Bitcoin Smart Accounts: Trust-Minimized Native Bitcoin DeFi Infrastructure

Cian Lalor
Lombard Finance

Matthew Marshall
Lombard Finance

Antonio Russo
Lombard Finance

March 26, 2026

Abstract

Bitcoin's limited programmability and transaction throughput have historically prevented native Bitcoin from participating in decentralized finance (DeFi) applications. Existing solutions depend on honest-majority thresholds, or centralized custodial entities that introduce significant trust requirements. This paper introduces Bitcoin Smart Accounts (BSA), a novel protocol that enables native Bitcoin to access DeFi through trust-minimized infrastructure while maintaining self-custody of funds.

BSA achieves this through a combination of emulated Bitcoin covenants using Partially Signed Bitcoin Transactions (PSBTs) and Taproot scripts, a Trusted Execution Environment (TEE)-based arbitration system, and destination chain smart contracts that enable DeFi platforms to accept self-custodial Bitcoin as collateral without necessitating protocol-level modifications. The setup leverages liquidity secured by the Lombard Security Consortium which provides a twofold advantage: for a DeFi protocol, liquidators rely on fungible assets with deep liquidity to quickly exit positions, while for a depositor, the general trust assumptions of honest majority (m-of-n) are reduced to existential honesty (1-of-k).

We present the complete protocol design, including the Bitcoin architecture, the TEE-based arbitration mechanism, and the Smart Account Registry for protocol management. We provide a security analysis that demonstrates the correctness, safety, and availability properties under our trust model. Our design enables native Bitcoin to serve as collateral in lending markets and other DeFi protocols without requiring users to relinquish custody of funds.

Keywords: Bitcoin, Decentralized Finance, Trust-Minimized Bridges, Emulated Covenants, Trusted Execution Environments, Partially Signed Bitcoin Transactions, Taproot, Nitro Enclaves

1 Introduction

Bitcoin [1] has emerged as the dominant cryptocurrency by market capitalization, with more than $1 trillion in value locked in the network. However, Bitcoin's conservative design philosophy, which prioritizes security and decentralization, has resulted in limited transaction throughput and programmability. These constraints have prevented Bitcoin from participating in the rapidly growing decentralized finance (DeFi) ecosystem, where programmable smart contracts enable lending, borrowing, trading, and other financial primitives.

1.1 The Problem

To access DeFi functionality, Bitcoin holders have historically faced a fundamental trade-off: either trust a custodial service to hold their Bitcoin to issue fungible tokens, or use bridges that

require trusting an honest majority of operators (m-of-n threshold schemes). Both approaches introduce significant risk that undermines Bitcoin's core value proposition of self-custody and trust minimization.

Observed limitations of existing bridges and Bitcoin Layer 2 solutions include:

- **Custodial Risk:** Most wrapped Bitcoin solutions require users to deposit funds into centralized custody, creating single points of trust and failure, introducing excessive counterparty risk.
- **Trust Assumptions:** Threshold signature schemes require trusting that a majority of signers remain honest, which highlights an attack vector for adversarial actors. Current implementations regularly rely on complex key management (adding or removing signers, key rotation).
- **Novel Cryptography:** Recent advances in Bitcoin bridging solutions are based on succinct knowledge proofs implemented over garbled circuits [2]. These cryptographic primitives are novel, and complex to audit and guarantee correctness.
- **Limited Programmability:** Bitcoin's intentionally limited scripting language, and present lack of native covenant support, prevent the enforcement of complex spending conditions directly on the Bitcoin network.
- **Integration Challenges:** Existing solutions often create friction for DeFi protocols, involving liquidity delays and bespoke infrastructure for the Bitcoin network, particularly around liquidation mechanisms that are critical for lending markets, e.g., Babylon Bitcoin Vaults [2].

1.2 Our Contribution

This paper introduces **Bitcoin Smart Accounts (BSA)**, a protocol built using existing and well-established primitives that enables native Bitcoin to access DeFi applications while maintaining self-custody and minimizing trust assumptions. Our contributions include:

1. **Trust-Minimized Security Model:** We reduce the depositor's (the user bridging Bitcoin) trust assumptions from an honest majority (m-of-n) to existential honesty (1-of-k), where a depositor requires a minimum of only one other honest party in the protocol to be safe.
2. **TEE-Based Arbitration Oracle:** We introduce a Trusted Execution Environment (TEE)-based arbitration system (Sec. 5) that verifies cross-chain events and resolves disputes. The TEE ensures code integrity and provides cryptographic attestations of correct execution, enabling the arbitration logic to be executed by any third party.
3. **Emulated Covenant Architecture:** We design a Bitcoin script architecture using Taproot addresses [3] and Partially Signed Bitcoin Transactions (PSBTs) to emulate covenant-like functionality, i.e., transaction constraints enforced with a committee, without requiring Bitcoin consensus changes—an approach also used in Lightning Network [4] and BitVM [5]. This enables complex, multi-party spending conditions to be enforced without relying on a trusted third party.
4. **DeFi-Native Integration:** We design the Smart Account Registry, a smart contract that enables seamless integration with existing DeFi protocols, supporting standard liquidation mechanisms without creating impediments for liquidators or lending protocols by leveraging existing on-chain liquidity through the use of Lombard's existing Security Consortium (a

collection of validators requiring 2/3 consensus—see footnote marker * on Table. 1 for further detail).

5. **Security Analysis:** We provide security evaluations that demonstrate the safety, liveness, and integrity properties under our trust model (Sec. 6).

1.3 BSA Comparison to Existing Bridging Solutions

Trustless approaches such as BitVM [5] achieve the strongest theoretical security guarantees by verifying arbitrary computations on Bitcoin through fraud proofs. However, they introduce significant operational complexity, including multi-round interactive protocols, large on-chain footprints, and requirements for DeFi applications to integrate directly with the Bitcoin network, making them impractical for general-purpose DeFi use cases today. BSA takes a pragmatic approach: by combining emulated covenants with TEE-based arbitration, it achieves comparable security to fully trustless vaults, specifically 1-of-k existential honesty and self-custody, while being deployable today with significantly lower operational complexity and native compatibility with existing DeFi infrastructure.

Below, we provide a table 1 to compare the key properties of our contribution with related works.

Property	**Bitcoin Smart Accounts**	**Multisig Bridge**	**BitVM Bridge**	**Centralized Issuer (wrapped BTC)**
Trust Model	1-of-k	m-of-n*	1-of-k	1-of-1
Self-Custody	Yes	No	Yes	No
Native DeFi support (e.g., liquidations)	Yes, with programmable perimeter†	Yes	No, requires DeFi applications to integrate with Bitcoin network	Yes
Decentralized withdrawal of native BTC by depositor	Yes, assumes 1-of-$(k+1)$‡ honest.	Yes, assumes m-of-n honest.	Yes, assumes 1-of-k honest.	No. Requires permissions from centralized authority.
Required Operators to steal BTC from depositor	k-of-k offline AOs and dishonest TO‡	m-of-n operators	k-of-k operators	1 single operator
Withdrawal of native BTC by non-depositor	Indirect via TO rebalance	Yes	Indirect via Operator	Yes

Table 1: Comparison of Bitcoin Smart Accounts with related work.

** Lombard's Security Consortium (the TO), a multisig bridge, includes additional safeguards to reduce the trust model further, such as the integration of Cubist's Bascule* [6] *and Chainlink oracles* [7] *to provide a secondary validation for all fund movements during normal operation.*
† The BSA enables seamless liquidations without liquidator- or DeFi- customizations. For more information, see section 4.3.
‡ An AO cannot be dishonest, only offline. When considering attack scenarios, we view the protocol to have 1-*of-*$(k+1)$ *security, given that the protocol consists of k AOs and a single TO. This is explained in more detail in section 6.2.*

1.4 Paper Organization

The remainder of this paper is organized as follows. Section 2 presents our system model, assumptions, and security goals. Section 3 details the complete protocol design, including the Bitcoin script architecture, PSBT mechanisms, and state transitions. Section 4 covers the Smart Account Registry and the components on the destination chain. Section 5 describes the Arbitration Oracle TEE implementation. Section 6 provides security analysis. Section 7 discusses limitations and future work. Section 8 concludes. The appendices outline additional key implementation details.

2 System Model and Security Goals

2.1 Network Model

We assume that Bitcoin and the destination chain (e.g., Ethereum) operate as robust public ledgers with persistence and high-availability properties. We make standard cryptographic assumptions: participants are computationally bounded, and standardized hash functions, signatures, and encryption schemes are utilized.

2.2 Participants

The BSA protocol involves the following participants:

- **Depositor (D):** A user who wants to utilize their native Bitcoin within DeFi applications. The depositor maintains control of their Bitcoin and destination chain private keys and participates in the Protocol User Setup Ceremony.
- **Token Operator (TO):** A single acting party responsible for minting BTC.b[1] and executing Bitcoin rebalances e.g., for DeFi liquidations. In practice, this is Lombard's existing Security Consortium, consisting of the existing set of validators that secures Lombard through 2/3 majority consensus. This actor can be viewed as the *canonical operator* for protocol instantiation: a requirement for an instance of the BSA to exist.
- **Arbitration Oracles (AO):** A software module that verifies protocol states and resolves disputes. A set of k arbitration oracles may exist, and the depositor requires at least one honest arbitration oracle to avoid trusting only the TO. The k AOs are interchangeable for a given protocol instance: any correct AO (see below Def. 2.1) included in the depositor's BSA instance setup can resolve its disputes.

2.3 Failure Model

The depositor and the Token Operator are Byzantine parties assumed to be selfish and rational, i.e. they will do anything which allows them to gain an economic advantage, and they do not harm themselves.

The Arbitration Oracle may fail due to lack of availability, i.e. it may not fulfill time-sensitive duties. However, it cannot act outside the protocol's design constraints or produce inaccurate results. Multiple independent entities can play the Arbitration Oracle role in the same protocol instance in such a way that the Arbitration Oracle duties are fulfilled as long as at least one performs within the protocol time constraints.

[1] BTC.b is a Bitcoin derivative minted by the Lombard Security Consortium, which is backed one-to-one by native Bitcoin.

Definition 2.1 (AO Correctness)**.** An Arbitration Oracle is *correct* if it is online and operational, i.e., it is available to verify protocol states and sign transactions resolving disputes in favor of the depositor, according to the deterministic dispute resolution logic as outlined in Sec. 5.4.

An AO operator (entity running the AO software) may be malicious, but due to the TEE-enforced code integrity (Sec. 6.5) and emulated covenant constraints (Sec. 3.4), the most severe adversarial action available to a malicious AO operator is to take the AO offline. A malicious AO operator cannot sign incorrect transactions or redirect funds to unauthorized addresses, as all spending paths are constrained by pre-signed PSBTs to commit to specific output addresses.

Whenever a party behaves according to the protocol rules, it is said to be *correct*. For the AO, correctness is equivalent to availability, as a malicious AO operator cannot violate protocol rules due to TEE and covenant constraints.

2.4 Security Goals

We define the security properties of the protocol according to the perspective of each participating party.

