if a vault is offline. As a result, XCC greatly improves the security

of user's funds, making vaults *non-custodial*, and decouples collateralization from the amount of issued CBAS improving capital efficiency.

The contributions of this paper can be summarized as follows:

- We present XCC (short for XCLAIM Commit), a novel framework for cryptocurrency-backed assets which combines XCLAIM with the concept of commit chains, making vaults non-custodial and significantly reducing collateralization requirements. We present a step-by-step migration from XCLAIM to XCC and provide a detailed security analysis.
- We provide a formal protocol specification and analyse blockchain requirements. While XCC requires smart contracts on the issuing chain, only limited scripting functionality (hash and timelocks) is necessary for ensuring the safety of backing funds, enabling XCC to support Bitcoin and similar systems as backing chain.
- We present multiple extensions to XCC, showcasing (i) how CBAS can be issued and maintained with *zero collateral* at the cost of reduced usability and stricter online requirements for users; (ii) cross-vault transfers; (iii) and how users can run their own vaults for personal use cases.
- We present a proof of concept implementation for XCC Bitcoin-backed assets on Ethereum, demonstrating practical feasibility. We analyse running costs and offer a comparison to XCLAIM, providing case studies for different user types. We observe XCC is significantly more cost-efficient for users who execute occasional transfers or moderate trading activity, while a temporary fallback to XCLAIM is recommended for periods of high frequency trading.

II. BACKGROUND

In this section, we provide the relevant background on cross-chain communication and XCLAIM. In consideration of space, we do not aspire to provide a complete description of the working of Bitcoin and distributed ledgers, hence recommend readers unfamiliar with these research fields to consult existing literature, such as [17, 21, 32].

A. Cross-Chain Communication

Given the proliferation of different cryptocurrencies with different properties and use cases, it becomes desirable to achieve some sort of interoperability between them. Cross-chain communication (CCC) can arise in a variety of different settings, such as between sibling chains in sharded systems (e.g. [28], [35]), merged mining parent/child chains ([3], [25]), or between completely separate, heterogeneous chains. Here, we focus on the latter case.

Impossibility, Assumptions, and Models: Unfortunately, cross-chain communication is not solvable without a trusted third party [36]. The simplest example of a heterogeneous CCC implementation is a fully centralised exchange, which allows a user to swap assets held on one chain for assets on another. However, the "trusted third party" can be abstracted away from a physical party; for instance, protocols such as Hash Time-Lock Contract (HTLC) Atomic Swaps [8] [24] implement a two-phase commit, requiring all parties to be online for correct execution of the swap. Here, the synchrony assumption is used to ensure protocol fairness: if a participant fails to be online throughout its execution, the protocol aborts. The consensus layer of the underlying blockchains stands in for the trusted third party, by enforcing timelocks on the transactions. Alternatively, on blockchains that support it, it is possible to use on-chain smart contracts as trusted third parties — again, the trust is put into the blockchain consensus.

The protocols mentioned so far implement a two-way swap between two blockchains. An alternative use case for CCC is one-way transfers, such as those used for cryptocurrency-backed assets [31] [37]: funds x on a backing chain X are locked, while on a different chain Y, assets $y(x)$ are *backed* by those funds. Ownership changes in $y(x)$ are propagated back to x. Similarly to two-way swaps, the trusted third party can be partially abstracted to e.g. a smart contract.

A useful way to consider cross-chain protocols is to aim for economic trustlessness: in other words, building protocols where rational actors acting in self-interest never benefit from misbehaving. An actor not acting rationally might still cause failure from a protocol point of view, but should not be able to inflict financial damage on other participants.

Cross-Chain State Verification: An important aspect of cross-chain communication is the ability to prove, on one chain, the state of another chain. In particular, if a chain supports smart contracts, one might wish to prove to a contract that, for instance, a transaction occurred on a different chain.

To achieve this practically, due to the cost of computation and memory on a blockchain, generally requires using light clients, which allow verification of blockchain state without running an entire node. Not all blockchains natively support light clients; for instance, the Ethereum specification is under development at the time of writing [12]. Bitcoin, on the other hand, was designed to support light clients from the outset [29]; its implementation is known as Simplified Payment Verification (SPV) [13]. SPV on Bitcoin operates by obtaining block headers, but not block data, which allows the client to eschew storing the transaction history for the entire blockchain. For instance, BTCRelay [10] is an Ethereum smart contract implementing Bitcoin SPV, allowing verification of Bitcoin transaction inclusion on the Ethereum chain for a fee.

As an alternative to "naive" SPV verification, wherein block headers are directly submitted for the smart contract to verify and store, using zero-knowledge proofs [16] can allow for off-chain proofs of header chain validity. This provides potentially more efficient — and therefore cheaper — on-chain verification [33].

B. XCLAIM

XCLAIM [37] is a framework for cryptocurrency-backed assets between two heterogeneous blockchains. It uses a smart contract as a trusted party, and financial incentives in the form of collateral to achieve an economically trustless protocol.

XCLAIM allows users to lock funds on a *backing* blockchain, and obtain an equivalent amount of tokens on

an *issuing* blockchain. Those cryptocurrency-backed tokens can then be traded and used entirely using transactions on the issuing chain. Any user holding tokens can then redeem them back for the same amount of backing funds on the original chain. Thus, the issuing chain — and its consensus and transaction mechanisms — can be employed to trade value across the backing chain.

The smart contract powering the framework resides on the issuing chain, which is therefore required to support Turing complete scripting. In contrast, the backing chain has fewer requirements to be suitable for use in XCLAIM — for instance, Bitcoin can be used as the backing chain. To allow the smart contract to act upon events occurring in the backing chain, a cross-chain relay between the backing chains and issuing chains is integrated.

XCLAIM relies on *vaults*, entities which lock funds on the backing blockchain. To be issued cryptocurrency-backed tokens, users send their backing funds to a Vault, submit proof of the transaction to the smart contract, and are issued newly-minted tokens in exchange. The Vault's responsibility is to hold the funds and to provide them back to a user who wishes to redeem their tokens for backing currency. XCLAIM allows a distributed, scalable architecture, where any number of vaults can operate and any user can register as a Vault.

Security via Chain Relays and Overcollateralization: To achieve financial security and trustlessness, XCLAIM employs collateralization of vaults: before a Vault can lock funds for a user, it must bond a sufficient amount of collateral with the smart contract, in the form of assets on the issuing blockchain. If a Vault refuses or is unable to operate correctly (e.g. fails to honour a request to redeem funds), then its collateral is slashed and used to reimburse affected users, who regain assets equal in value to the backing funds they lost (albeit on a different blockchain) and therefore suffer no economic loss.

However, since the collateral consists of assets on the issuing chain, while the collateralised funds are on the backing chain, exchange rate fluctuations allow the value of the collateral to fluctuate relative to the value of the funds it is meant to collateralise. To remedy this, XCLAIM employs *overcollateralization*. The security model assumes an upper bound on the rate of change of the exchange rate; based on this, collateral ratio thresholds are set, below which the Vault is considered to be under-collateralised and appropriate actions are taken. The original XCLAIM specification [37] suggests an ideal collateralization factor of 2, which could however be set higher if more security is desired; below a factor of 1.5, the specification suggests the Vault should no longer be permitted to serve further issuing requests by locking additional funds until the Vault rebalances its collateral. Additionally, there are automatic liquidation measures if a Vault refuses to rebalance, to avoid the possibility of the collateral's value ever falling below the value of the collateralised funds.

C. Commit Chains

Commit chains, first introduced by the NOCUST framework [26], are a blockchain scaling technique leveraging off-chain transactions. NOCUST employs a central operator who is responsible for managing an off-chain ledger and routing payments across users.

To ensure the correct behaviour of the operator, a smart contract is employed on-chain. At regular intervals, the operator commits the state of all user accounts via an on-chain checkpoint transaction, notifying the contract. Users may then challenge the checkpoint; if the operator fails to respond with a proof of correct operation, the contract rolls back ownership of funds to the approved previous checkpoint and halts the system, allowing users to recover their funds.

Properties and Security: Commit chains require users to come online at least once between every checkpoint to verify that the operator updated their balances correctly - and issue a challenge in case of inconsistencies. It is worth noting, however, that in a naive implementation, operators are capable of censoring users since transfers ignored by the operator are not visible to the smart contract.

A practical drawback of commit chain is that transfers are not final until they are committed to a checkpoint *and* no challenges are issued until the *next* checkpoint. To remedy this, some implementations, including NOCUST, allow operators to put down collateral as guarantee of correct behaviour, enabling instant transfer finalization.

III. System Model and Assumptions

In this section, we introduce the distributed ledger, network, and threat models for interlinked blockchains B and I, as well as system model and actors in XCC, including relevant notation. For a full table of the symbols used in this work, refer to Appendix A.

A. Distributed Ledger Model

We use the terms *distributed ledger* and *blockchain* as synonyms and adapt the distributed ledger model based off [21, 36]

Ledgers and State Evolution. We consider two distributed systems B and I that each consist of a set of participants and employ a consensus protocol to agree on a sequence of transactions. We assume the security model (threat, network and cryptographic assumptions) of the consensus protocol holds: the fraction of consensus participants f or computational power α corrupted by an adversary is bounded by the threshold necessary to ensure the security of the consensus protocol. For Proof-of-Work blockchains, we therefore assume $\alpha \leq 33\%$ [20, 23, 30].

To achieve consensus on the sequence of transactions, B and I each maintain a distributed ledger structured as a blockchain. i.e., an append-only sequence of blocks where each block contains a reference to its predecessor(s), e.g. via a hash. We define the state of a ledger L as a dynamically evolving sequences of transactions $\langle \mathsf{T}_1, ..., \mathsf{T}_n \rangle$. We assume that the state of the ledger progresses in discrete *rounds*, indexed[1] by a natural number $r \in \mathbb{N}$. At each round r, a new set

[1]The index of a ledger is often referred to as the blockchain *height*.

of transactions is *included* in the ledger L. We use $\mathsf{L}[r]$ to denote the state of ledger L at round r, i.e., after applying all transactions included in the ledger since round $r-1$. A transaction can be included in or *written* to L only if it satisfies the consensus rules of the system, given the current state of the ledger. This consistency is left for each particular system to define, and we describe it as a free predicate $\mathsf{valid}(\cdot)$.

Notion of Time. The state evolution of ledgers L_b and L_i may progress at different *time* intervals (i.e., B and I may exhibit different block generation rates [21]. To correctly capture the ordering of transactions across L_b and L_i, we define a clock function τ which maps a given round on any ledger to the time on a global, synchronized clock $\tau : r \rightarrow t$. We assume B and I are nevertheless synchronized and that there is no clock drift between them. We use this conversion implicitly in the rest of this paper.

Persistence and Liveness. Each party of the system maintains a local ledger that depicts its current view of the system. The views of two parties on the same ledger may differ (e.g., due to network delays or message loss). However, eventually, all honest parties in the distributed ledger will have the same view. This is encapsulated by the persistence and liveness properties of the distributed ledger, as defined in [22]:

- *Persistence:* Parameterized by persistence (or "depth") parameter k, if transaction T appears in the ledger of an honest party at time t, then it will eventually appear in the ledger of all other honest parties at time $t' \leq t + k$ ("stable" transaction).
- *Liveness:* Parameterized by liveness delay parameter u, if an honest party attempts to write a valid transaction T to its ledger at time t, then T will appear in its ledger at time $t' \leq t + u$.

In this work, we only consider distributed systems that maintain robust transaction ledgers [22], meaning they satisfy persistence and liveness with high probability to the security parameters.

B. Transaction and UTXO Model

We proceed to introduce the transaction and UTXO model for Bitcoin-like blockchains, based on [15]. Account-based systems, such as Ethereum [19], can replicate the functionality relevant for XCC and we hence use the UTXO model as a base framework. Specifically, we assume that B uses a UTXO transaction model, as seen in Bitcoin-like cryptocurrencies.

In the UTXO (Unspent Transaction Output) model, transactions consist of *inputs* and *outputs*: a transaction (i) takes as input or *consumes* coins locked in one or more unspent outputs of existing transactions, and (ii) maps these coins to new outputs, specifying how, when and by whom these coins can be spent, i.e., used as input for another transaction. The total amount of assets controlled by a user is hence computed from the set of outputs from which she can spend. UTXOs are outputs that have not yet been spent and *can only be spent as a whole*.

