

Trustless Bitcoin Vaults

Lukas Aumayr^{*†}, Xinshu Dong[‡], Runchao Han[‡], Srivatsan Sridhar[§], David Tse^{¶||}, Nikhil Vanjani[¶]

^{*}University of Edinburgh, [†]Common Prefix, [‡]Babylon Labs, [§]UC Berkeley, [¶]Byzantine Research, ^{||}Stanford University

Abstract—Bitcoin (BTC) is the largest crypto asset, yet the Bitcoin blockchain offers few applications in which BTC can be used. A standard approach to make BTC useful is bridging: BTC is deposited on the Bitcoin chain and a wrapped asset is minted on a chain such as Ethereum, where it can be used in smart contract applications. Unfortunately, all Bitcoin bridges to date require trust assumptions on a centralized party or a committee running the bridge to enable withdrawals of the deposited BTC, including state-of-the-art BitVM bridges.

In this paper, we observe that while bridges are application-agnostic, most BTC holders are interested in collateral-based applications, where BTC is deposited as collateral, and only the depositor or a set of known counterparties can withdraw the asset. For this class of applications, we propose a primitive, Trustless Bitcoin Vaults, that guarantees that the BTC depositor or the counterparties can withdraw the deposited BTC following only the logic of the application smart contract without trusting any third party. We present a protocol that achieves the security guarantees of this primitive and instantiate it for a lending application where the deposited BTC is used as collateral to borrow against a set of liquidators on the Aave v4 smart contracts on Ethereum. Our mainnet experiments show that at the current Bitcoin price and fee rate, a deposit costs about \$0.7 in fees while a withdrawal costs at most \$7.7. Our construction is built on recent advances in efficient proof verification for Bitcoin.

1. Introduction

Bitcoin [1] is the most valuable crypto asset, yet most BTC remains outside smart-contract applications. The technical mismatch is familiar: Bitcoin offers strong settlement and a conservative scripting model, while lending markets, stablecoins, perpetual exchanges, and the rest of decentralized finance live on ledgers with expressive smart contracts. Today, BTC enters those applications mostly through custodial products or wrapped assets [2], [3], [4]. These systems are useful, but they ask Bitcoin holders to exchange native BTC custody for a claim on an issuer, a bridge, or a committee—a trade that sits uneasily with the ethos Bitcoin was founded on.¹

A running example makes the problem concrete. Bob holds 1 BTC and wants to borrow \$50K worth of stablecoins

against it from a lender, Larry. Were the collateral ETH instead, the two would not need this paper: Bob would deposit into a lending protocol such as Aave [6], draw the loan, and reclaim his collateral upon repayment, with every step enforced by contract code and no third party in the loop. With BTC as collateral, nothing comparable exists. The demand is substantial: Aave and Morpho hold roughly \$6B of WBTC and cbBTC as collateral, while roughly \$3B of native BTC secures proof-of-stake networks through Babylon [7], [8], [9].

The natural question is:

How can the \$1.3 trillion worth of BTC participate trustlessly in smart-contract applications on chains like Ethereum?

Why Bitcoin bridges need trust. The standard way to use BTC in Ethereum applications is a *wrapped asset bridge*: Bob locks BTC on Bitcoin, a fungible token is minted on Ethereum, Bob posts the token as collateral, and whoever the application finally entitles to the collateral redeems the token for BTC. Until recently, all such bridges were custodial or federated: a centralized custodian (WBTC, cbBTC) or a threshold committee (tBTC, Liquid) holds the deposited BTC outright, and every redemption rests on the honesty of that committee [2], [3], [4], [10].

The BitVM line of work [11], [12], [13], [14], [15], [16], [17], [18], [19] addresses one part of the problem: Bitcoin’s scripting language is too limited to check that Ethereum has authorized a withdrawal—even a succinct SNARK verifier would occupy roughly 900 MB of block space [13], equivalent to 225 Bitcoin blocks—so BitVM verifies proofs *optimistically*: a claimer merely asserts a proof, and a compact on-chain dispute is needed only against invalid claims; the state-of-the-art BABE [15] brings the on-chain challenge cost under \$40.

One might expect this machinery to finally make bridges trustless. It does not, because proof verification is not where all the trust hides. The deeper limitation is Bitcoin’s lack of *covenants*: a UTXO cannot constrain its own future spending, so there is no way to lock a coin, at deposit time, such that it later pays out to *whichever* party proves a valid claim. Every admissible spend must instead be fixed in advance as a transaction pre-signed for a *known* recipient. Bridges emulate the missing covenant with an *n*-of-*n* signer committee that pre-signs the admissible spending transactions at setup and is then trusted to delete its keys; and because the redeemer of a fungible token is unknown at peg-in time, it cannot be a recipient in one of the pre-signed paths, so a set of

1. In Satoshi Nakamoto’s initial post introducing Bitcoin to the world, he described it as follows: “It’s completely decentralized, with no central server or trusted parties, because everything is based on crypto proof instead of trust.” [5]

TABLE 1: Trust assumptions for Bob (depositor) and Larry (lender and liquidator) in collateralized lending: posting the wrapped asset minted by a BitVM3 bridge [20] as collateral versus using the TBV protocol introduced in this work. Lending using TBV is a strict improvement over lending using the BitVM3 bridge.

	Lending using BitVM3 Bridge		Lending using TBV	
	Bob	Larry	Bob	Larry
Cooperation needed for setting up loan	Larry and n -of- n signers	Bob and n -of- n signers	Larry	Bob
Trust for withdrawing BTC	1-of- m operators & 1-of- n signers	1-of- m operators & 1-of- n signers	Trustless	Trustless

m operators must front BTC to redeemers from their own liquidity and reimburse themselves through those paths [20].

In our running example, the implication is this: to set up the loan, Bob and Larry must rely not only on each other—unavoidable, since a loan takes two—but also on the signer committee; and to ever recover the BTC, each must trust at least one of the m operators and one of the n signers [14], [20]. The left half of Table 1 summarizes these trust assumptions.

Removing the trust. The observation driving this paper is that this residual trust is forced by the bridge *interface*, not by the proof machinery. A bridge offers open redemption to an arbitrary future holder; collateral applications do not need it. The parties who may ultimately own the BTC are known the moment it is deposited: Bob and Larry in our example. They can therefore pre-sign the admissible transactions *themselves*: the covenant that Bitcoin script cannot express, and that bridges emulate with a third-party signer committee, is replaced by the signatures of exactly the parties whose funds are at stake. No standing redemption service is needed: after setup, each of them can withdraw unilaterally by proving *on Bitcoin* that the application has transferred ownership to it. The right half of Table 1 is the result: setting up the loan now requires only the counterparty, who must cooperate anyway for the loan to exist, and withdrawal is trustless for both Bob and Larry. (See Corollary 1 in Section 5.) The same idea applies when Bob borrows from a decentralized lending market like Aave: in this case, instead of Larry, a set of liquidators represents the interests of Aave, repaying the debt and seizing the Bitcoin when the loan position becomes unhealthy. (See Corollary 2 in Section 5.) The idea also applies to other collateral applications such as Bitcoin-backed stablecoins or perpetual exchanges.

Trustless Bitcoin Vaults. We realize this idea as a *Trustless Bitcoin Vaults* (TBV) primitive and a protocol that achieves the primitive. A TBV is a Bitcoin UTXO bound at setup to an *application App*—a smart contract on an external chain—and to a finite claimer set: the depositor and the application-designated *vault keepers* App.Keepers, such as liquidators, auction participants, or arbitrageurs. The TBV protocol is agnostic to the application’s internal logic and consumes a single transaction from it: $\text{Closed}(V, P)$, emitted

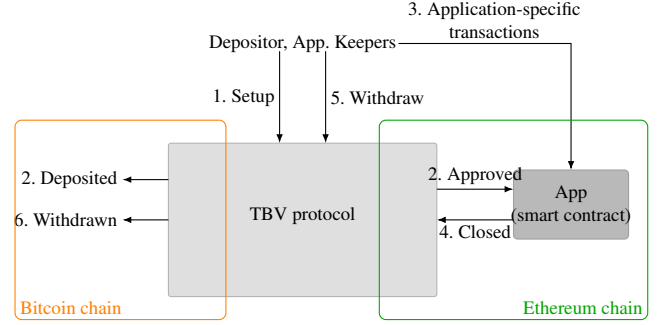


Figure 1: TBV protocol overview. The depositor and vault keepers invoke setup and withdraw on the TBV protocol, which spans Bitcoin (orange) and Ethereum (green). Application-specific transactions drive state in the App smart contract; the App signals vault approval and closure back to the protocol. Deposited/withdrawn BTC settle on Bitcoin.

TABLE 2: Comparison of the BitVM3 bridge and hybrid TBV-Bridge. The comparison scenario is for Larry, Bob and other users. Larry and Bob want to set up a BTC-backed loan between themselves but may want to participate in other applications later. Other users want to bridge to Ethereum to participate in other applications. All users are free to exchange the wrapped asset. The hybrid TBV-Bridge is a strict improvement over the pure bridge.

Users	BitVM3 Bridge			Hybrid TBV-Bridge		
	Bob	Larry	Others	Bob	Larry	Others
Cooperation needed for setting up BTC-backed loan	Larry & n -of- n signers	Bob & n -of- n signers	N/A	Larry & n -of- n signers	Bob & n -of- n signers	N/A
Trust for BTC deposit for others	N/A	N/A	n -of- n signers	N/A	N/A	n -of- n signers
Trust for withdrawing BTC	1-of- m operators & 1-of- n signers			Trustless		
						1-of- m operators & 1-of- n signers

when the application authorizes party P to withdraw the BTC from vault V . In the lending example, the application emits $\text{Closed}(V, \text{Bob})$ upon repayment and $\text{Closed}(V, \text{Larry})$ upon liquidation. Fig. 1 illustrates the interface; Sec. 4 gives the formal syntax. The TBV primitive is defined by five security properties—setup correctness, setup liveness, setup safety, withdrawal liveness, and withdrawal safety—where, e.g., withdrawal liveness states that an honest party that closes the vault in the application can withdraw the BTC, and withdrawal safety states that BTC cannot be withdrawn without the vault being closed. *Trustless* has a precise meaning here: each property protects a party as long as *that party alone* is honest, with no assumption on signers, operators, the counterparty, or anyone else. No existing Bitcoin bridge achieves this.

Hybrid TBV-Bridge resolves cost of trustlessness. Table 1 is silent about everyone except Bob and Larry, and that silence is exactly what a pure TBV pays for trustlessness: there is no fungible token, so users outside the claimer set cannot participate at all. Our second contribution, the *hybrid TBV-Bridge*, simultaneously supports users who want

trustlessness and users who simply want wrapped BTC for arbitrary applications. Table 2 compares the bridge with the hybrid in terms of trust assumptions. The hybrid is *not* a TBV and a bridge running side by side: all parties hold the *same* fungible wBTC token and can freely exchange it with one another. When Bob deposits, he receives freshly minted wBTC. As long as he uses it only inside the designated application, he and Larry retain the trustless withdrawal of a pure TBV (last row of Table 2, middle); any wBTC that leaves the application—transferred, sold, or composed into other DeFi protocols—remains redeemable by its holder with the help of an operator, exactly as in a bridge (last row, right).

