TOOP: A transfer of ownership protocol over Bitcoin

Ariel Futoransky[1], Fadi Barbara[1,2], Ramses Fernandez[1], Sergio Demian Lerner[1], and Gabriel Larotonda[3,4]

[1] Fairgate Labs
{futo, fadi.barbara, ramses.fernandez, sergio}@fairgate.io, glaroton@dm.uba.ar
[2] Università di Roma La Sapienza, Italy
[3] Universidad de Buenos Aires, Argentina
[4] CONICET, Argentina

Abstract. We present the Transfer of Ownership Protocol (TOOP). TOOP solves a limitation of all existing BitVM-like protocols (and UTxO blockchains at large) that restricts the unlocking transfers to addresses known and preregistered during lock and setup. Accordingly, our protocol avoids the financially costly, regulatory problematic, and congestion-prone front-and-reimburse paradigm.
Furthermore, we note that one of the main applications of TOOP is as an enabler of secure transfer of assets between UTxO blockchains, and back. We showcase this via sketching a committee-based validation protocol that requires only 1-out-of-n honest security. This protocol operates in distinct phases: the lock phase, where the initial setup and individual assets are locked on Bitcoin, and the unlocking with the ownership transfer phase, where the asset is transferred to a possibly different legitimate owner.
This cross-chain bridge protocol, where TOOP plays a key role, is being formalized in concurrent work, and has been implemented for the first time in Cardinal, a protocol for wrapping Bitcoin Unspent Transaction Outputs (UTxOs) onto the Cardano blockchain, with Bitcoin Ordinals represented as Cardano Non-Fungible Tokens (NFTs).

Keywords: Bitcoin · BitVMX · Interoperability.

1 Introduction

Blockchain interoperability is a concept that has become increasingly important for the widespread adoption and practical utility of blockchain technology. Interoperability means that several blockchains can interact securely and consistently while avoiding new strong trust assumptions.

1.1 Fundamental approaches to cross-chain technology

Several methods to achieve cross-chain interoperability can be found in the literature. The interested reader can find in [30] a rigorous treatment of the technology

and in SoK by Augusto et al. [2] a treatment of the security and privacy implication of those methods. Here we briefly present these results for completeness.

We model a cross-chain exchange between blockchains (or ledgers) $\mathcal{BC}_1$ and $\mathcal{BC}_2$. Assume entities P and Q operate on blockchains $\mathcal{BC}_1$ and $\mathcal{BC}_2$ respectively. P and Q may be single parties, federations, or even the same party. Our model presents a cross-chain transfer as a four-steps protocol:

1. *Setup.* The parties exchange information about the involved blockchains $\mathcal{BC}_1$ and $\mathcal{BC}_2$. This includes verification and agreement schemes, asset details and time constraints.
2. *Locking transaction on* $\mathcal{BC}_1$. The party P broadcasts a *locking* transaction to blockchain $\mathcal{BC}_1$[5]. The transaction is finalized through its consensus protocol. This transaction acts as a *commitment* on chain $\mathcal{BC}_1$
3. *Verify.* Party Q verifies the commitment on X using the agreed-upon verification scheme. Based on the result, the protocol either proceeds to commit on $\mathcal{BC}_2$ or aborts.

4a. *Unlocking transaction on* $\mathcal{BC}_2$. Upon successful verification, the party Q sends an *unlocking* transaction to blockchain $\mathcal{BC}_2$.

4b. *Abort.* If verification fails or Q cannot complete the process on Y, the protocol aborts by reverting the effects on $\mathcal{BC}_1$, typically through a compensating transaction.

In practice, the entity Q may serve either as a direct beneficiary of the exchange or as a facilitator. In the latter role, Q can be a single entity or a collection of entities. We use the term group in a broad sense, encompassing configurations with heterogeneous governance structures.[6] This distinction gives rise to a variety of implementation strategies, each with its own trade-offs. For clarity, we categorize these strategies based on the number of independent entities involved, distinguishing between cases where Q is a *Single* party and those where it is a *Group*.

Single Q This is the easiest and older approach. In general, P is a user aiming to exchange funds, while Q can either be another user with the opposite goal or a facilitator. We now see both cases.

Atomic Swaps In the first case, we say that the exchange is "decentralized" since no centralizing process is involved. This kind of exchange is generally called Atomic Swap. Atomic here refers to the atomicity of changes in databases: either both P and Q receive their new asset, or P and Q retain the older one. In

[5] Such a transaction is said to *lock* the funds of P on chain $\mathcal{BC}_1$, preventing P from spending them. Similarly, to *unlock* funds from party Q means that Q can now use them as it pleases.

[6] Specifically, we consider a company (or user) operating multiple nodes as a single entity, since control is centralized. In contrast, a committee (or federation), where each node is governed by a distinct company or user, is treated as a group.

particular, a secure atomic swap avoids the case where, e.g. P receives the new asset *and* retains its older one.

The first atomic swap method is based on the Hash-Time Lock contract (HTLC), introduced by Tier Nolan [24]. In an HTLC, after P has found a potential buyer Q, P performs a locking transaction on chain $\mathcal{BC}_1$ by sending a transaction that can be redeemed either by showing the preimage of a hash h and the usual signature (hash-lock) or after some time elapsed and the showing of the signature (time-lock). Notably, Q sends a similar transaction on chain Y using the same hash h.

If P is honest, P shows its preimage on chain Y redeeming Q's transaction before the time has elapsed. If Q is honest, Q uses P's preimage to redeem P's transaction on chain $\mathcal{BC}_1$, completing the exchange.

Conversely, if P is dishonest and does not redeem the transaction on chain $\mathcal{BC}_2$, Q needs to wait for the time to elapse on chain $\mathcal{BC}_2$. Considering the time-value of money[7], this is a serious obstacle.

Company based exchanges When Q represent a company, we say that the method is "centralized". In that case, Q's guarantees derive from a legal (or trust based) approach, instead of cryptographic methods. An example of this approach is the company operating the bitcoin wrapped token WBTC [17] on Ethereum.

The advantage for P is in its simplicity. In fact, when Q is a company, P does not have to find the buyer nor perform complex non-standard transaction. In a centralized method, P only sends a transaction on $\mathcal{BC}_1$ redeemable by Q. Furthermore, since guarantees are trust based, Q performs a normal transaction on chain Y too. Here, commitments are trivial.

The tradeoff is that P has to trust both Q and the legal apparatus behind it. For example, if Q does not broadcast a correct transaction on chain $\mathcal{BC}_2$, then P needs to rely on lawyers or Q's customer service to retrieve its funds, since no time-lock mechanism has been set.

Group Q This is the case of *committees* (sometimes called federations). In those cases, Q generally acts as facilitator.

In a committee, Q is a set of n nodes requiring a subset of size t to act together to sign transactions for locking and unlocking funds. An example of a committee is the Liquid Federation [22].

Similarly to companies based exchange, P sends the transaction to the designated address of the committee Q. Then at least t parties need to sign a transaction releasing funds for P on chain $\mathcal{BC}_2$.

Since *only a subset* of the committee is required to act, the probability of successful malicious action is greatly diminished since it requires a collusion of t parties. Clearly, larger t mean a lower probability of success since different companies have different aims at different times.

[7] The time-value of money is the idea that a sum of money is worth more now than the same sum in the future due to its potential earning capacity.

Surprisingly, this reduction of risk does not come with a reduction of usability on the part of P. In fact, P only sends a transaction to an address and the complexity is hidden inside the federation.

1.2 Current research

Cross-chain asset transfer protocols must satisfy fundamental security and correctness requirements to ensure reliable interoperability between blockchain networks. We model an asset A as a set whose elements represent the atomic units that can be transferred independently, enabling precise analysis of partial asset transfers and fractional ownership scenarios common in decentralized finance applications.

The research by Sober et al. [27] establishes five essential requirements that any secure cross-chain asset transfer protocol must satisfy. These requirements provide a formal framework for evaluating protocol correctness and security properties.

1. Asset ownership verification requires that asset destruction operations respect legitimate ownership boundaries. When a sender attempts to burn an asset subset X on the source blockchain $\mathcal{BC}_1$, the operation should succeed only if $X \subseteq A^{\mathcal{BC}_1}_{\text{Sender}}$, where $A^{\mathcal{BC}_1}_{\text{Sender}}$ represents the assets legitimately controlled by the sender. The burning operation permanently removes assets from circulation on the source blockchain, making them unavailable for future transactions, while enabling their recreation on the destination blockchain.
2. Transfer authenticity ensures that asset recreation occurs only after verified destruction on the source blockchain. When transferring asset subset X from source blockchain $\mathcal{BC}_1$ to destination blockchain $\mathcal{BC}_2$, the assets should be recreated on $\mathcal{BC}_2$ only after cryptographic proof demonstrates that X has been legitimately burned on $\mathcal{BC}_1$. This requirement prevents counterfeit asset creation and maintains the fundamental invariant that cross-chain transfers neither create nor destroy value.
3. Double-spending prevention mandates that burned assets can be recreated exactly once across all destination blockchains. If an asset subset X is burned on one blockchain, it can be recreated on at most one other blockchain, preventing scenarios where the same assets appear simultaneously on multiple chains and violate conservation of value.
4. Guaranteed recreation establishes that burned assets will eventually be recreated on the intended destination blockchain within a bounded timeframe. This liveness property ensures that legitimate transfers do not result in permanent asset loss due to protocol failures or network partitions.
5. Confirmation feedback provides optional verification that asset recreation has completed successfully on the destination blockchain. In this case, the source blockchain should eventually receive confirmation that the asset subset X has been successfully recreated, enabling applications that require

end-to-end transfer verification. This requirement supports decentralized finality, allowing any network participant to submit confirmation proofs rather than relying on trusted intermediaries.

XClaim [29] implements cross-blockchain transfers through designated vault operators, which lock assets on the source blockchain and issue corresponding wrapped tokens on the destination blockchain. In this system, vault operators provide over-collateralization in the destination blockchain's native currency to guarantee the value of issued tokens. When users request cross-chain transfers, they deposit assets with a vault operator on the source blockchain, and the vault operator mints equivalent wrapped tokens on the destination blockchain after the deposit is confirmed.

The protocol employs blockchain relays to verify locking transactions on the source blockchain, enabling automatic verification of asset deposits without requiring trusted oracles. This relay-based verification mechanism satisfies the asset ownership verification requirement by ensuring that only legitimately deposited assets trigger wrapped token issuance on the destination blockchain.

XClaim's reliance on individual vault operators for transfer execution creates significant centralization risks that violate fundamental decentralization requirements. Each transfer depends on a single vault operator who controls both the locked assets and the minting process for wrapped tokens. This centralized control structure fails to prevent double-spending scenarios where malicious vault operators could potentially issue wrapped tokens without proper asset backing or refuse to process legitimate redemption requests. The protocol also lacks guaranteed recreation mechanisms because vault operators can unilaterally decide whether to complete transfer operations, potentially causing permanent asset loss if operators become unavailable or act maliciously.

This centralized vault model contrasts sharply with distributed approaches where multiple independent parties participate in transfer finalization, reducing single points of failure and improving protocol resilience. The concentration of control in individual vault operators fundamentally limits XClaim's ability to provide the trustless guarantees expected from decentralized cross-chain protocols.