Transactions are specified by a list of inputs and outputs; for instance, $\mathsf{T}_i = \left[o_j^1(w_1)\right] \mapsto \left[o_i^1, o_i^2\right]$ denotes a transaction T_i which spends the first output o_j^1 of transaction T_j, providing witness data w_1 (such as a user's signature), and creates two outputs o_i^1 and o_i^2. We use $\#$ to denote when it does not matter what outputs are used as inputs for a given transaction.

Outputs are described by the amount of funds locked into them, and the conditions that need to be met to spend them. In this paper, we differentiate between the following conditions in accordance with Bitcoin, which can be composed and combined using logical operators $\wedge$, $\vee$ and $\neg$:

- **Signature locks**, denoted as σ_A, require that a valid signature over the spending transaction, generated by the private key of A (Alice), be provided. The combination of multiple signature locks is called a *multisignature* lock.
- **Hashlocks**, denoted as $H(s)$, requires the revelation of the pre-image s of $H(s)$, where $H()$ is a secure cryptographic hash function.
- **Timelocks**, denoted as $\Delta(t)$, specify a delay t after which a UTXO can be spent. In the rest of the paper, we make use of "relative" timelocks, i.e., $\Delta(t) = \Delta(r)$ determines the number of rounds that must elapse after the transaction T generating the associated UTXO has been included in the underlying ledger L.

For example, $(\sigma_{\mathsf{A}} \mid 5b)$ denotes an output which holds 5 b coins and can be spent by anyone able to generate a signature associated with a user A's private key — assumed to be A herself.

C. Network Model

For the underlying network, we make the same assumptions as in prior work [27], [28], [37], namely that (i) honest nodes are well connected and (ii) communication channels between these nodes are (semi-)synchronous, i.e., messages sent between honest participants will be received within a known maximum delay Δ. Additionally, we assume that all participants are aware of the smart contract iSC, and hence of the values publicly stored in it.

D. Threat model

We assume that the cryptographic primitives of B and I are secure and that adversaries are computationally bounded. Adversaries are fully adaptive, i.e can freely choose controlled/corrupted participants (Vaults and users) in the XCC protocol for each time window. Adversaries may also perform arbitrary actions to maximize their economic value, such as delay or withhold transactions on protocol-level, read unconfirmed transactions in the network, and perform Sybil attacks. Note that under these assumptions (and the persistence and liveness properties of I) the adversary cannot tamper with the correct execution of the smart contract iSC.

To keep track of and react to exchange rate fluctuations between i and b, we assume an oracle $\mathcal{O}$ provides the iSC with the exchange rate $\varepsilon_{(i,b)} \in \mathbb{R}_{\geq 0}$. We further assume the fluctuations in the exchange rate within a time interval Δ_ε is bounded by a maximum value $\overline{\varepsilon_{(i,b)}}$.

E. XCC System Model and Actors

XCC allows users to create symmetric cryptocurrency-backed assets (CBAs), as defined by XCLAIM [37]. Specifically, XCC operates between two blockchains: a backing blockchain B of a cryptocurrency b and an issuing blockchain I with native currency i. A cryptocurrency-backed asset on I backed by B is a ("wrapped") representation of B's native currency b on I and is denoted as $i(b)$. In this paper we consider symmetric CBAs, i.e., assets $i(b)$ issued on I are backed by the *equivalent* amount of funds b on B, i.e., $|i(b)| = |b|$.

XCC differentiates between the following actors, following XCLAIM:

- **User**. A user in XCC can take up any of the following roles:
  - **Requester**. Locks b on B to request $i(b)$ on I.
  - **Sender**. Owns $i(b)$ and transfers ownership to another user on I.
  - **Receiver**. Receives ownership over $i(b)$ on I.
  - **Redeemer**. Destroys $i(b)$ on I to receive the corresponding amount of b on B.
- **Vault**. A Vault is a (non-trusted) intermediary responsible for securely holding backing funds b locked on B until they are redeemed for $i(b)$.
- **Smart Contract** (iSC). A public smart contract responsible for managing the correct issuing and transfer of $i(b)$ and ensuring correct behaviour of Vaults.

All actors are identified by their public/private key pairs (or the representation thereof, e.g. accounts or addresses) on B and I.

F. System Goals

XCC is an extension deployed on top of the XCLAIM framework[37]. As such, under the blockchain, network and threat models specified above, in Sections III-E-III-D, we derive the following desirable *security* properties for XCC, based on XCLAIM [37]:

- **Auditability.** Any user with read access to blockchains B and I can audit the operation of XCC 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., $|b| = |i(b)|$.
- **Strong Redeemability.** Any user is guaranteed to be able to redeem CBA $i(b)$ for backing currency b on B during publicly known *secure* periods S_Δ. In the intervals between *secure* periods, denoted as T_Δ, a user can redeem CBA $i(b)$ for backing currency b on B or will be reimbursed with equivalent economic value on I in case of failure.
- **Weak Liveness.** Any user in XCC redeem CBAs without requiring a third party, relying only on the secure operation of B and I. Issuing and transferring CBAs requires interaction with the Vault holding b locked on B.
- **Atomic Swaps.** Users can atomically swap XCC CBAs against other assets on I or the native currency i.

In addition, we derive the following desirable *functional* properties for XCC:

- **Strong Scale-Out.** The total amount of CBAs $i(b)$ available for circulation increases with the total amount of backing currency b locked up on blockchain B but is *independent* of the collateral locked up on I. *Any* user can contribute to the amount of $i(b)$ by assuming the role of the *Vault* and/or locking b.
- **Weak Compatibility.** XCC does not depend on the features of any specific blockchain and is compatible with any pair of blockchains meeting a minimum set of requirements: smart contract support[2] on the issuing blockchain I; time-locks, hashlocks, and basic fund transfers on the backing blockchain B.

IV. Design Roadmap: From XCLAIM to XCC

In this section, we provide a high-level overview of the core protocols of the XCLAIM framework, highlight XCLAIM's main drawbacks, and lay out a design roadmap for the more secure and collateral-efficient design of XCC.

A. The XCLAIM Protocols

We provide a high-level step-by-step description of the XCLAIM protocols to issue, transfer, and redeem assets $i(b)$ on an issuing blockchain I backed by b locked on a backing blockchain B. A detailed protocol specification can be found in [37].

XCLAIM Protocol: Issue. Alice (*requester*) locks units on b with the Vault on B to mint $i(b)$ on I:

1) *Setup.* One or more Vaults register with the smart contract iSC, bonding collateral i. The amount of backed-assets $i(b)$ that can be issued is determined by the exchange rate $\varepsilon_{(i,b)}$ between b and i, and XCLAIM's over-collateralization rate r_{col}: $\max(i(b)) = \frac{i_{col}}{r_{col} \cdot \varepsilon_{(i,b)}}$. Before starting the issue process with a Vault, Alice verifies that the Vault has sufficient collateral locked in the iSC.
2) *Request.* Alice requests to issue an amount of $i(b)$ with iSC, specifying her public key on I and temporarily reserving the corresponding amount of collateral $i_{col} = i(b) \cdot \varepsilon_{(i,b)}$.
3) *Lock.* Alice locks funds b ($|b| = |i(b)|$) with one or more Vaults on B using via a standard on-chain asset transfer.
4) *Prove.* Alice submits a transaction inclusion proof to iSC, attesting that she correctly locked b.
5) *Issue.* The smart contract iSC verifies the transaction inclusion proof via a chain relay and, if successful, issued $i(b)$ to Alice on I.

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

1) *Transfer.* Alice notifies the iSC that she wishes to transfer her $i(b)$ to Dave on I. The state of iSC is updated and Dave becomes the new owner of $i(b)$.
2) *Witness.* The Vaults witnesses the change of ownership on I by observing iSC, and will no longer allow Alice to withdraw the transferred $i(b)$ for locked b.

[2]Turing completeness is not required, cf. [37] Section VI-B.

XCLAIM Protocol: Redeem. Dave (*redeemer*) locks $i(b)$ with the iSC on I to receive b from one or more Vaults on B; $i(b)$ is then destroyed and the Vaults' collateral i_{col} unlocked:

1) *Lock.* Dave locks $i(b)$ with the iSC on I, requesting redemption of $i(b)$ and specifying his / the recipient public key on B.
2) *Release.* One or more Vaults are selected to execute the redeem request by the iSC and proceed to release funds b to Dave on B, such that $|b| = |i(b)|$.
3) *Prove.* Next, each Vault involved in the redeem process submits a transaction inclusion proof to the chain relay component of the iSC, attesting that they correctly released b to Dave.
4) *Burn.* Finally, the iSC verifies the proofs and, if successful, destroys or burns the corresponding amount of $i(b)$ and releases the associated locked collateral i_{col} to the Vaults on I.

B. XCLAIM Properties and Limitations

XCLAIM allows users to issue, transfer and redeem CBAS without requiring a third party (**Liveness**) and logs all transactions either B or I (**Auditability**). Since users must prove correct lock-up of b with Vaults when issuing $i(b)$, XCLAIM also achieves **Consistency**. Thereby, XCLAIM only requires standard transfers to be supported by the backing chain B and supports fully fungible CBAS (**Compatibility**), making $i(b)$ easily tradeable against any other asset on I (**Atomic Swaps**).

However, XCLAIM exhibits two main drawbacks hindering its practicability. When issuing CBAS, users transfer custody over b to Vaults, who are required to bond collateral to protect users against financial damage. Nevertheless, Vaults may still choose to steal funds b despite losing collateral i, e.g. triggered by private information such as speculation on future exchange rates. As a result, when redeeming CBAS, XCLAIM can only guarantee that a user will either be able to redeem $i(b)$ for b or will be reimbursed in the equivalent (or higher) amount of i in case of Vault failure (**Redeemability**). While this provides significantly better guarantees than any existing centralized service, unexpectedly receiving collateral i instead of b may impose challenges on applications making use of $i(b)$.

Since the collateral is locked up in a different currency than b, XCLAIM must employ *over-collateralization* to mitigate exchange rate fluctuations and prevent financial damage to users. In practice, this makes the supply of available $i(b)$ dependent on the amount of b locked by users on B *and* the collateral i locked by Vaults on I (**Scale-out**). For example, if we assume XCLAIM's suggested collateralization rate of 2.0, the overall capital requirement of minting a single unit of $i(b)$ is 300% of its economic value. Another challenge is that there is no time limit on the lifetime of CBAS. This means Vaults do not know for how long they must bond their collateral on I, making it difficult to estimate the opportunity costs of their invested capital. As a result, Vaults in XCLAIM must charge high fees to users during the issue and redeem processes, yielding a framework less attractive to users from an economical standpoint than trusted but more cost-efficient, centralized services.

C. XCC Design Roadmap

To address the security and scalability limitations of XCLAIM, we outline the design roadmap for XCC, introducing the core components used in its construction.

- In Section V-B, we prevent vaults from accessing users' funds, save for short, pre-determined periods, and ensure users can always redeem CBAS without a Vault's involvement, thus achieving **Strong Redeemability**. For this, we introduce a novel *checkpointing* mechanism on the backing chain B. When issuing CBAS, users lock b in *timelocked commitments* on B, which ensure b cannot be moved. Vaults aggregate and renew these commitments periodically via on-chain checkpoints on B. Pre-signed recovery transactions, spending from dedicated multisignature outputs in commitments, ensure users can always recover their funds on B.
- In Section V-C, we achieve **Strong Scale-out** by significantly reducing the collateral requirements for Vaults and allowing *fractional collateralization* without loss of security. Specifically, Vaults must only bond collateral during short periods when checkpoints are renewed and, through parallelization and cascading, can use the same collateral to securely maintain numerous checkpoints and scale to thousands of users.
- Finally, in Section V-D we achieve **Atomic Swaps** by ensuring users can instantly transfer CBAS, independent of the checkpointing scheme, via on-demand collateralization by Vaults.

V. XCC PROTOCOL DESIGN

This section presents the design of XCC. We first provide a high-level overview and intuition of the protocol, and then follow the design roadmap laid out in Section IV-C to detail XCC's design.

A. Overview

XCC overcomes the limitations of XCLAIM by adding theft prevention mechanisms for vaults on the backing chain B, thus removing the need for collateralization on I, save for very short intervals. Similarly to XCLAIM, users must lock b_{lock} on B and submit a proof of this to the iSC when issuing CBAS $i(b)$ on I. However, instead of transferring funds directly into a Vault's custody, users lock b in transactions encumbered by timelocks, guaranteeing that b cannot be spent until the timelock expires. To ensure CBAS can exist for an indefinite period and are easily transferable, vaults temporarily receive custody over b_{lock} to renew timelocks upon expiry. Thereby, vaults aggregate funds from multiple users into checkpoints — transactions on B which move users' b_{lock} into new timelocked commitments. During the short period when funds b_{lock} are under a Vault's control, the Vault locks up collateral i_{col} on I to guarantee economic security to users.