To see why exchanging tokens with *Others* matters even inside Bob and Larry’s loan application, consider how liquidation works in practice. In the simple loan of Table 1, Larry liquidates for himself; real lending markets instead rely on third-party *professional liquidators*, who compete to repay unhealthy debt the moment it appears and who operate with little capital of their own. A pure TBV is hostile to this role: a liquidator must be named in the claimer set at setup, must front Bob’s entire stablecoin debt from its own pocket, and then must wait out the days-long dispute window of the Bitcoin withdrawal before recovering any value. The hybrid removes every one of these obstacles. Because the seized collateral is fungible wBTC rather than a claim on a specific UTXO, *any* liquidator can act, with zero capital, atomically: take a USDC flash loan, repay Bob’s debt, receive the wBTC collateral, swap it for USDC on a DEX, repay the flash loan, and pocket the liquidation bonus—all within a single Ethereum transaction [21]. The DEX traders left holding the wBTC are precisely the *Others* of Table 2, free to redeem through an operator. A pure bridge supports this flash flow too. What it cannot offer is the happy path: on a bridge, Bob repaying his loan and recovering his own BTC still depends on 1-of- m operators and 1-of- n signers (Table 2, left). In the hybrid, that path is trustless. Every user thus gets the strongest guarantee compatible with how they use their tokens, rather than everyone inheriting the trust assumptions of the most demanding use case.

Key challenge: per-vault versus global accounting. The flash-liquidation flow above is also exactly what makes the hybrid non-trivial to build: fungibility makes vaults interact. The wBTC minted for Bob’s vault may end up with an unrelated DEX trader who redeems it against a *different* vault, so security can no longer be argued one vault at a time, as it can in a pure bridge (where vaults are interchangeable) or a pure TBV (where they are independent). Our *accounting layer* (Sec. 8) resolves this with a single notion of *backing*: Bob’s vaults remain reserved for his trustless withdrawal exactly while his token balance and collateral cover them; once he spends wBTC outside the application, vaults become free, in equal number, for bridge-style redemption. We construct the hybrid as an application HybridApp(innerApp) on top of TBV for an inner application, e.g., the lending contract.

Contributions. In summary, this paper makes the following contributions:

- **The TBV primitive (Sec. 4).** We define Trustless Bitcoin Vaults, an application-agnostic primitive for using BTC as collateral in smart-contract applications on external chains, together with five security properties under which each party is protected as long as it alone is honest.
- **Construction and security proofs (Sec. 5).** We construct TBV using the state-of-the-art BABE proof verification protocol [15] and prove all five security properties.
- **The hybrid TBV-Bridge (Sec. 8).** We combine TBV trustlessness with bridge fungibility through an accounting layer whose backing invariant determines, globally across all vaults, which vaults remain reserved for trustless withdrawal and which become free for bridge-style redemption.
- **Implementation and mainnet evaluation (Secs. 6 and 7).** We implement the full stack—a BTCVaultRegistry contract, an adapter that integrates vault collateral into Aave v4² [6] without modifying Aave, and withdrawal proofs of Ethereum finality via a Helios [22] light client—and deploy it end-to-end on Bitcoin and Ethereum mainnet. A deposit costs about \$0.7 in Bitcoin fees, an honest withdrawal at most \$7.7, and blocking an invalid claim about \$36; off-chain storage is 84 MB per claimer, and TBV adds no on-chain or off-chain overhead over the underlying BABE protocol.

2. Related Work

Trustless solutions for specific types of applications.

Payment-channel protocols such as Lightning [23] introduced Bitcoin-backed trustless off-chain payments, but do not support general applications. In the spirit of state channels for Ethereum [24], Generalized Channels [25] extend Bitcoin channels beyond payments by supporting off-chain execution of arbitrary Bitcoin-script functionality. However, their expressiveness remains constrained by Bitcoin Script. Other approaches move on-chain functionality off-chain using trusted execution environments (TEEs) [26], [27], [28], or replace TEEs with MPC-based techniques [29], [30], which require additional collateral. All these approaches target two-party settings, or require some k -of- n threshold assumption, e.g., [31].

BitML [32] and BitMLx [33] provide a language for specifying Bitcoin contracts and unrolling them into Bitcoin transactions. They can trustlessly express simple contracts that are programmable directly on Bitcoin; BitMLx provides a loan example, but relies on a trusted mediator.

Babylon’s Bitcoin staking [34], [35] allows a staker to deposit BTC as collateral for validating a proof-of-stake chain. As in TBV, the staker can trustlessly withdraw BTC. However, the protocol can only enforce slashing conditions

2. Aave is the largest DeFi lending protocol by total value locked (over \$12 B as of this writing) and Aave v4 is its latest version.

based on double-signing, and cannot enforce liquidation or other application-specific withdrawal conditions.

Ownership-preserving Bitcoin applications. Cardinal [19] introduces a trust-minimized BitVM-based bridge that preserves coin identity: users withdraw the same UTXOs they deposited. Compared to other BitVM-based bridges, this prevents a user from receiving tainted UTXOs during withdrawal. TBV also achieves ownership preservation.

Bitcoin-backed lending. Atomic Loans [36], DLC-based loans [37], and practical multisig/escrow lending systems [38], [39], [40] keep BTC on Bitcoin, but settle using atomic-swap logic, oracle attestations, or trusted escrows. Recent cryptographic loan protocols [41] compile the loan into a finite state machine, where repayment transactions unlock encrypted signatures for the corresponding collateral distribution. Bitcoin Smart Accounts [42] instead target DeFi integration, using TEE-based arbitration to let BTC-backed tokens be used for lending. These approaches are practical, but they rely on bespoke oracles, TEEs, or trusted mediator or escrow assumptions.

3. Preliminaries

We present informal preliminaries here; formal definitions are in Appendix D.

Model. Parties communicate over secure, authenticated channels. Communication is assumed to be synchronous, i.e., messages are delivered within a known bounded time delay. All parties run probabilistic polynomial-time machines. We model hash functions $\text{Hash}(s)$ as a random oracle, and digital signatures as EUF-CMA secure. An adaptive adversary $\mathcal{A}$ may corrupt any party and access the full state of that party’s machine, and send and receive messages on its behalf. The adversary is Byzantine, i.e., may deviate arbitrarily from the protocol.

Ledgers. Time is measured in rounds, and I denotes the polynomially bounded set of rounds of an execution. Let $\mathcal{L}_{BTC}^{P,r}$ be the Bitcoin ledger and $\mathcal{L}_{ETH}^{P,r}$ the external application ledger at round r in party P ’s view. Whenever the party/round is not important, we simply write $\mathcal{L}_{BTC}$ and $\mathcal{L}_{ETH}$, respectively. We assume both ledgers satisfy safety, validity, and liveness with confirmation delays, as in standard analyses of Nakamoto-style settlement and smart-contract ledgers [44], [45].

Bitcoin. The ledger’s state is represented as a set of unspent transaction outputs (UTXOs). Each UTXO is a pair $\text{out} = (a, \text{lockScript})$ where $a \in \mathbb{R}_{\geq 0}$ is the amount of coins (in BTC) in that UTXO, and lockScript is a program (the *locking script*) that determines under which conditions the UTXO can be spent. In BABE, we widely use Taproot Trees [46], or Taptrees, which make a UTXO spendable by satisfying one of multiple locking scripts. We will represent the locking script of such a UTXO as $\text{lockScript} = \langle \text{leaf}_0, \dots, \text{leaf}_{k-1} \rangle$. A transaction $\text{tx} = (\text{inputs}, \text{outputs}, \text{tx_witnesses})$ consumes UTXOs inputs and creates new UTXOs outputs and is valid if it contains tx_witnesses satisfying the locking

scripts of inputs. The pair $\overline{\text{tx}} = (\text{inputs}, \text{outputs})$ (without tx_witnesses) is called the *transaction skeleton*.

The locking scripts of Bitcoin that are used in this paper are: signature check $\text{CheckSig}(\text{pk})$ (requires a signature on the transaction in the witness), $\text{RelTimelock}(\tau)$ (requires at least τ blocks before unlocking), $\text{HashLock}(h)$ (requires a hash pre-image of h). Using the above scripts and logical operators $\wedge$ (AND) and $\vee$ (OR), we can also define $\text{CheckLampSig}(\text{lpk})$ (requires a Lamport signature on some message) and $\text{CheckLampSigsMatch}(\text{lpk}_A, \text{lpk}_B)$ (requires Lamport signatures for each public key on the same message). See Appendix D.2 for a formal definition of Lamport signatures and their Bitcoin scripts.

BITVM-CORE. We use a multiple verifier extension of BITVM-CORE [15]. Appendix E describes a protocol that extends the BABE protocol [15] to multiple verifiers.

A BITVM-CORE protocol is an interactive protocol run between a Prover P and a set of Verifiers $\mathcal{V}$ interacting with the Bitcoin ledger in two phases: the setup phase and proving phase. A BITVM-CORE protocol for a relation $\mathcal{R}$ consists of the following *p.p.t.* algorithms:

- $\text{Gen}(\mathcal{R}) \rightarrow \text{crs}$: Takes as input the relation and outputs a common reference string crs .
- $P_{\text{Setup}}(\text{crs}, x)$: Run by the Prover on a statement x during the setup phase, interacts with the Verifiers and the Bitcoin ledger, and outputs the Prover’s state st_P and a set of success transactions $\mathcal{S}$. Alternatively, the Prover may output $\perp$ to indicate that the setup failed.
- $V_{\text{Setup}}(\text{crs}, x)$: Run by each Verifier during the setup phase, interacts with the Prover and the Bitcoin ledger, and outputs the Verifier’s state st_v and a set of success transactions $\mathcal{S}$. Alternatively, the Verifier may output $\perp$ to indicate that the setup failed.
- $P_{\text{Prove}}(\text{st}_P, w)$: Run by the Prover knowing a witness w , interacts with the Bitcoin ledger, and outputs a bit $b \in \{0, 1\}$ ($b = 1$ indicates success of the proving phase).
- $V_{\text{Prove}}(\text{st}_v)$: Run by each Verifier during the proving phase, interacts with the Bitcoin ledger.

Robustness ensures that as long as the Prover is honest, has accepted that the setup was performed correctly, and has a valid witness, malicious Verifiers cannot prevent the Prover from succeeding (i.e., one of the transactions in $\mathcal{S}$ appears on the ledger) except with negligible probability. Thus, the worst thing a malicious Verifier can do is to cause the Prover to abort the setup, in which case no Bitcoin is ever lost by any party. Knowledge soundness ensures that if any honest Verifier accepts the setup, then an adversary controlling the Prover and all other verifiers cannot successfully include any transaction from $\mathcal{S}$ on Bitcoin unless he knows a valid witness for the chosen relation. Together, robustness and knowledge soundness make a BITVM-CORE protocol *trustless*.

Atomic Swaps. An atomic cross-chain swap is a protocol by which two or more parties exchange assets across distinct blockchains without a trusted intermediary [47], [48]. Security requires that if both parties are honest, they both

receive the expected assets, and if one party is dishonest, the honest party does not lose assets.

Ethereum. A *smart contract* $\mathcal{C}$ is a stateful program with a fixed address $a \in \{0, 1\}^{160}$. A contract exposes an interface of typed *functions*; each function may read or update its state, transfer Ether, invoke functions of other contracts (*internal calls*, which complete before the caller resumes), and *emit events*. An event $E(\dots)$ is a log entry permanently recorded on the blockchain and publicly observable. We say $E(\dots) \in \mathcal{L}_{\text{ETH}}$ if the transaction that emitted the event is in $\mathcal{L}_{\text{ETH}}$.