Definition 2.2 (Protocol Liveness)**.** A protocol is said to have *liveness* if funds cannot be prevented from exiting the protocol after they are transferred to the address VA, as defined in Sec. 3.3.1.

Definition 2.3 (Depositor Safety)**.** A protocol instance is *Depositor Safe* if there exists a finite and time bounded list of operations that allows a correct depositor to regain full and unilateral control of its funds on the Bitcoin blockchain.

Definition 2.4 (Token Operator Safety)**.** A protocol instance is *Token Operator Safe* if in any protocol state where the corresponding token collateralized by Bitcoin deposits is minted on the destination chain, there exists a finite and time-bounded list of operations that makes the Token Operator gain full and unilateral control of funds on the Bitcoin blockchain when collateral tokens exit the protocol perimeter as defined in Sec. 4.2.2.

Definition 2.5 (Protocol Safety)**.** A protocol instance is *Protocol Safe* if and only if both Depositor Safety 2.3 and Token Operator Safety 2.4 hold.

Given the Failure Model described in section 2.3, the general security goal of this work is to guarantee Protocol Safety 2.5 under the assumption that at least 1-of-k entities playing the Arbitration Oracle role are correct.

As a consequence, if the TO is malicious, the depositor must trust that at least 1-of-k AOs remain honest. This is a significant improvement over traditional m-of-n threshold schemes.

The protocol description following in the next section will make apparent that from the depositor's point of view, for k AOs, the 1-of-k requirement reduces to 1-of-$(k+1)$ since the honest behavior of the TO does not require any intervention of AO.

3 Bitcoin Architecture

3.1 Overview

The BSA protocol enables a depositor to lock native Bitcoin in a self-custodial vault while receiving BTC.b receipt tokens on a destination chain for use in DeFi applications. The protocol uses four Taproot script addresses with specific spending conditions, enforced through pre-signed PSBTs exchanged during the *Protocol User Setup Ceremony* (Sec. 3.4.1).

3.2 Protocol Parameters

The BSA protocol uses the following time lock parameters:

- T_1: The relative time lock period during which the TO can challenge a withdrawal, or unbonding, request. After T_1 expires without challenge by the TO, the depositor can unilaterally claim funds.
- T_2: The relative time lock period for any AO to resolve a state in favor of the depositor. After expiration, the funds can be pulled by the TO.
- T_3: The relative time lock period for any core protocol modifications to take effect (e.g., upgrading the destination chain smart contracts), which may impact the security of the protocol. Hence, all security guarantees (see Sec. 6.3) of the protocol are upheld until T_3.

where the time lock relation:

$$T_3 > (T_1 + T_2) \tag{1}$$

ensures correctness. The depositor and an AO can verify this relation before depositing Bitcoin.

3.3 Bitcoin Script Architecture

We design four Taproot script addresses [3] that form the core architecture on the Bitcoin blockchain. Taproot scripts were chosen for their reduced transaction size and fees, and ability to enable smart contract-like spending logic.

Every user has their own suite of unique script addresses, directly tied to both their destination chain address and Bitcoin network public key. The usage of PSBTs exchanged by the parties allows funds to be directed between the addresses of the protocol in a verifiable manner. Verifiability is achieved through a public code repository which outlines formulae for address derivation. Appendix A provides more details on the topic.

3.3.1 Script Addresses

The four Taproot script addresses represent states within the protocol. Funds move through these addresses according to the protocol rules. A user's deposit may contain multiple UTXOs, which can be held at any of a user's set of unique addresses while the protocol is live. An instance of the protocol is considered to be live if these UTXOs are locked to any one of the four addresses:

1. **Vault Address (VA):** a 2-of-2 multisig between a single depositor and the TO. This is where the depositor initially locks their Bitcoin.
2. **Unbond Timelock Address (UTA):** a 2-of-2 multisig between a single depositor and TO, with an additional 1-of-1 spending path for the depositor after the timelock T_1 expires.
3. **Unbond Challenge Address (UCA):** a 2-of-2 multisig between a single depositor and a set of AOs, with an additional 1-of-1 spending path for the TO after timelock T_2 expires.
4. **Rebalance Challenge Address (RCA):** a 2-of-2 multisig between a single depositor and a set of AOs, with an additional 1-of-1 spending path for the TO after timelock T_2 expires.

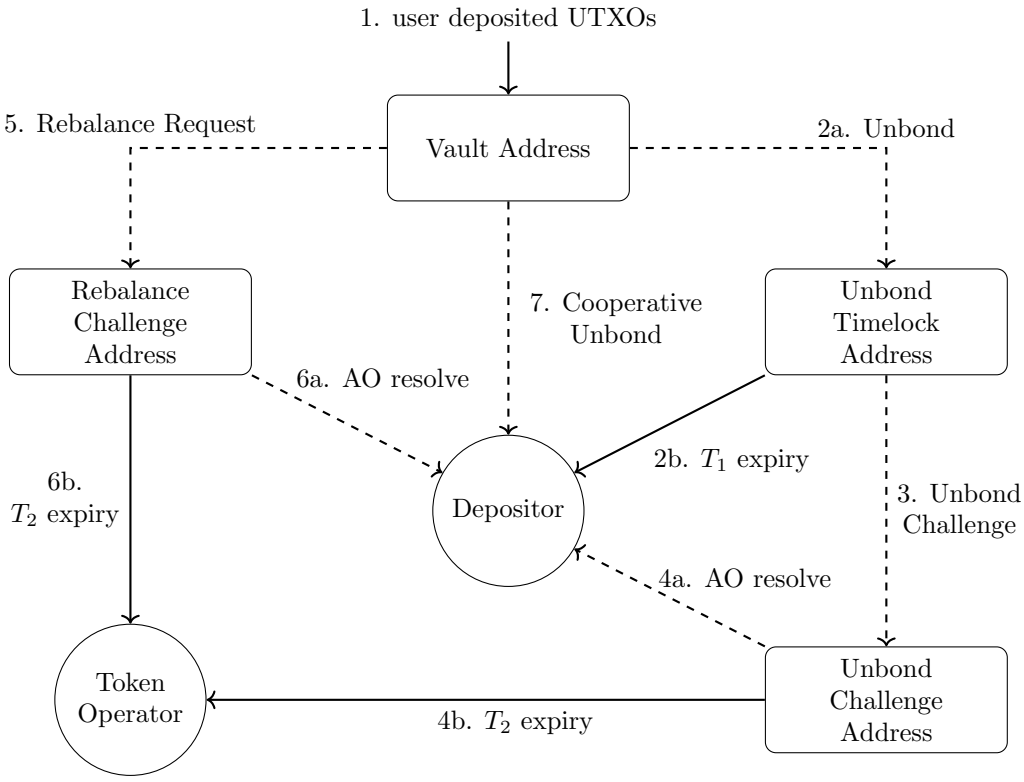

Figure 1: State transition diagram showing the vault unbond paths. We use boxes to represent addresses with spending paths by two or more members, and circles to represent addresses controlled by an individual actor. Terms in diagram reference state transitions as enumerated in section 3.5. Executors of state transitions can be seen in section 3.4.1. Dotted lines reference 2-of-2 spending paths, solid lines reference 1-of-1 spending paths.

In order to map transactions to corresponding state transitions, each taproot address has its internal key tweaked by an address-specific label. In this way, each of the listed Bitcoin addresses is unique with respect to an instance of the protocol.

The internal key of all Taproot addresses is a Nothing Up My Sleeves (NUMS) key, according to BIP-0341 [3]. See Appendix A for more information.

3.4 PSBT-Based Covenant Emulation

Bitcoin Script does not natively support covenants (constraints on transaction outputs). Native covenant support would require opcodes such as `OP_CHECKTEMPLATEVERIFY` [8] that enable scripts to inspect and constrain transaction outputs. We emulate covenant functionality by combining multi-signature spending constraints with pre-signed transactions that commit to specific outputs. During Protocol User Setup Ceremony (Sec. 3.4.1), the TO, AO, and depositor exchange a set of PSBTs. These PSBTs commit to specific spending paths and output addresses, effectively constraining future transaction outputs. The key insight is that by pre-signing transactions with fixed outputs, we can enforce spending conditions that would otherwise require native covenant support. This approach builds upon the techniques used by other protocols, e.g., the Lightning Network [4].

A covenant emulation is established between two parties: the party who creates and provides a

PSBT (the *creator*), which pre-commits transaction outputs to specific destination addresses, and the party who receives the PSBT (the *executor*), who can sign and execute the transaction using the inputs and outputs specified in the PSBT received by the creator.

In a valid protocol instance, all parties have signed and provided the expected PSBTs. Addresses in PSBT outputs commit to the expected spending scripts.

3.4.1 Protocol User Setup Ceremony

Bitcoin's UTXO model enables precise control over which specific UTXOs are spent. For each UTXO deposited into the VA, the following set of PSBTs are created and exchanged for the *Protocol User Setup Ceremony* to start a valid protocol instance:

1. The depositor receives a PSBT from the TO enabling them to initiate an unbonding request (moving funds from VA to UTA) unilaterally.
2. The TO receives PSBTs from the depositor, enabling them to unilaterally:
   (a) Challenge depositor unbonding requests (moving funds from UTA to UCA).
   (b) Initiate rebalancing requests (moving funds from VA to RCA).
3. The AO receives PSBTs from the depositor enabling them to unilaterally resolve:
   (a) Challenged unbonding requests in favor of the depositor (moving funds from UCA to Dep).
   (b) Rebalance requests in favor of the depositor (moving funds from RCA to Dep).

The number of PSBTs required from the depositor can be reduced through optimization techniques, such as having the TO provide PSBTs for certain spending paths, though this extends trust assumptions. For a complete overview of transactions, please refer to the table in Appendix B.

3.4.2 Dynamic PSBT Updates

While the initial PSBT exchange occurs during protocol setup, additional PSBTs can be added to the protocol while it is live to optimize workflow and enable new functionalities, such as a fast unbond from the Vault in case the TO cooperates and the depositor behaves correctly (see **Cooperative Unbond** below).

3.5 State Transitions

The protocol supports the following state transitions:

1. **Deposit:** Depositor sends Bitcoin to VA. TO observes the deposit, and if protocol is considered valid according to section 3.4, mints the corresponding BTC.b on the destination chain.
2. **Unbond Request:** The depositor burns BTC.b on the destination chain and then (a) initiates an unbonding transaction from VA to UTA, and if the TO does not challenge, then (b) after time lock T_1 expires, the depositor pulls the funds from UTA.
3. **Unbond Challenge:** If TO observes an unbonding request but no corresponding burn event, TO challenges by moving funds from UTA to UCA.