Metronome's approach centres on their proprietary MET token, enabling cross-blockchain transfers through a receipt-based validation system. While validators assess receipt validity to prevent unauthorized transfers (addressing the requirement 2), the protocol fails to meet the requirements 4 and 5 due to its centralized validators structure and lack of transfer confirmation mechanisms.

Karantias et al. [14] provide a cryptographic treatment of proof-of-burn mechanisms, establishing rigorous foundations for protocols that enable verifiable cryptocurrency destruction. While proof-of-burn has been utilized in various blockchain applications, previous implementations lacked formal security analysis and standardized definitions, limiting their theoretical understanding and practical deployment.

The research introduces a comprehensive cryptographic framework consisting of two fundamental functions that together constitute a complete burn protocol.

The GenBurnAddr function generates cryptocurrency addresses with the critical property that any funds sent to these addresses become permanently and irrevocably destroyed. The BurnVerify function provides cryptographic verification that a given address represents a legitimate burn address, enabling independent parties to confirm the validity of burn operations without requiring trusted intermediaries.

The formal analysis establishes three security properties that define robust burn protocols. Unspendability ensures that addresses verified as burn addresses cannot be used for spending operations, providing mathematical guarantees that burned funds remain permanently inaccessible. Binding enables the association of arbitrary metadata with specific burn operations, allowing burn addresses to encode information such as destination blockchain identifiers or transfer parameters. Uncensorability mandates that burn addresses remain computationally indistinguishable from regular cryptocurrency addresses, preventing censorship attempts that might block burn transactions based on address analysis.

The protocol design achieves broad compatibility with existing cryptocurrency systems through its construction simplicity and flexibility. The scheme operates effectively across all popular cryptocurrencies without requiring protocol modifications or specialized wallet software, enabling users to perform burn operations through standard transaction mechanisms. Security analysis demonstrates protocol correctness under the Random Oracle model, providing formal guarantees for the cryptographic constructions.

Although satisfying the requirement 1, the protocol does not address the requirements 3 and 5, lacking decentralized finality and transfer confirmation mechanisms. Pillai et al. [25] introduce a burn-to-claim protocol which requires specialized gateway nodes for cross-network mining and burn transaction verification. While meeting requirement 1, the protocol fails to address requirements 3, 4, and 5, lacking decentralized finality, true decentralization, and transfer confirmations.

AucSwap [20] models cross-chain transfers as Vickery auctions, using HTLCs for asset exchange. Although this approach satisfies requirements 1 and 2, it functions more as an asset exchange system than a transfer protocol and does not fully address requirements 3, 4, and 5.

Zendoo [9] uses zero-knowledge proofs for transaction verification (requirement 1) but does not address requirements 3, 4, and 5. Its specific sidechain architecture requirements limit implementation potential.

Van Glabbeek [11] suggests a protocol that adapts multi-hop payment channel concepts, emphasizing finality (requirement 3). While satisfying the requirement 2 (double-spend prevention), implementation details remain underspecified for requirements 4 and 5.

DeXTT [5] synchronizes the existence of assets across blockchains. Its claim-first transaction model violates requirement 2, and the protocol lacks transfer confirmations (requirement 5), although it partially addresses requirements 1 and 4 through its decentralized synchronization process.

Polkadot [28] implements cross-chain Message Passing (XCMP) using Merkle tree-based queuing. Cosmos employs the TCP/IP-inspired Interblockchain Communication (IBC) protocol. While both platforms address requirements 1 and 2, their specialized blockchain requirements impact requirement 4, and they lack specific implementations for requirements 3 and 5.

The Ownership Transfer and Execution (OTEx) system [6] facilitates asset ownership transfer without asset movement, comprising:

1. Inter Blockchain Communication (IBC) protocol for message formatting.
2. IBC Gateways for smart contract execution and ownership state transfers.
3. Auxiliary blockchain logging for transaction traceability.

While addressing requirements 1 and 2 through its comprehensive logging and gateway system, the reliance of the protocol on specialized gateways affects the requirement 4.

1.3 Contributions of this research

Our work advances cross-chain asset transfer protocols through several key innovations that address critical limitations in existing systems. Building upon the requirements established by Sober et al., our research introduces a proposal for a bidirectional ownership transfer protocol with dynamic address support. Most importantly, the proposed Transfer of Ownership Protocol (TOOP) is, to our knowledge, the first protocol to enable locking an UTxO towards an address that is undetermined at the time of locking. The usefulness of this functionality is mainly showcased in our work as a core part of cross-chain transfers in the Cardinal project, bridging Bitcoin Ordinals to the Cardano blockchain.

This feature distinguishes our protocol from existing cross-chain systems such as DeXTT and OTEx, which require recipients to be specified and registered during the initial asset locking phase. Our approach enables ownership updates to arbitrary legitimate addresses after the lock phase has completed, providing operational flexibility that addresses a limitation in BitVM-based cross-chain protocols.

2 Preliminaries

2.1 Notation

We establish the notation which will be used along this paper. Let $\mathcal{BC}_1$ and $\mathcal{BC}_2$ be two blockchain systems. In our setting $\mathcal{BC}_1$ will be Bitcoin. Let $\mathcal{O} = \{O_1, \ldots, O_n\}$ be the set of operators, each modelled as probabilistic polynomial-time (PPT) machines. Let $\mathcal{A}$ denote an adversary, also modelled as a PPT machine. We denote by $\mathcal{ID}$ the power set of operator subsets, and by $\mathcal{U}$ denote the universe of all possible assets. Let κ be the security parameter.

2.2 Security definitions and protocol properties

Definition 1 (Owner). *Let us consider $\mathcal{BC}$ be a blockchain and $\mathcal{U}$ be the universe of all digital assets on $\mathcal{BC}$. An entity Owner is the owner of asset subset $X \subseteq \mathcal{U}$ if and only if Owner possesses the private key material sk_i corresponding to the public keys pk_i that control the spending conditions for all assets in X.*

Definition 2 (Asset validity). *A protocol Π satisfies asset validity if, for any asset subset $X \subseteq \mathcal{U}$ and any $Owner_0$, the probability that a lock transaction $Lock(X, Owner_0)$ is accepted on $\mathcal{BC}_1$ when $X \nsubseteq Assets(Owner_0, \mathcal{BC}_1)$ is negligible in κ:*

$$\Pr\begin{bmatrix} params \leftarrow Setup(1^\kappa); \\ (X, Owner_0) \leftarrow \mathcal{A}(params); \\ X \nsubseteq Assets(Owner_0, \mathcal{BC}_1); \\ Lock(X, Owner_0) \text{ accepted on } \mathcal{BC}_1 \end{bmatrix} \leq negl(\kappa)$$

Here, $Assets(Owner, \mathcal{BC})$ denotes the set of all assets that are legitimately owned and controllable by $Owner$ on $\mathcal{BC}$ at the time of evaluation, and the function $Lock(X, Owner)$ represents a transaction where $Owner$ transfers asset subset X to the operators' aggregate multi-signature address.

Definition 3 (Transfer authenticity). *A protocol Π satisfies transfer authenticity if, for any asset subset $X \subseteq \mathcal{U}$, the probability that a recreation transaction $Create(X, \mathcal{BC}_2)$ is accepted without a corresponding valid burn transaction $Burn(X, \mathcal{BC}_1)$ having occurred is negligible in κ:*

$$\Pr\begin{bmatrix} params \leftarrow Setup(1^\kappa); \\ X \leftarrow \mathcal{A}(params); \\ Create(X, \mathcal{BC}_2) \text{ accepted}; \\ \nexists \text{ valid } Burn(X, \mathcal{BC}_1) \end{bmatrix} \leq negl(\kappa)$$

Here, $Create(X, \mathcal{BC}_2)$ represents the minting of a wrapped asset X on the secondary blockchain $\mathcal{BC}_2$. Furthermore, $Burn(X, Owner_1)$ represents a transaction on $\mathcal{BC}_2$ where the wrapped asset X is burnt.

Definition 4 (Double-spend prevention). *A protocol Π satisfies double-spend prevention if, for any asset subset $X \subseteq U$ that has been burnt in $\mathcal{BC}_1$, the probability that X is successfully recreated more than once is negligible in κ:*

$$\Pr\begin{bmatrix} params \leftarrow Setup(1^\kappa); \\ X \leftarrow \mathcal{A}(params); \\ Burn(X, \mathcal{BC}_1) \text{ accepted}; \\ Create(X, \mathcal{BC}_2) \text{ accepted}; \\ Create'(X, \mathcal{BC}'_2) \text{ accepted} \end{bmatrix} \leq negl(\kappa)$$

where $Create'$ represents a second creation operation, possibly on a different blockchain $\mathcal{BC}'_2$.

Definition 5 (Ownership transfer security). *A protocol Π satisfies ownership transfer security if for any asset subset $X \subseteq \mathcal{U}$ initially owned by $Owner_0$ on $\mathcal{BC}_1$ and any adversary $\mathcal{A}$ oblivious on the secret key of the recipient $Owner_1$, the probability that $\mathcal{A}$ can transfer X to an address not controlled by $Owner_1$ on $\mathcal{BC}_1$ after the unlocking phase is negligible in κ. If we write $\mathcal{E}(Owner_1, X)$ to denote the event that assets X are transferred to an address under the control of $Owner_1$, then:*

$$\Pr\begin{bmatrix} params \leftarrow Setup(1^\kappa); \\ (X, Owner_0, Owner_1) \leftarrow \mathcal{A}(params); \\ Lock(X, Owner_0) \text{ accepted on } \mathcal{BC}_1; \\ Burn(X, Owner_1.pubKey) \text{ accepted on } \mathcal{BC}_2; \\ \neg\mathcal{G}(Owner_1, X) \end{bmatrix} \leq negl(\kappa)$$

Definition 6 (Liveness). *A protocol Π satisfies liveness if, for any subset $X \subseteq \mathcal{U}$ locked by $Owner_0$ and burned $Owner_1$ with a valid public key, and assuming at least one honest operator follow the protocol, the assets X will eventually be transferred to $Owner_1$ on $\mathcal{BC}_1$. If we denote Δ a time bound specified by the system then, given at least one honest operator:*

$$\Pr\begin{bmatrix} params \leftarrow Setup(1^\kappa); \\ Lock(X, Owner_0) \text{ accepted at time } t_0; \\ Burn(X, Owner_1.pubKey) \text{ accepted at time } t_1; \\ \exists t_2 \leq t_1 + \Delta : \mathcal{G}(Owner_1, X) \text{ at time } t_2 \end{bmatrix} \geq 1 - negl(\kappa)$$

Definition 7 (1-out-of-n security). *A protocol Π satisfies 1-out-of-n security if for any adversary $\mathcal{A}$ that can corrupt up to $n-1$ operators, the protocol maintains asset validity, transfer authenticity, double-spend prevention, and ownership transfer security, with overwhelming probability. That is, for any security property $\mathcal{P}$ defined above and any adversary $\mathcal{A}$ controlling a subset $\mathcal{C} \subset \mathcal{O}$ such that $|\mathcal{C}| \leq n-1$:*

$$\Pr\begin{bmatrix} params \leftarrow Setup(1^\kappa); \\ \mathcal{C} \leftarrow \mathcal{A}(params) \text{ where } |\mathcal{C}| \leq n-1; \\ \mathcal{A} \text{ violates property } \mathcal{P} \end{bmatrix} \leq negl(\kappa)$$