Funds in timelocked commitment can be recovered earlier from multisig outputs by providing both the user's and

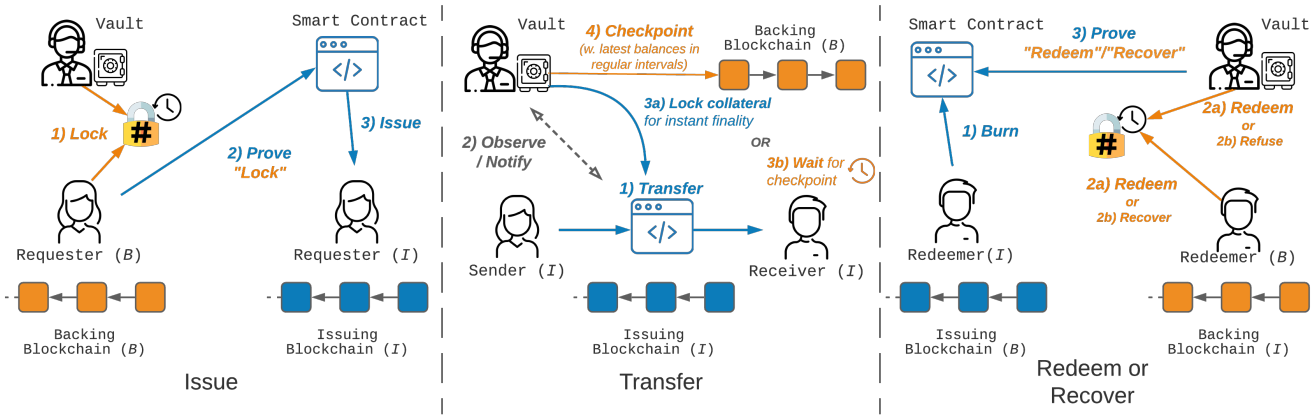


Fig. 1: High-level overview of the Issue, Transfer, and Redeem protocols in XCC.

Vault's signatures. To ensure users can always exit the system, vaults distribute *pre-signed recovery transactions* to all users involved in a checkpoint.

Coordination on timelock duration and checkpoint composition happens via the iSC on I and, in some cases, can be offloaded to off-chain channels. Just like in XCLAIM, users can run their own vaults, while relying on third-party vaults is a matter of convenience and usability.

Since backing funds b_{lock} cannot be stolen by Vaults except during checkpoint renewal, there is no need for Vaults to maintain full collateralization continuously. Instead, Vaults only provide collateral shortly before executing a checkpoint, which can be limited to e.g. 2 hours in the case of Bitcoin. If a Vault fails to bond collateral, a single honest, online party can broadcast a recovery transaction to return b_{lock} to affected users. To service a large number of users, Vaults can cascade independent checkpoints and, by avoiding overlaps, re-use the same collateral i_{col} to secure funds of thousands of users, s.t. $|i(b)| = |b_{lock}| > |i_{col}|$.

The ability of users to always recover their backing funds introduces a limitation to XCC CBAS in terms of transferability. Changing the ownership of $i(b)$ on I must be reflected in checkpoints on B (update of recovery conditions), which can incur delays of several hours in the worst case. To enable instant transfers, XCC vaults hence offer *on-demand collateralization*: users can request a Vault to bond collateral $|i_{col}| > |b_{lock}|$ until the update in $i(b)$ ownership is reflected in a checkpoint on B.

In the following sections we introduce six XCC protocols encompassing the above functionality - *Issue, Checkpoint, Transfer, Redeem, Recovery, Instant Transfer*, sometimes extending protocols with additional steps, necessary for features introduced in later sections. A final and formal protocol specification is then presented in Figure 5.

B. Timelocked Commitments, Checkpoints and Recovery

As outlined in Section IV-C, XCC makes use of timelocked commitments and an interactive checkpointing scheme to secure backing funds b_{lock} on B, while allowing users to independently recover their funds. We now proceed to detail the necessary transactions on the backing chain B and the resulting modifications to XCLAIM's issue, transfer and, redeem processes.

Initially, we assume the Vault is *still fully (over-) collateralized*, following the XCLAIM model. As we shall see in Section V-C, the newly introduced timelocked commitments and checkpointing mechanism allow us to relax the collateralization requirements - described in Section V-C below.

1) Issuing to Timelocked Commitments: First, we modify how backing funds are locked in the XCLAIM Issue protocol. Instead of simply transferring b_{lock} to a Vault and giving away custody over the assets, a user A locks b_{lock} in a contract which prevents them from being spent by the Vault for a pre-defined period - the so-called *checkpoint duration* Δ_V. Specifically, A creates a transaction T_{Issue} as follows:

$$\mathsf{T}_{Issue} = [\#] \mapsto [(\sigma_{\mathsf{V}} \wedge \Delta_V) \vee (\sigma_{\mathsf{V,A}} \wedge \Delta_{Rec})].$$

That is, user A creates a transaction the output of which can be spent under two conditions:

1) $(\sigma_{\mathsf{V}} \wedge \Delta_V)$: the Vault can spend b_{lock} after a delay Δ_V, or
2) $(\sigma_{\mathsf{V,A}} \Delta_{Rec})$: the Vault and A can cooperate to spend the locked assets from the multsig output after a delay Δ_{Rec}, where $\Delta_{Rec} < \Delta_V$.

The execution of Issue then proceeds as follows:

XCC Protocol: Issue. User A locks b_{lock} on B with the Vault V and obtains $i(b)$ on I:

1) *Registration.* A verifies the smart contract iSC is available on the issuing blockchain I, and records her public key on B, as well as her preferences for Δ_V and Δ_{Rec}, with the contract.
2) *Setup.* A selects the Vault V she wishes to use on the backing chain B; and comes to an agreement on Δ_V and

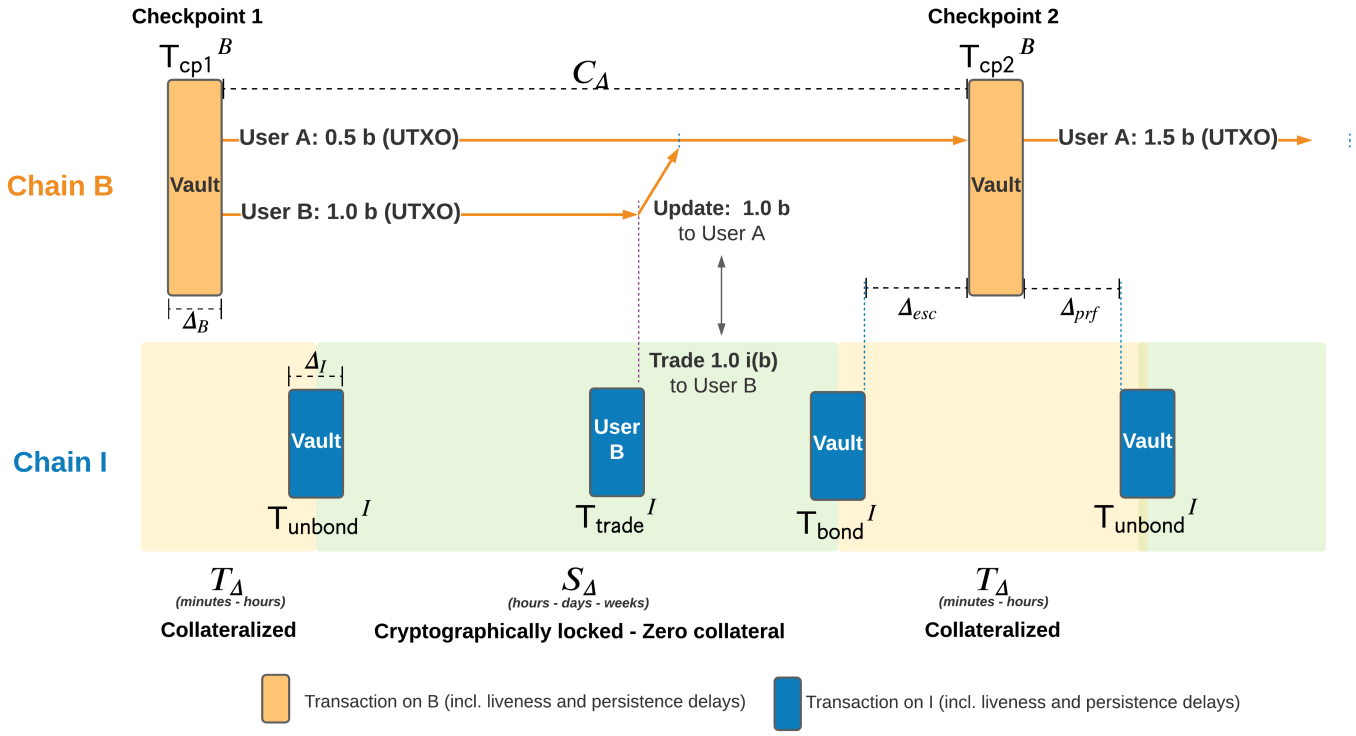


Fig. 2: Overview of the XCC checkpoint process. User A and B have b timelocked with a Vault on B for duration Δ_C (Checkpoint 1). The Vault unbonds its collateral i_{col} once Checkpoint 1 is completed. User B trades 1.0 $i(b)$ to user B, which will be reflected in Checkpoint 2 on B. Shortly before timelock Δ_C expires (Δ_{esc}), the Vault re-bonds collateral on I. Once Checkpoint 2 is stable on B, the Vault submits a proof to the iSC and unbonds its collateral.

Δ_{Rec}, and any service fees, with V (if necessary, updating her account information with the iSC). [3]

3) *Lock.* A broadcasts T_{Issue}, as described above, on B.
4) *Prove.* A submits a transaction inclusion proof for T_{Issue} to the the iSC, proving that the funds b_{lock} are locked with V.
5) *Issue.* The iSC verifies T_{Issue}, ensuring the σ_{V} signature lock corresponds to the specified Vault's public key, σ_{A} corresponds to A's public key recorded during *Registration*, and that the timelocks Δ_V and Δ_{Rec} are correct. The contract then creates and credits to A the corresponding amount of $i(b_{lock})$, such that $|i(b_{lock})| = |b_{lock}|$.

2) Periodic Checkpoints: Once the *Issue* protocol is executed correctly, the user's backing assets b_{lock} are locked in a timelock contract and cannot be moved/spent until the timelock expires. Once the timelock expires, Vault V receives full custody over the locked funds and is instructed to renew the timelock such that b_{lock} remain locked while $i(b_{lock})$ exists. This process is defined in a new *Checkpoint* protocol.

To renew timelocks of locked backing assets, the Vault creates and broadcasts a new, so-called *checkpoint* transaction on B. The checkpoint transaction takes as input (spends) the outputs of (a) an issue transaction, or (b) a previous checkpoint transaction, and exhibits the same output structure as T_{Issue} described above. Thereby, a single checkpoint transaction can refresh timelocks of multiple users at the same time: a separate input and output are included for every user participating in the *checkpoint*.

We define a *checkpoint timeout* Δ_{CP} such that $\Delta_{CP} - \Delta_V$ provides sufficient time for the Vault to broadcast the checkpoint on B and notify the iSC.

The execution of Checkpoint therefore proceeds as follows:

XCC Protocol: Checkpoint. The Vault V refreshes the timelocks of one or more of its users.

1) *Construct.* The Vault creates a transaction $\mathsf{T}_{\mathrm{CP}_{i+1}}$, where i is the index of the last checkpoint, as follows. For each user U (whose Δ_V timelock has expired):
   - *Add as input* the corresponding output of the previous checkpoint transaction $\mathsf{T}_{\mathrm{CP}_i}$ or, if this is the first renewal for this user, the output of the user's issue transaction T_{Issue}.
   - *Add an output* $((\sigma_{\mathsf{V}} \wedge \Delta_V) \vee (\sigma_{\mathsf{V,A}} \wedge \Delta_{Rec}) \mid b_{lock}{}^{\mathsf{U}})$ to $\mathsf{T}_{\mathrm{CP}_{i+1}}$
2) *Broadcast.* V broadcasts $\mathsf{T}_{\mathrm{CP}_{i+1}}$ (providing the necessary signatures as witness data), and notifies the smart contract

[3] It is possible for the user to proceed here without the Vault's agreement; however in such a case the Vault needs to take no action and will suffer no penalty, while the user's only course of action will be to exit the system.

iSC.