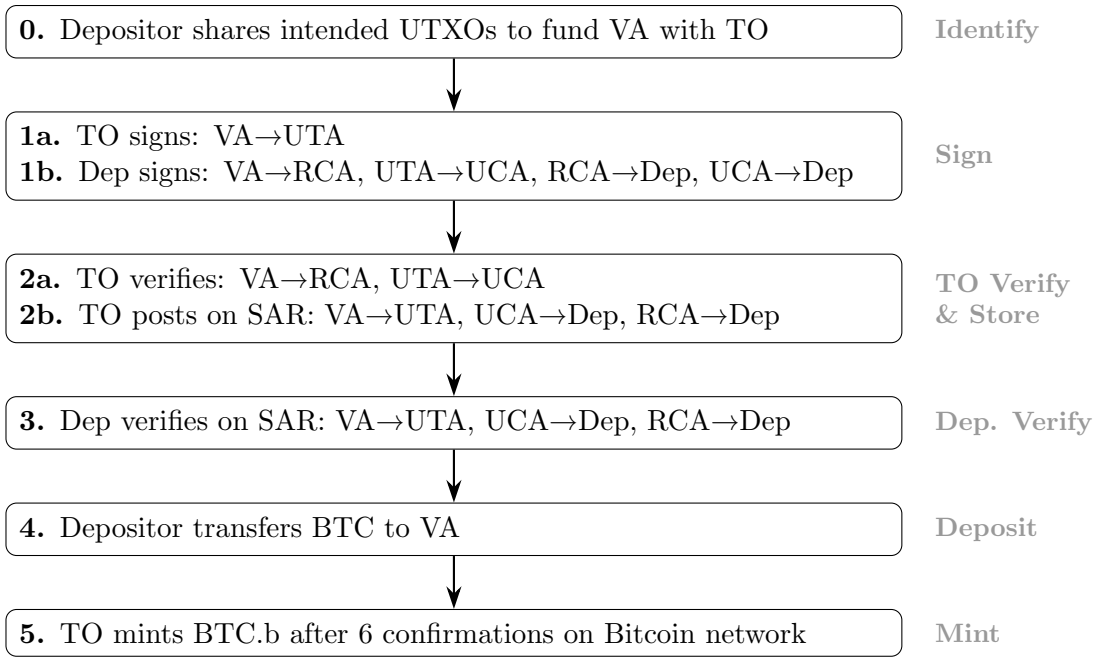

Figure 2: Protocol User Setup Ceremony. PSBTs are exchanged and signed, posted to the SAR (Sec. 4), and independently verified before the depositor funds the Vault Address and the TO mints BTC.b at the corresponding destination address.

4. **Unbond Challenge Resolution:** AO verifies the destination chain state and either: (a) resolves the dispute in favor of the depositor, moving funds from UCA to the depositor's address, or (b) chooses inaction, effectively resolving the dispute in favor of the TO, enabling the TO to pull funds from UCA after T_2.

5. **Rebalance Request:** TO observes an under-collateralized position and triggers a rebalance event on the destination chain, moving funds from VA to RCA.

6. **Rebalance Finalization:** AO verifies the destination chain state and either: (a) resolves the rebalance request in favor of the depositor, moving funds from RCA to the depositor's address, or (b) chooses inaction, effectively finalizing the rebalance in favor of the TO, enabling the TO to pull funds from RCA after T_2.

7. **Cooperative Unbond:** *this state transition is not strictly necessary for protocol functionality, and is introduced only to improve user experience.* If a user requests the TO to verify that they may legitimately unbond (i.e., have already burnt BTC.b), the TO will sign an additional PSBT enabling a spending path from the vault directly to the depositor's return address, circumventing the required time lock for manual unbonds.

   Note: There is no further reference to Cooperative Unbond in this paper as it does not impact participants' trust or protocol security.

As outlined in section 3.3.1, a depositor's instance of the protocol contains four unique Bitcoin addresses. A depositor can deposit multiple UTXOs to an instance of the BSA, effectively splitting funds across UTXOs which can move independently across protocol states. Splitting funds is beneficial for rebalance and unbond mechanisms, e.g., enabling partial liquidations (see Sec. 3.6.2).

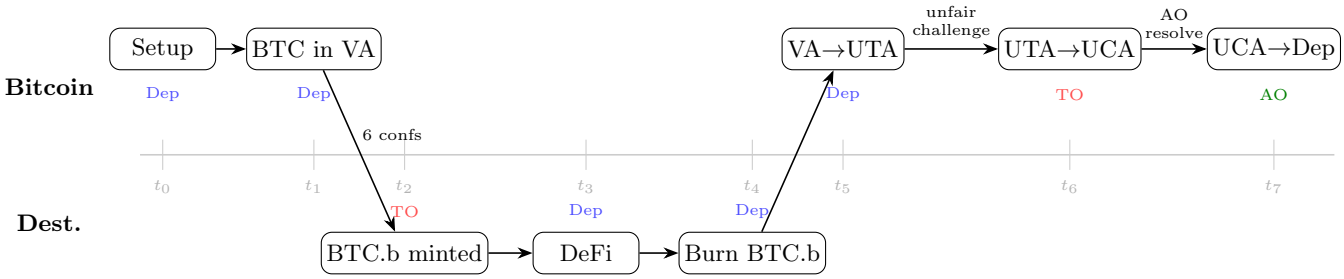

Figure 3: Protocol lifecycle timeline: onboarding through to offboarding with an illegitimate challenge.

3.6 Rebalance Models

When the TO is honest, the *Rebalance Request* is a reactionary state transition triggered by the TO whenever an arbitrary amount of BTC.b tokens exits the protocol perimeter as defined in Sec. 4.2.2. Given this amount may be arbitrary, it will likely not correspond to the specific UTXO amounts locked in the VA for a given user's instance of the BSA.

To overcome this misalignment, we propose two rebalancing models to balance user experience and protocol flexibility.

Note: When the TO is dishonest, this state transition will be corrected by an AO (see Sec. 6.1).

3.6.1 Model 1: Collaborative Rebalancing

A user deposits a single UTXO in the VA. When a rebalance is required, the TO requests a new set of PSBTs from the depositor within a time window, enabling new protocol states that reflect the amount to be rebalanced from the original deposit and the remaining amount. If the depositor fails to sign within the agreed-upon time window, the TO rebalances the full amount of the deposited UTXO to the RCA.

3.6.2 Model 2: UTXO-Based Rebalancing

A user transfers multiple UTXOs to the same VA, and a corresponding amount of tokens are minted to the destination address. The number of UTXOs determines the granularity of rebalancing. When a rebalance occurs, the TO moves sufficient UTXOs from VA to RCA to cover the imbalance. As PSBTs, signed during the Protocol User Setup Ceremony, commit to amounts in deposited UTXOs, it is not feasible to partially liquidate any whole UTXOs.

Importantly, in both models, for any over-seized amount, the depositors' DeFi position(s) remains active. The only effective change is that part of the underlying collateral backing said position is transferred from the depositor's self-custodial BSA vault to the LSC's general reserves: any amount of over-seized collateral does not impact the fungibility of the corresponding on-chain BTC.b. Any amount remaining in the VA is unchanged, and still fully collateralizes a DeFi position as before a rebalance request. Consequently, when the depositor wishes to unbond they need to claim any over-seized amount from the TO in the event the required rebalance does not match the user's exact deposited amounts.

3.7 Fee Management

In the BSA protocol, without proper fee management, UTXOs moving through state transitions would "consume themselves" to pay for transaction costs. However, as PSBTs are pre-signed during the Protocol User Setup Ceremony, and since these signatures commit to both inputs and outputs, fees cannot be updated if network conditions change and the preset fees become insufficient. This introduces a problem: if a pre-signed transaction's fee becomes too low relative to current network conditions, the UTXO could become stuck and the protocol would not progress.

To prevent the protocol from being unable to execute a state transition, we use a combination of *Anchor Outputs* [9] and *Child-Pays-For-Parent (CPFP)* transactions [10], along with *SigHash* flags [3].

Both the mechanisms outlined below in the remainder of this section allow the protocol to avoid committing to unnecessarily high fees during the Protocol User Setup Ceremony, while ensuring transactions can always be executed and confirmed, independent of network conditions.

3.7.1 Anchor and CPFP Transactions

For all state transitions involving a covenant emulation between the TO and the depositor, the *creator* includes an additional anchor output which the *executor* can spend unilaterally.

If a pre-signed transaction tx_0 is broadcast but not confirmed due to insufficient fees ($f_0 < f_{\text{required}}$), the executor can construct a CPFP transaction tx_{anchor} that spends both the anchor output from the unconfirmed tx_0 and an additional UTXO controlled by the executor. By setting the fee of tx_{anchor} such that $f_{\text{anchor}} \geq f_{\text{required}} + \tilde{f}_{\text{required}} - f_0$, where $\tilde{f}_{\text{required}}$ is the additional required fee for tx_{anchor}, the total effective fee for tx_0 becomes,

$$f_{\text{total}} = f_0 + f_{\text{anchor}} \geq f_{\text{required}} + \tilde{f}_{\text{required}}, \tag{2}$$

ensuring that tx_0 is confirmed.

3.7.2 Sighash Additional UTXO

For AO-invoked transitions where the AO does not commit to future protocol state, we adopt transaction signatures that specify the flag `SIGHASH_ANYONECANPAY | SIGHASH_ALL` instead of adding Anchor outputs to reduce protocol costs. These flags are set during the Protocol User Setup Ceremony when the depositor signs the PSBT for the AO's resolutions. This flag enables any entity, e.g. the AO, to add an additional UTXO as an input, which can be used to pay higher fees during network congestion.

If a pre-signed transaction tx_0 is broadcast but not confirmed due to insufficient fees ($f_0 < f_{\text{required}}$), the executor can add an additional input as a fee f_1, to tx_0 ensuring that $f_{\text{total}} = f_0 + f_1 \geq f_{\text{required}}$.

4 Smart Account Registry

4.1 Overview

The Smart Account Registry (SAR) is a smart contract deployed on the destination chain (e.g. Ethereum) that serves as the central coordination point for the protocol. The SAR is responsible for:

- Maintaining a public registry of deposit UTXOs and protocol parameters. This includes a status for every deposited UTXO, used by the AO to determine the correct resolution of any dispute (Sec. 5.4).
- Defining the *perimeter* (Sec. 4.2.2) and triggering events when depositor imbalances are detected, legitimizing Bitcoin rebalancing (Sec.4.2.1).
- Defining the rebalancing order of UTXOs, set by the depositor, described in section 4.2.3.
- Storing of pre-signed PSBTs for data availability, ensuring they cannot be corrupted or modified after protocol setup
- Storing authorized AO software versions with corresponding expiry timestamps, signed by the TO, (Sec.5.6).

4.2 Rebalance

A rebalance is the process when tokens minted on the destination chain against BTC held in an instance of the BSA exit the *perimeter* (as defined in Sec. 4.2.2) requiring the native Bitcoin acting as collateral to be transferred to the Lombard Security Consortium's general reserves. The token minted on the destination chain can be burnt and redeemed for native BTC from these general reserves after exiting the perimeter, while funds held in the BSA can only be transferred to predefined states as outlined in the Protocol User Setup Ceremony (Sec. 3.4.1). As redemptions are resolved using BTC from the general reserve, the rebalance mechanism is used to pull Bitcoin from BSA instances into the general reserves in the event said instance of the BSA is undercollateralized, rectifying any depositor imbalance.