Definition 8 (Key share confidentiality). *A protocol Π satisfies key share confidentiality if, for any adversary $\mathcal{A}$ that does not know $Owner_1$ private key, the probability that $\mathcal{A}$ can extract any operator's key share from the encrypted communications is negligible in κ. Formally, for any operator O_i and its key share k_i:*

$$\Pr\begin{bmatrix} params \leftarrow Setup(1^\kappa); \\ (Owner_1.pubKey) \leftarrow \mathcal{A}(params); \\ encShare_i \leftarrow ECIES.Encrypt(Owner_1.pubKey, k_i); \\ k_i' \leftarrow \mathcal{A}(encShare_i); \\ k_i' = k_i \end{bmatrix} \leq negl(\kappa)$$

2.3 Dependencies with respect to the 1-out-of-n assumption

The 1-out-of-n honesty assumption is basic in this proposal. This assumption fundamentally shapes how the protocol achieves its security guarantees and determines which properties can be maintained even under adversarial conditions. The following analysis examines each security property and clarifies its relationship to this assumption. We consider 3 categories:

- Critical dependency:
  1. Asset validity relies entirely on the 1-out-of-n assumption. The protocol requires operators to verify that lock transactions correspond to assets legitimately owned by the sender before proceeding with the setup process. Without at least one honest operator performing this verification, corrupted operators could approve lock transactions for non-existent or already-spent assets. The honest operator serves as the essential gatekeeper that prevents the protocol from accepting invalid asset locks.
  2. Transfer authenticity depends fundamentally on honest operator participation during the unlocking phase. When assets are burned on the secondary blockchain, operators must verify this burn transaction before providing their encrypted key shares. The honest operator ensures that assets can only be unlocked on Bitcoin when a corresponding legitimate burn has occurred on the secondary chain. Without this verification by at least one honest participant, the protocol could not maintain the crucial invariant that prevents unauthorized asset recreation.
  3. Double-spend prevention requires honest operators to maintain consistent protocol state and refuse participation in multiple unlocking operations for the same asset. The honest operator acts as a safeguard against attempts by corrupted operators to facilitate multiple unlocks of the same locked asset. This property cannot be maintained without the honest operator's disciplined adherence to the protocol's state management requirements.
  4. Liveness depends on the 1-out-of-n assumption to ensure that legitimate transfer requests eventually complete. The honest operator guarantees that valid lock and burn transactions will be processed, preventing corrupted operators from indefinitely blocking legitimate transfers. Without at least one honest operator willing to participate in the protocol, the system could deadlock, preventing any asset transfers from completing.
  5. Secure group selection relies on honest operators to provide valid encrypted key shares to legitimate recipients. The protocol's security depends on $Owner_1$ being able to select a group that includes at least one honest operator. The honest operator ensures that valid key shares are always available to legitimate recipients, preventing scenarios where only corrupted operators respond to transfer requests.
- Partial dependency: Ownership transfer security has a nuanced relationship with the 1-out-of-n assumption. While the ECIES encryption used for key share distribution provides confidentiality regardless of operator honesty, the overall ownership transfer security depends on honest operators to properly

verify recipients and refuse to provide key shares to unauthorized parties. The honest operator ensures that the protocol's ownership verification mechanisms function correctly.
- Independent of the 1-out-of-n honesty assumption: Key share confidentiality represents the only security property that does not directly depend on the 1-out-of-n assumption. This property is guaranteed by the cryptographic security of ECIES encryption, which prevents adversaries from extracting key shares without possession of the recipient's private key. The confidentiality holds regardless of how many operators are corrupted, as it relies solely on the underlying cryptographic assumptions rather than honest behaviour.

The 1-out-of-n honesty assumption is not just a convenience but an essential requirement for nearly all security properties of the protocol. The assumption enables a trust-minimized approach that avoids the need for honest majorities while still providing strong security guarantees. The protocol's architecture leverages this assumption efficiently by ensuring that a single honest operator can prevent all major classes of attacks while enabling legitimate operations.

2.4 Threat model

We present a threat model for the formal characterization of potential adversaries, their capabilities, and the attack vectors they might exploit. The model below considers adversaries who can corrupt a subset of protocol participants, manipulate network communications, and exploit timing differences between blockchain confirmations. One focuses on the 1-out-of-n honest operator assumption, which distinguishes our approach from traditional multi-signature schemes. The model of this protocol is given by the following entities and participants:

One assumes the source blockchain $\mathcal{BC}_1$ to be Bitcoin. Let us assume that $\mathcal{BC}_2$ is the destination blockchain: A smart contract-enabled blockchain that can execute arbitrary logic

The protocol involves four key participants. Operators $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$ are entities responsible for facilitating cross-chain transfers. Owner_0 is the initial asset owner on $\mathcal{BC}_1$, while Owner_1 is the recipient of the asset after cross-chain transfer. The adversary $\mathcal{A}$ represents any entity attempting to compromise the protocol.

The protocol consists of four main technical components. The BitVMX Program executes the verification logic for unlocking assets on Bitcoin. The Smart Contract manages wrapped asset creation and burning on $\mathcal{BC}_2$. MuSig2 Aggregation enables multi-signature creation among operators, while ECIES Encryption secures communication between operators and owners.

The protocol relies on five trust assumptions, namely: the 1-out-of-n honest operator assumption requires that at least one operator among n follows the protocol correctly, serving as the fundamental trust assumption. Both $\mathcal{BC}_1$ and $\mathcal{BC}_2$ operate correctly, maintain consensus, and execute transactions as programmed. BitVMX security assumes the verification system operates according

to its specifications and correctly enables challenge-response protocols. Non-collusion requires that owners and operators maintain independence, with no universal collusion between all participants. Finally, key management assumes that participants securely generate and store their cryptographic keys.

The adversary model defines five core capabilities that establish the security boundaries of the protocol. The adversary can achieve operator corruption by compromising up to $n - 1$ operators, gaining complete access to their private keys and the ability to make them deviate arbitrarily from the established protocol. Through network observation capabilities, the adversary can monitor all public network communications, including comprehensive visibility into all transactions occurring on both blockchain networks. The adversary possesses network delay capabilities, allowing it to delay network messages and blockchain transactions up to a specified bound Δ, though it cannot indefinitely block these communications. Additionally, the adversary can execute transaction creation by generating arbitrary valid transactions on both blockchain networks. The model constrains the adversary through computational bounds, defining it as a probabilistic polynomial-time (PPT) machine that operates within established computational limitations.

On the other hand, The adversary is explicitly unable to corrupt all n operators simultaneously, break the underlying cryptographic primitives, violate the consensus rules of either blockchain, indefinitely prevent transactions from being confirmed, and access encrypted communications without the corresponding private key.

The protocol operates within the following structured communication framework. Public blockchain communication ensures that all blockchain transactions remain publicly visible and immutable once confirmed, providing transparency and verifiability for all network participants. Private operator-owner communication utilizes encrypted channels through ECIES encryption, protecting sensitive information exchanges between these critical parties. The system operates under asynchronous communication assumptions, where the network experiences variable delays with a maximum message delay bound of Δ. Despite these delays, reliable delivery guarantees ensure that messages and transactions are eventually delivered within the established time bound Δ. The framework explicitly operates without trusted broadcast capabilities, requiring that all multi-party communication be secured through cryptographic means rather than relying on trusted intermediaries.

The security model relies on the following cryptographic assumptions. The discrete logarithm problem (DLP) establishes that given $Y = x \cdot G$ where $G \in E(\mathbb{Z}_q)$ is a generator, determining the scalar x remains computationally infeasible for adversaries. Building upon this foundation, the computational Diffie-Hellman problem (CDHP) assumes that when provided with $a \cdot G$ and $b \cdot G$ for unknown scalars $a, b \in \mathbb{Z}_q$, computing $a \cdot b \cdot G$ presents an insurmountable computational challenge. The elliptic curve discrete logarithm problem (ECDLP) represents the specific instantiation of the DLP within elliptic curve groups $E(\mathbb{Z}_q)$, providing the cryptographic hardness for elliptic curve-based operations. The

analysis employs the random oracle model, treating hash functions as perfect random oracles for security evaluation purposes. The encryption components rely on IND-CPA security, ensuring that the symmetric encryption component of ECIES maintains indistinguishability under chosen-plaintext attacks. Finally, the framework requires existential unforgeability, guaranteeing that the digital signature schemes employed remain existentially unforgeable under chosen-message attacks.

This threat model addresses the following categories of attacks:

1. Protocol flow attacks: The protocol must defend against several categories of attacks that target the execution flow and timing aspects of cross-chain operations. Replay attacks represent a fundamental threat, where adversaries attempt to reuse signatures or protocol messages from previous legitimate transactions to achieve unauthorized asset transfers. Race conditions pose timing-based vulnerabilities that adversaries can exploit by manipulating the sequence or timing of cross-chain events to create inconsistent states or bypass security checks. Griefing attacks involve malicious operators who deliberately attempt to block or delay protocol execution, potentially causing asset freezing or denial of service without necessarily stealing assets. Eclipse attacks present network-level threats where adversaries isolate honest operators from the broader network, preventing them from receiving critical updates or participating in consensus mechanisms.
2. Cryptographic attacks: The cryptographic foundation of the protocol faces multiple attack vectors that target the underlying mathematical primitives and their implementations. Key compromise attacks occur when adversaries obtain private keys through side-channel analysis, implementation vulnerabilities, or other technical exploitation methods that bypass the intended cryptographic protections. Signature forgery represents attempts by adversaries to create valid signatures for unauthorized actions without possessing the corresponding private keys, potentially enabling unauthorized asset transfers or protocol manipulations. Rogue key attacks specifically target multi-signature schemes by allowing adversaries to manipulate their public key contributions during the aggregation process, potentially gaining disproportionate influence over collective signature operations.
3. Asset-specific attacks: The protocol must address attacks that directly target asset integrity and ownership throughout the cross-chain transfer process. Double-spending attacks represent attempts by adversaries to unlock or transfer the same asset multiple times, potentially creating artificial inflation or theft scenarios across the participating blockchain networks. Unauthorized transfer attacks involve adversaries attempting to redirect legitimate asset transfers to unauthorized recipients, effectively stealing assets during the cross-chain migration process. Asset freezing attacks occur when malicious operators deliberately prevent asset unlocking or completion of transfers, potentially resulting in permanent loss of access to legitimate user assets.
4. Cross-chain consistency attacks: The inherent complexity of managing state across multiple blockchain networks creates opportunities for attacks that ex-

ploit inconsistencies between chains. Invalid state transition attacks involve adversaries creating or exploiting inconsistent states between the source and destination blockchains, potentially allowing unauthorized asset creation or destruction. Finality differences present attack vectors where adversaries exploit the varying finality guarantees and confirmation times between different blockchain networks to create temporary or permanent inconsistencies. Chain reorganization attacks leverage the possibility of blockchain reorganizations or forks to undo completed asset transfers, potentially enabling sophisticated double-spending scenarios that span multiple blockchain networks.