ERC-20 [49] is the standard interface for *fungible tokens* on Ethereum. An *ERC-20 token contract* has state variables `bal` (token balance of each address) and `allow` (spending allowance from each owner to each spender), and `totalSupply` (total token supply), and exposes functions `balanceOf(owner)`, `transfer(to, v)`, `approve(spender, v)`, and `transferFrom(owner, to, v)`. Token contracts commonly designate a *minter* address authorized to call `mint(to, v)`, and any user can call `burn(spender, v)`.

Ethereum Light Client. An *Ethereum light client* is a p.p.t. algorithm $\text{LC}(\text{tx}, \mathcal{L}, w) \rightarrow \{0, 1\}$ that takes a transaction tx , a ledger $\mathcal{L}$, and a witness w , and outputs 1 iff it accepts tx as confirmed in $\mathcal{L}$. It satisfies safety, liveness, and timeliness.

4. TBV Primitive

In this section, we formally define the trustless Bitcoin Vault (TBV) primitive. A TBV is a general-purpose primitive for building applications that use Bitcoin as collateral. We state the model and syntax of the TBV interface, security properties, and example applications.

Notation. We write V for a vault identifier and $V.\text{Dep}$ for the depositor of vault V .

External Application. In our approach, while the external application can have its internal logic, we require that its interface with the TBV protocol satisfies some properties. Formally, an application is a tuple $\text{App} = (\text{Keepers}, \text{Closed})$, where `Keepers` is a set of vault keepers, such as lenders, liquidators, arbitrageurs, or other application-designated keepers that will interact with the TBV protocol and may later claim or challenge a vault, and `Closed` is an event emitted by the application contract with parameters vault identifier V and party P (the same application instance may govern multiple vaults, each identified by a distinct V). $\text{Closed}(V, P)$ where $P \in \{V.\text{Dep}\} \cup \text{Keepers}$ is an event that indicates that the vault with id V has been closed on the external application ledger $\mathcal{L}_{\text{ETH}}$ by party P . The confirmation of this event authorizes P to withdraw the UTXO for vault V from the Bitcoin ledger.

Actors. The parties that participate in the vault protocol are

- *The Depositor:* The party which initially funds the vault and who wishes to participate in the vault application, and may later want to lay claim to or challenge vault keepers' claims to the vault. We call the depositor `Dep`.
- *The Vault Keepers:* The vault application names a finite set of vault keepers `Keepers`, which, depending on the

application, can act as lenders, liquidators, arbitrageurs, or other application-specific actors, that may later lay claim to or challenge the depositor's claims to a vault.

We refer to $\{V.\text{Dep}\} \cup \text{Keepers}$ as *the claimer set* for vault V .

Syntax. A TBV protocol for `App` is a tuple $\text{TBV}[\text{App}] = (\text{Setup}, \text{Withdraw}, \text{Deposited}, \text{Approved}, \text{Withdrawn})$, where `Setup` and `Withdraw` are procedures that parties can invoke, `Deposited` and `Withdrawn` are transactions defined on the Bitcoin ledger $\mathcal{L}_{\text{BTC}}$, and `Approved` is an event defined on the application ledger $\mathcal{L}_{\text{ETH}}$, as follows:

- `Setup[Dep, Keepers]` is a procedure that `Dep` can invoke to set up a vault with the help of the keepers in `Keepers`. Its common input is the deposit amount v together with the public keys of the claimer set $\{V.\text{Dep}\} \cup \text{Keepers}$, and each party's private input is its own signing key. On success all parties output a common vault identifier V ; any party may instead output $\perp$, aborting the setup. Its side effects are the inclusion of `Deposited(V)` on $\mathcal{L}_{\text{BTC}}$, the emission of `Approved(V)` on $\mathcal{L}_{\text{ETH}}$, and the local state each party retains for use by `Withdraw`;
- `Withdraw` is a procedure that any party P among `Dep` and `Keepers` can invoke to attempt to withdraw from the vault. Its input is a vault identifier V together with the local state P retained from `Setup`, and its output is a bit recording if the attempt succeeded; its side effect on success is the inclusion of `Withdrawn(V, P)` on $\mathcal{L}_{\text{BTC}}$;
- `Deposited(V)` is a transaction that creates the vault with id V on the Bitcoin ledger $\mathcal{L}_{\text{BTC}}$;
- `Approved(V)` is an event indicating that a vault with id V is approved on the external application ledger $\mathcal{L}_{\text{ETH}}$;
- `Withdrawn(V, P)` where $P \in \{V.\text{Dep}\} \cup \text{Keepers}$ is a transaction by which P withdraws the vault UTXO V on the Bitcoin ledger $\mathcal{L}_{\text{BTC}}$.

Remark 1. We highlight that the `Withdraw` procedure is allowed to read whether the `Closed` event has occurred on the external application ledger $\mathcal{L}_{\text{ETH}}$. Further, the external application `App` is allowed to read whether `Approved` has occurred on the external application ledger $\mathcal{L}_{\text{ETH}}$. However, the TBV protocol itself only has the above interface with the application, and does not need to know about the internal logic of the application or when it will execute `Closed`. The TBV protocol only requires an application to have a mild soundness property (Def. 1). This allows TBVs to be used as a general-purpose primitive for building various applications that use Bitcoin as collateral, without requiring any specific design for the application. The interface between the TBV protocol and the application is illustrated in Fig. 1.

Definition 1 (Application soundness). For all valid ledgers $\mathcal{L}_{\text{ETH}}$, for all vault identifiers V :

- if $\text{Closed}(V, P) \in \mathcal{L}_{\text{ETH}}$, then $\text{Approved}(V) \in \mathcal{L}_{\text{ETH}}$,
- at most one $\text{Closed}(V, \cdot)$ event appears in $\mathcal{L}_{\text{ETH}}$.

Security Properties. Each security property below is a predicate over the ledger views of honest parties, parameterized by an explicit time bound T_i and by the honesty of the

particular party whose guarantee is at stake. Appendix A proves that the construction of Sec. 5.1 satisfies all five.

Definition 2 (T_1 -Setup correctness). Suppose that Dep invokes Setup with Keepers at round r . If Dep and all keepers in Keepers are honest, then there exists a vault identifier V such that for all honest parties P_1 of $\mathcal{L}_{BTC}$ and P_2 of $\mathcal{L}_{ETH}$, and for all rounds $r' \geq r + T_1$, we have $\text{Deposited}(V) \in \mathcal{L}_{BTC}^{P_1, r'}$ and $\text{Approved}(V) \in \mathcal{L}_{ETH}^{P_2, r'}$.

Definition 3 (T_2 -Setup liveness). Suppose that honest Dep invokes Setup with Keepers for vault identifier V . Then for all honest parties P_1 of $\mathcal{L}_{BTC}$ and P_2 of $\mathcal{L}_{ETH}$, and for all rounds $r_1, r_2 \in I$ with $r_2 \geq r_1 + T_2$, if $\text{Deposited}(V) \in \mathcal{L}_{BTC}^{P_1, r_1}$ then $\text{Approved}(V) \in \mathcal{L}_{ETH}^{P_2, r_2}$.

Definition 4 (T_3 -Setup safety). For all vault identifiers V , if one of $\{V.\text{Dep}\} \cup \text{Keepers}$ is honest, then for all honest parties P_1 of $\mathcal{L}_{BTC}$ and P_2 of $\mathcal{L}_{ETH}$, and for all rounds $r_1, r_2 \in I$ with $r_1 \geq r_2 + T_3$, if $\text{Approved}(V) \in \mathcal{L}_{ETH}^{P_2, r_2}$, then $\text{Deposited}(V) \in \mathcal{L}_{BTC}^{P_1, r_1}$.

Definition 5 (T_4 -Withdrawal liveness). For all vault identifiers V , for all $P \in \{V.\text{Dep}\} \cup \text{Keepers}$, if P is honest, then for all honest parties P_1 of $\mathcal{L}_{BTC}$ and P_2 of $\mathcal{L}_{ETH}$, and for all rounds $r_1, r_2 \in I$ with $r_1 \geq r_2 + T_4$, if $\text{Closed}(V, P) \in \mathcal{L}_{ETH}^{P_2, r_2}$ then $\text{Withdrawn}(V, P) \in \mathcal{L}_{BTC}^{P_1, r_1}$.

Definition 6 (T_5 -Withdrawal safety). For all vault identifiers V , for all $P \in \{V.\text{Dep}\} \cup \text{Keepers}$, if P is honest, then for all $P' \in \{V.\text{Dep}\} \cup \text{Keepers}$, for all honest parties P_1 of $\mathcal{L}_{BTC}$ and P_2 of $\mathcal{L}_{ETH}$, and for all rounds $r_1, r_2 \in I$ with $r_2 \geq r_1 + T_5$, if $\text{Withdrawn}(V, P') \in \mathcal{L}_{BTC}^{P_1, r_1}$, then $\text{Closed}(V, P') \in \mathcal{L}_{ETH}^{P_2, r_2}$.

It is *trustless* in the sense that setup safety, withdrawal liveness, and withdrawal safety each hold for a given party as long as that party alone is honest; no party need trust any other. These goals create a trade-off between setup liveness and withdrawal safety. A k -of- n setup improves liveness, but an omitted keeper cannot have trustless withdrawal safety. We prioritize safety because a safety failure can lead to a loss of funds, whereas a failed setup lets the depositor refund the BTC.

Relationship to the bridge primitive. A *unidirectional bridge* is a primitive that relays transactions included on a source ledger to a destination ledger (cf. [20], restated in Appendix D.7). Colloquially, the term “bridge” is used to mean a *wrapped asset bridge* (such as the BitVM3 bridge [20]) which consists of two unidirectional bridges:

- a peg-in bridge that relays Deposited transactions from Bitcoin (user locks BTC) to Minted transactions on Ethereum (bridge mints a wrapped BTC asset and transfers it to the user),
- a peg-out bridge that relays peg-out requests (user gives up his wrapped BTC) on Ethereum to transactions paying BTC to the corresponding user on Bitcoin.

In that sense, the TBV primitive too consists of two unidirectional bridges:

- a setup bridge that relays Deposited transactions on Bitcoin to Approved events on Ethereum,
- a withdrawal bridge that relays Closed events on Ethereum to Withdrawn transactions on Bitcoin.

The security properties of a wrapped asset bridge are that in both the peg-in and peg-out bridges, when a transaction occurs on one of the ledgers, the corresponding transaction must occur on the other ledger. Similarly, the TBV safety and liveness properties guarantee that when a transaction occurs on one ledger, the corresponding transaction occurs on the other ledger.

A *trustless* wrapped asset bridge primitive is one in which peg-in and peg-out can be done trustlessly without relying on other parties. It is a powerful primitive but unfortunately there are no known constructions that achieve it. BitVM bridges have tried to achieve this powerful primitive but unnatural trust assumptions are needed on some external committees. The TBV primitive instead *restricts* the trustless bridge primitive’s power by only allowing a *known* set of parties to unilaterally withdraw, the set known at deposit time. This restriction enables trustlessness, but at the same time is useful enough for many applications.

4.1. Example Applications

Recall from Sec. 4 that the TBV protocol reads only the application’s $\text{Closed}(V, P)$ event, so specifying an application reduces to giving the rules under which it emits $\text{Closed}(V, P)$. We illustrate this with three applications – a lending vault, a BTC-backed stablecoin, and a perps DEX. We use $\text{VaultOwner}(V)$ informally here to denote the party currently entitled to withdraw vault V ($\perp$ if no one is, e.g., while the BTC backs an outstanding loan); it is defined formally in Sec. 8. $\text{Closed}(V, P)$ is emitted when the current owner P requests to close vault V . Table 5 gives the conditions for each application’s VaultOwner transitions.