3) *Prove.* The Vault submits a transaction inclusion proof for the checkpoint transaction $\mathsf{T}_{\mathrm{CP}_{i+1}}$ to the iSC.
4) *Verify.* The iSC verifies for each user/output the correctness of (i) the updated timelocks, (ii) the amount of b_{lock} locked, and (iii) the signature and timelock parameters. In case of any inconsistencies, the Vault's collateral is slashed, and the affected users are reimbursed using the Vault's collateral i_{col}.
5) *Timeout.* For any given user, if Δ_{CP} since their last checkpoint has passed and the Vault has not submitted a checkpoint, it is considered to be timed out. The iSC slashes Vault collateral for that user, if any, and burns that user's tokens.

With the introduction of checkpoints, specifically multisignature outputs, we implicitly altered the process required to redeem CBAS: both the user's and the Vault's signatures are required for a cooperative *Redeem*. Note: we introduce a Recovery mechanism allowing users to exit the system even if the Vault is non-cooperative in the next section.

XCC Protocol: Redeem (Cooperative). The user redeems their tokens for backing funds, with cooperation from the Vault.

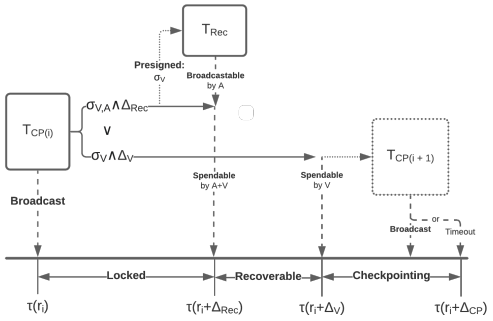
1) *Setup.* The user and Vault prepare and co-sign a transaction spending from the user's last checkpoint's output, and returning the funds to the sole custody of the user.
2) *Unlock.* Any time after Δ_{Rec} has passed since the checkpoint, the transaction is broadcast.
3) *Burn.* The contract burns the corresponding tokens, either upon notification from any party, or at the latest after the checkpoint timeout Δ_{CP} since the user's last checkpoint.

The checkpoint scheme further requires us to redefine the XCLAIM *Transfer* protocol: in XCC, CBA transfers on I only become final once they have been included in a checkpoint on B and the later has been verified by the iSC. We introduce a protocol for *instant transfers* in Section V-D.

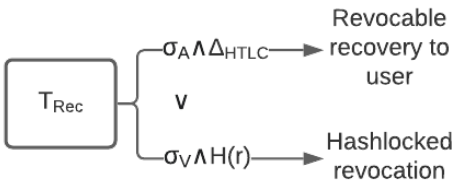
XCC Protocol: Transfer. A user transfers XCC CBA funds to another user on I, finalized by a checkpoint on B.

1) *Request.* The user submits the transfer request to the iSC, which records it. The Vault observes this.
2) *Finalise.* At some point in the future, the Vault broadcasts the next checkpoints for both the sender and the receiver, with the updated output amounts.
3) *Record.* Once the iSC has verified correct backing fund allocations for both users, the token balances are updated.

3) User Recovery: While holding the backing funds in a timelock prevents theft by the Vault, in practice that will merely delay a malicious Vault, as the user has no recourse to recover their funds if the Vault is dishonest: once the timelock expires, the Vault gains control over b_{lock} is can commit theft. We remedy this by introducing the *Recovery* protocol: a mechanism allowing users to pre-empt a malicious Vault and regain control of their backing funds b_{lock} if necessary, which makes use of a *recovery transaction* spending from the multisignature lock/condition of the user's checkpoint output.



(a) Each user's full output conditions from a checkpoint transaction



(b) The pre-signed (but not yet broadcast) recovery transaction and its outputs

Fig. 3: Visualization of a checkpoint for a single user. (a) shows the checkpoint transaction, its output conditions and timelocks, and the pre-signed recovery transaction. (b) shows in detail the output of the recovery transaction.

XCC Protocol: Checkpoint - Extension. When creating the checkpoint CP_i for a user A (during the *Construct* phase of the *Checkpoint* protocol) the Vault pre-signs a *recovery transaction*, $\mathsf{T}_{Rec_i^{\mathsf{A}}}$. This transaction spends from A's output in CP_i, and returns the backing funds ${b_{lock}}^{\mathsf{A}}(\mathrm{CP}_i)$ to A on B. The Vault publishes this signature to the iSC when submitting the checkpoint for verification during the *Prove* step of the *Checkpoint* protocol. User A can add her signature and broadcast $\mathsf{T}_{Rec_i^{\mathsf{A}}}$ to recover funds on B - and is are the only user able to do so (digital signature).

XCC Protocol: Recovery. A user unilaterally regains custody of their backing funds.

1) *Recover.* Any time after Δ_{Rec} has passed since the last checkpoint, the user signs the already partially-signed recovery transaction T_{Rec} stored in the iSC and broadcasts it on B.
2) *Verify.* Any user or Vault (latest during the next checkpoint) submits an SPV proof for T_{Rec} to the iSC.
3) *Burn.* The iSC verifies the recovery transaction and burns user A's $i(b)$.

Security Argument for Strong Redeemability. XCC by design ensures that the user's funds b_{lock} are always (a) cryptographically locked on the backing chain B in multisig and timelocked outputs (S_Δ), or (b) are in the Vault's custody but secured by

locked collateral i_{col} on I (T_Δ). In the edge case that the Vault fails to bond collateral i_{col} before Δ_{Rec}, the user can recover her b_{lock} using the **Recovery** protocol, i.e., by broadcasting the recovery transaction T_{Rec} on B.

To steal a user's b_{lock} the Vault would hence have to achieve one of the following:

1) Forge user A's signature or to include an invalid transaction in B to incorrectly spend from the outputs of the checkpoint. This is only possible if Persistence of B does not hold or if the adversary is not computationally bound (i.e., can obtain the private key from A's public key) - which is a contradiction to the model.
2) Tamper with the smart contract iSC on I to illicitly withdraw collateral i_{col}. This is a contradiction to the model.
3) Forge a transaction proof from B to the iSC. This is a contradiction to the model, as this is equivalent to proving a valid double-spend, but the adversary is computationally bound.
4) Prevent the user from broadcasting the recovery transaction T_{Rec}. This again is a contradiction to the model (Persistence and Liveness of B, and synchronous channels between honest parties).

Alternatively, a dishonest user could attempt to steal from a different user by broadcasting a **Redeem** or **Recover** transaction, but without having notified the contract to burn the corresponding tokens. The dishonest user could then attempt to execute **Transfer**. In this edge case, the delayed finality of **Transfer** in XCC ensures that the receiver will not receive the unbacked fraudulent tokens, and the transfer will fail.

To successfully steal from another user by transferring unbacked tokens, the malicious user would have to achieve one of the following:

1) Tamper with the smart contract iSC on I to force the invalid transfer to go through. This is a contradiction to the model.
2) Forge a transaction proof for a successful checkpoint from B to the iSC. As above, this is a contradiction to the model.

We conclude that under the model, a user can redeem $i(b)$ for b during S_Δ, or be reimbursed in collateral i_{col} during T_Δ. Hence XCC achieves **Strong Redeemability**. □

Security Argument for Auditability. All the parameters necessary to reconstruct protocol execution, including the Vault's signature for the Recovery transaction, B addresses for the user and Vault, timelock durations, etc., are recorded in the iSC. By the assumptions of the model, it directly follows that all participants can observe these values, and hence verify the correct execution of the protocol. Thus, **Auditability** is achieved. □

Security Argument for Weak Liveness. A user is able to:

- Execute Issue by broadcasting T_{Issue} on B, and notifying the iSC on I: no third party is required.
- Redeem their funds by broadcasting $\mathsf{T}_{Rec_i^{\mathsf{A}}}$ on B as soon as Δ_{Rec} has passed since the last checkpoint: no third party is required.
- Execute Transfer by notifying the iSC on I, and waiting until the user's Vault updates the balances of b_{lock} for themselves and their recipient: cooperation of the vault is required.

Therefore, all the conditions for **Weak Liveness** are satisfied. □

C. Collateral Reduction

Due to **Strong Redeemability**, full collateralization of b_{lock} funds is no longer necessary in XCC. In particular, during the secure period S_Δ, a user is guaranteed the ability to redeem her $i(b_{lock})$ for the corresponding backing funds b_{lock}. As a direct consequence, the Vault is not required to lock collateral while timelocks are active (S_Δ), but only during the short periods T_Δ when the Vault gains full control over b_{lock} to renew time timelocks.

This section outlines how collateral can be more optimally managed, leading to satisfying the **Scale-out Collateralization** property by reducing total collateral required below the total value of the locked funds. In Section V-C1, we describe how security is maintained during checkpoints, by requiring that the Vault put down collateral a short period in advance. In Section V-C2 we describe how this can be used to minimise the amount of funds required to operate a Vault, by interleaving checkpoints and reusing collateral between them.

1) Periodic Checkpoint Collateralization: We specify that the funds must be collateralised by the Vault prior to the user's timelock expiring, i.e. prior to Δ_{Rec} since the latest checkpoint. We the define $\Delta_{col} = \Delta_{CP} - \Delta_{Rec}$, i.e Δ_{col} is the duration of time between Δ_{Rec} and the timelock expiring.

To achieve economic security, we therefore require that users come online at any point during Δ_{col} prior to their scheduled checkpoint, and verify that the Vault has correctly collateralised their funds. Should a Vault attempt to avoid collateralising funds and wait for the timelock to expire to steal the funds with no repercussions, the user(s) involved, upon observing this, broadcast their recovery transactions and regain custody of their backing funds.

After the checkpoint has been broadcast, this collateral is then released again once the iSC has verified its correctness.

2) Fractional Collateralization: Thanks to periodic collateralization, the Vault may interleave groups of users such that, after each group's checkpoint, the collateral is released before Δ_{Rec} of the next group's checkpoint. This allows the Vault to reuse the same liquid funds to collateralise alternating groups of users. As a result, the total amount of collateral required at any one time can be reduced below the total amount of funds locked across all users.

Arguments for Strong Scale-Out. Taking advantage of fractional collateralization, a Vault can securely lock the backing funds b_{lock} of a large number of users, while reusing the same i_{col} as collateral across multiple, cascaded checkpoints/user groups. Thus, the total amount of CBAs $i(b)$ that can be minted in XCC depends directly on the amount of b locked with Vaults and not on the total amount of collateral i_{col} the Vault can put down at any given point in time. By design, any user can

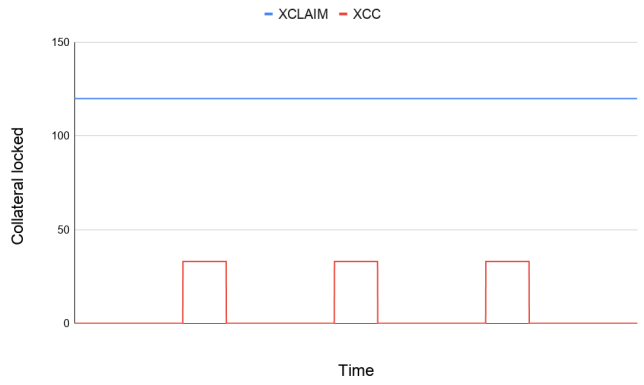


Fig. 4: A comparison between the collateral required for XCLAIM and XCC in an example scenario, with $\frac{1}{2}$ fractional collateralization (i.e. two even user groups participating in checkpoints, interleaved by the Vault), and 1.1 overcollateralization in XCC.

lock b_{lock} with one or more Vaults, or take up the role of a Vault herself by registering with the iSC and providing some amount of collateral i_{col}. The iSC by design does not restrict the set of Vaults, only verifying the collateral requirements at checkpoint renewal. We conclude that XCC achieves **Strong Scale-Out**. □

3) Reducing Overcollateralization: An additional benefit of the period collateralization is the significantly reduced requirement for overcollateralization. As collateral is locked in the smart contract in the issuing chain, its value relative to the backing funds (on the backing chain) may fluctuate as the exchange rate between the cryptocurrencies on the two chains fluctuates; thus, in both XCLAIM and XCC, overcollateralization is necessary to guard against the collateral becoming less valuable than the funds it secures.