4.2.1 Rebalance Mechanism

For the state transition *Rebalance Request* on the Bitcoin network to be legitimate, it must be preceded by a rebalance event on the destination chain, which is a result of a user *imbalance*. An imbalance occurs when the sum of initially deposited UTXOs D is greater than the sum of the user's balances across all DeFi platforms and their personal address.

Formally, let $B_{\text{defi}} = \sum_{i=1}^{k} b_i$ denote the aggregate balance across k DeFi platforms, and B_{personal} denote the personal address balance, we let the total,

$$B_{\text{total}} = B_{\text{defi}} + B_{\text{personal}} \tag{3}$$

and an imbalance is detected when,

$$B_{\text{total}} < D \tag{4}$$

When an imbalance is detected, the TO can then submit a transaction to the Smart Account Registry to mark the depositor as requiring rebalancing, emitting an event from the smart contract. This event allows the TO to initiate a legitimate rebalance request for the depositor's underlying Bitcoin without the risk of AO intervention.

4.2.2 Perimeter Definition

The *perimeter* is defined as an additional set of smart contracts registered on the SAR, called *Adapters*, which collectively track the value of B_{total}. These adapters enable the efficient querying of B_{total} for each user. This design choice was motivated to help both facilitate integration with DeFi protocols and allow BTC.b to be simultaneously backed by Bitcoin in BSA protocol and the TO's general reserves, without redemption risk to any BTC.b holder.

Adapter management follows a conservative security model: adding new adapters (integrating a new DeFi protocol) is effective immediately, as adapters only return positive or zero balances, posing no risk to depositors. However, removing or replacing an adapter requires adherence to time lock T_3, allowing depositors sufficient time to manage their DeFi positions in the event they are affected.

4.2.3 UTXO Rebalance Ordering

Every user address has an ordered set of UTXO outputs, $O = [o_1, \ldots, o_n]$, where each $o_i \in O$ is an active deposit on the SAR for any i. Users can customize the rebalance ordering by defining a permutation function $\pi : [n] \to [n]$, where $[n] = (1, \ldots, n)$, mapping initial output indices to the desired rebalance order, returning a sequence $L = [o_{\pi(1)}, o_{\pi(2)}, \ldots, o_{\pi(n)}]$ where $L[1]$ is rebalanced first.

4.3 Rebalancing Process

When a rebalance is detected, as defined in section 4.2.1, the TO initiates the process of rebalancing, sending from either VA $\to$ RCA, or UTA $\to$ UCA if a depositor's illegitimate unbond request was confirmed on the Bitcoin network. The rebalancing takes at most T_2 to complete, after which the TO can pull the funds into Lombard's general reserves.

For the sake of clarity, we stress that the derivative token BTC.b, issued by the TO (Security Consortium)[2], is backed by both BTC held in Lombard's general reserves and BSA instances.

Example 4.1 (Liquidation on a Lending Protocol)**.** Suppose a liquidation occurs on a lending protocol from an unhealthy position that uses BTC held in a BSA instance as collateral. A liquidator repays the user's bad debt (borrowed asset) to the lending platform, and as a result, acquires the collateral for this position, BTC.b, backed by native BTC from a user's BSA instance[3]. This causes the user's BSA to become imbalanced (see Sec. 4.2.1) as their BTC.b has exited the perimeter.

After receiving BTC.b from a liquidation, the liquidator has two options: (1) swap or sell the BTC.b on a decentralized exchange (DEX), or (2) redeem the BTC.b one-to-one for native Bitcoin from Lombard's general reserve.

In the second case, the redemption time will vary based on the proportion of liquidity immediately available in Lombard's general reserves. The longest time this can take is when Lombard's general reserve contains no Bitcoin. The redemption will then be completed within at most $T_2 + \omega$, where ω is the (nominal) time required for the TO to execute the transaction to pull the funds into its general reserves and transfer them to the liquidator's specified Bitcoin address. The address

[2]Lombard's Security Consortium's general reserves operate as a permissionless bridge: depositors generate a unique deterministic Bitcoin deposit address using a set of parameters including destination chain and address, and BTC.b is minted on the corresponding chain and address.

[3]In practice, a subset of DeFi platforms will rely on third-party oracles to track the reserves of BTC.b, ensuring that collateral is tracked regardless of whether funds are held in individual BSA instances or in Lombard's general reserve. In this scenario, the reserve oracles track and attest that addresses for every instance of the BSA are included in this list, which a depositor can verify before depositing BTC.

is provided as an input parameter by the liquidator[4] (required for a standard redemption) when calling the `redeem` function on the BTC.b contract.

UTXOs are rebalanced according to rebalancing order L (as described in Sec. 4.2.3) to cover the amount of imbalance $\Delta = D - (B_{\text{defi}} + B_{\text{personal}})$. The minimal set of UTXOs from L needed to cover Δ is selected for rebalance, with the remaining UTXOs unaffected. As mentioned in section 3.6.2, as PSBTs commit to set amounts, in the event a user's deposited UTXO amounts cannot exactly cover the required rebalance, an over-seizure will occur where the TO will rebalance an amount sufficiently large to cover the rebalance. This over-seized amount will (1) still actively collateralize the user's DeFi position without interruption and (2) can be redeemed by the user during unbonding from the TO. A user can mitigate this concern by increasing the liquidation granularity by depositing more, and smaller, UTXOs.

4.4 Upgrades

To maintain the ability to introduce new features and optimize protocol design in the future, the SAR implementation can be upgraded. The time lock for any proposed change is T_3, providing sufficient time for review by all participants in the protocol, and for participants to exit the protocol in the event they do not agree with scheduled changes.

5 Arbitration Oracle Architecture

5.1 Overview

The Arbitration Oracle (AO) plays a key role in ensuring that the Smart Account Registry (SAR) and the relevant protocol aspects for the Bitcoin network have consistent states. The AO is analogous to a neutral arbitrating party in a tri-party agreement.

For an instance of the protocol to be live, the Enclave codebase, proposed by the TO, must be accepted by both the depositor and the AO operator. Firstly, the TO signs a specific source code and configuration for an AO version. Secondly, the AO reviews this version and can accept it by deploying it. Following the deployment of this AO version, a cryptographic attestation is made available by the AO to both the depositor and the TO, providing a guarantee of AO integrity (see below Guarantee 6.5). Lastly, based on this attestation, the TO accepts this specific AO deployment as a legitimate AO party (making the AO Bitcoin signing key eligible to partake in instances of BSA), and the depositor can review that the core logic in the enclave follows a set of codified arbitration rules to which they agree.

5.2 Trusted Execution Environment

The main characteristic of the AO is that it can only fail due to lack of availability (offline), as described in the Failure Model (Sec. 2.3), i.e., the AO cannot sign maliciously outside of logic detailed in section 5.4. This guarantee is enforced by having all operations that validate protocol states and sign transactions deployed inside a hardware enforced trusted execution environment (TEE).

[4]Incidentally, as the BSA protocol is agnostic to liquidations and liquidators, i.e., the liquidator is not required to update the SAR and mark a position as `Imbalanced` in order to retrieve the collateral, the user has a theoretical time window to restore their position by purchasing BTC.b on the open market before the imbalance is recorded by the TO. The TO, however, can check imbalances arbitrarily, so this may not be feasible in every scenario.

TEEs are technologies provided by major CPU vendors that enable the creation of execution environments with hardware-enforced isolation from other software running on the same platform. These environments can produce a cryptographic measurement of their initial code and configuration, and generate an attestation report authenticating this measurement. This procedure uses an attestation key rooted in hardware and certified by the vendor's public key infrastructure.

By pairing such an attestation report with a key management service capable of verifying the report and conditioning key release on the measured code identity, an AO can be constrained to execute only the logic described in section 5.4, even if the operator hosting the AO behaves adversarially with respect to other protocol participants.

Our initial implementation is designed for the cloud provider Amazon Web Services (AWS). AWS provides a solution by integrating AWS Key Management Service (KMS) [15] with AWS Nitro Enclaves, a TEE solution based on the Firecracker [12] virtualization technology. For an overview of how key management is implemented with AWS for an AO, refer to Appendix D.

5.3 Enclave Architecture

The enclave implements the core logic of the AO. This includes the list of logical operations that brings the AO to a protocol decision. This application is made up of two main components:

- **Arbitration Logic:** Core implementation of the dispute resolution enforcing protocol rules
- **Helios Light Client:** Implementation of the Altair Light Client protocol [14] for SAR state verification.

The Enclave is packaged as an Enclave Image Format (EIF) file containing both the arbitration logic and the light client. The AO software package is built in a reproducible manner, so any resulting EIF always yields the same PCR0 measurement; participants can thus verify that the attested enclave matches the published source. Please see Appendix C for an overview of implementation details.

The Enclave generates an AWS KMS key with policy restrictions that deny access outside the Enclave application, publishes the public key with AWS attestation including Platform Configuration Registries (PCRs) values that identify the exact codebase, and is limited to providing Bitcoin network signatures only under conditions enforced by the Enclave logic. Please see Appendix D.1.2 for an overview of implementation details.

5.4 TEE Logic

The aims of the TEE logic are to ensure the AO (i) processes only valid and relevant Bitcoin transactions, (ii) identifies both Bitcoin and SAR protocol states, and (iii) decides on a resolution based on the consistency or inconsistency of these states.

The AO operates in a fail-closed mode: it will only sign transactions (see figure 1, labels 4a. & 6a.) if all verification steps outlined below are completed successfully. If the AO cannot verify the SAR state (e.g., due to light client desynchronization, Sec. 5.5), it cannot sign. This design prioritizes safety over *liveness* (see Def. 2.2): no incorrect signatures can be produced under ambiguous conditions, thus, an AO that cannot verify is operationally equivalent to an offline AO. The depositor's liveness guarantee is preserved by the 1-of-k redundancy model. That is, a single correct AO with a synchronized view of the destination chain and legitimate inputs (Bitcoin transactions) is sufficient to resolve disputes within T_2.

5.4.1 Transaction Verification

If the enclave receives raw transaction data for $tx_{\text{VA}\to\text{RCA}}$; or $tx_{\text{VA}\to\text{UTA}}$ and $tx_{\text{UTA}\to\text{UCA}}$, the enclave:

1. verifies signatures match the public keys in the Bitcoin transaction's witness stack according to the UTXO value fetched from the SAR,
2. verifies that one public key is the TO's public key pk_{TO} (fetched from the SAR),
3. verifies the address tweaking parameters (fetched from the SAR) and recognizes the state transition as described at the very end of section 3.3.1.

For $tx_{\text{VA}\to\text{RCA}}$, the enclave:

4. identifies $\text{UTXO}_{\text{deposit}}$ from the input of $tx_{\text{VA}\to\text{RCA}}$.

Or, for $tx_{\text{UTA}\to\text{UCA}}$, the enclave:

5. verifies the output of $tx_{\text{VA}\to\text{UTA}}$ is the input of $tx_{\text{UTA}\to\text{UCA}}$,
6. identifies $\text{UTXO}_{\text{deposit}}$ as the input of $tx_{\text{VA}\to\text{UTA}}$.