Lending. In an over-collateralized lending vault, the depositor Dep deposits BTC as collateral and borrows loan tokens against it, while the application receives loan tokens from lenders. The application tracks the health of the borrowing position via its loan-to-value ratio (LTV), defined as the current value of the total loan (including interest) divided by that of the total collateral, against a liquidation loan-to-value threshold (LLTV); the position becomes unhealthy when LTV exceeds LLTV. The parties in Keepers act as liquidators. The application tracks vault ownership as follows: $\text{VaultOwner}(V) \leftarrow V.\text{Dep}$ upon $\text{Approved}(V)$ and after Dep repays the full debt, $\text{VaultOwner}(V) \leftarrow \perp$ when Dep borrows against the collateral, and $\text{VaultOwner}(V) \leftarrow P \in \text{Keepers}$ when P repays the full debt and $\text{LTV} > \text{LLTV}$.

Stablecoin. A BTC-backed stablecoin protocol works like the lending application, except that the debt – stablecoins – is minted by the application rather than received from lenders, and accrued stability fees play the role of interest. For simplicity, we reuse LTV and LLTV for the position’s debt-to-value ratio and its liquidation threshold. The keepers are auction participants: an unhealthy position is opened for

auction, and the winner that burns the full debt becomes the owner, whereas Dep redeems a healthy position by burning the full debt himself.

Perp DEX. A perpetual decentralized exchange (Perp DEX) trades perpetual futures – cash-settled derivative contracts for speculating on asset prices without an expiry date – with profit and loss (P&L) settled in quote assets such as USD stablecoins. Here Dep deposits BTC as margin for his trades. As long as his margin value (M) is no less than the maintenance margin requirement (MMR), he can close his position, settle the P&L using the quote assets, and redeem the BTC. Once M drops below MMR, any keeper of the DEX can close the position, cover the resulting P&L obligation to the counterparty (or the liquidity pool), and thereby become the vault owner.

5. TBV Protocol

5.1. Construction

We instantiate the TBV interface $\text{TBV}[\text{App}] = (\text{Setup}, \text{Withdraw}, \text{Deposited}, \text{Approved}, \text{Withdrawn})$ of Sec. 4 from two building blocks introduced in Sec. 3:

- 1) a multi-verifier BITVM-CORE protocol $\Pi_{\text{BITVM-CORE}} = (\text{Gen}, P_{\text{Setup}}, V_{\text{Setup}}, P_{\text{Prove}}, V_{\text{Prove}})$ (Appendix D.3); and
- 2) an Ethereum light client LC (Appendix D.6).

The application $\text{App} = (\text{Keepers} = (K_1, \dots, K_n), \text{Closed})$ is deployed as a smart contract $\mathcal{C}_{\text{App}}$ on $\mathcal{L}_{\text{ETH}}$.

Parameters and notation. Throughout, λ denotes the security parameter. The construction is parameterized by three *protocol timeouts* and one *confirmation depth*, all fixed at deployment and *not* inherited from any building block: T_{activate} (Ethereum blocks), the deadline for Dep to activate the vault (Phase 6); T_{stale} (Bitcoin blocks), the greatest depth of $\text{tx}_{\text{PreDeposit}}$ at which the BITVM-CORE setups may still begin (Phase 3); T_{refund} (Bitcoin blocks), the relative timelock on the refund leaf (Phases 1 and 8); and k (Bitcoin blocks), the confirmation depth each keeper awaits before acknowledging the deposit (Phase 5). They must satisfy the compatibility conditions (C1)–(C2) of Appendix A, which require each timeout to outlast the protocol steps it covers. The parameters $T_1, \dots, T_5$ of the security definitions (Defs. 2 to 6) are *derived* bounds rather than protocol inputs: they are expressed purely in the liveness and delay parameters of the ledgers and of the building blocks, and are stated in Thms. A.1 to A.5. The three timeouts above enter the analysis only through (C1)–(C2), and none of them appears in any T_i . The confirmation depth k is the exception: honest keepers wait for depth k in Phase 5, so it appears directly in T_1 , as the term k/τ_{BTC} .

The Vault Manager $\mathcal{C}_{\text{VM}}$. On the Ethereum side, the TBV protocol itself deploys a singleton smart contract $\mathcal{C}_{\text{VM}}$ (the *Vault Manager*) that coordinates the cross-chain setup. $\mathcal{C}_{\text{VM}}$ maintains per-vault state indexed by vault ID V and exposes the following methods, invoked during Setup (Sec. 5.1.1): `registerDeposit`, `postDepositSignatures`, `postACK`,

`activateVault`. It emits the events `DepositRegistered(V)`, `DepositSignaturesPosted(V)`, `ACKAccepted(V)`, and the event `Approved(V)`. The event `DepositRegistered(V)` doubles as the public assignment of the vault identifier $V \in \{0, 1\}^\lambda$. We identify each emitted event with the inclusion on $\mathcal{L}_{\text{ETH}}$ of the corresponding $\mathcal{C}_{\text{VM}}$ transaction.

The BITVM-CORE statement. For a vault V and a claimer (prover) $P \in \{V.\text{Dep}\} \cup \text{Keepers}$, the BITVM-CORE protocol is set up for the NP relation

$$\mathcal{R}_{V,P} := \left\{ (x, w) : \begin{array}{l} x = (V, P) \wedge w = (\mathcal{L}_{\text{ETH}}^*, w) \wedge \\ \text{LC}(\text{Closed}(V, P), \mathcal{L}_{\text{ETH}}^*, w) = 1 \end{array} \right\},$$

where $\mathcal{L}_{\text{ETH}}^*$ is an Ethereum ledger and w is the witness consumed by LC. A witness for $\mathcal{R}_{V,P}$ therefore certifies that the application contract has emitted `Closed(V, P)`.

5.1.1. The Setup procedure. `Setup[Dep, Keepers]` is invoked jointly by Dep and Keepers. The procedure proceeds in eight phases that interleave off-chain coordination with on-chain steps on $\mathcal{L}_{\text{BTC}}$ and $\mathcal{L}_{\text{ETH}}$ (Fig. 2a). The on-chain steps realize an atomic swap between Bitcoin and Ethereum, binding the release of the depositor’s BTC into a vault UTXO on Bitcoin to the emission of `Approved(V)` on Ethereum.

Inputs. The common input is the deposit amount v , the application contract $\mathcal{C}_{\text{App}}$ with its keeper set `Keepers`, and the public keys $\text{pk}_{\text{Dep}}, \text{pk}_{K_1}, \dots, \text{pk}_{K_n}$ of the claimer set; each party’s private input is its own signing key, and Dep additionally holds the BTC to be deposited. *Outputs.* On success every party outputs the vault identifier V assigned in Phase 1; any party may instead output $\perp$ to abort, in which case Dep recovers the deposit through the refund path of Phase 8. *Side effects.* `Deposited(V)` is included on $\mathcal{L}_{\text{BTC}}$ and `Approved(V)` is emitted on $\mathcal{L}_{\text{ETH}}$; each party retains the local state recorded after Phase 8, which `Withdraw` consumes.

Phase 1 – Deposit registration. Dep samples a fresh secret $s \xleftarrow{\$} \{0, 1\}^\lambda$ and computes $h := \text{Hash}(s)$. Dep then constructs the skeletons of the two Bitcoin transactions that the swap will use:

- the *pre-deposit* transaction $\text{tx}_{\text{PreDeposit}}$, funded from Dep’s own UTXOs and locking $v \text{ BTC}$ into a Taproot output with two leaves:
 - the *deposit leaf* $\text{HashLock}(h) \wedge \text{CheckSig}(\text{pk}_{\text{Dep}}) \wedge \bigwedge_{K \in \text{Keepers}} \text{CheckSig}(\text{pk}_K)$;
 - the *refund leaf* $\text{RelTimelock}(T_{\text{refund}}) \wedge \text{CheckSig}(\text{pk}_{\text{Dep}})$;
- the *deposit* transaction $\text{tx}_{\text{Deposit}}$, spending the deposit leaf of $\text{tx}_{\text{PreDeposit}}$ and creating the vault UTXO whose locking script is the single leaf

$$\text{CheckSig}(\text{pk}_{\text{Dep}}) \wedge \text{CheckSig}(\text{pk}_{K_1}) \wedge \dots \wedge \text{CheckSig}(\text{pk}_{K_n}).$$

Pre-signed spending transactions for this leaf, one per potential claimer (prover) P , are produced in Phase 3.

Dep calls $\mathcal{C}_{\text{VM}}.\text{registerDeposit}(\text{tx}_{\text{PreDeposit}}, \text{tx}_{\text{Deposit}}, h, \text{pk}_{\text{Dep}})$. $\mathcal{C}_{\text{VM}}$ validates the input, stores the two transaction skeletons on $\mathcal{L}_{\text{ETH}}$, where they are publicly readable by every party

(Remark 2 (i)), assigns a fresh identifier V to the vault, and emits the event $\text{DepositRegistered}(V)$.

Phase 2 – $\text{tx}_{\text{PreDeposit}}$ on Bitcoin. After $\text{DepositRegistered}(V)$ appears on $\mathcal{L}_{\text{ETH}}$, Dep signs $\text{tx}_{\text{PreDeposit}}$ and writes it to $\mathcal{L}_{\text{BTC}}$.

Phase 3 – Per-claimer BITVM-CORE setup. Each claimer (prover) $P \in \text{Keepers}$ monitors $\mathcal{L}_{\text{ETH}}$ for $\text{DepositRegistered}(V)$ and $\mathcal{L}_{\text{BTC}}$ for the inclusion of $\text{tx}_{\text{PreDeposit}}$. Once $\text{tx}_{\text{PreDeposit}}$ is at least 1 block deep on a Bitcoin chain and not more than T_{stale} blocks deep, all parties run the multi-verifier BITVM-CORE setup phase; Remark 2 (ii) explains both bounds. Specifically, for each $P \in \{V.\text{Dep}\} \cup \text{Keepers}$ the parties jointly execute

$$\text{crs}_P \leftarrow \text{Gen}(\mathcal{R}_{V,P}),$$

$$\begin{aligned} &(\mathcal{S}_{V,P}, \text{st}_P^{V,P}, \{\text{st}_v^{V,P}\}_{v \in \mathcal{V}_P}) \\ &\leftarrow \langle P_{\text{Setup}}(\text{crs}_P, (V, P)), \mathcal{V}_{P,\text{Setup}}(\text{crs}_P, (V, P)) \rangle, \end{aligned}$$

where P plays the Prover and the verifier set $\mathcal{V}_P := (\{V.\text{Dep}\} \cup \text{Keepers}) \setminus \{P\}$ plays the multi-verifier of $\Pi_{\text{BITVM-CORE}}$. The setup pre-signs every transaction in $\mathcal{S}_{V,P}$ with Dep's signature and every keeper's signature, so that it spends the future vault UTXO under its multi-sig leaf and pays the unlocked BTC to P . Any party may abort if its local view of any of the $1 + |\text{Keepers}|$ setup transcripts is invalid. This ordering limits depositor griefing: keepers incur the expensive setup cost only after $\text{tx}_{\text{PreDeposit}}$ confirms, so a depositor must first lock BTC and pay Bitcoin fees. Applications can also rate-limit requests or require minimum deposits and setup fees. Because keepers are permissioned, the application can exclude a keeper that disrupts future setups.