In particular, this model considers the following specific threat scenarios:

1. Operator collusion: The most significant threat to protocol security appears when up to $n-1$ operators coordinate to compromise the system through coordinated malicious behaviour. These collusive operators can pursue multiple attack vectors designed to subvert the protocol's intended functionality and security guarantees. They may attempt to unlock assets without corresponding legitimate burn transactions on the secondary blockchain, effectively creating unauthorized asset transfers that bypass the protocol's cross-chain verification mechanisms. Additionally, colluding operators can refuse to process legitimate unlock requests from honest users, creating denial-of-service conditions that prevent rightful asset access. The provision of incorrect key shares to legitimate recipients represents another attack vector, where operators deliberately distribute invalid cryptographic material to prevent successful asset recovery. Furthermore, malicious operators may manipulate the BitVMX execution environment to favour their coordinated interests, potentially compromising the verification logic that ensures protocol correctness. To address these threats, the protocol must maintain security even when facing $n - 1$ colluding operators, ensuring that at least one honest operator retains the capability to prevent unauthorized actions. The BitVMX challenge-response mechanism serves as a critical security requirement, enabling the detection of invalid executions and maintaining protocol integrity despite widespread operator collusion.
2. Owner impersonation: Impersonation attacks employ various technical approaches to compromise the recipient identification and verification mechanisms within the protocol. Adversaries may attempt to intercept encrypted key shares that operators transmit to the intended Owner_1, positioning themselves to capture sensitive cryptographic material during the communication process. Once intercepted, attackers seek to decrypt these key shares without possessing Owner_1 legitimate private key, requiring them to break the ECIES encryption or exploit implementation vulnerabilities. Another attack vector involves creating forged proofs of ownership on the secondary blockchain, where adversaries attempt to present false evidence of their authority to receive cross-chain transferred assets. To counter these threats, the protocol must implement robust cryptographic verification of Owner_1 identity, ensuring that only legitimate recipients can access transferred assets.

Key shares must maintain strict confidentiality, remaining accessible exclusively to the intended Owner_1 throughout the entire transfer process. Additionally, the protocol must verify that burn transactions originate from legitimate asset owners, preventing unauthorized parties from initiating transfers using assets they do not rightfully control.

3. Cross-chain exploitation: Coordinating operations across multiple blockchain networks creates opportunities for adversaries to exploit inherent differences and inconsistencies between the participating chains. These exploitation attacks target the complex synchronization requirements and varying operational characteristics of different blockchain systems. Adversaries may attempt to create inconsistent states between the source and destination chains, leveraging timing differences or operational discrepancies to establish contradictory asset states that benefit their malicious objectives. Finality differences between blockchain networks present particularly sophisticated attack opportunities, where adversaries exploit varying confirmation requirements and finality guarantees to revert transactions on one chain after achieving completion on another. This temporal manipulation can enable complex double-spending scenarios that span multiple networks. Additionally, adversaries may manipulate timestamps or block confirmations to create artificial timing conditions that bypass the protocol's synchronization safeguards. To address these cross-chain vulnerabilities, the protocol must implement consistent state verification mechanisms that ensure coherent asset states across all participating blockchain networks. Sufficient confirmation waiting periods serve as essential security requirements, providing adequate time for transaction finality to prevent exploitation of timing differences. The protocol must also enforce atomic execution of cross-chain operations, ensuring that asset transfers either complete successfully across all involved chains or fail entirely, preventing partial states that adversaries could exploit.

2.5 Blockchain interoperability

One of the most important challenges in the blockchain ecosystem has been the complexities and insecurity of communication and asset transfer between different networks.

Blockchain bridges solve this problem by connecting two or more blockchain networks, enabling message passing and cross-chain transactions, enabling flawless interoperability

The basic process involves locking an asset on one blockchain and representing it on another through a wrapped token. Smart contracts handle the transaction, ensuring accuracy and security throughout the process:

1. A user locks an asset on the source blockchain.
2. A wrapped token, representing the locked asset, is minted on the destination blockchain.
3. Smart contracts manage this process, ensuring that transactions are valid and secure.

2.6 Types of blockchain bridge

There are different approaches to bridging blockchains, each with its own methods and trust requirements. These can be categorized into trusted, trust-minimized and trustless models.

1. Trusted bridges: trusted bridges rely on a central entity or a group of validators to oversee operations and ensure security.
   (a) Custodial bridges: A centralized entity holds the original assets and mints their equivalents on the destination chain.
   (b) Federated bridges: These are operated by a group of validators who manage assets and verify transactions. One finds two models:
      i. m - of - n: requires a majority ($m > n$) of validators to approve operations.
      ii. 1 - of - n: requires just one honest validator to process a transaction.
2. Trustless bridges: trustless bridges eliminate the need for centralized entities and replace them with cryptographic schemes and smart contracts for transaction validation. The essential functioning follows:
   (a) A user deposits an asset into a smart contract on the source chain, which locks it, making it unusable until the process completes.
   (b) The contract generates a proof of both the deposit and the lock.
   (c) A relayer (or oracle) forwards the proof to a smart contract on the destination chain.
   (d) The destination contract verifies the proof and mints the equivalent asset.
   (e) Upon successful minting confirmation, the locked asset in the source chain is burned, ensuring that no duplicate assets exist across chains.
3. Trust-Minimized Bridges: Trustless bridges cannot be built for the Bitcoin blockchain given today's technology. Trusted bridges, either custodial or federated, require a level of centralization that is not acceptable for many Bitcoin use cases.
   Trust-minimizing bridges achieve a balanced solution in which a group of operators oversees the management of the bridge, while a single honest one can prevent invalid operations from being accepted.

Key components Blockchain bridges are powered by several components that play distinct roles in ensuring secure and efficient operations.

1. Custodians: manage locked assets in centralized bridges and handle deposits and withdrawals.
2. Oracles: feed external data into blockchains and monitor activity, such as prices or contract events.
3. Validators: verify transactions, enforce rules, and confirm deposits before minting wrapped assets.

Each component contributes to the functionality and reliability of the bridge, whether centralized or decentralized.

Challenges in using bridges While blockchain bridges offer powerful interoperability solutions, they face notable challenges that must be addressed for widespread adoption:

1. Security: bridges are high-value targets for attackers. Trustless bridges, while decentralized, can be exploited due to their complexity. Regular audits and layered security are essential.
2. Technical complexity: building bridges for multiple networks requires navigating diverse protocols and architectures.
3. Regulation: operating across jurisdictions with varying laws makes compliance challenging.
4. Scalability: managing large transaction volumes and optimizing costs while maintaining efficiency can strain bridge operations.

Bitcoin bridges The development of Bitcoin bridges offers particular challenges, indeed:

Bitcoin's scripting limitations create fundamental constraints for cross-chain interoperability protocols. The UTxO model's inherent restrictions on state management, combined with Bitcoin Script's limited expressiveness, shape the design space for bridging solutions. The absence of covenant support in Bitcoin presents a challenge, since funds cannot be programmatically constrained to specific destination addresses based on external validation conditions. This limitation forces bridges to implement multi-party coordination protocols rather than direct programmatic constraints.

State management represents another significant challenge. Without the ability to maintain mutable state within transactions, bridges must implement complex UTxO management schemes to track cross-chain assets and their state. This creates additional complexity in ensuring atomicity and preventing replay attacks across bridge operations.

The computational constraints of Bitcoin Script further restrict the types of validation logic that can be performed on-chain. Complex operations like Merkle proof verification or dynamic validator set management cannot be implemented directly within Bitcoin's scripting system.

These constraints have led to several architectural patterns in the ecosystem.

1. Federation-based approaches leverage multi-signature schemes to implement bridge security through distributed trust. While this provides a practical solution, it introduces additional trust assumptions compared to fully programmatic approaches possible on more expressive platforms.
2. Hash time-locked contracts (HTLCs) represent another pattern that works within Bitcoin's constraints. However, their application is primarily limited to atomic swap scenarios rather than general-purpose bridging protocols. The inability to implement complex validation logic on-chain means that HTLC-based approaches struggle to scale beyond simple two-party exchanges.
3. Layer-2 protocols represent an alternative approach, introducing additional programmability through separate security domains. Solutions like Liquid

and Rootstock implement more expressive scripting capabilities, enabling complex bridge logic at the cost of additional trust assumptions in their security models.

These architectural patterns demonstrate how Bitcoin's security-focused design decisions continue to influence the development of cross-chain interoperability solutions. The trade-off between trustlessness and functionality remains a central consideration in bridge design, with different approaches optimizing for different points along this spectrum.

2.7 BitVMX

BitVMX [18] is a virtual CPU design that allows arbitrary programs to run on Bitcoin through an optimistic verification system. It builds on the challenge-response framework introduced by BitVM [3], implementing a general-purpose CPU that can be verified using Bitcoin's scripting capabilities. The system is compatible with common processor architectures like RISC-V and MIPS.

The key innovation of BitVMX lies in its streamlined approach. It employs hash chains to track program execution, uses memory-mapped registers, and introduces an enhanced challenge-response protocol. The system includes a message linking protocol that enables authenticated communication between participants, effectively creating stateful smart contracts by maintaining shared state across transactions.

The verification process uses presigned transactions to manage challenge-response interactions. When disputes arise, the system leverages the hash chain of program execution to pinpoint and investigate computational errors through n-ary search. This represents a significant improvement over BitVM, as BitVMX eliminates the need for Merkle trees in CPU instructions and memory operations, while also removing dependence on signature equivocations. These improvements reduce complexity and make BitVMX a strong alternative to BitVM2 [19].

In practice, BitVMX operates as a two-party system between a prover (operator) and a verifier, though it can be expanded to accommodate multiple verifiers. When funds are locked in a Bitcoin UTxO, they can only be spent based on the outcome of a predefined program with specific inputs. The operator must demonstrate correct program execution to access the funds. If the verifier agrees with the execution results, the process proceeds smoothly. If not, both parties enter a dispute resolution protocol on the Bitcoin blockchain.

The execution tracking system in BitVMX represents a notable advancement. Both parties execute the program locally, generating both an execution trace and a hash chain for each step. The hash chain is built by combining each step's trace with the previous hash and applying a secure hashing function. This creates an unforgeable record of execution, where any computational differences between parties result in divergent hash chains.

BitVMX's n-ary search capability allows for faster identification of computational conflicts compared to BitVM's binary search approach. Once an error is located, specialized challenges can determine its precise nature, such as tracking

memory access patterns to identify incorrect memory operations. The system's response verification ensures that all responses directly correspond to specific challenges, allowing the challenger to prove dishonest behaviour using the operator's own signed messages.

This design offers flexibility in balancing various trade-offs, including transaction costs versus rounds of interaction, prover versus verifier computational costs, and precomputation requirements versus protocol rounds. While BitVMX can still utilize signature equivocations like BitVM, they are not fundamental to its operation, resulting in a more streamlined protocol.