Lastly, the enclave queries the light client to:

7. identify the user account on the SAR which is indexed against $\text{UTXO}_{\text{deposit}}$,
8. identify the SAR UTXO status,
9. verify the software the AO is running is up-to-date (see 5.6.1 for details).

The enclave then decides the appropriate action based on the logical rules outlined in sections 5.4.2 and 5.4.3, using the UTXO status to determine the correct resolution.

5.4.2 Rebalance Resolution

For rebalancing requests by the TO ($tx_{\text{VA}\to\text{RCA}}$), after completing the verification steps above, the Enclave checks the UTXO status on the SAR:

- If the UTXO status is not `SpentOnRebalance` (indicating an invalid rebalance request by the TO), the enclave retrieves the PSBT from the SAR that enables the AO to resolve the dispute in favor of the depositor and signs it.
- Otherwise, the Enclave elects to perform no action, effectively resolving the dispute in favor of the TO, who can pull the funds after time lock T_2 expires.

Algorithm 1 AO Enclave Logic for Rebalance Request Resolution.

Require: $tx_{\text{VA}\to\text{RCA}}, \text{sk}_{AO}$ (secret key from AO)

$\text{UTXO}_{deposit} \leftarrow tx_{\text{VA}\to\text{RCA}}.\text{input}$
$pk_{TO} \leftarrow \text{GetPKTOFromSAR}(\text{UTXO}_{deposit})$
if $!\text{SigValid}(tx_{\text{VA}\to\text{RCA}}, pk_{TO})$ **then return** $\perp$
if $tx_{\text{VA}\to\text{RCA}}.output.address \neq \text{GetRCAFromSAR}(\text{UTXO}_{deposit})$ **then return** $\perp$
$utxoStatus \leftarrow \text{GetUTXOStatusFromSAR}(\text{UTXO}_{deposit})$
if $utxoStatus \neq SpentOnRebalance$ **then**
    $PSBT_{\text{depositor}} \leftarrow \text{LookupPSBTonSAR}(\text{UTXO}_{deposit})$ ▷ PSBT to resolve in favour of depositor
    $tx_{\text{resolved}} \leftarrow \text{Sign}(PSBT_{\text{depositor}}, \text{sk}_{\text{AO}})$
    return tx_{resolved}
else
    return $\perp$ ▷ Do nothing: rebalance authorized, TO can claim after T_2

5.4.3 Unbonding Challenge Resolution

For challenged unbonding requests ($tx_{\text{UTA}\to\text{UCA}}$), after completing the verification of section 5.4:

- If the UTXO status is `Withdrawn` (indicating the depositor has burned BTC.b tokens) or `Rejected` (indicating the BTC.b was not yet minted and the depositor requested to exit the protocol), the Enclave retrieves the PSBT from the SAR that enables resolving the dispute in favor of the depositor and signs it.
- Otherwise, the Enclave elects to perform no action, effectively resolving the dispute in favor of the TO, who can pull the funds after time lock T_2 expires.

Algorithm 2 AO Enclave Logic for Unbonding Challenge Resolution

Require: $tx_{\text{UTA}\to\text{UCA}}$ and $tx_{\text{VA}\to\text{UTA}}, \text{sk}_{AO}$ (secret key from AO)

$\text{UTXO}_{deposit} \leftarrow tx_{\text{VA}\to\text{UTA}}.\text{input}$
$pk_{TO} \leftarrow \text{GetPKTOFromSAR}(\text{UTXO}_{deposit})$
if $!\text{SigValid}(tx_{\text{VA}\to\text{UTA}}, pk_{TO})$ **then return** $\perp$
if $!\text{SigValid}(tx_{\text{UTA}\to\text{UCA}}, pk_{TO})$ **then return** $\perp$
if $tx_{\text{VA}\to\text{UTA}}.output \neq tx_{\text{UTA}\to\text{UCA}}.input$ **then return** $\perp$
if $tx_{\text{VA}\to\text{UTA}}.output.address \neq \text{GetUTAFromSAR}(\text{UTXO}_{deposit})$ **then return** $\perp$
if $tx_{\text{UTA}\to\text{UCA}}.output.address \neq \text{GetUCAFromSAR}(\text{UTXO}_{deposit})$ **then return** $\perp$
$utxoStatus \leftarrow \text{GetUTXOStatusFromSAR}(\text{UTXO}_{deposit})$
if $utxoStatus \in \{Withdrawn, Rejected\}$ **then**
    $PSBT_{\text{depositor}} \leftarrow \text{LookupPSBTonSAR}(\text{UTXO}_{deposit})$ ▷ PSBT to resolve in favour of depositor
    $tx_{\text{resolved}} \leftarrow \text{Sign}(PSBT_{\text{depositor}}, \text{sk}_{\text{AO}})$
    return tx_{resolved}
else
    return $\perp$ ▷ Do nothing: unbonding challenge authorized, TO can claim after T_2

5.5 Light Client Synchronization

The TEE runs a light client of the destination chain in order to verify SAR events. As the first implementation of the protocol targets the Ethereum blockchain, the following discussion describes mechanics and constraints according to the Ethereum post-merge [13] consensus protocol. Hence, depending on the destination chain, considerations for implementing BSA protocol roles will differ.

In Ethereum, the process of synchronizing with the network is subject to a weak subjectivity model [16], where a Weak Subjectivity Period (WSP) T_{sub} defines the maximum time a node can be offline while still resynchronizing autonomously. We stress the fact that an AO can resync autonomously to the network after both (1) a network interruption and (2) system downtime, both points detailed in Appendix E.2. Period T_{sub} is initialized with a conservative hardcoded value[5] and, after the light client is synchronized, the TEE keeps updating the period length based on network conditions following the Ethereum specification.

When the TEE comes back online after downtime, it follows a resynchronization protocol: if downtime is less than T_{sub}, it resynchronizes from its last known block using a self-attested checkpoint [6]; if the downtime exceeds T_{sub} or during initial boot, it uses a TO-signed checkpoint not older than the default hardcoded weak subjectivity period.

When an AO resynchronizes using a TO-provided checkpoint, they are responsible for verifying the validity of the checkpoint to ensure depositor safety. An AO can decline to resync if they disagree with the proposed checkpoint statement. An AO that declines to resync defaults to exiting the protocol until such a further time when it agrees with the TO's attested checkpoint. This is operationally equivalent to an offline AO. If any of a depositor's chosen AOs disagrees with a checkpoint, the depositor must be notified. The depositor can decide to maintain their BSA instance with their remaining AOs, or can choose to exit the protocol with sufficient time, as ensured by relation 1. We explore how this impacts AO availability requirements in Appendix E.

5.6 Upgrades

The Enclave application can be upgraded to introduce new features, fix bugs, and for general code maintenance according to the following mechanics:

1. Enclave Image Files (EIF) are signed by the TO.
2. The AWS KMS key that decrypts the AO secret key is constrained via the PCR8 value, providing a guarantee from the Nitro Enclave system that only artifacts signed by the TO can operate it (i.e., the TO is the only entity able to release new software upgrades).
3. AO providers are responsible for deploying and running such software in their cloud accounts; therefore, before updating the old image with the latest upgrade, they are responsible for verifying the contents of each upgrade.

The above points ensure all protocol parties verify and approve the AO software while guaranteeing that only agreed software can use the AO KMS key. Depositors and the TO can request to verify the expected software is operated by the AO using the deterministic nature of Enclave Image Files builds and the PCR0 value reported in any attestation document. See Appendix C for info on PCR values and Appendix F for additional consideration on the AO version lifecycle.

[5] A value is set to be shorter than any reasonable presumption for a WSP.

[6] self-attested means that the TEE periodically, or upon request (e.g., in preparation for a scheduled reboot), creates an attestation as per section 5.8 containing the currently computed WSP which is stored in a persistent storage by the entity running the enclave application

5.6.1 Version Expiration

Authorizing and verifying the new AO software does not provide additional security without the ability to deprecate previous versions.

We achieve deprecation using a time-based expiration mechanism. The Smart Account Registry holds a mapping between versions and their respective expiration. The TO can set the expiration for a specific version by referring to its PCR0 value. The Enclave application, before performing any protocol action, verifies that its own version is not expired by retrieving the expiration value from the SAR. If the expiration date is past, the AO is not able to proceed with signing a transaction for dispute resolution (as shown in Sec. 5.4.1 step 9.).

Due to the time-based expiration mechanism, AO operators have a time window within which they can update the Enclave without any downtime. The protocol parameter relation 1 is enforced by the AO operator deploying the initial versions of the AO software only if $T_3 > T_1 + T_2$ and by only accepting new versions with increasing expiration time. Importantly, the depositor can always exit the protocol whenever the expiration time of the deployed AO software approaches T_3. We develop this topic in more detail in Appendix F.

5.7 Security Properties

The TEE architecture provides the following security properties:

Property 5.1 (TEE Code Integrity)**.** Once deployed, the TEE code cannot be modified by an AO operator, AWS administrators, or any other party, except as outlined in sections 5.6 and 5.6.1.

Note: We analyze this property in more detail as Guarantee 6.5 in Sec.6.3.

Property 5.2 (No Pre-Signing)**.** The AO cannot pre-sign any PSBT until after the steps outlined in section 5.4 and subsections 5.4.3 and 5.4.2.

5.8 Attestation and Verification

Attestations are documents produced by the Enclave application to certify that a computation, resolution, or state assessment was performed according to the core logic implemented in a specific EIF image. Each attestation always reports the AO public key, all PCR registries of an Enclave application deployment, and is signed by the AWS Nitro Enclave service to demonstrate such values are legitimate. The attestation document can be verified using the AWS Nitro Enclave Public Key Infrastructure [17].

This provides a guarantee to all protocol parties that a specific AO deployment runs a codebase that both the TO and the AO operator agreed upon. The TO would not participate in the Protocol User Setup Ceremony without having verified an attestation that reports the AO public key and the expected PCR registry values. Appendix C describes how PCR values and attestation contents define this process.

6 Security Analysis

6.1 Overview

The BSA protocol achieves a trust model where depositors only need to trust that at least one AO (1-of-k) or the TO remains honest, achieving 1-of-$(k+1)$ security (see section 6.2). This represents a significant improvement over traditional m-of-n threshold schemes that require an

honest majority. The security boundary is reached when all AOs and the TO are both corrupted. Additionally, in Lombard's model, the TO is a set of validators, called the Security Consortium, which self-coordinate via a Tendermint [18] based consensus protocol.

6.2 Trust Model: 1-of-k vs 1-of-$(k+1)$

We use the term 1-of-$(k+1)$ security to refer to k independent AOs and 1 TO. The TO is itself a Byzantine Fault Tolerant distributed system running the Tendermint [18] consensus protocol, progressing correctly if less than one-third of validators are dishonest. The distinction between 1-of-k and 1-of-$(k+1)$ refers specifically to the requirement that either 1-of-k AOs must remain *correct*, or, if all AOs are offline, then the protocol still achieves Protocol Safety 2.5 under the Failure Model 2.3 if the TO is honest. If the depositor's security threshold is strictly reliant on AOs, excluding the TO, then we have 1-of-k security; otherwise, we have 1-of-$(k+1)$ security.