XCLAIM recommends a collateralization factor of 2 or greater for these purposes. By contrast, in XCC, collateral is only locked for the duration of Δ_{col}, followed by the delay necessary for the checkpoint transaction to be confirmed on the backing chain, then verified on the issuing chain. Depending on the value of Δ_{col}, the total duration during which collateral is necessary, Δt, may be as short as several hours. As a result, the maximum change in exchange rate $\mathcal{E}(t)$ will generally be bounded much lower than for XCLAIM, which operates with an unbounded Δt (can be days, weeks, even months). Depending on the volatility of the cryptocurrencies XCC operates on at any given time, a collateralization factor of 1.2 or lower may be appropriate.

Further, if we assume that the rate of change of volatility — which is $\frac{\partial}{\partial t}\mathcal{E}(t)$, or the second derivative of the exchange rate — is bounded, then, given an appropriate oracle, the collateralization factor can be adjusted based on the volatility at any given time. During periods of low volatility, the collateralization factor could approach 1, thanks to the short Δt and bounded rate of change of volatility. During periods of increased volatility, the factor could be increased as necessary, allowing for continued secure operation of the protocol.

D. Instant Collateralised Transfer

In this section, we introduce the *revocable recovery* mechanism, which allows instant transfer finality at the cost of extra collateralization when the Vault is available and cooperating.

Note that at the time CP_i and Rec_i^{A} are created, A owns $i(b)^{\mathsf{A}}(\text{CP}_i)$ and therefore her checkpoint output — from which the recovery transaction spends — locks ${b_{lock}}^{\mathsf{A}}(\text{CP}_i)$. However, if prior to the next checkpoint, A trades some of her tokens to some other user B, it will be the case that $(i(b)^{\mathsf{A}} = {b_{lock}}^{\mathsf{A}}) < {b_{lock}}^{\mathsf{A}}(\text{CP}_i)$. But the recovery transaction cannot be adjusted after it has been created. As A can broadcast Rec_i^{A} at any time, this would result in theft, by A, of backing funds belonging to B. Therefore, currently, the Transfer protocol requires that users wait until the sender's next checkpoint before it is considered finalised.

For transfers to be securely final before the next checkpoint, the sender's current Recovery transaction must therefore be invalidated. To this end, a hashed timelocked contract (HTLC) is used as Rec_i^{A}'s output (illustrated in Fig. 3b):

$$\mathsf{T}_{Rec_i^{\mathsf{A}}} = \left[o^{\mathsf{A}}_{CP_i}(\sigma_{\mathsf{V}})\right] \mapsto \left[((\sigma_{\mathsf{A}} \wedge \Delta_{Rec}) \vee (\sigma_{\mathsf{V}} \wedge H(r_i^{\mathsf{A}})) \mid {b_{lock}}^{\mathsf{A}})\right]$$

In other words, to spend from Rec_i^{A}, one of two conditions must be specified: either A can simply provide her signature and claim the output, but only after the *recovery delay timelock*, Δ_{Rec}, has expired; or the Vault can spend the output with its signature, but only if it has the hash preimage r_i^{A}. This preimage is known to the user, who sets the corresponding hashes in the iSC in advance, allowing the Vault to construct and pre-sign the recovery transactions as required as well as maintaining **Auditability**.

Thus, to invalidate the recovery transaction, A records the preimage corresponding to its hashlock in the iSC, publicly revealing it. After doing so, should A try to broadcast the Recovery anyway, the Vault will be able to regain the funds before Δ_{Rec} expires, while A is yet unable to spend them.

This allows the A to potentially execute transfers, without the receiver bearing the risk of A then broadcasting their Recovery and gaining back the ostensibly traded away funds. However, a consequence of invalidating the Recovery transaction is that the secure period S_Δ ends, and the T_Δ period begins: therefore, the user's funds *must* be collateralised prior to performing this; ensuring this is the user's responsibility. We introduce a modified **Transfer** protocol for this.

XCC Protocol: Instant Transfer. A user transfers XCC-locked funds to another user with instant finality, with cooperation from the Vault.

1) *Request.* The user submits the transfer request to the iSC, which records it. Additionally, the user requests that the Vault put down collateral for their funds, if it has not already done so.

Operations:

Off-chain:

Off-chain, operations can be executed by the $user$ and the $Vault$.

- verifyCollateral(b_{lock}) $\rightarrow \top|\bot$: *(Executed by: user)* The user observes the state of the iSC and the current exchage between b and i, and ensures that $Vault$ has locked sufficient collateral locked to cover b_{lock}.

Additionally, we assume that both the user and Vault are able to observe and verify relevant transactions on both B and I, and will not proceed with a procedure if any party fails to perform an earlier step.

Backing Blockchain:

On the backing chain B, again the operation can be executed by the $user$ and the $Vault$.

- createLock(b$_{lock}$, $Vault$) $\rightarrow$ T$_{lock}$: *(Executed by: user)* Outputs a transaction $\mathsf{T}_{lock} = [\#] \mapsto [((\sigma_{Vault} \wedge \Delta_i^{user}) \vee (\sigma_{Vault,user}) \mid b_{lock})]$, which can be used to lock b_{lock} with $Vault$ on B.
- createUnlock(i(b), $user$) $\rightarrow \mathsf{T}_{unlock}$: *(Executed by: Vault)* Outputs a transaction $\mathsf{T}_{unlock} = [\#] \mapsto [(\sigma_{user} \mid b)]$, which returns the backing funds for $i(b)$ to the sole control of $user$.
- createCP($user, b_{lock}$) $\rightarrow \mathsf{T}_{CP}$: *(Executed by: Vault)* Outputs a transaction $\mathsf{T}_{CP} = [\#] \mapsto [((\sigma_{Vault} \wedge \Delta_i^{user}) \vee (\sigma_{Vault,user}) \mid b_{lock})]$ which re-locks the user's funds for another round (i.e. a checkpoint transaction).
- createRecovery($user, \mathsf{T}_{CP}$) $\rightarrow \mathsf{T}_{Rec_{user}}$: *(Executed by: Vault)* Creates a transaction $\mathsf{T}_{Rec_{user}} = [o_{\mathsf{T}_{CP}}(\sigma_{Vault})] \mapsto [((\sigma_{user} \wedge \Delta_{Rec}) \vee (\sigma_{user} \wedge H(r_i^{user})) \mid b_{lock}{}^{user})]$, which provides the recovery mechanism for the T_{CP}, immediately signs it with σ_{Vault}, and outputs this pre-signed transaction.
- broadcast(T): *(Executed by: any)* Broadcasts T on B, provided the party executing the procedure is able to satisfy and outstanding requirements to spend from the inputs of the transaction.

Issuing Blockchain:

On the issuing chain I, operations are executed by the iSC. Some operations are restricted to being executed by certain parties; where this is not specified, any party can execute the operation.

- verifyIssue($\mathsf{T}_{lock}, user$) $\rightarrow b_{lock}|\bot$: Verifies that $\mathsf{T}_{lock} \in \mathsf{L}_b$ and is a valid locking transaction, and observes the amount of b_{lock}.
- issue($i(b_{lock}), user$): Mints $i(b_{lock})$ tokens and assigns them to the ownership of the $user$.
- verifyHash($round, user$) $\rightarrow \top|\bot$: Verifies that the user has supplied a preimage corresponding to their Recovery transaction this round, and that the preimage matches the hashlock.
- storeHashPreimage($round, user, preimg$): *(Invoked by: user)* Publicly stores the user's Recovery preimage for $round$.
- transfer($i, user, receiver$): *(Invoked by: user)* Subtracts i from the balance of $user$ and adds it to the balance of $receiver$.
- burn($i, user$): *(Invoked by: user)* Destroys i of tokens from the user's balance.
- verifyRedeem($i(b), \mathsf{T}_{unlock}, user$): Verifies that $\mathsf{T}_{unlock} \in \mathsf{L}_b$ and is a valid transaction, and returns b corresponding to $i(b)$ to the sole control of $user$.
- verifyCP($Vault, user, \mathsf{T}_{CP}, \mathsf{T}_{Rec_{user}}$) $\rightarrow \top|\bot$: Verifies that $\mathsf{T}_{CP} \in \mathsf{L}_b$, that is a valid checkpoint transaction broadcast by $Vault$, that it locks b_{lock} corresponding to the user's $i(b_{lock})$ token holdings into a new checkpoint, and that $\mathsf{T}_{Rec_{user}}$ is a matching, valid, and correctly pre-signed recovery transaction.
- verified($round$) $\rightarrow \top|\bot$: Returns $\top$ if the checkpoint at round $round$ has been successfully verified using verifyCP, and $\bot$ otherwise.
- hasCollateral($vault, user$) $\rightarrow \top|\bot$: Returns $\top$ if $Vault$ has non-zero collateral bonded for $user$.
- bondCollateral(i, $Vault$, $user$): *(Invoked by: Vault)* Locks i as collateral to back funds of $user$.
- unbondCollateral($Vault, user$): Releases the collateral locked by $Vault$ for $user$.
- slashAndReimburse($Vault, user, i_{col}$): Slashes i_{col} locked by $Vault$ against $user$, and transfers ownership of the slashed collateral assets to the user as reimbursement.

Algorithms:

procedure ISSUE
  $user$.createLock(b_{lock}, $Vault$) $\rightarrow \mathsf{T}_{lock}$
  $user$.broadcast(T_{lock})
  if (iSC.verifyIssue($\mathsf{T}_{lock}, user, Vault$) $\rightarrow i(b_{lock})) \neq \bot$ **then**
    iSC.issue($i(b_{lock}), user$)

procedure INSTANT TRANSFER
  if $\neg$iSC.verifyHash($r_i, user$) **then**
    if $user$.verifyCollateral(b_{lock}) **then**
      $user$ calls iSC.submitHashPreimage($\tau(r_i), user, r_i^{user}$)
  if iSC.verifyHash($r_i, user$) **then**
    $sender$ calls iSC.transfer($i, user, receiver$)

procedure CHECKPOINT
  before $\tau(r_i + \Delta_{Rec})$:
    $Vault$ calls iSC.bondCollateral(i_{col}, $Vault$, $user$)

  after $\tau(r_i + \Delta_{Rec})$:
    if $\neg user$.verifyCollateral(b_{lock}) **then**
      $user$ executes RECOVER

  after $\tau(r_i + \Delta_V)$, **before** $\tau(r_i + \Delta_{CP})$:
    $Vault$.createCP($user, b_{lock}$) $\rightarrow \mathsf{T}_{CP}$
    $Vault$.createRecovery($user, \mathsf{T}_{CP}$) $\rightarrow \mathsf{T}_{Rec_{user}}$
    $Vault$.broadcast(T_{CP})
    if iSC.verifyCP($Vault, user, \mathsf{T}_{CP}, \mathsf{T}_{Rec_{user}}$) $= \top$ **then**
      iSC.unbondCollateral($Vault, user$)
      return
    else
      iSC.slashAndReimburse($Vault, user, i_{col}$)
      return

  after $\tau(r_i + \Delta_{CP})$:
    if $\neg$iSC.verified(r_i) **then**
      iSC.slashAndReimburse($Vault, user, i_{col}$)

procedure REDEEM
  $user$ calls iSC.burn($b_{lock}, user$)
  $Vault$.unlock($b_{lock}, user$) $\rightarrow \mathsf{T}_{unlock}$
  $Vault$.broadcast(T_{unlock})
  if iSC.verifyRedeem($i(b_{lock})$), $\mathsf{T}_{unlock}, user$) $= \top$ **then**
    if iSC.hasCollateral($Vault, user$) $= \top$ **then**
      iSC.unbondCollateral($Vault, user$)

procedure RECOVER
  after $\tau(r_i + \Delta_{Rec})$:
    $user$.broadcast($\mathsf{T}_{Rec_{user}}$)

Fig. 5: Formal specification of XCC protocols.

2) *Collateralise.* The Vault bonds the required collateral.
3) *Finalise.* The user reveals the hashlock to their recovery HTLC, recording it in the iSC, if they have not already done so since the last checkpoint. If they have, the transfer is automatically considered finalised.
4) *Record.* Once the iSC has verified that collateral has been bonded and the recovery has been revoked, the token balances are updated.

If the user is next scheduled to participate in a checkpoint far into the future, this may result in the Vault locking collateral for an extended period of time. To remedy this, the user and Vault may cooperate to make use of the last checkpoint's multisignature output to release the funds into custody of the Vault. This will not affect security since the Vault must have already put down collateral and allows the Vault to spend the funds and include them in an earlier checkpoint.