2.8 Cryptographic components

Let us denote with $\mathcal{M}$ and $\mathcal{C}$ the sets of messages and ciphertexts, respectively. One assumes that the following data is public: an elliptic curve E with a prime q such that the discrete logarithm problem is hard over $E(\mathbb{Z}_q)$, a hash function $H : E(\mathbb{Z}_q)^\star \to \{0,1\}^{l_1+l_2}$, for parameters $l_1, l_2 > 0$, and a MAC function $\text{MAC} : \mathcal{C} \to \mathbb{Z}_q$. Let $Sym = (\varepsilon, \delta)$ be a symmetric encryption mechanism.

This protocol involves a phase where operators set $1-1$ communications with owners using the ECIES protocol [8], which one introduces informally below.

Algorithm 1 ECIES.KeyGen

1: Choose $A \in E(\mathbb{Z}_q)$ of order q, secret $s \in \mathbb{Z}_q$, compute $B = s \cdot A$
2: **Output:** public key (A, B), private key s

Algorithm 2 ECIES.Encrypt(m, pk)

1: Choose random $k \in [1, q-1]$
2: Compute $R = k \cdot A$, $Z = k \cdot B$, $\kappa_E \,\|\, \kappa_M = H(R, Z)$
3: Compute $C = \varepsilon_{\kappa_E}(m)$, $t = \text{MAC}_{\kappa_M}(C)$
4: **return** (R, C, t)

Algorithm 3 ECIES.Decrypt(C, sk)

1: Compute $Z = s \cdot R$, $\kappa_E \,\|\, \kappa_M = H(R, Z)$, $t' = \text{MAC}_{\kappa_M}(C)$
2: **if** $t \neq t'$ **then reject** C
3: **elsereturn** $m = \delta_{\kappa_E}(C)$
4: **end if**

This protocol also makes use of MuSig2 [23] for multi-signatures involving the committee. In this setting one considers a point $G \in E(\mathbb{Z}_q)$ of prime order q, and

two hash functions $H_{agg} : E(\mathbb{Z}_q)^\star \times \mathcal{M} \to \mathbb{Z}_q$, and $H_{sig} : E(\mathbb{Z}_q) \times E(\mathbb{Z}_q) \times \mathcal{M} \to \mathbb{Z}_q$.

Algorithm 4 MuSig.KeyGen

1: Each participant O_i selects secret key $k_i \in \mathbb{Z}_q$ and computes $K_i = k_i \cdot G \in E(\mathbb{Z}_q)$
2: Each O_i computes $a_i = H_{agg}(K_i, \{K_1, \ldots, K_n\}) \in \mathbb{Z}_q$
3: Aggregate public key: $K = \sum_{i=1}^{n} a_i \cdot K_i \in E(\mathbb{Z}_q)$
4: Implied aggregate secret key: $k = \sum_{i=1}^{n} a_i \cdot k_i \mod q$

Algorithm 5 MuSig.Sign(m)

1: **Round 1:** Each O_i generates $r_{i1}, r_{i2} \in \mathbb{Z}_q$, computes $R_{i1} = r_{i1} \cdot G$, $R_{i2} = r_{i2} \cdot G$
2: Each O_i broadcasts (R_{i1}, R_{i2})
3: **Round 2:** All compute $b = H_{agg}(R_{11} \,\|\, R_{12} \,\|\, \ldots \,\|\, R_{n1} \,\|\, R_{n2}, m) \in E(\mathbb{Z}_q)$
4: Each O_i computes $R_i = R_{i1} + b \cdot R_{i2}$ and all compute $R = \sum_{i=1}^{n} R_i$
5: All compute challenge $c = H_{sig}(K, R, m) \in \mathbb{Z}_q$
6: Each O_i computes and broadcasts $s_i = r_{i1} + b \cdot r_{i2} + c \cdot a_i \cdot k_i \mod q$
7: Final signature: $\sigma = (R, s) = \left(R, \sum_{i=1}^{n} s_i \mod q\right)$

Algorithm 6 MuSig.Verification(m, σ)

1: Compute $c = H_{sig}(K, R, m)$
2: Check if $s \cdot G = R + c \cdot K$
3: **if** equality holds **then return** valid
4: **elsereturn** invalid
5: **end if**

3 Protocol overview

Let us assume that a user $Owner_0$ has an asset (UTxO) on Bitcoin and wants to wrap it to use it on a secondary blockchain with smart contracts support.

A group of $n \geq 1$ operators is in charge of managing the protocol and have published an aggregate address to receive assets and help with the wrapping, following MuSig2.

3.1 Locking keys and power set

For each locked asset in Bitcoin, the operators generate a corresponding list of private and public keys. To this end, each operator O_i computes a random secret key k_i and public key $K_i = k_i \cdot G$, for $G \in E(\mathbb{Z}_q)$.

Let $\mathcal{ID}$ be the power set of all the subsets of operators which will collaborate with $Owner_1$ in forthcoming steps of the protocol.

In forthcoming steps, users will have the ability to consider elements of $\mathcal{ID}$, and define both their aggregate public key and the associated signing key. This can be done restricting the aggregation process to the involved operators, where the coefficients are computed following MuSig2 and using the public keys of the willing operators.

3.2 Fundamental phases

One can divide the protocol in three phases, locking, asset usage, and unlocking, which are described below.

1. Locking: ensures the secure transition of assets from Bitcoin into its wrapped form on the second blockchain.
   (a) A committee of operators generates, and shares, an aggregate address on Bitcoin following MuSig2.
   (b) Owner0 transfers the asset to the operator's aggregate address on Bitcoin using a *lock request* transaction, which includes:
      i. The assets to be locked.
      ii. Protocol fees to support Bitcoin and the second blockchain operations.
      iii. The second blockchain address selected to receive the wrapped assets.

      The lock request transaction offers two potential outcomes:
      i. It can be redeemed by all operators to complete the lock process.
      ii. If not accepted within a specified time-frame, the owner can reclaim the asset after a time-lock expires.
   (c) The user communicates with the operators off-chain, sharing details of the lock request transaction.
   (d) The operators identify the request and initiate the setup process, which involves:
      i. Preparing BitVMX presigned transactions and secrets.
      ii. Creating aggregate keys for the transfer of ownership. This is done using MuSig2.
   (e) The operators create a BitVMX program that will validate an unlocking request in the future, and will transfer the asset into one of the predefined 2^n addresses.
   (f) A wrapped asset linking to Owner0 original asset is created on the second blockchain.
2. Asset usage: in this phase, Owner0 uses the asset on the secondary chain, possibly transferring it to other users or smart contracts.
3. Unlocking:
   (a) $Owner_1$ initiates the transfer by sending the wrapped asset to the smart contract for burning. This is done by signing a burn transaction for the token, including an additional public key to encrypt the transfer of ownership information.

(b) Operators monitoring the smart contract will detect the burn request and verify the asset origin. The operators will use the provided public key to encrypt their shares of the aggregated key for that UTxO.
(c) $Owner_1$ will initiate off-chain communication with the operators to retrieve their encrypted shares and associated public keys. These shares are encrypted using the public key provided by $Owner_1$. Here, the cryptographic scheme in use is ECIES.
(d) $Owner_1$ verifies that the decrypted keys match the public keys linked to the locked UTxO. This allows the identification of the subset of valid keys and assign an element in $\mathcal{ID}$ to this subset.
(e) $Owner_1$ sends a message to the smart-contract selecting a group-id which corresponds to the aggregate public key of the subset of operators who sent valid keys. (Address (S)).
(f) An operator starts the execution of the BitVMX unlocking program on Bitcoin, giving as input the selected group-id as well as a zero-knowledge proof that the owner burned and selected that group-id on a smart-contract transaction.
(g) If step (f) is successful, the locked asset is sent to address(S).
(h) $Owner_1$ calculates an aggregated private key for address(S) and transfers the asset to one address of his choice.

3.3 Interaction between owners and operators

Once detected the burning transaction in the smart contract, a subset of operators set communication with $Owner_1$ using the ECIES scheme. This communication is possible because $Owner_1$ will generate a key pair, specific to $1-1$ communications, given by a random secret key $skc \in \mathbb{Z}_q$, and a public key $(G, skc \cdot G)$.

After checking the validity of the received secrets, $Owner_1$ can derive a set S in $\mathcal{ID}$ and compute the associated aggregated key following MuSig2.

4 Security assumptions and analysis

4.1 Security assumptions

The security of this proposed protocol relies on several cryptographic assumptions and the robustness of the underlying primitives, in particular MuSig2 and ECIES. Below, there is an outline of the key assumptions and their implications for the protocol's security.

The protocol operates under the 1-out-of-n honesty assumption, which assumes that among n operators, at least one is honest and follows the protocol specification correctly. This assumption is critical for ensuring the integrity and correctness of the protocol, as the honest participant serves as a safeguard against malicious behaviour by the remaining $n-1$ operators. The honest participant ensures that the protocol's outputs are verified correctly, that BitVMX challenges are set when required, and that no adversarial manipulation goes undetected.

The protocol employs MuSig2 for the multisignatures involving the committee of operators, and ECIES for setting secure communications between owners and operators. It therefore inherits the following security assumptions:

1. Discrete logarithm problem over elliptic curves: the hardness of computing discrete logarithms in the underlying elliptic curve group ensures that an adversary cannot forge signatures without knowledge of the private keys.
2. Random oracle model: one assumes that cryptographic hash functions behave as random oracles. This model is essential for proving the security of the scheme against chosen-message attacks.
3. Elliptic curve Diffie-Hellman problem: the hardness of solving the Diffie-Hellman problem ensures that an adversary cannot obtain the shared secret used for symmetric key derivation.
4. Indistinguishability under chosen-plaintext attack (IND-CPA): the symmetric encryption component of ECIES is assumed to be IND-CPA secure, ensuring that ciphertexts do not leak information about the plaintext.
5. Existential unforgeability under chosen-message attacks: The MAC component in ECIES ensures that ciphertexts cannot be forged or tampered with without detection.

One observes that MuSig2 incorporates a key aggregation technique that mitigates rogue-key attacks, ensuring that malicious participants cannot manipulate their public keys to gain an unfair advantage during signature generation.

4.2 Security analysis

In this section, we provide the main ideas which set the security properties of protocol. Each proof demonstrates how the design of the protocol ensures specific security guarantees under the cryptographic assumptions which have been established.

Theorem 1. *Under the discrete logarithm assumption in elliptic curve groups and assuming at least one honest operator, the protocol satisfies asset validity with overwhelming probability.*

Proof. Let us assume that there exists a PPT adversary $\mathcal{A}$ that violates asset validity with non-negligible probability ϵ. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve the discrete logarithm problem.

Given a discrete logarithm challenge $(G, Y = x \cdot G)$ where $x \in \mathbb{Z}_q$ is unknown, $\mathcal{B}$ runs the protocol setup, embedding the challenge public key Y as the public key for some UTxO not controlled by $Owner_0$. When $\mathcal{A}$ attempts to violate asset validity by locking assets $X \not\subseteq \text{Assets}(Owner_0, \mathcal{BC}_1)$, $\mathcal{A}$ must produce a valid signature for the UTxO corresponding to public key Y.