6.3 Security Guarantees

We now outline the security guarantees of the BSA protocol with respect to the definitions in section 2.4. The protocol achieves 1-of-$(k+1)$ security, requiring that at least one of k AOs (see definition 2.1) is correct or the TO remains honest. Security guarantees ensure that funds cannot be stolen or lost under the Failure Model 2.3.

Guarantee 6.1 (Depositor Safety)**.** Under the assumption that at least one AO remains correct, a depositor's funds are safe from theft by a malicious TO. Specifically:

- A correct depositor can withdraw their native Bitcoin unilaterally after time lock T_1 (if no challenge) or at most $T_1 + T_2$ (after challenge resolution).
- A malicious TO cannot steal funds through false rebalances (VA $\rightarrow$ RCA) or unjustified challenges (UTA $\rightarrow$ UCA), as any correct AO (online and operational) verifies the SAR state and resolves the dispute in favor of the depositor within T_2.
- Even if an AO keypair is compromised, they cannot steal funds, as AO spending is constrained by emulated covenants to the depositor address only (see section 3.4). The most severe action available to a malicious AO operator is to take the AO offline, which has no impact if other correct AOs remain online.

See Appendix F.3 for additional remarks.

Guarantee 6.2 (Token Operator Safety)**.** The TO is safe from malicious depositors and malicious AO operators:

- If a depositor initiates an illegitimate unbonding request (VA $\rightarrow$ UTA without updating SAR to `Withdrawn` or `Rejected`), the TO can challenge within T_1, and funds remain in UCA until T_2 expires, enabling the TO to claim them. All correct AOs verify the SAR state and observe the challenge to be legitimate, allowing the TO to claim funds after T_2 expires.
- If a depositor's position becomes under-collateralized, the TO can initiate a rebalancing (VA $\rightarrow$ RCA). Correct AOs verify the destination chain imbalance exists and do not dispute, allowing the TO to claim funds after T_2 expires. Incorrect AOs cannot deviate from the prescribed behavior due to property 5.1.

Guarantee 6.3 (Protocol Safety)**.** The protocol progresses towards a state ensuring Protocol Safety under Guarantees 6.1 and 6.2.

That is, Protocol Safety holds if at least one of the following holds: (1) at least one AO is correct, or (2) the TO is correct. If at least one AO is correct, disputes are resolved within time lock T_2 through the AO's deterministic dispute resolution logic (sections 5.4.2 and 5.4.3). If all AOs are offline but the TO is correct, protocol state transitions occur via Bitcoin script timelocks: the TO can claim funds from challenge addresses (UCA, RCA) after T_2 expires, ensuring progress to a TO-safe state. The deterministic nature of Bitcoin timelocks ensures progress, regardless of AO availability.

Note: we do not consider protocol liveness 2.2 as a property in this section given that we are unconcerned with finality that is reached in a non-protocol-safe manner. However, the reader may observe that eventually, the protocol will always progress to an end state (unilateral control of funds by the depositor or TO) if either the depositor or TO is online.

Guarantee 6.4 (Covenant Enforcement)**.** Even if AWS Nitro Enclaves or KMS are compromised, Bitcoin can only be transferred to either: (1) the depositor's address, or (2) the TO's address (if AO does not act within T_2). This is enforced by emulated covenants (section 3.4): all PSBTs exchanged during the Protocol User Setup Ceremony commit to specific output addresses through pre-signing with fixed outputs. Even if the TEE or KMS is compromised, the pre-signed PSBTs cannot be modified to create new spending paths, as they are cryptographically committed. The only valid spending paths are defined during the original setup ceremony, preventing the establishment of new spending paths to third parties.

Guarantee 6.5 (TEE Code Integrity)**.** The TEE cannot be modified by AO operators, AWS administrators, or any other party, except through the upgrade mechanism in sections 5.6 and 5.6.1. The TEE enforces code integrity through hardware-enforced isolation. The Enclave Image File (EIF) is cryptographically measured via Platform Configuration Registers (PCRs), and any modifications to the codebase results in different PCR values, which can be verified through attestations (section 5.8). KMS policy restrictions prevent access to signing keys outside the TEE, and the dispute resolution logic (sections 5.4.2 and 5.4.3) requires state verification before signing, preventing pre-signing attacks.

Guarantee 6.6 (Protocol Immutability)**.** No changes can be made to protocol design (adapter removal 4.2.2, SAR upgrades 4.4, TEE version deprecation 5.6) before the time lock T_3 expires. All protocol modifications that impact security goals require time lock T_3. During this period, depositors can withdraw their funds unilaterally (see Guarantee 6.1) using previously agreed upon TEE versions, ensuring no party is forced to accept changes they disagree with.

6.4 Hypothetical Risks with Failure Analysis

In the failure model (Sec. 2.3) we assume the TO and the depositor to be Byzantine, i.e., they may deviate arbitrarily, by acting dishonestly or maliciously, to maximize their own gain. Throughout the paper, security is analyzed under the assumption that AOs can only fail by being offline, i.e., failing to fulfill time-sensitive duties. We now characterize how the protocol behaves when this assumption is relaxed.

The system fails when Protocol Safety 2.5 is violated: either the depositor cannot regain control of their funds (Depositor Safety fails Def. 2.3), or the TO cannot reclaim collateral when tokens

exit the perimeter (Token Operator Safety Def. 2.4 fails). Failures arise from: (1) AO unavailability or key compromise, or (2) TO quorum corruption or inability to reach consensus.

6.4.1 AO Failure Types

Availability (offline). An AO is *unavailable* if it does not respond within T_2 to sign dispute resolutions. If there are offline AOs but at least one correct AO, all guarantees hold. If *all* AOs are offline, the guarantee of Protocol Safety 6.3 can only be satisfied by an honest TO: a correct TO can claim funds from challenge addresses after T_2; a malicious TO can steal depositor funds via false rebalances (violating Depositor Safety 6.1) or fail to challenge a depositor's illegitimate unbonding request (violating Token Operator Safety 2.4).

Key compromise. An AO's signing key is *compromised* if the key is leaked such that an adversarial actor can use it to execute transactions on behalf of the AO. The leak of one AO key is enough for a malicious depositor to independently perform the following attack: initiate an illegitimate unbond, receive a legitimate challenge from the TO, and sign the resolution in their favor using the leaked key. Depositor Safety 6.1 holds, and Token Operator Safety 6.2 fails. The Guarantee of Covenant Enforcement 6.4 fails in a non-significant way, the depositor can still send funds to anywhere they control.

TEE Failure. An AO operator can access and update the code deployed in the enclave to something not approved by the TO. In this event the guarantees of TO Safety 6.2, Protocol Safety 6.3, TEE Code Integrity 6.5, and Protocol Immutability 6.6 (specifically, TEE version deprecation) all fail. Covenant Enforcement 6.4 holds, and Depositor Safety 6.1 holds if there is at least one correct AO remaining, or if the AO with TEE failure is colluding with said depositor.

6.4.2 TO Failure Types

Inability to reach consensus (no sign). If at least 1/3 of LSC validators are offline the TO cannot sign. The TO cannot initiate rebalances or challenge depositor unbonding requests. Protocol Safety 6.3 is degraded. Depositor Safety 6.1 is ensured if (1) they have a UTXOs in their VA address, or (2) they do have UTXOs in a challenged state, then they require at least one AO to be correct. Respectively, a depositor (1) can unbond and will not be (illegitimately) challenged, and (2) if a TO illegitimately challenges and then goes offline, the depositor still requires one correct AO to resolve the dispute. Token Operator Safety 6.2 is degraded: the TO cannot reclaim funds from challenge addresses if it cannot sign.

Quorum corruption. The TO requires a 2/3 supermajority to sign. If two-thirds or more of LSC validators are Byzantine, they can form a quorum and produce malicious outcomes, e.g., signing false rebalances, signing false unbond challenges, or refusing to challenge illegitimate unbonding. In this case, Protocol Safety 2.5 is not achieved: a malicious depositor can steal funds if the TO does not challenge illegitimate unbond requests, or a corrupted TO together with all AOs offline can steal depositor funds. Token Operator Safety 6.2 fails and Depositor Safety 6.1 can fail if all AOs are offline (see Sec. 6.4.1).

6.4.3 Guarantee Summary by Failure Type

Table 2 summarizes which guarantees remain enabled under each failure scenario. "Yes" indicates the guarantee holds; "No" indicates it fails. For the table below, unless otherwise specified, we assume the depositor to be Byzantine.

Table 2: Guarantees under failure types. Dep. = Depositor Safety 6.1; TO = Token Operator Safety 6.2; Safety = Protocol Safety 6.3

Failure	**Condition**	Dep.	TO	Safety
1 AO offline	≥ 1 AO correct	Yes	Yes	Yes
All AOs offline	TO Honest	Yes	Yes	Yes
All AOs offline	TO Malicious	No	Yes	No
AO key leak	Dep. Malicious	Yes	No	No
TO no consensus	≥ 1 AO correct	Yes	No	No
TO no consensus	All AOs offline	Yes	No	No
TO quorum corruption (non-rational)	≥ 1 AO correct	Yes	No	No
TO quorum corruption (non-rational)	all AOs offline	No	No	No

6.5 Monitoring Requirements

Security guarantees depend on monitoring of both the Bitcoin network and the destination chain.

The TO must monitor Bitcoin transactions to detect illegitimate unbonding (VA→UTA) within T_1 and challenge if necessary. Monitoring the Smart Account Registry is necessary to observe legitimate rebalancing events before executing the transition (VA→RCA) on the Bitcoin blockchain.

Correct AOs must monitor Bitcoin to detect unfair rebalance (VA→RCA), or unfair challenges (UTA→UCA) within T_2 and resolve the dispute if necessary. Also, by monitoring the SAR, AOs acquire knowledge about new instances of the BSA protocol to safeguard.

Failure to monitor within relevant timelock windows can lead to violation of the security guarantees outlined in section 6.3.

7 Limitations and Future Work

7.1 Current Limitations

1. **Infrastructure Integrity:** The protocol trusts AWS Nitro and KMS infrastructure for the AO role. While this is a reasonable assumption, it introduces a dependency on a single technology and service provider. We view this risk as acceptable, as in the event of a full compromise of AWS KMS keys, the architectural design choices limit the only possible action to be the unbonding of Bitcoin in favor of the depositor.

2. **Timelock Trade-offs:** The timelock periods T_1 and T_2 create a trade-off between security and user experience. Longer time locks provide greater security but also introduce potential delays for depositor unbonds and TO rebalances.

3. **Liquidation Granularity:** Model 1 (Collaborative Rebalance, Sec. 3.6.1) requires user availability to create new PSBTs, which may not be ideal for all use cases. Model 2 (UTXO-Based Rebalancing, Sec. 3.6.2) may impede user experience by requiring multiple signing of PSBTs during the setup ceremony (Sec. 3.4.1).