Phase 4 – Pre-signing of $\text{tx}_{\text{Deposit}}$. After all $1 + |\text{Keepers}|$ BITVM-CORE setups complete without abort, Dep and each $K \in \text{Keepers}$ jointly co-sign the $\text{tx}_{\text{Deposit}}$ transaction (the hash-lock witness s is still absent at this point). Note that $\text{tx}_{\text{Deposit}}$ is the only protocol transaction not covered by the pre-signing inside the BITVM-CORE setups of Phase 3, since it belongs to no $\mathcal{S}_{V,P}$. The collected pre-signatures are submitted to $\mathcal{C}_{\text{VM}}$. $\text{postDepositSignatures}$ for data availability; $\mathcal{C}_{\text{VM}}$ checks that the submission is *complete* and *well-formed* — one signature of the expected form for each member of $\{V.\text{Dep}\} \cup \text{Keepers}$ — and emits $\text{DepositSignaturesPosted}(V)$. Checking that those signatures are cryptographically *valid* is left to the parties, each of which does so off-chain before it acts on them (Phases 5 and 6), which keeps the on-chain cost of a deposit independent of the size of the claimer set.

Phase 5 – ACK round. Each $K \in \text{Keepers}$ waits for $\text{tx}_{\text{PreDeposit}}$ to be at least k blocks deep on the longest Bitcoin chain (i.e., it appears on the ledger $\mathcal{L}_{\text{BTC}}$) and for $\text{DepositSignaturesPosted}(V)$ to be confirmed on $\mathcal{L}_{\text{ETH}}$. K verifies the artifacts posted in Phase 4, signs the message $\text{Hash}(\text{"DEPOSIT_ACK"} \parallel V)$, and submits the signature to $\mathcal{C}_{\text{VM}}$. postACK . Once $\mathcal{C}_{\text{VM}}$ has accepted ACKs from every keeper, it emits $\text{ACKAccepted}(V)$.

Phase 6 – Vault activation = $\text{Approved}(V)$. Dep first verifies the pre-signatures posted in Phase 4: that they are complete and that each is a valid signature on the $\text{tx}_{\text{Deposit}}$ skeleton. If this check fails, Dep does not activate and instead takes the refund path of Phase 8. Otherwise, at any round before $\text{DepositRegistered}(V)$ becomes T_{activate} blocks deep on $\mathcal{L}_{\text{ETH}}$, Dep calls $\mathcal{C}_{\text{VM}}$. $\text{activateVault}(V, s)$. The method is permissionless — any party may call it — but only a caller who knows s can pass the checks below, and by Lem. 1 that is Dep until he reveals it. $\mathcal{C}_{\text{VM}}$ checks $\text{Hash}(s) = h$ and that $\text{ACKAccepted}(V)$ has appeared; if both checks pass it marks V as active and emits the event $\text{Approved}(V)$, which carries s as part of its public payload. If activateVault is not invoked in time, $\mathcal{C}_{\text{VM}}$ permanently disallows activation of V and the protocol proceeds to the refund path below.

Phase 7 – $\text{tx}_{\text{Deposit}}$ on Bitcoin. Once $\text{Approved}(V)$ is on $\mathcal{L}_{\text{ETH}}$, the secret s is publicly readable. Any party — in practice, any keeper or relay — can read s from $\text{Approved}(V)$, complete the hash-lock witness of the pre-signed $\text{tx}_{\text{Deposit}}$, and write the resulting transaction to $\mathcal{L}_{\text{BTC}}$. We require this to happen before T_{refund} blocks have elapsed since the inclusion of $\text{tx}_{\text{PreDeposit}}$ in $\mathcal{L}_{\text{BTC}}$. The inclusion of $\text{tx}_{\text{Deposit}}$ in $\mathcal{L}_{\text{BTC}}$ is by definition the $\text{Deposited}(V)$ transaction; the output it creates is the *vault UTXO* of V .

Phase 8 – Refund path. If activateVault is not invoked within T_{activate} (Phase 6), then $\mathcal{C}_{\text{VM}}$ permanently disallows activation of V , so $\text{Approved}(V)$ never appears. In this case Dep may, after T_{refund} blocks have elapsed since the inclusion of $\text{tx}_{\text{PreDeposit}}$, broadcast a refund transaction $\text{tx}_{\text{Refund}}$ spending the refund leaf of $\text{tx}_{\text{PreDeposit}}$ back to Dep. Neither $\text{Approved}(V)$ nor $\text{Deposited}(V)$ ever confirms, and the vault is never created (Remark 2 (iii)).

Local state (side effects of Setup). At the end of a successful Setup, each party X retains: its BITVM-CORE state $\text{st}_X^{V,P}$ for every P (prover state if $P = X$, verifier state otherwise); and a local copy of the pre-signed transaction subsets $\{\mathcal{S}_{V,P}\}_P$ relevant to X 's roles. These states are used by Withdraw below.

Remark 2 (Design rationale for Setup). Three steps above warrant comment. (i) Because $\mathcal{C}_{\text{VM}}$ stores the skeletons on $\mathcal{L}_{\text{ETH}}$ in Phase 1, any keeper can rebuild and complete $\text{tx}_{\text{Deposit}}$ in Phase 7 from on-chain data alone; the proof of setup safety rests on this. (ii) The one-block lower bound in Phase 3 only ensures that $\text{tx}_{\text{PreDeposit}}$ is on-chain before the keepers invest in the expensive setups; no step of the analysis depends on it. The upper bound T_{stale} does matter: T_{refund} runs from the inclusion of $\text{tx}_{\text{PreDeposit}}$, so a setup begun too late would leave too little time before the refund path opens ((C2)). (iii) The refund path of Phase 8 stays available to Dep because s stays secret (Lem. 1), so no other party can satisfy the hash lock of the deposit leaf.

5.1.2. The Withdraw procedure. Withdraw is invoked by any party $P \in \{V.\text{Dep}\} \cup \text{Keepers}$ that the application has designated as the owner of vault V . The procedure has 3 phases: an Ethereum-side application step, off-chain witness generation, and an on-chain Bitcoin proving phase (Fig. 2b).

Inputs. P takes the vault identifier V and the local state it retained from Setup — its BITVM-CORE prover state $\text{st}_P^{V,P}$ and the pre-signed subset $\mathcal{S}_{V,P}$; every other member of the claimer set takes its corresponding verifier state and acts as a member of $\mathcal{V}_P$. *Output.* A bit $b \in \{0, 1\}$ recording whether the proving phase succeeded. *Side effect.* On success, $\text{Withdrawn}(V, P)$ is included on $\mathcal{L}_{\text{BTC}}$, spending the vault UTXO to P .

Phase 1 – Closed(V, P) on Ethereum. P waits until the application App emits $\text{Closed}(V, P)$ on $\mathcal{L}_{\text{ETH}}$ in accordance with App’s internal logic. The action that triggers $\text{Closed}(V, P)$ is application-specific (Sec. 4.1 gives repayment, liquidation, and settlement examples); P may itself take such an action, but the emission of $\text{Closed}(V, P)$ is performed by $\mathcal{C}_{\text{App}}$, not by P directly.

Phase 2 – Off-chain witness generation. P waits until $\text{Closed}(V, P)$ is confirmed in its local view of $\mathcal{L}_{\text{ETH}}$ and computes a witness w such that $\text{LC}(\text{Closed}(V, P), \mathcal{L}_{\text{ETH}}^*, w) = 1$, where $\mathcal{L}_{\text{ETH}}^*$ is the relevant snapshot of $\mathcal{L}_{\text{ETH}}$. P sets $w := (\mathcal{L}_{\text{ETH}}^*, w)$.

Phase 3 – On-chain BITVM-CORE proving phase. P runs the proving phase of its BITVM-CORE instance against the verifier set $\mathcal{V}_P$ established in Phase 3 of Setup:

$$b \leftarrow \text{out}_P(\langle P_{\text{Prove}}(\text{st}_P^{V,P}, w), \mathcal{V}_{P, \text{Prove}}(\{\text{st}_v^{V,P}\}_{v \in \mathcal{V}_P}) \rangle_r),$$

starting at some round r . Every verifier $v \in \mathcal{V}_P$ runs $V_{\text{Prove}}(\text{st}_v^{V,P})$ in parallel, interacting with the Bitcoin ledger.

5.1.3. On-chain identification of TBV transactions. The three ledger-side transactions of the TBV interface are realized as follows:

- $\text{Deposited}(V) := \text{tx}_{\text{Deposit}}$ as included in $\mathcal{L}_{\text{BTC}}$ in Phase 7 of Setup.
- $\text{Approved}(V) :=$ the event emitted on $\mathcal{L}_{\text{ETH}}$ by the call to $\mathcal{C}_{\text{VM}}.\text{activateVault}(V, s)$, as included in $\mathcal{L}_{\text{ETH}}$ in Phase 6 of Setup.
- $\text{Withdrawn}(V, P) :=$ the unique transaction in $\mathcal{S}_{V,P}$ that is included in $\mathcal{L}_{\text{BTC}}$ at the end of Phase 3 of Withdraw.

5.2. Security Analysis

We first state our vault-level security guarantees informally, matching the security properties of Sec. 4, and give proof sketches. We keep the standard property names: setup correctness, setup liveness, setup safety, withdrawal liveness, and withdrawal safety. In the lending corollaries below, setup liveness gives depositor-side progress after BTC is committed, whereas setup safety gives lender- or pool-side collateral backing after a position is accepted on Ethereum. The formal theorem statements — with explicit time bounds, the standing assumptions, and the supporting lemmas — and the full proofs are deferred to Appendix A. All vault-level theorems below additionally assume that the Bitcoin and Ethereum ledgers are safe and live, that $\Pi_{\text{BITVM-CORE}}$ satisfies setup correctness, robustness, and knowledge soundness, that the light client LC is safe, live, and timely, that the application

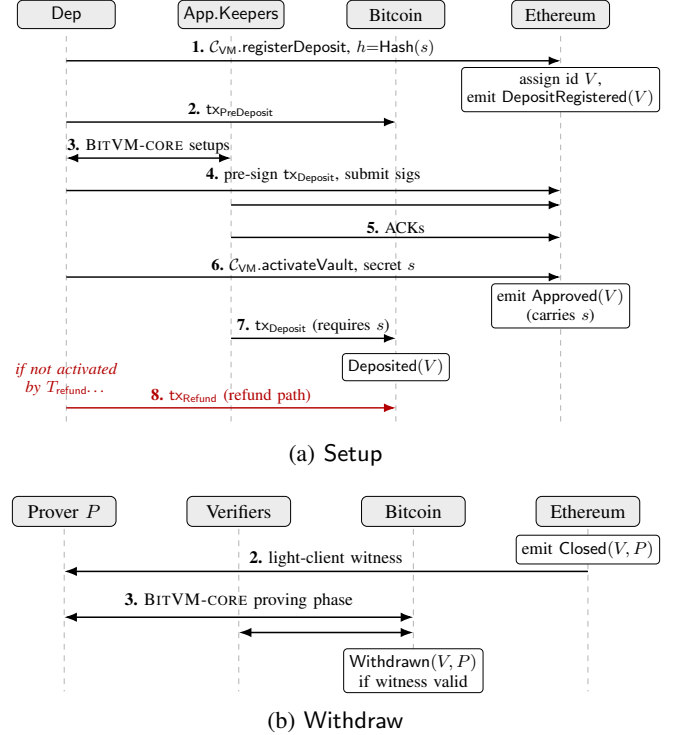


Figure 2: Chronological phases of the TBV protocol. Boxed annotations on a lifeline denote on-chain processing by that party. In a, Phase 8 (red) occurs only on the refund path when Dep does not invoke activateVault in time. In b, the prover P is any member of $\{V, \text{Dep}\} \cup \text{App.Keepers}$ that has been designated as owner by the application; the verifiers are all other members of that set.

satisfies application soundness (Def. 1), that signatures are EUF-CMA secure, that Hash is a random oracle, and that the protocol timeouts are mutually consistent. The end-to-end lending discussion below adds only the application-level correctness assumptions stated there; each vault-level statement names the honesty assumption it needs.