If $\mathcal{A}$ succeeds, then $\mathcal{A}$ has produced a valid ECDSA signature (r, s) for some message m under public key Y. Using the forking lemma, $\mathcal{B}$ can extract two signatures (r, s_1) and (r, s_2) for the same r but different hash values h_1 and

h_2. This gives $\mathcal{B}$ the discrete logarithm x, contradicting the discrete logarithm assumption.

However, even if $\mathcal{A}$ could forge signatures, the honest operator O_j will verify the lock transaction before participating. The honest operator checks that all input UTxOs in the lock transaction belong to $\text{Assets}(Owner_0, \mathcal{BC}_1)$. Since $X \not\subseteq \text{Assets}(Owner_0, \mathcal{BC}_1)$, the honest operator will reject the lock request, preventing protocol progression.

Theorem 2. *Under the discrete logarithm assumption and BitVMX security assumptions, with at least one honest operator, the protocol satisfies transfer authenticity.*

Proof. One considers two different cases in this situation:

- Lock to wrap: in this situation, one assumes that the adversary $\mathcal{A}$ creates wrapped assets on $\mathcal{BC}_2$ without corresponding lock on $\mathcal{BC}_1$. The wrapped asset creation requires operator signatures on $\mathcal{BC}_2$. At least one honest operator O_j will only sign after verifying the lock transaction exists on $\mathcal{BC}_1$ with sufficient confirmations. If no valid lock exists, O_j refuses to sign, preventing wrapped asset creation.
- Burn to unlock: Let us suppose that $\mathcal{A}$ unlocks assets on $\mathcal{BC}_1$ without corresponding burn on $\mathcal{BC}_2$. The unlock requires BitVMX program execution with proof π that demonstrates:
  1. Existence of burn transaction tx_{burn} on $\mathcal{BC}_2$.
  2. Correct group selection $S \in \mathcal{ID}$.

  If no valid burn exists, the honest operator will challenge the BitVMX execution. Under BitVMX security assumptions, invalid executions can be proven incorrect through the challenge-response protocol. The challenger provides evidence that π does not correspond to a valid burn transaction, causing the execution to fail.

  Formally, let $\text{Verify}(\pi, stmt)$ be the BitVMX verification predicate and statement. If $stmt$ asserts existence of burn transaction tx_{burn} but no such transaction exists, then an honest challenger can prove $\neg\text{Verify}(\pi, stmt)$ with overwhelming probability.

Therefore, both directions of transfer authenticity hold with probability $1 - \text{negl}(\kappa)$.

Theorem 3. *The protocol prevents double-spending with overwhelming probability under the discrete logarithm assumption and honest operator assumption.*

Proof. Let us consider the following situations:

- Multiple Bitcoin unlocks: Each locked asset corresponds to a unique UTxO on $\mathcal{BC}_1$ with specific transaction identifier and output index. The BitVMX program execution creates a spending transaction that consumes this UTxO by transferring it to the aggregate address corresponding to the selected operator subset $S \in \mathcal{ID}$. Bitcoin's consensus mechanism enforces that each

UTxO can be spent exactly once across all valid transactions. Any subsequent unlock attempt must reference the same UTxO, but after successful BitVMX execution, this UTxO no longer exists in the blockchain state. The adversary cannot construct valid Bitcoin transactions that spend non-existent UTxOs, providing absolute protection against multiple Bitcoin unlocks of the same asset.

- Multiple secondary chain burns: The smart contract on $\mathcal{BC}_2$ maintains explicit state tracking for each wrapped asset through mappings that record burn status. When an asset X undergoes burning, the contract atomically updates its internal state to mark X as burned and rejects subsequent burn operations for the same asset identifier. The atomic execution guarantees provided by $\mathcal{BC}_2$ ensure that state updates and asset destruction occur as indivisible operations. Even under concurrent burn attempts, the blockchain's transaction ordering ensures that only the first operation succeeds while subsequent attempts fail due to the updated contract state.
- Cross-chain coordination attacks: Adversary $\mathcal{A}$ might attempt to exploit timing differences or coordination failures between blockchains to create inconsistent states, where assets appear burned multiple times or unlocked without corresponding burns. The honest operator O_j maintains consistent state across both blockchains and tracks which assets have been processed. When a legitimate burn occurs on $\mathcal{BC}_2$, the operator O_j records this event and provides encrypted key shares exactly once for each unique asset. The operator refuses participation in subsequent unlock attempts for previously processed assets.

The discrete logarithm assumption ensures that $\mathcal{A}$ cannot forge operator signatures or manipulate cryptographic proofs required for unauthorized operations. Under BitVMX security assumptions, invalid program executions can be detected and challenged by the honest operator, preventing unauthorized unlocks that might enable double-spending scenarios.

The combination of UTxO consumption mechanics, smart contract state management, and honest operator coordination creates multiple independent security barriers. The failure of any single mechanism does not compromise overall double-spending prevention, ensuring the property holds with probability $1 - \text{negl}(\kappa)$ where κ represents the security parameter.

Theorem 4. *Under the ECDHP and IND-CPA security assumptions, the protocol ensures ownership transfer security with overwhelming probability.*

Proof. Let us consider an adversary $\mathcal{A}$ attempts to redirect the transfer without knowing $Owner_1$'s private key sk_1. The adversary must:

1. Decrypt ECIES ciphertexts $C_i = \text{ECIES.Encrypt}(pk_1, share_i)$ for operator key shares.
   Under ECDHP assumption, $\mathcal{A}$ cannot decrypt C_i without sk_1. Formally, consider the ECIES encryption:
   - $(R, C, t) = \text{ECIES.Encrypt}(pk_1, share_i)$ where $R = r \cdot G$, $pk_1 = sk_1 \cdot G$

- Shared secret $Z = r \cdot pk_1 = sk_1 \cdot R$
- Decryption requires computing Z from (R, pk_1), which is the ECDHP

If $\mathcal{A}$ can decrypt with non-negligible probability ϵ, we construct ECDHP solver $\mathcal{B}$: Given (G, aG, bG), $\mathcal{B}$ sets $pk_1 = aG$, $R = bG$, runs $\mathcal{A}$ on the resulting ciphertext. If $\mathcal{A}$ succeeds, $\mathcal{B}$ extracts abG from the decryption process.

2. Compute a valid aggregate key for some subset $S \in \mathcal{ID}$.
   Even if $\mathcal{A}$ obtains some key shares through other means, the honest operator O_j only provides $share_j$ encrypted under pk_1. Without $share_j$, $\mathcal{A}$ cannot form a valid subset S that includes O_j. Any subset excluding O_j cannot unlock the asset because the protocol requires the honest operator's participation.

Therefore, ownership transfer security holds with probability $1 - \text{negl}(\kappa)$.

Theorem 5. *Under the cryptographic assumptions established, all security properties hold when at least one operator is honest, regardless of the corruption of up to $n - 1$ operators.*

Proof. Let $\mathcal{C} \subset \mathcal{O}$ be the set of corrupted operators with $|\mathcal{C}| \leq n - 1$, and let $O_j \in \mathcal{O} \setminus \mathcal{C}$ be the honest operator.

For each security property $\mathcal{P}$, namely: asset validity, transfer authenticity, double-spend prevention, and ownership transfer security, one shows that the honest operator O_j can maintain $\mathcal{P}$ despite corruption of $\mathcal{C}$:

1. Asset validity: O_j independently verifies lock transactions. Even if all operators in $\mathcal{C}$ approve invalid locks, O_j's refusal prevents protocol progression.
2. Transfer authenticity: O_j verifies burn transactions before providing key shares and challenges invalid BitVMX executions.
3. Double-spend prevention: O_j maintains consistent state and refuses multiple unlock participation. The security follows from Theorem 3 arguments.
4. Ownership transfer security: O_j correctly encrypts key shares for legitimate recipients.

Each property's security reduces to the honest behaviour of O_j combined with the underlying cryptographic assumptions. Since we assume at least one honest operator exists, all properties hold simultaneously. Therefore, 1-out-of-n security holds with overwhelming probability.

Theorem 6. *Under the assumption of having at least one honest operator, the protocol satisfies liveness.*

Proof. Let us consider a legitimate asset transfer where $Owner_0$ locks asset X and $Owner_1$ burns the corresponding wrapped asset. We must show the transfer eventually completes despite potential interference from up to n-1 corrupted operators.

The protocol achieves liveness through a combination of honest operator capabilities and protocol design features that prevent permanent blocking attacks. The honest operator O_j maintains the following capabilities that ensure forward progress:

1. Lock-to-wrap progression: After detecting a valid lock transaction on Bitcoin, the honest operator can unilaterally trigger wrapped asset creation on the secondary blockchain. Since wrapped asset creation requires only honest operator participation rather than full consensus, corrupted operators cannot prevent this step. The operator's verification of lock transaction validity within time Δ_1 and subsequent participation in wrapped asset creation within time Δ_2 ensures this phase completes.
2. Burn detection and response: When $Owner_1$ burns wrapped assets, the honest operator detects this event within blockchain confirmation bounds Δ_3. The operator provides encrypted key shares to $Owner_1$ within network message bounds Δ_4, and corrupted operators cannot prevent this communication since it occurs through independent channels between the honest operator and $Owner_1$.
3. BitVMX execution completion: The critical liveness mechanism operates through BitVMX's challenge-response framework. When corrupted operators attempt to block progress by refusing to participate or by initiating invalid challenges, the honest operator can force resolution, indeed: if corrupted operators refuse to execute the BitVMX program for valid burn transactions, the honest operator can initiate execution unilaterally. If corrupted operators attempt invalid executions to block progress, the honest operator challenges these executions through BitVMX's dispute resolution protocol. The security properties of BitVMX ensure that honest challenges succeed against invalid executions, forcing the protocol to progress to the correct state.
4. Prevention of permanent blocking: Corrupted operators cannot permanently block legitimate transfers because the protocol requires only one honest participant to maintain progress. The honest operator's economic incentives align with completing legitimate transfers, while the BitVMX challenge mechanism provides technical enforcement against blocking attempts.

   The protocol's time-lock mechanisms create bounded windows for each phase, but these bounds serve to prevent indefinite delays rather than enable blocking attacks. Within each bounded period, the honest operator can complete the necessary protocol steps despite non-cooperation from corrupted operators.
5. Completion guarantee: The total completion time $T \leq \sum_{i=1}^{5} \Delta_i$ represents the worst-case progression time when corrupted operators provide no cooperation. The honest operator's ability to complete each phase independently, combined with BitVMX's enforcement of correct execution, ensures that legitimate transfers progress to completion within these bounds.

Since the honest operator cannot be prevented from executing the protocol correctly and has both the technical capability and economic incentive to complete legitimate transfers, liveness holds with overwhelming probability.

4.3 Analysis of the combination of MuSig2, ECIES and BitVMX

The security of our cross-chain transfer protocol relies on the cryptographic composition of MuSig2, ECIES, and BitVMX, where each primitive operates within its established security model while maintaining independence from the others. This section outlines the rationale supporting the claim that their composition preserves the underlying security assumptions.

MuSig2 operates under the discrete logarithm assumption in elliptic curve groups, providing signature aggregation without compromising individual private keys. The protocol maintains MuSig2's security by ensuring that the key aggregation process $K = \sum_{i=1}^{n} a_i K_i$ and subsequent signature generation remain isolated from other cryptographic operations.