4. **Downstream Oracle Failure:** The BSA protocol does not mitigate risks arising from downstream DeFi protocol price oracle failures or manipulations, which would lead to unfair rebalances, as the perimeter only tracks on-chain balances.

7.2 Future Work

1. **Resynchronization Dependency**: Supporting multiple sources beyond the Lombard Security Consortium (e.g., other trusted sources and services) to provide trusted check points

for the destination chains—used during resynchronization after extended AO downtime (see section 5.5)—would improve AO liveness.

2. **Multi-Client Destination Chain Architecture:** Implement a selection of destination chain clients and consensus between them to leverage code diversity and expand the set of supported destination chains.

3. **Bitcoin Protocol Upgrades:** If Bitcoin adopts native covenant opcodes (e.g., `OP_CAT` [19], `OP_CHECKTEMPLATEVERIFY` [8]), the protocol could be simplified and efficiency improved.

4. **Alternative TEE Providers:** Explore support for TEE providers beyond AWS Nitro to reduce infrastructure dependency on both the cloud provider (e.g., Google Cloud Platform, Microsoft Azure) and CPU manufacturer (AMD SEV-SNP, Intel TDX).

8 Conclusion

This paper introduced Bitcoin Smart Accounts (BSA), a novel protocol that enables native Bitcoin to access DeFi applications while maintaining self-custody and minimizing trust assumptions. We achieve this through the combination of emulated Bitcoin covenants (using PSBTs and Taproot scripts), a TEE-based arbitration system, and smart contracts that enable DeFi platforms to operate without necessitating protocol-level modifications.

BSA achieves a trust model where depositors only need to trust that at least one Arbitration Oracle or the Token Operator remains honest, achieving 1-of-$(k+1)$ security, a significant improvement over traditional m-of-n threshold schemes. A single honest AO can counter any decision made by the TO, providing strong safety guarantees for depositors.

We provided security evaluations demonstrating correctness, safety, and liveness properties under our trust model. Our design enables native Bitcoin to serve as collateral in lending markets and other DeFi protocols without requiring users to relinquish custody of their funds, mimicking a tri-party agreement.

The protocol represents a significant step toward unbonding Bitcoin's vast capital for DeFi applications while preserving Bitcoin's core values of self-custody and trust minimization. We believe that Bitcoin Smart Accounts provide a practical and secure foundation for the growing Bitcoin DeFi ecosystem.

Appendices

A Bitcoin Taproot Address Tweaking

A Taproot address commits to an *internal key* P and a list of spending scripts, organized in a Merkle tree, via its Merkle root m [3], by combining them in a single *tweaked key*. Such a *tweaked key* Q is included in the `ScriptPubKey` that yields the Taproot address, and it is computed as follows: $Q = P + tG$ where $t = \mathsf{hash}(p \| m)$, p is the 32-byte encoding of the x-coordinate of point P and the point G is the generator point of the `secp256k1` elliptic curve. Any variation either of the internal key, or of the scripts list produces a different `ScriptPubKey`, so a different taproot address. For full specifications, please refer to BIP-341 [3].

A.1 Nothing Up My Sleeves (NUMS) Internal Key

All BSA Taproot addresses use a *Nothing Up My Sleeves* (NUMS) point as the internal key P [3]. A NUMS point is an elliptic curve point assumed to have an unknown discrete logarithm with respect to the `secp256k1` generator point G, implying it is computationally infeasible to use it to satisfy a signature verification algorithm. Additionally, NUMS keys can be constructed deterministically starting from a known set of inputs so that it is verifiable that no party can have chosen it to embed a backdoor. BIP-341 [3] describes as an example the NUMS point,

$$H = \mathsf{lift\_x}(\texttt{0x50929b74c1a04954b78b4b6035e97a5e078a5a0f28ec96d547bfee9ace803ac0}) \quad (5)$$

constructed by taking the hash of the standard uncompressed encoding of G and using the result as the x-coordinate. A number of NUMS keys can be generated starting from H by taking $H + rG$ where r is any integer smaller than the order of the `secp256k1` curve.

Since a valid taproot address includes a key spending path, BSA disables it using a NUMS construction, resulting in only script path spending being enabled, used for multi-signature in covenant emulation and time-lock conditions.

BSA leverages the following NUMS construction to commit protocol-specific information in each protocol addresses. Each NUMS point is computed following the BIP-341 [3] suggestion $H + rG$ by taking $r = sha256(\text{address type}||\text{tweak data}) \bmod n$, where n is the order of the `secp256k1` curve. The *address type* is a simple label identifying which of the four address types should be generated (VA, UTA, UCA, RCA), while the *tweak data* includes all the information that characterizes the protocol instance such as the keys of all involved parties, all timelocks, and the destination address information.

B Transaction Reference Table

Table 3 lists the essential Bitcoin state transitions. Abbreviations *Dep*, *TO*, *AO*: Depositor, Token Operator, Arbitration Oracle. *Creator*: creates and signs the PSBT; *Stored*: where it is held (Dep, TO, SAR); *Executor*: signs, finalizes, and broadcasts.

Table 3: Transaction reference.

Transaction	Path	Signers	Creator	Stored	Executor	Fee	Motivation
Unbond request	VA → UTA	TO & Dep	TO	SAR	Dep	Anchor; CPFP [10]	Dep initiates withdraw request
Unbond finalize	UTA → Dep	Dep	—	—	Dep	Dep's UTXOs	Unbond request not challenged, claimed after T_1
Unbond challenge	UTA → UCA	Dep & TO	Dep	TO	TO	Anchor; CPFP	TO challenges unbond request within T_1
Unbond resolve	UCA → Dep	Dep & AO	Dep	SAR	AO	SIGHASH [3]	AO sides with depositor within T_2
Unbond resolve (expired)	UCA → TO	TO	—	—	TO	TO UTXOs	TO claims after T_2; challenge was valid
Rebalance request	VA → RCA	Dep & TO	Dep	TO	TO	Anchor; CPFP	TO initiates rebalance request
Rebalance resolve	RCA → Dep	Dep & AO	Dep	SAR	AO	SIGHASH	AO sides with Dep; unfair rebalance
Rebalance resolve (expired)	RCA → TO	TO	—	—	TO	TO UTXOs	TO claims after T_2; rebalance was valid

Note: in the above table, the terms Creator and Executor are as defined in section 3.4. The Executor column indicates the recommended party, though, after both parties have signed, either signer may execute the state transition. Storage by the TO refers to Lombard Ledger, the Security Consortium's ledger.

C Reproducible Build and Outcome for Arbitration Oracle

An important desired characteristic of Trusted Execution Environment (TEE) is the ability to attest, or provide proof, that the outcome of a computation from within the TEE is the result of the execution of a specific codebase with a corresponding set of inputs and states. AWS Nitro Enclaves provides this feature via *attestations*: documents signed-off by the Nitro hardware and software, validated via the AWS Public Key Infrastructure, which report measurements of the runtime state inside a running enclave. In particular, there are two values, or Platform Configuration Registries (PCR), that are used to verify the AO operation correctness:

1. **PCR0**: reports the sha384 digest of the Enclave Image File running in the enclave, effectively certifying that a specific codebase was built and assembled in a determined runtime.

2. **PCR8**: reports the sha384 digest of the certificate that signed the Enclave Image File running in the enclave, certifying that a deployed artifact has been signed by the TO.

While PCR8 is a simple certificate-match verification, the reliability of PCR0 depends on the ability to reproduce the same Enclave Image File from the same source code. To achieve this, the AO implementation relies on a deterministic and reproducible toolchain, which includes:

- A deterministic Rust compiler,

- Hash-locked dependencies for libraries needed at compile time to always produce the same statically linked binary,
- Hash-locked dependencies for tools needed at execution time, i.e. network proxies,
- Pinned base image to construct the enclave runtime,
- Pinned Nitro Enclave toolchain to pack together the Enclave Image File from the runtime definition and the AO logic build artifacts.

C.1 AO Attestation Contents

Beyond the PCR measurements included in all AWS Nitro Enclave attestations, such documents offer the possibility to include additional application-specific data. The AO implementation always sets for every attestation:

- The Bitcoin public key of the AO to correctly identify the attester identity,
- The latest Ethereum finalized checkpoint to assess what Ethereum state the AO decision was based on,
- The hash of the AO resolution and the Bitcoin input data used to derive this resolution.

D Arbitration Oracle Key Lifecycle

The correctness of the AO running in the TEE is directly connected to the security of the key pair representing its Bitcoin identity. The main challenge to overcome in having a TEE operating a Bitcoin identity is the lack of persistent storage to rely on in case of restarts of the AO-deployed artifact. In fact, once a keypair is securely generated in an enclave using a crypto secure pseudo-random number generator, they are used in the Bitcoin scripts, addresses, and PSBTs reported in Sec. 3 which remains valid for an indefinite amount of time. However, a restart of the AO instance would lose any previous state, thus permanently losing the previous keypair and ostensibly implying it could only generate a brand new identity. This issue is overcome using AWS KMS and its native integration with AWS Nitro Enclaves attestations.

We report below all the operations an AO enclave and the operator deploying its logic in a cloud instance (referred to as the *AO operator*) carry out to achieve a secure persistent usage of a Bitcoin identity in case of restarts, upgrades, or crash failures of the AO TEE logic.

As a brief summary of the solution (explored in the proceeding section), we can say that the enclave logic uses AWS KMS to encrypt its Bitcoin private key. Then, it creates an attestation using the Nitro Secure Module (NSM) that new instantiations of the enclave can use to verify that the decoded identity originates from a legitimate execution of an AO in an AWS Nitro Enclave. Such NSM attestations are also validated by AWS KMS, which only decrypts the encrypted backup of the Bitcoin private key to the authorized software.

D.1 Key Initialization

On initial boot, the AO running in the enclave has no identity set, and it receives from the AO operator the `Init` command which makes it:

1. Generate a Bitcoin keypair (secp256k1) requesting a source of randomness from the Nitro Secure Module. Additional and independent sources of randomness can be fed to contribute to this process.

2. Request from AWS KMS the creation of a new key with encrypt and decrypt capabilities configured with the constraints described in Sec. D.1.2.

3. Encrypt the Bitcoin secret key using the created AWS KMS key (locally in the enclave or via secure communication with AWS KMS)

4. Produce, using the Nitro Secure Module available inside the enclave, an attestation that includes, in the user data field, the hash of the concatenation of the following data: the newly generated Bitcoin public key, the encrypted private key, and the KMS Key ID.