Security Arguments for Strong Redeemability. We first consider the funds b_{lock} held by the sender. According to the **Instant Transfer** protocol execution, the sender's funds are always either (a) locked on the backing chain B under multisig and timelock outputs, with **Recovery** available; or (b) either locked on B with **Recovery** unavailable, or in the Vault's custody, but secured by bonded collateral i_{col} on the issuing chain I.

Case (a) satisfies **Strong Redeemability** as demonstrated in Section V-B3. In case (b), to steal the sender's b_{lock}, the Vault must tamper with the execution of the contract iSC to withdraw collateral i_{col}. This contradicts the model. Thus, the **XCC Instant Transfer** satisfies **Strong Redeemability** for the sender.

We now consider the receiver. According to the protocol, the b_{lock} from referenced by the transferred $i(b)$ are fully collateralised by the Vault's i_{col}. As such, if the Vault and/or the sender collude to commit theft, the received will be reimbursed in i_{col}. To withdraw i_{col} without reflecting the correct balance updates (sender to receiver) on B, the Vault must tamper with the verification logic of iSC, which contradicts the model. We conclude, **Strong Redeemability** is also satisfied for the receiver. □

Arguments for Atomic Swaps. The **XCC Transfer** protocol does not allow performing atomic swaps of XCC tokens for other assets on I, due to the delayed finality of the transfer. However, with the introduction of the instant transfer mechanism, swaps become possible, using the following protocol:

XCC Protocol: Atomic Swap. A sender and a receiver atomically exchange $i(b)$ for a pre-agreed amount of any asset on I.

1) *Prepare.* The sender executed steps 1-3 of the **Instant Transfer** protocol if they have not done so already since the last checkpoint. This ensures their funds are collateralised and their recovery has been revoked.
2) *Lock.* The sender locks $i(b)$ in the iSC.
3) *Swap.* If the receiver locks the pre-agreed assets in the iSC within the delay Δ_{swap}, the iSC executes both transfers atomically, performing the swap.
4) *Revoke.* If the receiver fails to perform the swap within Δ_{swap}, the iSC released i back to the sender.

□

VI. Security and Incentive Considerations

In this section, we provide an informal analysis of potential security flaws and attack vectors, and their mitigation, supplementing the design choices from Sections V-B, V-C and V-D.

A. Chain Relay Security

The contract iSC operating on the issuing chain I must be aware of the state of the backing chain B; in particular, it must be able to verify the validity and inclusion of relevant transactions on B. A *chain relay* is used for this purpose, as described for XCLAIM.

As the relay's design is reused directly, the security considerations for the chain relay are identical. We present an overview of these; a more detailed discussion can be found in XCLAIM citezamyatin2018xclaim.

1) **Chain Relay Poisoning.** An adversary may attempt to submit fake data to the relay. To remedy this, a maturity period is introduced before which submitted block headers are not considered final, similar to the confirmation period used on most blockchains themselves.
2) **Replay Attacks on Inclusion Proofs.** An adversary may attempt to replay transaction inclusion proofs to e.g. issue duplicate tokens. To mitigate this, a unique identifier can be required to be included in relevant transactions. Note that, unlike in XCLAIM, the timelock-based nature of XCC limits the scope in which replay attacks are possible: for instance, a *Redeem* transaction can no longer be replayed after the next checkpoint, nor can it be used as proof for any user except the original one.
3) **Chain Splits.** It is possible, when consensus rules are modified in a conflicting way (hard fork) without full adoption by all consensus and network participants, that two instances of the same blockchain will propagate at the same time [38]. In the case of B chain splits (B and B'), the iSC on I will be default follow the non-upgraded chain B and must be modified to support B'. In this scenario, correct implementation of replay protection on B and B' is critical for the security of XCC (and all other B applications for that matter) to distinguish between $i(b)$ and $i(b)$'.
   In case of a I chain split, two distinct instances of $i(b)$ are created, backed by the same locked funds b_{lock} but by different collaterals: i_{col} and i_{col}'. To ensure physical redemption is possible, can be required to chose between holding CBAS on I and I'. Alternatively, one can allow a self-managed transition, where b_{lock} are redeemed on a first-come-first-served basis and remaining CBAS become synthetic assets backed only by collateral on I and I' respectively. In the latter case, to prevent incorrect slashing of vaults, checkpoint verification rules must be loosened to account for the imbalance between b_{lock} and the sum of

i_{col} and i_{col}'. Economic security of users is ensured by design in both cases.

B. Ledger Synchronisation

Various parameters of XCC can be adjusted, either when instantiating the smart contract iSC, or even individually for every user. In particular, the durations of the timelocks Δ_{Rec}, Δ_V and Δ_{CP}, expressed in rounds of L_b, are left up to the implementation.

The choice of these parameters, however, must take into account the persistence and liveness parameters k and u for both L_b and L_i. In particular, $\Delta_V - \Delta_{Rec}$ — that is, the time interval during which users are able to execute **Recovery** if a Vault is misbehaving or failing to collateralise, before the funds are released into Vault custody — must be greater than $u_b + k_b$, i.e. the delay for a user to include a stable transaction in B.

Similarly, $\Delta_{CP} - \Delta_V > u_b + k_b + u_i$ must be satisfied, i.e. the time a Vault has to broadcast a checkpoint before timeout must allow for both the delay to include and stabilise the checkpoint transaction on L_b and additionally the delay for a transaction to be included in L_i, as the iSC, located on I, must verify the checkpoint (as part of a transaction on I) before Δ_{CP}.

C. Counterfeiting

A common concern in both XCLAIM and XCC is of a Vault, and in the case of XCC potentially a Vault colluding with a user, reusing already locked funds b_{lock} to re-issue additional $i(b_{lock})$.

In XCLAIM, this required restricting any spending of b_{lock}, except as required for the execution of the protocols. In XCC, locked funds cannot normally be moved while the various timelocks are active, as part of predefined protocols such as **Redeem** or **Recover**. During the brief period between Δ_V and Δ_{CP}, while the Vault has sole custody of the funds, the opportunity to counterfeit is very limited. Due to this, fraud proofs become a practically feasible way to prevent counterfeiting, and the Vault can remain otherwise unrestricted in its handling of the funds in preparation for a checkpoint.

Interestingly, counterfeiting only becomes a potential problem on a macro-economic scale, specifically in case of bank runs for b. Since every issued $i(b)$ is secured by i_{col} even if b_{lock} have been removed, $i(b)$ owners remain financially unaffected. In this scenario, $i(b)$ represents *synthetic asset* backed by collateral and pegged to the price to b through price oracles.

D. DoS

While XCC operates in a decentralised model where any user can function as a Vault, unlike XCLAIM, tokens have limited fungibility due to being tied to a single Vault through the multisignature timelock transactions. Therefore, if a particular Vault is the target of a denial-of-service (DoS) attack, all the users of that Vault may be forced to exit the system and liquidate their tokens. Furthermore, due to **Weak Liveness**, executing **Transfer** becomes impossible if the sending user's Vault is not available. This does not cause direct financial damage due to **Strong Redeemability**, but may be used to execute a targeted DoS attack against a particular user or users. Such an attack may also be caused by a Vault itself deliberately going offline.

As a partial remedy, an affected user may collateralise their own funds in order to transfer them to a different Vault by means of the **Recovery** protocol. This approach allows users of an unresponsive Vault to avoid liquidating their tokens, at the cost of requiring users to provide sufficient collateral to cover their own funds, and incurring a delay due to the Recovery HTLC timelock.

E. Fee Model Considerations

Both XCLAIM and XCC are protocols that rely on rational behaviour of Vaults - and as such must provide sufficient fee income to cover costs and generate a net return. The XCC (and XCLAIM) costs can be split as follows:

- *Operation costs*: The costs of operating clients (ideally, full nodes) on both B and I, as well as additional client software to automate Vault operation, in terms of both server storage and bandwidth.
- *Capital costs*: The costs of the locked i_{col} collateral on I.

While both XCLAIM and XCC require over-collateralization to mitigate exchange rate fluctuations, XCC significantly improves over XCLAIM by exhibiting predictable collateral lockup times. Specifically, while in XCLAIM Vaults lock collateral indefinitely (at user's discretion), Vaults in XCC only require collateral during the checkpoint renewal period T_Δ, or when user's request collateralization for instant transfers. In both cases, the collateral lockup duration is *predictable*, allowing the Vault to estimate the capital and opportunity costs and hence to offer *collateral as a service*.

A challenge arising in XCC is the need for vaults to incentivize users to pick suitable checkpoint frequencies. A straightforward approach is to charge checkpoint costs (transaction fees and bandwidth) to users, yet especially during the initial stages of protocol adoption lower fees may be preferred.

VII. Extensions

In this section, several extensions to the core XCC protocol are discussed, which can provide improved decentralisation or reduced collateralization.

A. Interactive, Zero-Collateral Checkpoints

Section V-B describes how timelocks can be renewed through collateralised checkpoints. We presently describe an extension of the protocol which can be used to renew checkpoints without requiring collateralization, with the tradeoff of requiring all participating users to be online.

After Δ_{Rec} since the last checkpoint but before Δ_V, a user's multisignature output from a checkpoint can be used to sign a subsequent checkpoint transaction, before the timelock expires. This is performed as follows:

XCC Protocol: Zero-Collateral Checkpoint. The Vault V refreshes the timelocks of one or more of its users, in an interactive but zero-collateral manner.

- *Construct.* The Vault creates T_{CP} in an identical manner to the **Checkpoint** protocol.
- *Sign.* Every user participating in the checkpoint adds their signature to the relevant input and output of the transaction. Any users which fail to respond at this point are dropped from the transaction.
- *Broadcast and Verify.* V broadcasts T_{CP} on the chain B and notifies the iSC which verifies it, identically to steps 2-4 of the **Checkpoint** protocol.

Since the transaction's inputs are signed by every user, it can be broadcast as soon as Δ_{Rec} has expired for all participating users, and before Δ_V. As the Vault therefore never gains custody of the backing funds, no collateralization is necessary at any point during the execution of this protocol.

Partially-signed, Malleable Transactions: If Bitcoin is used as the backing blockchain, it is possible to use `SIGHASH_SINGLE | SIGHASH_ANYONECANPAY` as the signing parameters to allow each user to only sign their specific input and output. This ensures that unresponsive users do not impede the broadcasting of the checkpoint. If a backing chain without such functionality is used and each user must sign the entire transaction, then the *Sign* step must use a multi-step commit phase and may need to be retried multiple times should any committed user fail to uphold their commitment, potentially rendering this protocol impractical.

B. Cross-Vault Transfers

Transferring tokens between users assumes that both participants in the transfer are trading within a single Vault. However, to ensure the system is scalable and resilient, it should allow any number of vaults to interoperate.

Unlike in XCLAIM, this cannot be achieved transparently to users with full fungibility of tokens, as vaults must actively track the backing funds corresponding to a user's tokens. Instead, at any time, a given token balance is held by a given Vault. As such, if a user A holding a balance with Vault V_1 wishes to send tokens to user B whose primary balance is secured by Vault V_2, the Transfer executed by A will create a new account for B with V_1, effectively splitting B's funds across two vaults.

This introduces two new considerations: firstly, B might not wish to do business with A's Vault, or A's Vault might intend to censor B. Secondly, even if no such problems arise, B would then have to pay fees to both vaults for maintaining his backing funds and monitor multiple locations to ensure funds are collateralised during checkpoint security periods.

This fragmentation can be fixed by transferring funds between vaults. This can be modelled as tokens simultaneously being redeemed at one Vault, and issued at another. This can be executed as a single transaction spending from the multisignature checkpoint output at V_1, and with its output being a well-formed Issue transaction for V_2. Incidentally, this does not require collateralization from either Vault.

C. Personal vaults

While we discuss vaults as dedicated parties providing a service to multiple users, it is worth mentioning that it is possible for a single user to operate their own Vault. This has identical security implications as a user colluding with a Vault, which is permitted by the threat model.

This offers the advantage of near-total decentralisation, as well as reduced costs to the user as the "Vault" in such a system would not be operating for profit. The disadvantage, however, is that transferring tokens will generally require a transaction in the backing chain, as described above; this precludes this model from being used in use cases involving frequent trading of CBAs. However, this could still be suitable for cases where CBAs are not intended to be transferred, e.g. when a user only wishes to use them to invest into a DeFi contract - or for constructing physically settled call and put cross-chain option contracts for the backing asset.