The composition preserves the random oracle model assumptions underlying MuSig2's security proof by ensuring that hash function calls H_{agg} and H_{sig} operate only on public values derived from the multi-signature protocol itself. The nonce generation process for each operator remains independent of ECIES encryption keys and BitVMX program execution, preventing potential correlation attacks that might compromise the Wagner's algorithm resistance properties of MuSig2.

The protocol ensures that MuSig2 private keys k_i never appear as inputs to ECIES encryption or BitVMX computation, maintaining the key independence required for the security reduction to the discrete logarithm problem. The aggregation coefficients $a_i = H_{agg}(K_i, \{K_1, \ldots, K_n\})$ depend only on public keys, preventing leakage of secret information through cross-component interactions.

On another hand, ECIES security depends on the computational Diffie-Hellman assumption and the semantic security of the underlying symmetric encryption scheme. The protocol composition maintains these guarantees by ensuring that ECIES encryption operates with independent key material that remains isolated from MuSig2 operations and BitVMX execution.

The key derivation process $\kappa_E \| \kappa_M = H(R, Z)$ where $Z = sk_1 \cdot R$ uses hash functions that operate independently of the hash functions employed in MuSig2 signature generation. This separation prevents potential hash collision attacks that might compromise the random oracle assumptions underlying both schemes simultaneously.

The protocol ensures that $Owner_1$'s ECIES private key sk_1 remains distinct from any MuSig2 private keys and never appears as input to BitVMX programs. This isolation maintains the CDH assumption by preventing adversaries from leveraging knowledge of MuSig2 signatures or BitVMX execution traces to gain advantage in solving ECIES decryption challenges.

Concerning BitVMX, its security relies on the challenge-response framework and the underlying hash function security for execution trace verification. The protocol composition maintains BitVMX's security model by ensuring that verification programs operate exclusively on public blockchain state rather than private cryptographic material from MuSig2 or ECIES operations.

The BitVMX program inputs consist of publicly verifiable blockchain transactions and state commitments, avoiding any dependence on private key material

or intermediate cryptographic values from other protocol components. The hash chain construction for execution trace verification $H(trace_i \| hash_{i-1})$ operates independently of the hash functions used in MuSig2 aggregation and ECIES key derivation.

The challenge-response mechanism maintains its security properties because honest operators can generate valid challenges without requiring access to secret information from other protocol components. The verification process depends solely on the computational soundness of the hash functions and the economic incentives for honest challenging behaviour.

The security composition succeeds because each primitive operates within its established threat model without requiring trust in the security of other components. The protocol design ensures that private key spaces remain disjoint across all three systems, preventing key reuse attacks that might compromise multiple components simultaneously. Hash function domains are separated to prevent collision attacks across different cryptographic contexts, while random number generation processes for each component operate independently, maintaining the randomness assumptions underlying each primitive's security analysis.

The 1-out-of-n security model provides robustness against partial system compromise by ensuring that honest operators can maintain security properties even when individual cryptographic components face attack. An honest operator will detect invalid MuSig2 signature requests, refuse to provide ECIES-encrypted shares to unauthorized recipients, and challenge invalid BitVMX executions, creating multiple independent security barriers.

5 Future research

This protocol involves several trade-offs related to scalability, communication overhead, and resilience against specific attack scenarios. These trade-offs, discussed below, also help define key directions for our future research.

5.1 Scalability

This protocol is designed with fixed on-chain costs, regardless of the number of operators involved. However, it faces challenges in managing computational complexity and communication overhead as the number of operators n increases. Indeed, concerning signature generation complexity, the protocol requires signature generation for each subset of operators, resulting in a total of 2^n signatures for n operators. This exponential growth limits practical application to scenarios where $n < 30$. While n remains small in most practical scenarios, the exponential growth could become a bottleneck in larger systems.

There exist several options to tackle the problem with the exponential growth in signature calculations, being one of them key aggregation; key can be aggregated into groups of up to a fixed size $N < n$. One observes that this approach reduces the computational complexity to $2^N \frac{n}{N}$ signature calculations for n operators, nevertheless, this optimization requires publishing $\frac{n}{N}$ on-chain group

signatures instead of 1, increasing the on-chain data requirements but substantially reducing computational costs. Another options would be considering all possible groups of size N, and for each group compute a multi-signature using the data published by operators O_i. At the unlocking phase, $Owner_1$ would receive communication of N operators with their secret keys. Once checked the validity of these keys, $Owner_1$ would be able to create the secret key associated to one of the possible subgroups. This approach reduces the computational complexity to $\binom{N}{n}$ signature calculations for n operators, significantly improving scalability while maintaining the integrity of the protocol.

When it comes to the communications overhead, communications between operators are on the order of $O(n^2)$, which may become a limiting factor for very large committees. As the number of operators grows, the increase in communication requirements could strain resources and delay operations. In any case, this is the same scalability requirements present in other parts of the system.

5.2 Attacks

This is a protocol designed to ensure security even in the presence of dishonest operators. However, certain attacks can still be attempted by malicious participants. Below, one outlines the possible attack vectors and the corresponding mitigation strategies to be explored in the future.

1. Misleading information to $Owner_1$: dishonest operators may send invalid or incorrect secret key information to $Owner_1$ during the unlocking phase. This could lead to delays or errors in the unlocking process.
   Mitigation: $Owner_1$ verifies all secret keys received against the public keys generated during the lock phase. Any invalid keys are discarded, ensuring that only valid keys are used for the unlocking process.
2. Refusal to collaborate: dishonest operators may refuse to participate in the unlocking process by withholding their secret keys from $Owner_1$.
   Mitigation: The protocol only requires one honest operator to participate for the unlocking to succeed. As long as at least one operator follows the protocol, the unlocking process remains functional.
3. Key sharing with unauthorized users: dishonest operators may share their secret keys with unauthorized users, enabling them to impersonate $Owner_1$ or gain access to the locked assets.
   Mitigation: $Owner_1$ independently verifies the validity of received keys. Additionally, since each operator's secret key is encrypted specifically for $Owner_1$ using ElGamal encryption, unauthorized users cannot use the keys without compromising the encryption.
4. Collusion between operators: operators may collude to manipulate the unlocking process by selecting a different group $\mathcal{ID}$ or bypassing $Owner_1$'s input. This could potentially allow them to access the locked assets improperly.
   Mitigation: An honest operator can challenge any unlocking attempt that deviates from the protocol by leveraging the BitVMX dispute mechanism.

This ensures that any unauthorized unlocking attempts are detected and prevented.

Acknowledgements The authors would like to thank Nicolas Biri, Torben Poguntke, Dimitar Jetchev, Thomas Vellekoop, Jesús Díaz Vico, and Romain Pellerin from Input Output Global, and Martin Jonas, Diego Tiscornia, Emilio García, Hernán Pérez Rodal, and Agustin Zaballa from Fairgate Labs for their insightful contributions to this research.

References

1. Abebe, Ermyas, et al. "Enabling enterprise blockchain interoperability with trusted data transfer (industry track)." Proceedings of the 20th international middleware conference industrial track. 2019.
2. Augusto, André, Rafael Belchior, Miguel Correia, André Vasconcelos, Luyao Zhang, and Thomas Hardjono. "Sok: Security and privacy of blockchain interoperability." In 2024 IEEE Symposium on Security and Privacy (SP), pp. 3840-3865. IEEE, 2024.
3. Aumayr, Lukas, et al. "BitVM: Quasi-Turing Complete Computation on Bitcoin." Cryptology ePrint Archive (2024).
4. Back, Adam, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. "Enabling blockchain innovations with pegged sidechains." URL: https://blockstream.com/sidechains.pdf (2014).
5. Borkowski, Michael, et al. "DeXTT: Deterministic cross-blockchain token transfers." IEEE access 7 (2019): 111030-111042.
6. Dangat, Shantanu, et al. "OTEx: Ownership Transfer and Execution Protocol for Blockchain Interoperability." 2024 IEEE International Conference on Blockchain (Blockchain). IEEE, 2024.
7. Ellis, Steven; Juels, Ari; Nazarov, Sergey (4 September 2017). "ChainLink A Decentralized Oracle Network". chain.link. 6 October 2017.
8. Galbraith, Steven D. *Mathematics of public key cryptography*. Cambridge University Press, 2012.
9. Garoffolo, Alberto, Dmytro Kaidalov, and Roman Oliynykov. "Zendoo: A zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains." 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2020.
10. Gazi, Peter, Aggelos Kiayias, and Dionysis Zindros. "Proof-of-stake sidechains." 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019.
11. van Glabbeek, Rob, Vincent Gramoli, and Pierre Tholoniat. "Cross-chain payment protocols with success guarantees." Distributed Computing 36.2 (2023): 137-157.
12. Goes, Christopher. "The interblockchain communication protocol: An overview." arXiv preprint arXiv:2006.15918 (2020).
13. Haugum, Terje, et al. "Security and privacy challenges in blockchain interoperability-A multivocal literature review." Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering. 2022.

14. Karantias, Kostis, Aggelos Kiayias, and Dionysis Zindros. "Proof-of-burn." Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24. Springer International Publishing, 2020.
15. Kiayias, Aggelos, and Dionysis Zindros. "Proof-of-work sidechains." Financial Cryptography and Data Security: FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23. Springer International Publishing, 2020.
16. Kwon, Jae, and Ethan Buchman. "Cosmos whitepaper." A Netw. Distrib. Ledgers 27 (2019): 1-32.
17. Network, Kyber, and R. Protocol. "Wrapped Tokens—A Multi-Institutional Framework for Tokenizing Any Asset." Wrapped tokens: A multi-institutional framework for tokenizing any asset",[online] Available: https://wbtc. network/assets/wrapped-tokens-whitepaper. pdf (2022).
18. Lerner, Sergio Demian, et al. "BitVMX: A CPU for Universal Computation on Bitcoin." arXiv preprint arXiv:2405.06842 (2024).
19. Linus, Robin, et al. "BitVM2: Bridging Bitcoin to Second Layers." (2024): 2024-11.
20. Liu, Weiwei, et al. "AucSwap: A Vickrey auction modeled decentralized cross-blockchain asset transfer protocol." Journal of Systems Architecture 117 (2021): 102102.
21. Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." Satoshi Nakamoto (2008).
22. Nick, Jonas, Andrew Poelstra, and Gregory Sanders. "Liquid: A bitcoin sidechain." Liquid white paper. URL https://blockstream. com/assets/downloads/pdf/liquid-whitepaper. pdf (2020).
23. Nick, Jonas, Tim Ruffing, and Yannick Seurin. "MuSig2: Simple two-round Schnorr multi-signatures." Annual International Cryptology Conference. Cham: Springer International Publishing, 2021.
24. Nolan, Tier, "Alt chains and atomic transfers, 2013". https://bitcointalk.org/index.php?topic=193281.0 (accessed May 26, 2024).
25. Pillai, Babu, et al. "The burn-to-claim cross-blockchain asset transfer protocol." 2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE, 2020.
26. Poon, Joseph, and Thaddeus Dryja. "The Bitcoin lightning network: Scalable off-chain instant payments." 14 Jan. 2016
27. Sober, Michael, et al. "Decentralized cross-blockchain asset transfers with transfer confirmation." Cluster computing 26.4 (2023): 2129-2146.
28. Wood, Gavin. "Polkadot: Vision for a heterogeneous multi-chain framework." White paper 21.2327 (2016): 4662.
29. Zamyatin, Alexei, et al. "Xclaim: Trustless, interoperable, cryptocurrency-backed assets." 2019 IEEE symposium on security and privacy (SP). IEEE, 2019.
30. Zamyatin, A. et al. (2021). SoK: Communication Across Distributed Ledgers. In: Borisov, N., Diaz, C. (eds) Financial Cryptography and Data Security. FC 2021. Lecture Notes in Computer Science(), vol 12675. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-64331-0_1

A Algorithms

Below follows a description in pseudocode of all the algorithms involved in the transfer of ownership protocol.