Theorem 5.1 (Setup correctness, informal). *If Dep and all keepers in Keepers are honest, then a Setup invocation produces, within bounded time, the Deposited(V) transaction on $\mathcal{L}_{\text{BTC}}$ and the Approved(V) event on $\mathcal{L}_{\text{ETH}}$.*

Proof sketch. We walk forward through the phases of Setup. Every on-chain step confirms within the corresponding ledger’s liveness bound; the BITVM-CORE setups of Phase 3 complete without abort by setup correctness; all keepers ACK in Phase 5; Dep activates within the deadline in Phase 6; and any keeper completes $\text{tx}_{\text{Deposit}}$ in Phase 7 before the refund timelock expires, by the timeout consistency. Formal statement and proof: Thm. A.1. $\square$

Theorem 5.2 (Setup liveness, informal). *If Dep is honest and the Deposited(V) transaction confirms on $\mathcal{L}_{\text{BTC}}$, then the Approved(V) event confirms on $\mathcal{L}_{\text{ETH}}$ within bounded time.*

Proof sketch. The witness of $\text{tx}_{\text{Deposit}}$ contains a preimage of h . Since Dep is honest, the secret s becomes public only through Dep’s $\text{activateVault}(V, s)$ broadcast (Lem. 1), which Dep issues only after $\text{ACKAccepted}(V)$ is on-chain, so the call cannot revert; even if an adversary front-runs it with a copy carrying the same s , that copy also emits $\text{Approved}(V)$. Ledger liveness concludes. Formal T_2 statement and proof: Thm. A.2. $\square$

Theorem 5.3 (Setup safety, informal). *If at least one member of $\{V.\text{Dep}\} \cup \text{Keepers}$ is honest and the $\text{Approved}(V)$ event confirms on $\mathcal{L}_{\text{ETH}}$, then the $\text{Deposited}(V)$ transaction confirms on $\mathcal{L}_{\text{BTC}}$ within bounded time.*

Proof sketch. By the checks of $\mathcal{C}_{\text{VM}}$, $\text{Approved}(V)$ implies that all ACKs and the pre-signatures of $\text{tx}_{\text{Deposit}}$ precede it on $\mathcal{L}_{\text{ETH}}$, and the payload of $\text{Approved}(V)$ reveals s . Those pre-signatures are valid, because the honest member verified them itself before acting: an honest keeper before ACKing (Phase 5), or an honest Dep before activating (Phase 6). Hence a complete valid witness for $\text{tx}_{\text{Deposit}}$ is assemblable from on-chain data alone. The honest member — depositor or keeper — broadcasts $\text{tx}_{\text{Deposit}}$ per Phase 7, and until the refund timelock expires no other spend of the $\text{tx}_{\text{PreDeposit}}$ output is possible (Lem. 2), so $\text{tx}_{\text{Deposit}}$ confirms by ledger liveness. Formal T_3 statement and proof: Thm. A.3. $\square$

Theorem 5.4 (Withdrawal liveness, informal). *If $P \in \{V.\text{Dep}\} \cup \text{Keepers}$ is honest and the $\text{Closed}(V, P)$ event confirms on $\mathcal{L}_{\text{ETH}}$, then the $\text{Withdrawn}(V, P)$ transaction confirms on $\mathcal{L}_{\text{BTC}}$ within bounded time.*

Proof sketch. First, $\text{Closed}(V, P)$ implies $\text{Approved}(V)$ by application soundness, so the vault UTXO exists on $\mathcal{L}_{\text{BTC}}$ by setup safety. Second, the vault UTXO is unspent when P acts: its only spending leaf requires every party’s signature, and the honest P signed only the pre-signed transactions $\bigcup_{P'} \mathcal{S}_{V, P'}$ (Lem. 3); a spend by $P' \neq P$ would imply $\text{Closed}(V, P')$ on honest Ethereum views (Lem. 4), contradicting the at-most-one-Closed clause of application soundness. Third, honest P computes the light-client witness (light-client liveness) and runs P_{Prove} against all other parties; by robustness of $\Pi_{\text{BITVM-CORE}}$ — which holds even when all verifiers are malicious — the success transaction $\text{Withdrawn}(V, P)$ confirms. Formal statement and proof: Thm. A.4. $\square$

Theorem 5.5 (Withdrawal safety, informal). *If $P \in \{V.\text{Dep}\} \cup \text{Keepers}$ is honest and the $\text{Withdrawn}(V, P')$ transaction confirms on $\mathcal{L}_{\text{BTC}}$ for any $P' \in \{V.\text{Dep}\} \cup \text{Keepers}$, then the $\text{Closed}(V, P')$ event confirms on $\mathcal{L}_{\text{ETH}}$ within bounded time.*

Proof sketch. By Lem. 3, the spending transaction belongs to $\mathcal{S}_{V, P'}$. If $P' = P$: honest P broadcasts its success transaction only after observing $\text{Closed}(V, P)$ confirmed on $\mathcal{L}_{\text{ETH}}$, which persists by ledger safety. If $P' \neq P$: honest P is a verifier in P' ’s BITVM-CORE instance, so knowledge soundness extracts a light-client witness for $\text{Closed}(V, P')$, and light-client safety and timeliness place $\text{Closed}(V, P')$ in

all honest views of $\mathcal{L}_{\text{ETH}}$ (Lem. 4). Formal statement and proof: Thm. A.5. $\square$

End-to-end security for BTC-backed lending. The theorems above are vault-level: they relate the TBV events $\text{Deposited}(V)$, $\text{Approved}(V)$, $\text{Closed}(V, P)$, and $\text{Withdrawn}(V, P)$ for any application satisfying application soundness. We now instantiate them with the over-collateralized lending application of Sec. 4.1.

For both lending corollaries, we assume that the lending smart contract is correct relative to the rules of Sec. 4.1. In particular, it accepts a TBV position as collateral only after $\text{Approved}(V)$; maintains debt accounting, pool accounting when applicable, interest accrual, price feeds, and the liquidation predicate consistently with the advertised LTV and LLTV rules; sets $\text{VaultOwner}(V) = V.\text{Dep}$ upon approval and after full repayment; sets $\text{VaultOwner}(V) = \perp$ while the loan is outstanding; and sets $\text{VaultOwner}(V) = P \in \text{Keepers}$ when P repays the full outstanding debt while $\text{LTV} > \text{LLTV}$. In the pooled case, each lender’s pool claim is computed from the pool accounting, and borrower repayments and liquidation repayments for a position are credited to the pool before that position is treated as debt-settled. The current owner is the only party authorized by the application to close the position, and closing is terminal. These are application-level assumptions; TBV consumes only the resulting interface events.

Corollary 1 (End-to-end security for peer-to-peer BTC-backed lending, informal). *Consider the peer-to-peer loan of Table 1, where $\text{Keepers} = \{\text{Larry}\}$ and Larry is both the lender and the eligible liquidator. Under the TBV assumptions and the lending-application assumptions above, the following guarantees hold for vault V .*

- (i) Setup liveness. *If $V.\text{Dep}$ and Larry honestly run Setup, then V becomes both approved on Ethereum and deposited on Bitcoin within bounded time. Thus the only TBV setup counterparty for $V.\text{Dep}$ is Larry, and the only TBV setup counterparty for Larry is $V.\text{Dep}$.*
- (ii) Depositor-side setup safety. *If honest $V.\text{Dep}$ ’s BTC collateral deposit confirms on Bitcoin, then the position becomes approved on Ethereum within bounded time.*
- (iii) Lender-side setup safety. *If honest Larry funds the loan only after the lending application accepts V as approved collateral, then that position is backed by the corresponding BTC deposit within bounded time.*
- (iv) Trustless withdrawal. *If honest $V.\text{Dep}$ repays the full outstanding debt, then $V.\text{Dep}$ can withdraw the BTC collateral within bounded time. If honest Larry liquidates an unhealthy position by repaying the full outstanding debt while $\text{LTV} > \text{LLTV}$, then Larry can withdraw the BTC collateral within bounded time.*

Proof sketch. Item (i) is setup correctness with $\text{Keepers} = \{\text{Larry}\}$. Item (ii) is setup liveness. Item (iii) is setup safety, applied after the lending application accepts an approved position and using Larry’s honesty as the honest member of the claimer set. Item (iv) combines the lending ownership

rules with TBV withdrawal liveness. TBV withdrawal safety gives the converse direction, so a separate no-premature-withdrawal statement is not needed. $\square$

Corollary 2 (End-to-end security for BTC-backed lending pools, informal). *Consider a pooled lending market in which lenders fund a lending pool, borrowers open TBV-backed debt positions, and the application-designated arbitrageurs/keepers Keepers may liquidate unhealthy positions. Under the TBV assumptions and the lending-application assumptions above, the following guarantees hold for each accepted TBV position V .*

- (i) Setup liveness. *If V .Dep and all keepers in Keepers honestly run Setup, then V becomes both approved on Ethereum and deposited on Bitcoin within bounded time. Pool lenders do not participate in TBV setup for V .*
- (ii) Depositor-side setup safety. *If honest V .Dep’s BTC collateral deposit confirms on Bitcoin, then the position becomes approved on Ethereum within bounded time.*
- (iii) Pool-lender setup safety. *From the point of view of an honest lender who supplies liquidity only through the lending application, every accepted approved TBV position against which the pool lends is backed by the corresponding BTC deposit within bounded time, provided at least one party in $\{V$.Dep $\} \cup$ Keepers is honest.*
- (iv) Trustless withdrawal for possible TBV owners. *If honest V .Dep repays the full outstanding debt, then V .Dep can withdraw the BTC collateral within bounded time. If an honest arbitrageur/keeper $P \in$ Keepers liquidates an unhealthy position by repaying the full outstanding debt while $LTV > LLTV$, then P can withdraw the BTC collateral within bounded time.*
- (v) Pool-lender repayment safety. *For any honest lender who supplies liquidity to the pool, and any accepted position V whose debt is funded by that pool, if at least one party in $\{V$.Dep $\} \cup$ Keepers is honest, then Bitcoin-side withdrawal of V ’s BTC cannot bypass the pool’s debt claim: the collateral is withdrawn only after the full outstanding debt of V has been repaid to the pool, and the lender’s pool claim includes its share of the recovered debt according to the pool accounting.*

Proof sketch. Items (i) and (ii) are setup correctness and setup liveness. Item (iii) is setup safety, applied to each approved position accepted by the lending application. Item (iv) combines the lending ownership rules with TBV withdrawal liveness. Item (v) combines the lending ownership and pool accounting rules with TBV withdrawal safety: any Bitcoin withdrawal is tied back to an application-authorized owner, and for a position with pool-funded debt such ownership is available only after borrower repayment or keeper liquidation repays the full outstanding debt to the pool. $\square$

6. Implementation

This section describes an implementation of the TBV protocol and its integration with Aave v4.³ We instantiate BITVM-CORE with BABE [15]. We then summarize the smart contracts and their integration with Aave v4.