VIII. Implementation and Evaluation

This chapter discusses the proof-of-concept implementation of XCC that was completed as part of this project, using Bitcoin as the backing blockchain and Ethereum as the issuing chain. A technical summary of the implementation is presented, followed by an analysis of the running costs.

Note that at the time of writing, the Ethereum network is highly congested, leading to high costs for most smart contract operations. As the XCLAIM protocol is generic, it can be implemented on any pair of chains that satisfy the requirements laid out as part of the **Weak Compatibility** protocol property, defined in Section III-F; for instance, Polkadot [34] [18] could be used as the issuing chain, which may result in significantly lower costs than those obtained from the proof of concept Ethereum implementation.

A. Implementation

The smart contract functionality, as detailed in Figure 5, was implemented in Solidity for the Ethereum blockchain, in about 620 lines of code total. The Interlay Solidity implementation [11] of BTCRelay [10] was used to verify transaction inclusion in the Bitcoin blockchain, and parse Bitcoin transaction data. The `bitcoin-spv` library [14] was used for utilities helping manipulate Bitcoin transactions.

The resulting tokens are not strictly compatible with the ERC-20 standard, due to the possibility that they make become unusable at any time (if a user executes Recovery). This is a drawback in comparison with XCLAIM, the tokens in which are fully ERC-20 compliant.

B. Costs Evaluation

Protocol execution entails costs in fees on both blockchains. On Bitcoin, where the fee depends on the transaction size in bytes, we use a rate of 6 satoshi per byte, which should provide prompt confirmation in the majority of cases [2]. For Ethereum, where fees are determined in gas which depends on both the data size and computation required for a transaction, a

conservative gas price of 140 Gwei is used, based on historical data [5].

For the Register and UpdateHashlist protocols, we assume 4 new (previously unset) hashes are set by the user in each case.

At the time of writing, the BTC exchange rate was approximately 35400.00 USD, and the ETH exchange rate was approximately 2440.00 USD. [4]. Based on these figures, the following costs are incurred by users:

Protocol	USD cost
Issue	$69.70
Transfer	$18.40
Redeem	$38.80
Recover	$38.80

Finally, we consider the Checkpoint protocol. The costs of broadcasting and verifying scale with the number of users. However, since for every user a checkpoint entails identical transaction sizes (on Bitcoin) and identical operations (on Ethereum), the costs scale close to linearly. Thus, the cost was found to be approximately $9.24 per checkpoint per user.

C. Feasibility

The above results show that the protocol is feasible to implement on BTC-ETH. The overall costs for a user will vary greatly depending on their usage pattern.

The greatest contributor to checkpoint costs are the Bitcoin fees. However, the Ethereum block gas limit provides a cap on maximum practical checkpoint sizes: at the time of writing, the limit for a single block is slightly above 12,000,000 gas [6], meaning a 500 user checkpoint would consume over a third of a block. Thus, the expected method for vaults to operate would be to limit their checkpoints to a few hundred users and instead broadcast them more frequently. At an average of 500 users per checkpoint, a single Vault broadcasting hourly checkpoint transactions for users scheduled for daily checkpoints could serve around 12000 users. In practice, this number might differ due to users selecting different checkpoints, or being involved in an earlier checkpoint due to collateralization/trading.

If a party operating a Vault finds it must handle more users every round than is practical to handle in a single transaction, the checkpoints can be effectively "split" simply by registering a new Vault with the smart contract. Recall: any user can create an unlimited number of Vaults - the bottleneck is merely the transaction throughput of Bitcoin and Ethereum. For example, 100 such Vaults can cater to 1.2 million users, broadcasting merely 100 Bitcoin transactions per hour in total.

D. Costs in Practice

The primary other scheme for cryptocurrency backed assets is XCLAIM [37], which XCC extends. Thus, we compare the costs associated with the two frameworks, as well as the collateral required for operation.

The cost of running the basic protocols — Issue, Transfer, Redeem — is nearly identical to XCLAIM, comprising the cost of broadcasting a Bitcoin transaction in the cases of Issue and Redeem, followed by an Ethereum contract invocation. Using the same fee price assumptions presented in the XCLAIM analysis [37], namely, a gas cost of 9 Gwei and Bitcoin fees of 40 satoshi/byte, an ETH cost of $105.71 and a BTC cost of $3717.38:

- **Issue** costs $0.40, or 15% cheaper
- **Redeem** costs $0.37, or 24% cheaper

The basic Transfer protocol in XCC has less functionality than the atomic Swap implemented in XCLAIM, and thus they cannot be directly compared. The differences observed are small enough that they may be down to implementation differences, as the core protocols are very similar.

The primary difference comes from the cost of the Checkpoint transaction. The costs of XCC vary greatly depending on the usage pattern, hence we consider several example scenarios and determine the costs and benefits of XCC in each case.

Frequent trader. Consider a user executing a large number of Transfers per day. Under such a scenario, the user would require near-constant collateralization of funds. In this case, the primary benefit brought on by checkpoints is that collateral is only ever locked for predetermined periods of time, reducing uncertainty and lowering the need for overcollateralization: at regular intervals, the user may choose to be included in a checkpoint, incurring a cost of approximately $1.30 (plus any Vault profit margins) under the assumptions above, and "locking in" an updated exchange rate for the collateral.

Additionally, XCC can fall back to emulating XCLAIM, simply by not scheduling a checkpoint and fully collateralising a user's funds, if desired.

The first two rows in Table I illustrate the costs and benefits of XCC for this use case.

Occasional trader. Consider next a user executing a trade twice a week on average, using a mixture of instant (collateralised) and delayed transfers, and who have in turn scheduled to be included in checkpoints at weekly intervals — resulting, on average, in each user being collateralised half the time between checkpoints. Assuming there are multiple such users registered with the Vault, the Vault would then be able to take advantage of fractional collateralization, reducing collateral required by up to 50% in an ideal scenario. Additionally, the benefits of reducing overcollateralization mentioned above would still apply, resulting in the Vault requiring to put down only 30% of the collateral which would be required by XCLAIM in a similar scenario. This is summarised in the third row of Table I.

Static user. Finally, consider a user who has either issued or received XCC tokens, but does not regularly use them, only rarely executing Transfer. Such a user could safely be placed on a schedule of extremely far-removed checkpoints — such as monthly, or even less frequently. This reduces the collateral required to almost nothing; the user's funds need only be collateralised for the duration of the security period every month, which may last a few hours. And in turn, overcollateralization is nearly eliminated, as exchange rate fluctuations are minimal in the span of a few hours, except

TABLE I: Summary of example XCC operation costs, relative to XCLAIM

Use case	Cost	XCC extra cost	Collateral
XCLAIM equivalent	\$20.58/week	\$0	200%
Frequent trading	\$30.38/week	\$9.80/week	110%
Moderate transfers	\$4.34/week	\$1.40/week	60%
Non-trader	\$2.87/transfer	\$1.40/transfer	$\leq 1\%$

during periods of extreme volatility. Meanwhile, the extra cost is compared to XCLAIM is negligible.

The drawback of such an arrangement is that should such a user wish to execute Transfer, they will almost certainly incur the cost of an extra checkpoint transaction as part of that.

E. Summary

In summary, XCC provides significantly improved flexibility to aid in reducing the collateral required, with the associated cost of including a user in a checkpoint transaction. Outside of checkpoints, the costs of XCC are extremely similar to those of XCLAIM; and if desired, XCC can fall back to an XCLAIM-like mode of operation. However, even where that would be advantageous, XCC provides the option to ensure regular re-collateralization of the funds, reducing the overcollateralization factor required. In other scenarios, especially when users do not frequently trade their tokens, using fractional collateralization can allow the Vault to significantly reduce collateral usage.

Ultimately, whether taking advantage of checkpoints is profitable depends on how much the Vault values freeing up collateral, which will be reflected in user fees. The expectation is that low volume users may see little benefit from participating in checkpoints, and may prefer to fall back to XCLAIM-like usage, including full overcollateralization for extended periods of time. However, for users storing significant amounts of value, the reduction in required collateral is very likely to be worth the cost of broadcasting the checkpoint transactions, not least because it will allow vaults to serve a significantly higher amount of users using the same amount of available liquid collateral.

IX. Conclusion and Future Work

In this paper, we presented the XCLAIM Commit framework, utilising a checkpointing scheme inspired by commit chains to extend XCLAIM and achieve significant collateral reduction. A contract for non-Turing-complete backing blockchains such as Bitcoin was described, enabling improved theft prevention in the new round-based setting. The regular checkpoints were used to define several ways in which collateral could be reduced depending on usage patterns down to a fraction of that required by XCLAIM. We conducted a security analysis of the protocol by modelling the system as a state machine. The economic security of the system was demonstrated, ensuring that no honest party can lose funds during its operation. We implemented and evaluated a proof of concept of XCC. Its operation is not more expensive than the equivalent XCLAIM operations where those exist. When making full use XCC, the potential for significant reduction in collateral required was demonstrated.

Possibly the most promising future work on XCC would be a reduction in the size of checkpoint transactions and hence their cost, by avoiding the need to include an input and an output for every user. The primary difficulty, in this case, arises from the need to provide a recovery mechanism. One promising avenue for implementing this is a mechanism to allow users to trigger the creation of a full-sized checkpoint with individual outputs when they detect misbehaviour from the Vault; on Bitcoin, this would require the `OP_CHECKTEMPLATEVERIFY` opcode introduced by BIP-119 [1], currently in the draft stage.

Additionally, XCC provides for flexible usage, ranging from emulating round-less XCLAIM to allowing users to lock funds practically indefinitely with no collateral required. In turn, vaults have some flexibility in how they manage the funds they are holding; in other matters, in particular checkpoint frequency, they can influence users by setting their fee schedule to encourage certain behaviours. Thus, there is a lot of potential for Vault implementations to optimise their behaviour. A detailed economic analysis of the incentives involved would be of great use to inform a profitable Vault design.

References

[1] bips/bip-0119.mediawiki at master · bitcoin/bips · github. https://github.com/bitcoin/bips/blob/master/bip-0119.mediawiki. Accessed: 2020-07-31.

[2] Bitcoin fee estimation. https://bitcoinfees.github.io/#1d. Accessed: 2020-09-02.

[3] Bitcoin wiki: Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification. Accessed: 2020-06-02.

[4] Cryptocurrency market capitalizations | coinmarketcap. https://coingecko.com/. Accessed: 2021-04-21.

[5] Ethereum average gas price chart | etherscan. https://etherscan.io/chart/gasprice. Accessed: 2021-04-21.

[6] Ethereum network status. https://ethstats.net/. Accessed: 2020-09-02.

[7] Global charts | coinmarketcap. https://coinmarketcap.com/charts/. Accessed: 2020-06-02.

[8] Hash time locked contracts. https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts. Accessed: 2020-06-02.

[9] The history of bitcoin & crypto exchange hacks - cryptosec. https://cryptosec.info/exchange-hacks/. Accessed: 2020-09-03.

[10] http://btcrelay.org/. http://btcrelay.org/. Accessed: 2020-06-02.

[11] interlay/btc-relay-solidity: Bitcoin light client on ethereum. https://github.com/interlay/BTC-Relay-Solidity. Accessed: 2020-08-31.

[12] Light client protocol · ethereum/wiki wiki · github. https://github.com/ethereum/wiki/wiki/Light-client-protocol. Accessed: 2020-06-02.

[13] Scalability - bitcoin wiki. https://en.bitcoin.it/wiki/Scalability#Simplified_payment_verification. Accessed: 2020-06-02.

[14] summa-tx/bitcoin-spv. https://github.com/summa-tx/bitcoin-spv/tree/6102ece76f846a53b68f1c2550f65caf53cbaa02. Accessed: 2020-08-31.

[15] G. Avarikioti, O. S. T. Litos, and R. Wattenhofer. Cerberus channels: Incentivizing watchtowers for bitcoin. Cryptology ePrint Archive, Report 2019/1092, 2019. https://eprint.iacr.org/2019/1092.

[16] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. https://eprint.iacr.org/2013/879.

[17] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *IEEE Symposium on Security and Privacy*, 2015.

[18] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood. Overview of polkadot and its design considerations. *CoRR*, abs/2005.13456, 2020.

[19] V. Buterin. https://ethereum.org/whitepaper/, 2013. Accessed: 2020-06-02.

[20] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[21] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. pages 281–310, 04 2015.

[22] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015*, pages 281–310. Springer, 2015.