Algorithm 7 Generate Operator Powerset

Require: Set of operators $O = \{O_1, O_2, \ldots, O_n\}$ where $n > 1$
Ensure: Powerset $P(O)$ containing all possible subsets of O

function GenCombinations($elems, size$)
  if $size = 0$ **or** $size = |elems|$ **then return** $size = 0?[\emptyset] : [elems]$
  end if
  $combs \leftarrow \emptyset$, $first \leftarrow elems[0]$, $rest \leftarrow elems[1:]$
  for all $smallComb \in$ GenCombinations($rest, size - 1$) **do** $combs \leftarrow combs \cup \{[first] + smallComb\}$
  end for
  $combs \leftarrow combs \cup$ GenCombinations($rest, size$) **return** $combs$
end function
function GenOperatorPowerset(ops)
  if $|ops| < 2$ **then return** "Error: Need at least 2 operators"
  end if
  $pset \leftarrow \emptyset$, $opList \leftarrow$ list(ops)
  for $subSize \leftarrow 0$ **to** $|opList|$ **do**
    $pset \leftarrow pset \cup$ GenCombinations($opList, subSize$)
  end forreturn $pset$
end function

Algorithm 8 Key Generation for Locked Assets

Require: Set of locked Bitcoin assets A and operators $O = \{O_1, O_2, \ldots, O_n\}$
Ensure: Dictionary mapping each operator to their key pairs for each asset

function GenKeys($assets, ops, G, q$)
  $aKeys \leftarrow \emptyset$
  for each $a \in assets$ **do**
    $opKeys \leftarrow \emptyset$
    for each $op \in ops$ **do**
      $k_i \leftarrow$ RandSecNum$(1, q-1)$, $K_i \leftarrow$ ScalMult(k_i, G)
      $opKeys[op] \leftarrow \{$"$priv$" $: k_i,$ "pub" $: K_i\}$
    end for
    $aKeys[a] \leftarrow opKeys$
  end forreturn $aKeys$
end function

Algorithm 9 Aggregate Address Generation

Require: Committee of operators $O = \{O_1, O_2, \ldots, O_N\}$, $Owner_0$, $Owner_1$
Ensure: Aggregate Bitcoin address and shared operator keys

function GenAggAddr(ops, own_0, own_1)
  $opKeys \leftarrow \emptyset$ ▷ Phase 1: Initial Key Generation
  for each $op \in ops$ **do**
    $secKey \leftarrow$ GenSecRand, $pubKey \leftarrow$ GenPubKey($secKey$)
    $opKeys[op] \leftarrow \{"sec" : secKey, "pub" : pubKey\}$
  end for
  $allPubKeys \leftarrow [keys["pub"]$ for $keys$ in $opKeys.values()]$
  $aggPubKey \leftarrow$ CompAggKey($allPubKeys$)
  $aggAddr \leftarrow$ PubKeyToAddr($aggPubKey$)
  $ownInfo \leftarrow \emptyset$
  $ownInfo[own_0] \leftarrow \{"addr" : aggAddr, "ops" : ops, "aggKey" : aggPubKey\}$
  $ownInfo[own_1] \leftarrow \{"addr" : aggAddr, "ops" : ops, "aggKey" : aggPubKey\}$
  $opInfo \leftarrow \emptyset$
  $opInfo["addr"] \leftarrow aggAddr$
  $opInfo["keys"] \leftarrow opKeys$
  $opInfo["aggKey"] \leftarrow aggPubKey$
  $opInfo["owners"] \leftarrow [own_0, own_1]$
  return $\{"addr" : aggAddr, "ownInfo" : ownInfo, "opInfo" : opInfo\}$
end function
function CompAggKey($pubKeys$) ▷ Compute aggregate public key using MuSig2
end function
function PubKeyToAddr($pubKey$) ▷ Convert public key to Bitcoin address
end function
function GenSecRand ▷ Generate cryptographically secure random number
end function
function GenPubKey($secKey$) ▷ Generate public key from secret key
end function

Algorithm 10 Create Lock Request Transaction

Require: Owner address own_0, assets A, operator aggregate address $opAddr$, protocol fees $fees$, second chain address sc_addr
Ensure: Lock request transaction structure

function CreateLockReq($own_0, A, opAddr, fees, sc\_addr$)
  $lockReq \leftarrow \{\}$
  $lockReq[\text{"}from\text{"}] \leftarrow own_0$, $lockReq[\text{"}to\text{"}] \leftarrow opAddr$, $lockReq[\text{"}locked\_assets\text{"}] \leftarrow A$
  $lockReq[\text{"}fees\text{"}] \leftarrow \{\text{"}btc\_fee\text{"} : fees[\text{"}btc\_fee\text{"}], \text{"}sc\_fee\text{"} : fees[\text{"}sc\_fee\text{"}]\}$
  $lockReq[\text{"}sc\_dest\text{"}] \leftarrow sc\_addr$
  $lockReq[\text{"}cond\text{"}] \leftarrow \{$ "op_redeem" : {"req" : "all_op_sigs","$timelock$" : $NULL$}, "own_redeem" : {"req" : "own_sig","$timelock$" : "rel_blocks_timeout"}}
  return $lockReq$
end function

Algorithm 11 Operator Setup Protocol

Require: Set of operators ops, powerset P, lock request req
Ensure: Setup components for BitVMX

function PrepOpSetup(ops, P, req)
  $setup \leftarrow \{\text{"}tx\text{"} : \{\}, \text{"}secrets\text{"} : \{\}, \text{"}keys\text{"} : \{\text{"}ind\text{"} : \{\}, \text{"}agg\text{"} : NULL\}\}$
  for each $subset \in P$ **do**
    $subId \leftarrow$ _join(sort([$op.id$ for $op \in subset$]))
    $setup[\text{"}tx\text{"}][subId][\text{"}ops\text{"}] \leftarrow subset$
    for each $op \in ops$ **do**
      $setup[\text{"}tx\text{"}][subId][\text{"}txs\text{"}][op.id] \leftarrow$ "presigned_tx_for_" + subId
    end for
  end for
  $setup[\text{"}secrets\text{"}] \leftarrow \{op.id$: "secret_for_" + op.id for $op \in ops\}$
  return $setup$
end function

Algorithm 12 Transfer Protocol

Require: Owners own_0, own_1, operators ops, powerset P
Ensure: Transfer protocol execution

function CreateBurnTx($own_1, wrap\_asset, contract$)
  $burn \leftarrow \emptyset$
  $burn[\text{"}asset\text{"}] \leftarrow wrap\_asset$
  $burn[\text{"}action\text{"}] \leftarrow$ "burn"
  $burn[\text{"}redeem\_key\text{"}] \leftarrow own_1.ecies\_pub$
  $burn[\text{"}time\text{"}] \leftarrow$ CurrTime
  return $burn$
end function
function PrepEncShares($ops, own_1\_pub, utxo$)
  $enc\_data \leftarrow \emptyset$
  for each $op \in ops$ **do**
    $enc\_share \leftarrow$ ECIESEnc($own_1\_pub, op.key\_share, op.ecies\_priv$)
    $enc\_data[op.id] \leftarrow \{\text{"}enc\_share\text{"} : enc\_share, \text{"}pub\_key\text{"} : op.ecies\_pub\}$
  end for
  return enc_data
end function
function VerifySelectGroup(own_1, enc_shares, P)
  $valid\_ops \leftarrow \emptyset$
  for each $(op\_id, share\_data) \in enc\_shares$ **do**
    $dec\_share \leftarrow$ ECIESDec($own_1.ecies\_priv, share\_data[\text{"}enc\_share\text{"}]$)
    if VerifyShare(dec_share, share_data["pub_key"]) **then**
      $valid\_ops \leftarrow valid\_ops \cup \{op\_id\}$
    end if
  end for
  for each $group \in P$ **do**
    if $group = valid\_ops$ **then return** GroupToId($group$)
    end if
  end for
  return $NULL$
end function
function CreateSelMsg($group\_id$)
  return $\{\text{"}group\_id\text{"} : group\_id, \text{"}proof\text{"} :$ GenProof($group\_id$)$\}$
end function

Algorithm 13 Interactive Burn Protocol

Require: Constants: MAX_CERTS, MIN_SGRS, MAX_SGRS

function ValidCertChain($snap, burn\_block$)
  $chain\_val \leftarrow$ {"$snap$" : {"$height$" : $snap.height$,"agg_sig" : $ops.agg\_sig$,"$init\_state$" : $snap.state$}, "$cert\_chain$" : {"max_len" : MAX_CERTS,"$certs$" : []}, "$target$" : {"$height$" : $burn\_block.height$,"$cert$" : $burn\_block.cert$}}
  return $chain\_val$
end function
function CreateProverInput($cert\_chain$)
  $prover \leftarrow$ {"$cert\_hashes$" : [$sha256(cert)$ for $cert \in cert\_chain$], "unk_cert" : {"$sgrs$" : {"$count$" : $|cert\_chain.sgrs|$,"pub_keys" : $cert\_chain.sgr\_keys$,"agg_sig" : $cert\_chain.bls\_sig$}, "$merkle$" : {"$root$" : $cert\_chain.merkle\_root$,"$path$" : $cert\_chain.incl\_proof$}}}
  return $prover$
end function
function CreateVerifierChallenge($cert\_chain$)
  $verifier \leftarrow$ {"$chal\_opts$" : { "sgr_incl" : {"$type$" : "SGR_VERIFY","sgr_idx" : $NULL$}, "sig_valid" : {"$type$" : "BLS_SIG","$chal\_data$" : $NULL$}, "blk_incl" : {"$type$" : "BLK_CERT","blk_data" : $NULL$}}, "unk_cert" : {"seq_num" : $NULL$,"$snap\_ref$" : $NULL$}}
  return $verifier$
end function
function ExecProofVerification($prover\_in, verifier\_chal$)
  $proof \leftarrow$ {"$comps$" : {"$cert\_chain$" : $prover\_in.cert\_hashes$, "$chal\_resp$" : {"$type$" : $verifier\_chal.type$,"$proof\_data$" : $NULL$}}, "$constr$" : {"$chain\_len$" : MAX_CERTS,"sgr_count" : {"min" : MIN_SGRS,"max" : MAX_SGRS}}}
  return $proof$
end function