Ethereum Light Client. In our implementation we use Helios [22], an Ethereum light client in which the ledger is checkpointed at periodic intervals and the witness w consists of signatures for each checkpoint from the *sync committee*, a dynamically evolving subset of the Ethereum validators. We compile the Helios verification logic into a SNARK (Groth16 [50]) proof with statement $x = tx$ and witness $w = (\mathcal{L}, w)$, which is then verified on Bitcoin by the BABE protocol. Helios accepts only ledgers ending at a finalized checkpoint; since finalized prefixes are shared by all honest views, Helios satisfies timeliness with u_{LC} equal to Ethereum’s finality delay. Helios is efficient, but it trusts the sync committee, a randomly sampled subset of Ethereum validators. Signal [51] instead verifies Ethereum consensus from validator attestations, giving TBV a drop-in light client with weaker trust assumptions.

BABE as the BITVM-CORE instantiation. A TBV protocol can be instantiated from any BITVM-CORE-style proof-verification protocol, such as BitVM2 [13], BitVM3 [20], or BABE [15]. We instantiate TBV from BABE for its on-chain and off-chain efficiency. BABE realizes BITVM-CORE through witness encryption and garbled circuits. During setup, each verifier encrypts a secret under a statement as defined in Sec. 5. A garbled circuit defers the witness-dependent part of this encryption. During the proving phase, the prover generates a succinct Groth16 [50] proof using the light client witness, commits to its proof on Bitcoin, obtains the garbled circuit input labels from the verifiers, and evaluates the circuit to recover the secret. This succeeds only when the proof is valid. Recovering the secret lets the prover withdraw Bitcoin from the vault. An invalid proof leaves the secret unrecoverable, and the prover cannot withdraw the Bitcoin.

The transaction graph. BABE already specifies a transaction graph that realizes BITVM-CORE for a single prover and a single verifier (cf. [15, Fig. 5]). The transaction graph comprises Assert, ChallengeAssert, WronglyChallenged, NoWithdraw, and Withdraw transactions anchored at a Deposit output. We build on this transaction graph with four changes.

- 1) We support multiple verifiers. Each party in the claimer set $\{V$.Dep $\} \cup$ App.Keepers acts as the prover of its own BITVM-CORE instance and as a verifier in every other party’s instance, through the multiple-verifier BITVM-CORE extension (cf. Appendix D.3, Appendix E).
- 2) We replace BABE’s Deposit with Pre-Deposit and Deposit transactions to support the HTLC-based deposit protocol (Sec. 5).

3. Due to licensing requirements, we cannot share the code. To facilitate independent verification, we report the protocol parameters, deployed mainnet contract addresses, transaction identifiers, and measured on-chain costs, allowing the mainnet results to be checked against public block explorers.

- 3) The transaction graph includes an additional `Claim` transaction. The prover submits the `Claim` transaction to set up the input required for `Assert` and start the withdrawal process on Bitcoin.
- 4) To avoid requiring challengers to reserve substantial BTC for pre-signed transaction fees, we implement Child Pays For Parent (CPFP) [52], where the pre-signed parent pays zero fees and its child supplies the required fee at broadcast time. Since CPFP requires the parent transaction to comply with Topologically Restricted Until Confirmation (TRUC) [53], which imposes a size limit of 10,000 vBytes, we split BABE’s Challenge-Assert into two transactions, `ChallengeAssertX` and `ChallengeAssertY`.

Fig. 3 describes the resulting transaction graph for one depositor `Dep`, one prover `P`, and one verifier `Vf`. A deposit realizes `Setup` in two transactions: the depositor posts `Pre-Deposit`, which locks the BTC behind a hashlock with a timed refund path, and once the vault is activated on Ethereum, anyone posts `Deposit`, which creates the vault UTXO, i.e., the `Deposited(V)` transaction of Sec. 4.

A withdrawal starts with the prover posting `Claim` and `Assert`, which commit the Groth16 proof in a Lamport signature. If no verifier challenges within the challenge period, the prover posts `Withdraw`, i.e., the `Withdrawn(V, P)` transaction of Sec. 4. A verifier challenges by posting `ChallengeAssertX` and `ChallengeAssertY`, which reveal the garbled circuit input labels encoding the prover’s proof. If the proof is invalid, the prover cannot recover the encrypted secret, and the verifier posts `NoWithdraw` after the response period to block `Withdraw`. If the proof is valid, the prover evaluates the garbled circuits to recover the secret, posts `WronglyChallenged`, and proceeds to `Withdraw`.

TBV smart contracts. The Ethereum-side smart contracts can be categorized into two parts: the protocol core contracts and the Aave v4 adapter contracts. Appendix B.2 provides details of the smart contracts and Aave v4 integration.

The protocol core contracts realize the Vault Manager C_{VM} of Sec. 5.1. They coordinate vault registration, deposit, and withdrawal. Their methods realize `registerDeposit`, `postDepositSignatures`, `postACK`, and `activateVault`. They also verify the Bitcoin signatures natively on Ethereum, so no trusted relay attests to Bitcoin state.

The Aave v4 adapter contracts integrate the protocol core contracts with Aave v4 [6], a major Ethereum lending protocol, *without modifying Aave*. The core contracts invoke the adapter contracts, and the adapter contracts drive Aave v4 only through its public interface. The adapter contracts track each vault’s lifecycle, authorized owner, and the borrower position it backs. The adapter contracts also handle the positions through Aave’s standard interface, using operations such as `borrow`, `repay`, `withdraw`, and `liquidate`.

7. Evaluation

We evaluate the implementation of Sec. 6 in terms of its on-chain and off-chain cost. The evaluation shows that

the TBV protocol runs end-to-end on Bitcoin and Ethereum mainnet at small on-chain cost. In Table 3, at the current Bitcoin price of \$65,000 and fee rate of 2 sat/vB, a deposit costs about \$0.7 in Bitcoin fees and a withdrawal at most \$7.7, and challenging a dishonest withdrawal about \$36. This also shows that the TBV protocol does not introduce additional overhead on top of the underlying Bitcoin proof verification protocol.

Evaluation goals. The evaluation aims to answer three questions:

- **On-chain cost (Sec. 7).** What is the Bitcoin and Ethereum on-chain cost of a deposit and of each withdrawal path, in transaction size and in dollars? Every dispute-path transaction competes for block space in a live fee market, so an excessive footprint would make withdrawals uneconomical or, in the worst case, unconfirmable.
- **Off-chain cost (Sec. 7).** What is the off-chain storage overhead of the TBV protocol? Storage is the dominant off-chain cost of the BABE protocol we use.
- **Overhead over Bitcoin proof verification protocol.** Does the TBV protocol introduce any extra on-chain or off-chain overhead on top of the underlying Bitcoin proof verification protocol [15]?

Experiment setting. We run the experiment end-to-end against *both* Bitcoin and Ethereum mainnet, driving the real daemons through the complete lifecycle. We develop a Rust harness that first deploys the protocol contracts on Ethereum, sets the protocol permissions and parameters, and registers the on-chain actors. We deploy our own *unmodified* Aave v4 instance because Aave v4 on mainnet is not permissionless, not because the integration requires any change to Aave. Creating a spoke or listing a new collateral asset is gated by Aave governance. Appendix B.3 provides the deployed contract addresses on Ethereum mainnet. We then run the daemons to interact with both chains, posting every transaction on-chain. Bitcoin on-chain costs are reported at the run’s ≈ 2 sat/vB feerate and BTC = \$65,000; Ethereum costs use the observed gas price (≈ 0.5 gwei) and ETH = \$1,700. These fee rates, gas prices, and spot prices are the values observed during the live mainnet run; the dollar figures scale linearly with them, whereas the vByte and gas footprints remain constant. Each run of the lifecycle starts with a deposit of BTC, which we supply as collateral to the Aave v4 protocol and borrow against. We then exercise the following paths for the BTC: `repay` $\rightarrow$ `withdraw` under the *no challenge*, *correctly challenged*, or *wrongly challenged* dispute path, or `liquidate` under the no-challenge path.

Bitcoin on-chain cost. Table 3 reports the live Bitcoin run in full, with one column block per dispute path – no-challenge, correctly challenged, and wrongly challenged – and an em-dash marking each transaction a path does not use. Every txid resolves on the mempool.space block explorer [54] for independent verification. We report each footprint in vBytes (fee-market-independent) and in dollars, split by the role (depositor, prover, or verifier) that posts each transaction; in this run the depositor also acts as the prover. The protocol

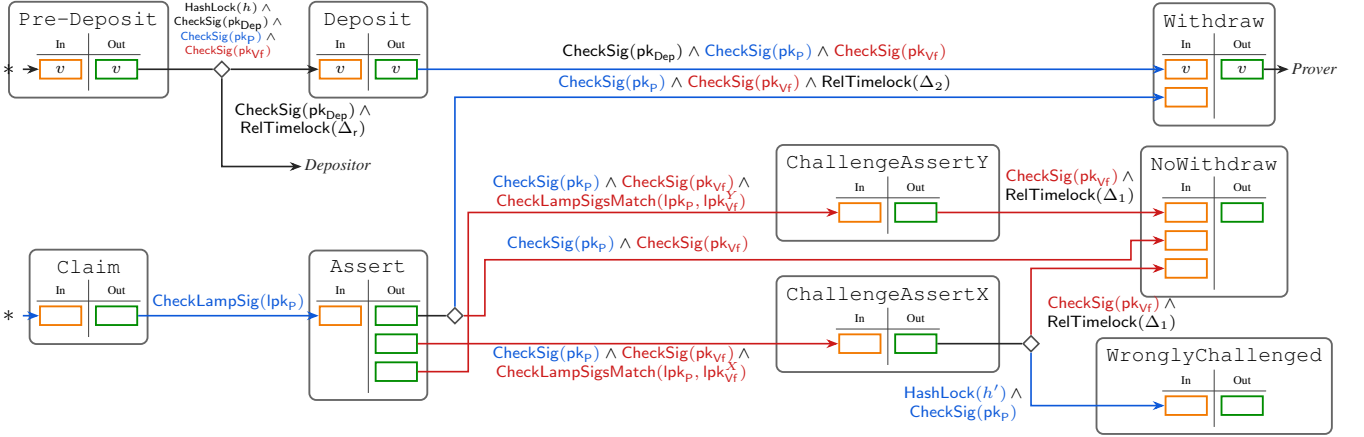


Figure 3: Transaction graph with the three withdrawal paths, shown for one depositor Dep, one prover P, and one verifier Vf. Each box is a Bitcoin transaction with its inputs (orange) and outputs (green); an edge from an output to an input is a spending path, annotated with its script condition (cf. Appendix D.2); a diamond marks an output with alternative spending paths. A condition is **blue** (**red**) if the prover (verifier) provides the witness that satisfies it, and an edge is **blue** (**red**) if the prover (verifier) submits the spending transaction; timelocks and the depositor’s conditions and spends are black.

fixes which role posts each transaction, so the cost falls predictably.