[23] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdo rf, and S. Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC*, pages 3–16. ACM, 2016.

[24] M. Herlihy. Atomic cross-chain swaps. pages 245–254, 07 2018.

[25] A. Judmayer, A. Zamyatin, N. Stifter, A. Voyiatzis, and E. Weippl. Merged mining: Curse or cure? pages 316–333, 09 2017.

[26] R. Khalil, A. Zamyatin, G. Felley, P. Moreno-Sanchez, and A. Gervais. Commit-chains: Secure, scalable off-chain payments. Cryptology ePrint Archive, Report 2018/642, 2018.

[27] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 357–388, Cham, 2017. Springer International Publishing.

[28] 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)*, pages 583–598. IEEE, 2018.

[29] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2008.

[30] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin, 2015. Accessed: 2016-08-22.

[31] J. Teutsch, M. Straka, and D. Boneh. Retrofitting a two-way peg between blockchains, 08 2019.

[32] F. Tschorsch and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. In *IEEE Communications Surveys Tutorials*, volume PP, pages 1–1, 2016.

[33] M. Westerkamp and J. Eberhardt. zkrelay: Facilitating sidechains using zksnark-based chain-relays. Cryptology ePrint Archive, Report 2020/433, 2020. https://eprint.iacr.org/2020/433.

[34] G. Wood. Polkadot: Vision for a heterogenous multi-chain framework. https://polkadot.network/PolkaDotPaper.pdf, 2016.

[35] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 931–948, New York, NY, USA, 2018. Association for Computing Machinery.

[36] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt. Sok: Communication across distributed ledgers. In N. Borisov and C. Diaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 3–36. Springer, 2021.

[37] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. J. Knottenbelt. XCLAIM: trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 193–210. IEEE, 2019.

[38] 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
SYMBOLS TABLE

Table II provides an overview of the symbols and notation used in this paper.

TABLE II: Summary of used symbols and notation.

Symbol	Description
ε	Exchange rate between the issuing and backing chains
$\mathcal{E}(t)$	Maximum assumed change in exchange rate during time t
T_t	Transaction t; refer to Section III-B for full transaction notation
σ_{A}	Signature of A
$H(s)$	Cryptographic hash digest, with preimage s
L	A distributed ledger; refer to Section III-A for more details on the ledger notation
L_b	The distributed ledger corresponding to B
L_i	The distributed ledger corresponding to I
$\tau(r)$	Function mapping round r on any ledger to a global clock; refer to Section III-A for more details
CP_i	Checkpoint transaction i
Δ_{Rec}	Time period after a Checkpoint (or Issue) transaction is broadcast, after which the multisignature condition becomes spendable: also *recovery delay*; defined in number of rounds on L_b
Δ_V	Time period after a Checkpoint (or Issue) transaction is broadcast, after which the Vault gains custody of the funds: also *checkpoint duration*; defined in number of rounds on L_b
Δ_{CP}	Time period after a Checkpoint (or Issue) transaction is broadcast, after which that checkpoint is considered expired; defined in number of rounds on L_b
B	The backing chain
b	Cryptocurrency unit on B
${b_{lock}}^{\mathsf{A}}$	Quantity of funds user A owns (and has locked) on B, corresponding to $i(b)^{\mathsf{A}}$
${b_{lock}}^{\mathsf{A}}(\text{CP}_i)$	Quantity of funds A owned on the backing chain at the time round i ended
I	The issuing chain
i	Cryptocurrency unit on I
${i_{col}}^{\mathsf{A}}$, ${i_{col}}^{\mathsf{A}}(\text{CP}_i)$	Quantity of funds on the issuing chain that are collateralising ${b_{lock}}^{\mathsf{A}}$, and similarly at the time round i ended
$i(b)^{\mathsf{A}}$, $i(b)^{\mathsf{A}}(\text{CP}_i)$	Number of CBA tokens user A owns, and similarly at the time round i ended
Rec_i^{A}	Recovery transaction for user A, corresponding to the checkpoint at the end of round i
r_i^{A}	The recovery hashlock preimage for Rec_i^{A} (which will have $H(r_i^{\mathsf{A}})$ as the hashlock)
Red_i^{A}	The redeem transaction for user A, spending from the checkpoint at round i
Esc_i^{A}	The escape transaction for user A, spending from the checkpoint at round i
S_Δ	The time during which the funds are considered secured from theft, according to the **Theft Prevention** property
T_Δ	The time during which the funds can be stolen, according to the **Theft Prevention** property

APPENDIX B
REDUCING CHECKPOINT SIZE

One bottleneck in the scaling of XCC is the size of the checkpoint transactions on the backing blockchain, and hence their cost. Several avenues for reducing this were explored, though none led to a feasible solution.

The most straightforward way to reduce checkpoint sizes is to avoid creating separate outputs for every user in the transaction. Thus in the ideal case, the checkpoint might instead have a single input and a single output, drastically reducing costs; more practically, users would be grouped by their set checkpoint frequency, with different timelocks for each, giving perhaps a dozen inputs/outputs. However, this means that a user's recovery transaction must spend from an output containing coins belonging to many different users. This leads to the primary difficulty with this approach: the other users' funds must be somehow redistributed securely by the user executing their recovery, without incurring large costs or inconvenience on the system.

A. Multi-user Recovery

A naive option is to simply trigger the recovery process for all affected users. If recovering was limited to circumstances where the Vault misbehaved, this would even be desirable; however, currently, any user can execute their recovery at any time (unless they have executed Transfer since the last checkpoint). This leads to a very high griefing potential, as a malicious user could bring the system to a halt by forcing large numbers of users to go through the recovery process regularly.

B. Fallback to Large Checkpoint

A slightly improved alternative is to prepare full checkpoint transactions, with individual outputs for every user, spending from the output(s) of the condensed checkpoint transaction. This would effectively allow small checkpoints to be used whenever every party is honest, and when a user wishes to dispute the Vault's behaviour (or to inconvenience the system), they would cause it to fall back to the normal XCC large-checkpoint protocol. In the pessimistic case of malicious griefers attempting to hinder the system, the outcome would only be very slightly worse than just using XCC with large checkpoints from the start: the only extra costs incurred would be the additional small checkpoint transaction and the slightly higher processing cost in the smart contract to verify the extra complexity.

However, two issues arise with this. The first one is that of transaction fees — it is unclear who should be responsible for paying the cost of broadcasting the full checkpoint with a large number of outputs, and in turn of broadcasting the next contracted checkpoint which will require a correspondingly large number of inputs. If the Vault is required to pay these fees, then the cost of griefing the system is negligible compared to the cost to the Vault. This means that the Vault must then charge sufficient usage fees to cover the price of regular large checkpoints, which defeats the purpose of attempting to reduce costs. On the other hand, if the user must pay the cost, then legitimate users attempting to recover funds are heavily penalised.

A second issue with this scheme is that, on the Bitcoin blockchain (which is the primary target for a backing blockchain), there is currently no mechanism for a transaction

to restrict its output to be spent in a particular second transaction, with no malleable output in that second transaction. As such, typically non-malleable contracts are achieved by having a multisignature output, such that all parties must ratify a transaction spending from it. However, in checkpoints that may involve dozens of users, if not hundreds, this is infeasible. Moreover, this would bloat the script sufficiently to largely negate the benefits of having only a single output. On a different blockchain, this issue may not be relevant. BIP-119 [1] proposes a Bitcoin opcode that would solve this issue.

APPENDIX C
POC IMPLEMENTATION DETAILS

The following Script was used as the output of checkpoints (and Issue transactions) on Bitcoin:

```
<Vault public key> OP_CHECKSIGVERIFY
<user public key> OP_CHECKSIG
OP_IFDUP
OP_NOTIF
    <locktime> OP_CHECKSEQUENCEVERIFY
OP_ENDIF
```

All transactions were implemented as Pay to Witness Script Hash (P2WSH), allowing the scripts to be included as witness data and reducing costs.

Additionally, we provide a table showing the breakdown of the fees for the individual protocols:

Protocol	BTC fees (sat)	ETH fees (Gwei)
Register	-	11418060
UpdateHashlist	-	7261800
Recover	1230	6185640
Issue	1088	11853840
Transfer	-	3313080
Redeem	1230	6185640
Checkpoint	1580 (approx.)	623200 (approx.)

APPENDIX D
STATE DIAGRAM

Figure 6 illustrates the possible state transitions of the tokens during XCC operation.

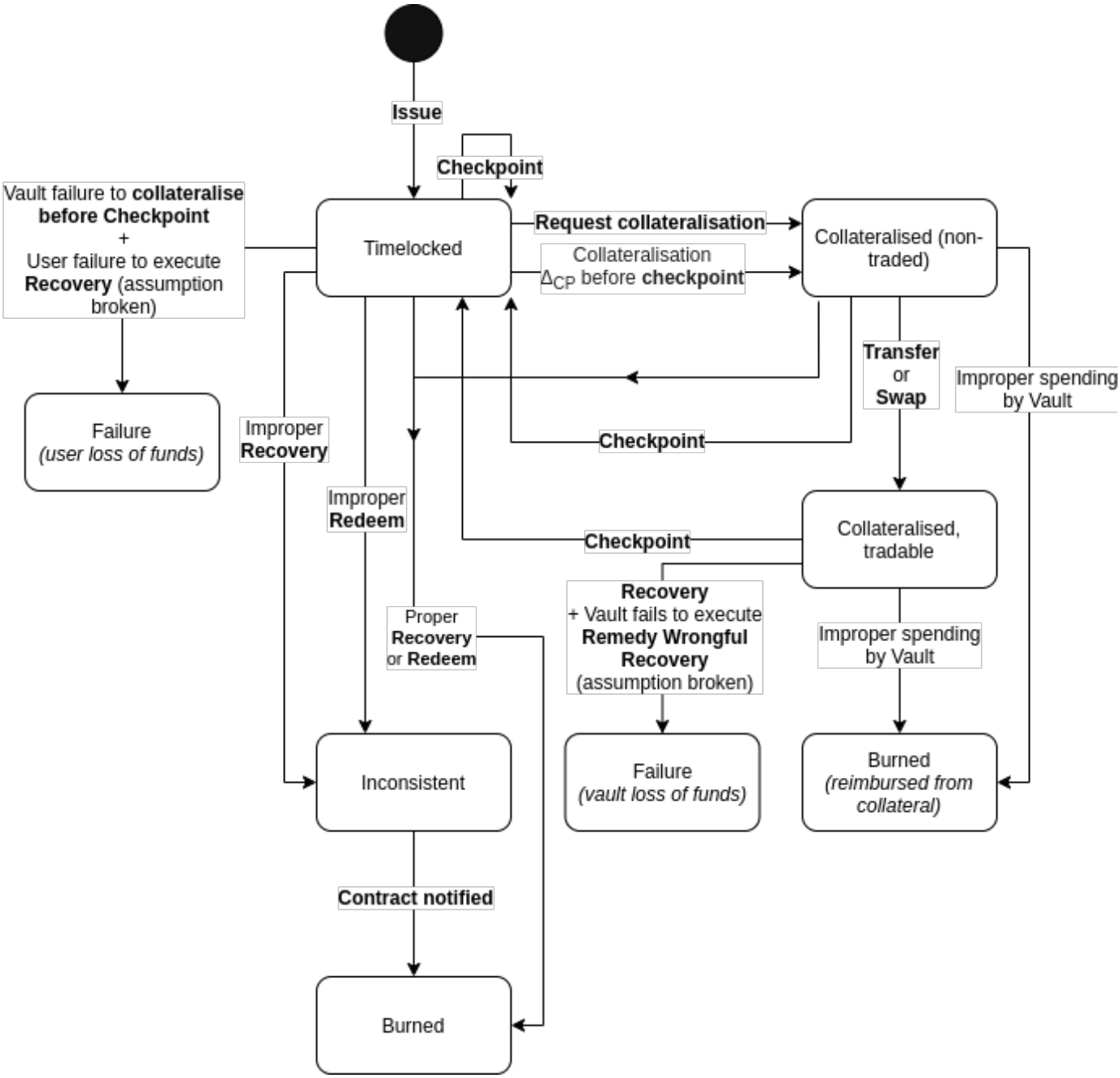


Fig. 6: The state machine for the lifecycle of tokens. Not shown are: the ability to properly Redeem from the three *Timelocked* and *Collateralised* states, which cleanly burns tokens with no inconsistent state; and the ability to properly Recover from both the *Timelocked* and *Collateralised (non-traded)* states, which also cleanly burns tokens.