5. Return the attestation to the cloud operator along with the information hashed in the user data.

At the end of a successful initialization process, the AO operator will obtain:

- A Nitro Enclave initialized with the logic of an AO and a legitimate identity suitable to play the AO role in BSA protocol instances.
- The Bitcoin public key of the AO to share to depositors for inclusion in the Protocol User Setup Ceremony.
- An encrypted version of the Bitcoin private key corresponding to the public key of the identity of the AO.
- The KMS Key ID used to encrypt the AO Bitcoin private key.
- An AWS Nitro Enclave attestation that the AO Bitcoin keypair has been generated according to the logic of a specific version of the AO (PCR0 measurement) released and signed off by the token operator (PCR8 which includes the hash of the TO certificate)

D.1.1 Key Restore

After a restart of the AO enclave, the cloud operator can restore a previously generated AO identity by sending to the TEE a Restore command and the information obtained during the first initialization: the Bitcoin public key, the encrypted secret key, the KMS Key ID, and the attestation committing to such information. The AO implementation running inside the Nitro Enclave:

1. Verifies the attestation using the AWS Nitro Enclave root certificate publicly provided by AWS [17] and embedded in the enclave image.

2. Decrypts the Bitcoin secret key using the AWS KMS Key referenced by the input Key ID.

3. Verifies the decrypted secret key corresponds to the Bitcoin public key passed as input.

D.1.2 AWS KMS Key constraints

Constraints on the key usage are expressed in AWS KMS via a usage policy. The core of the AWS Nitro Enclave integration with AWS KMS is the possibility to specify in the key usage policy constraints based on the PCR measurements of the Nitro Secure Module available in the Nitro Enclave. In the AO case, the KMS Key created during the initialization phase (Sec. D.1), includes a condition on the PCR8 measurement. Such registry is set to the hash of the certificate released by the entity that signed the Enclave Image File running inside the TEE. By setting such a value to a certificate in control of the Lombard Security Consortium, the AO operator can only restore a legitimate identity inside versions of the enclave software released and signed-off by the Consortium. KMS will not authorize any other usage of the decryption key.

Additionally, the key policy explicitly denies updates of the policy itself, locking out the AO operator or any root admin of the AWS Organization.

E AO Availability Requirement Estimation

The guarantee of Depositor Safety (Sec. 6.1) pivots on the assumption the AO can protect a depositor from potential unfair challenges by the TO. This section aims to outline the availability requirements for an AO operator to fulfill the 1-of-k requirement in the BSA protocol.

E.1 Protocol Parameter Constraints

An AO will run software authorized by the TO if it has an expiration time of at least T_3 (Sec. 3.2), which is the minimum time window any protocol change needs to take effect. The relation $T_3 > T_1 + T_2$, introduced in Sec. 3.2, implies any depositor, in order to be sure to leverage the protection of an online AO, has a time window δ to judge if their participation in the BSA protocol should come to an end (e.g., by withdrawing funds with an unbonding operation) if the new protocol conditions are not satisfactory, where,

$$\delta = T_3 - (T_1 + T_2) \tag{6}$$

Importantly, time parameter T_3 is mutable, whereas, for an instance of the protocol, time parameters T_1 and T_2 are immutable: they are set in the locking scripts committed by addresses 3.3.1.

In order to keep the aforementioned relation valid, there should always be a timeframe within any T_3 window set by the TO where an AO can correctly validate, and, if needed, resolve challenges by the TO: depositor-challenged unbonding requests or rebalancing requests. The minimum amount of time it takes for a malicious TO to challenge a correct unbond and unfairly collect depositor funds is T_2. In fact, the unbonding challenge transaction by the TO could be mined in the same block as the unbonding request, de-facto skipping the T_1 wait time, i.e., the T_1 window is intended singularly for TO security, and does not impact AO availability requirements. Therefore, on rebalancing or challenged-unbonding requests T_2 is the only time parameter to take into account. Also, we include in our considerations the time t_{op} needed by the AO to process a challenge, i.e., the time it takes for the AO to formulate a decision using the predefined logic outlined in 5.4, and ensure, if required, that any action is correctly applied on the Bitcoin blockchain. Thus, an initial constraint about the AO availability uptime'_{AO} is:

$$\text{uptime}'_{AO} > \frac{T_3 - (T_2 - t_{\text{op}})}{T_3} \tag{7}$$

where it is an implicit requirement that $t_{\text{op}} < T_2$ otherwise the AO duties could never be fulfilled, and that uptime$'$ is calculated per T_3. This ensures that for any point in time t within a T_3 window from deployment by a TO, the AO must be online for at least t_{op} within the interval $[t, t + T_2]$.

E.2 Ethereum Consensus Constraints

The global timelock for the TO to change protocol rules T_3 is not the only critical protocol parameter BSA parties should monitor to ensure safety guarantees hold. In fact, the AO software running in the enclave relies on an Ethereum light client to validate protocol states from the Smart Account Registry and resolve challenges accordingly. This means that the inability to produce a reliable view of the Ethereum blockchain is the same as preventing an AO from fulfilling their arbitration tasks.

An AO can access the Ethereum state in a reliable way by initializing its light client (running inside the TEE) with a trusted checkpoint obtainable in two possible ways: from the TO by taking a checkpoint signed-off by Lombard's Security Consortium (LSC), or from a previous attested checkpoint produced by the AO.

Independently from the source, according to the Ethereum consensus specification [16], each finalized checkpoint can be considered a reliable view of the network state from the moment it is finalized for a time period called the *Weak Subjectivity Period* (WSP). After the WSP expires, the given checkpoint may expose the client to a class of attacks from a minority of validators. To prevent that, an offline AO could only restart using a trusted checkpoint not older than the current time minus a default minimum WSP. This mechanic impacts the uptime requirements of the AO.

At initial startup, an AO is deployed using an LSC trusted checkpoint, which can be validated in an independent manner by the AO operator before use. After initial startup, the AO can autonomously produce a new trusted checkpoint while it is operational. Thus, a requirement on the AO uptime is derived by the need to be online at least once during the WSP of the latest available trusted checkpoint, for an amount of time t_{check}, defined as the time it takes to synchronize with the network and produce an attestation of a finalized checkpoint.

$$\text{uptime}''_{AO} > \frac{t_{\text{check}}}{\text{WSP}} \tag{8}$$

In conclusion, the complete availability requirement for an AO to fulfill their 1-of-k duty, uptime_{AO} is:

$$\text{uptime}_{AO} = \max\{\text{uptime}'_{AO}, \text{uptime}''_{AO}\} \tag{9}$$

F AO Version Lifecycle

The AO operator is responsible for verifying accepted software versions posted by the TO have an expiry such that $\delta = T_3 - (T_1 + T_2) > 0$, ensuring depositor safety, see Guarantee 6.1. However, the TO is not constrained to ensure this relation holds for software version expiration. In fact, this relation is only enforced by the SAR where T_3 reflects SAR upgrades and adapter removal, while the TO, may set an arbitrary expiration for any new AO version.

We outline below safety measures taken by the AO to ensure (1) initialization is done according to protocol specifications and (2) an optimal upgrade choice is taken by AOs when accepting a new version posted by the TO.

F.1 AO Version Deployment Criterion

The AO operator finds the current parameter T_3 on the SAR. We consider software version expiration times t_{exp} as valid if $t_{\text{exp}} - t_{\text{present}} > T_3$.

F.2 AO Version Upgrade Criterion

Assume the current accepted version by an AO has expiry time t_{exp}. A new version posted by the TO with expiry time t'_{exp} will be accepted by the AO if,

$$t'_{\text{exp}} > t_{\text{exp}} \tag{10}$$

F.3 Depositor Time-Based Decision Criterion

If the depositor wishes to join a protocol with a specific online AO, they must ensure that

1. $T_1 + T_2 + T_{\text{op}} < T_3$ from SAR
2. $t_{\text{present}} + T_1 + T_2 + T_{\text{op}} < t_{\text{exp}}$ for AO expiration

where T_{op} is the maximal operational time it takes for the depositor to execute an unbonding transaction and have this confirmed on the Bitcoin network.

If the AO is online, and a depositor wishes to exit the protocol the depositor can only safely exit, provided the AO remains online, confirming an unbonding on Bitcoin before $t_{\text{exp}} - (T_1 + T_2)$.

References

[1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf, 2008.
Last accessed: March 2026

[2] Babylon Team. *A Bitcoin-Charged Crypto Economy with Trustless Vaults*. https://docs.babylonlabs.io/papers/trustless-bitcoin-vaults.pdf, 2025
Last accessed: March 2026

[3] Pieter Wuille, Jonas Nick, and Anthony Towns. *BIP 0341: Taproot: Segwit version 1 spending rules*. https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki, 2020.
Last accessed: March 2026

[4] Lightning Network. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. https://lightning.network/lightning-network-paper.pdf.
Last accessed: March 2026

[5] Robin Linus, Lukas Aumayr, Alexei Zamyatin, Andrea Pelosi, Zeta Avarikioti, and Matteo Maffei. *BitVM2: Bridging Bitcoin to Second Layers*. https://bitvm.org/bitvm_bridge.pdf.
Last accessed: March 2026

[6] Cubist. *Introducing the Bascule Drawbridge for Bitcoin Bridge Security*. https://cubist.dev/blog/introducing-the-bascule-drawbridge-for-bitcoin-bridge-security.
Last accessed: March 2026

[7] Chainlink. *CCIP Token Developer Attestation Is Now Generally Available*. https://blog.chain.link/token-developer-attestation-generally-available/.
Last accessed: March 2026

[8] Jeremy Rubin. *BIP 0119: OP_CHECKTEMPLATEVERIFY*. https://github.com/bitcoin/bips/blob/master/bip-0119.mediawiki, 2020.
Last accessed: March 2026

[9] Bitcoin Optech. *Anchor Outputs*. https://bitcoinops.org/en/topics/anchor-outputs/.
Last accessed: March 2026

[10] Bitcoin Optech. *Child Pays For Parent (CPFP)*. https://bitcoinops.org/en/topics/cpfp/.
Last accessed: March 2026

[11] Amazon Web Services. *AWS Nitro Enclaves*. https://aws.amazon.com/ec2/nitro/nitro-enclaves/.
Last accessed: March 2026

[12] Firecracker Project. *Firecracker: Secure and fast microVMs for serverless computing*. https://github.com/firecracker-microvm/firecracker.
Last accessed: March 2026

[13] Ethereum *Ethereum—The Merge*. https://ethereum.org/roadmap/merge/#what-is-the-merge.
Last accessed: March 2026

[14] Ethereum Foundation. *Altair Light Client—Light Client*. https://ethereum.github.io/consensus-specs/specs/altair/light-client/light-client/.
Last accessed: March 2026

[15] Amazon Web Services. *AWS Key Management Service*. https://aws.amazon.com/kms/.
Last accessed: March 2026

[16] Ethereum Foundation. *Phase 0 – Weak Subjectivity Guide*. https://ethereum.github.io/consensus-specs/specs/phase0/weak-subjectivity/.
Last accessed: March 2026

[17] Amazon Web Services. *Verifying the root of trust - AWS Nitro Enclaves*. https://docs.aws.amazon.com/enclaves/latest/user/verify-root.html.
Last accessed: March 2026

[18] Ethan Buchman, Kwon Jae, and Milosevic Zarko. "The latest gossip on BFT consensus." arXiv preprint arXiv:1807.04938 (2018).

[19] Ethan Heilman and Armin Sabouri. *BIP 0347: OP_CAT*. https://github.com/bitcoin/bips/blob/master/bip-0347.mediawiki.
Last accessed: March 2026