The deposit cost is identical across paths: the depositor posts the Pre-Deposit/Deposit pair for 512 vB (\$0.7). On the no-challenge path, the prover alone posts Claim, Assert, and Withdraw (4633 vB, \$6.0), dominated by Assert (4006 vB); the verifier does not submit any transaction. A prover being wrongly challenged defends itself by submitting WronglyChallenged before Withdraw, raising its withdrawal cost only to 5046 vB (\$7.7). So an honest prover’s withdrawal costs at most \$7.7 regardless of the dispute path.

The challenge paths are dominated by the verifier’s footprint: 18358 vB on the correctly challenged path and 18024 vB on the wrongly challenged one. Within that footprint, the bulk comes from the CPF children that fee-bump the two zero-fee TRUC ChallengeAssert parents. During the experiments, these children overpaid the fee market, paying an effective package feerate of up to ≈ 8.4 sat/vB against 2 sat/vB. This put the verifier’s correctly challenged cost at \$98.9, whereas the same footprint costs about \$24 at 2 sat/vB. The vByte footprints are fee-market-independent. The gap is therefore purely a fee-estimation artifact, not a protocol cost.

Ethereum on-chain cost. Table 4 reports the Ethereum gas, fee, and tx hash of each TBV *protocol* transaction. We focus on the TBV protocol’s own footprint and exclude the Aave lending-market gas (supply, borrow, repay, withdraw, liquidate), which does not depend on TBV.

Across the five protocol steps, the TBV footprint on Ethereum mainnet costs 2.89 M gas (\$2.5 at \$1,700/ETH and the observed ≈ 0.5 gwei), dominated by the deposit request (1.06 M gas) and vault activation (0.88 M gas). Together with the Bitcoin footprint (Table 3), an honest deposit/withdrawal lifecycle settles for roughly \$9 on-chain across both chains

– small enough to deploy on today’s Bitcoin and Ethereum mainnet without protocol changes.

On-chain overhead over BABE. This on-chain footprint is comparable to that of the BABE SNARK verification protocol [15]: the verifier’s dispute is 18358 vB against BABE’s ≈ 17400 vB, and the prover’s 4633 vB withdrawal is in fact below BABE’s ≈ 9240 vB due to optimizations to the Lamport signature scheme. TBV thus adds no on-chain overhead on top of the proof verification results.

Off-chain cost. The off-chain cost of TBV is the storage of the garbled-circuit artifacts produced by the BABE cut-and-choose, held until the corresponding withdrawal resolves. We sample the garbled circuits from the protocol execution under (307, 6) cut-and-choose parameters: our optimizations shrink each circuit from 22.2 MB [15] to 14 MB, so a six-instance session is 84 MB per prover-verifier pair. This cost falls on the *prover* only: the verifier keeps only the short seed and regenerates the circuits deterministically, so its off-chain cost is negligible. The stored artifacts are exactly the BABE cut-and-choose circuits. TBV thus adds no off-chain overhead on top of the proof verification. The honest withdrawal path has constant on-chain cost in the number of keepers. On-chain cost grows only when verifiers challenge a malicious claim; in the worst case, it grows with the number of challengers. The main scaling cost is off-chain: for N claimers, the $N(N - 1)$ prover-verifier pairs each require 84 MB. We expect fewer than ten keepers, for which this cost is reasonable.

8. Hybrid TBV-Bridge

Our TBV protocol (cf. Sec. 4) deliberately fixes the set of parties that may withdraw a vault at setup time. The introduction (Sec. 1) showed what this costs in the lending application: liquidation is restricted to parties named

TABLE 3: Per-path on-chain results from the live Bitcoin mainnet run, merged into one table. For each transaction we report the vSize (vB, fee-market-independent), feerate (sat/vB), fee (USD, at BTC = \$65,000), and on-chain txid under the no-challenge, correctly challenged, and wrongly challenged paths. The depositor, who also acts as the prover in this run, posts the deposit transactions.

Transaction	No challenge				Correctly challenged				Wrongly challenged			
	vB	sat/vB	USD	txid	vB	sat/vB	USD	txid	vB	sat/vB	USD	txid
<i>Deposit procedure</i> (depositor)												
Pre-Deposit	240	2.01	\$0.31	eb009cb4	240	2.01	\$0.31	ba224e40	240	2.01	\$0.31	ca6d35d7
Deposit	272	2.00	\$0.36	23f15c76	272	2.00	\$0.36	63ce713f	272	2.00	\$0.36	91c9a86b
<i>Withdrawal procedure</i> (prover / verifier)												
Claim	195	2.00	\$0.25	fb1239a2	195	2.00	\$0.25	96c916a9	263	1.52	\$0.26	1c7f7feb
Assert	4006	2.00	\$5.21	bdd5f742	4006	2.00	\$5.21	fbe637e9	4006	2.00	\$5.21	c269e7cf
ChallengeAssertX [†]			–		8777	0	\$0.00	9acfe4d8	8777	0	\$0.00	bda62764
↪ CPFP child			–		234	115.1 [‡]	\$17.51	ddca3761	235	115.1	\$17.58	192d3161
ChallengeAssertY [†]			–		8777	0	\$0.00	ec880bcc	8777	0	\$0.00	6371ca3d
↪ CPFP child			–		234	115.1 [‡]	\$17.51	4d8c0ce2	235	115.1	\$17.58	05060029
WronglyChallenged [†]			–				–		178	2.01	\$0.23	3738203c
↪ CPFP child			–				–		167	13.02	\$1.42	7ddff85d
NoWithdraw			–		336	2.08	\$0.46	c4cd2711			–	
Withdraw	432	2.00	\$0.56	d0bb3e42			–		432	2.00	\$0.56	0f864a0a
Total (depositor, deposit)	512		\$0.67		512		\$0.67		512		\$0.67	
Total (prover, withdrawal)	4633		\$6.02		4201		\$5.46		5046		\$7.68	
Total (verifier, withdrawal)	0		\$0.00		18358		\$35.48		18024		\$35.16	

[†] The marked transactions are zero-fee TRUC parents bumped by a CPFP child (indented row); the child carries the package fee, so its high standalone feerate just reflects a small transaction paying for the whole package.

[‡] The correctly challenged run was executed during a fee spike; its CPFP children actually paid 323.7 sat/vB (\$49.24 each, verifiable from the linked txids). For a fair comparison with the wrongly challenged path, we report their fees normalized to that path’s 115.1 sat/vB rate; vSizes and txids are unchanged.

TABLE 4: Ethereum cost of the TBV *protocol* transactions: the measured gas, fee, and on-chain tx hash of each protocol step. The Aave lending-market operations (supply, borrow, repay, withdraw, liquidate) are excluded – their gas depends on the integrated market, not on TBV. Fees in USD use ETH = \$1,700.

Protocol step	gas	gas price (gwei) [†]	fee (USD)	tx hash
Deposit request	1 061 958	0.529	\$0.96	ca49de17
Input signatures	330 014	0.473	\$0.27	ba06bdfc
ACK signatures	270 682	0.506	\$0.23	f726829a
Vault activation	879 427	0.502	\$0.75	866859db
Redemption	344 212	0.563	\$0.33	2c6c531c
Total	2 886 293	–	\$2.54	–

at setup—shutting out the professional liquidators that real markets rely on—and the claim on the collateral cannot be used or traded outside the application. Both shortcomings have the same root: the claim on a vault does not circulate as a fungible token. The objective of this section is the hybrid vault promised there: a vault that issues such a token, wBTC, without giving up trustless withdrawal. Upon deposit, the hybrid vault mints fungible wBTC; any holder may redeem wBTC for BTC with the help of an operator, as in a bridge; and a party that keeps its tokens within the application, e.g., only does lending, retains the trustless withdrawal path of a pure TBV (Table 2). Neither pure model achieves this: a pure

bridge surrenders *every* withdrawal, including the depositor’s happy path, to the bridge’s trust assumptions [13], [20], while a pure TBV’s claims are bound to setup-time parties and cannot circulate.

The technical obstacle, previewed in Sec. 1, is that fungibility makes vaults *interact*. In both pure models, security can be analyzed one vault at a time: bridge vaults are homogeneous and interchangeable, and TBV vaults, while tailored to specific parties, never interact across instances. In the hybrid, a token minted from one vault may redeem against another. The simplest instance of the difficulty: suppose Bob deposits two vaults and repays both loans, so he holds two wBTC tokens and a trustless claim on each vault; he then transfers one token to Carol. Carol’s token must be redeemable for BTC, necessarily against one of Bob’s two vaults, and yet Bob, who still holds one token, should retain one trustless claim. Which vault remains Bob’s, and which becomes redeemable by Carol, cannot be decided by looking at either vault in isolation. Security must therefore be stated over the entire set of vaults $\mathcal{V}$, and the central question becomes: when does a vault remain *reserved* for its owner, and when does it become *free* for bridge-style redemption? The accounting layer introduced in this section answers this with a single notion of *backing*: a reservation is honored exactly while its owner holds enough wBTC to cover its reserved vaults.

The TBV properties (Defs. 5 and 6) guarantee the cross-chain relay of Closed and Withdrawn, but deliberately leave

open *who can validly close which vault*, which is application logic. The accounting layer we present in Sec. 8.1 settles it purely as Ethereum smart-contract validity; all cross-vault reasoning is confined to this single-ledger view. Appendix C then composes the accounting layer with the per-vault TBV properties into end-to-end guarantees, adding no new cross-vault reasoning.

Participants. We define our hybrid TBV-Bridge as a collection of vaults (cf. Sec. 4). A hybrid TBV-Bridge is parameterized by an inner application *innerApp*, such as the lending application of Sec. 5. Its participants are the *depositors*, who deposit BTC and receive freshly minted wBTC; the *local keepers* LocalKeepers named by *innerApp* (e.g., liquidators); the *operators* Operators, who enable bridge-style redemption for arbitrary token holders; and *other users*, arbitrary wBTC holders with no setup involvement. Every vault’s keeper set contains the common Operators, i.e., for every vault V in the Hybrid-TBV bridge, we have $\text{Operators} \subseteq V.\text{Keepers}$. In particular, operators participate in the setup ceremony of every vault, since a bridge-style redemption requires a party capable of spending any free vault. This common intersection of the keeper sets of all vaults, i.e., the operators, is the “cost” of fungibility. Note that the introduction’s permissionless flash liquidation is consistent with this participant model: a third-party liquidator settles the liquidation entirely at the token level on Ethereum, receiving wBTC like any other user, and never needs to appear in a vault’s claimer set on Bitcoin.

8.1. The Accounting Layer

Throughout this subsection, $\mathcal{L}_{\text{ETH}}$ denotes a valid Ethereum ledger (cf. Def. 17). Since validity is prefix-monotone and honest views are consistent (cf. Def. 8), every statement of this subsection holds in every honest party’s view at every round.

Inner application interface. The external application HybridApp, whose interface remains as defined in Sec. 4, internally delegates to *innerApp*, which exposes two functions of the ledger state to it (cf. Fig. 4).

- $\text{VaultOwner}(\mathcal{L}_{\text{ETH}}, V) \in \{V.\text{Dep}\} \cup V.\text{Keepers}$, the party currently holding the trustless claim on vault V according to *innerApp*’s logic;
- $\text{inApp}(\mathcal{L}_{\text{ETH}}, V) \in \{0, 1\}$, indicating whether the wBTC coin associated with V is currently committed inside *innerApp* (e.g., posted as collateral).

For now, each vault contains exactly 1 BTC, and coins are committed to *innerApp* as whole units, so *inApp* is binary. No $\perp$ owner state is needed: while a coin is committed inside the application (e.g., a loan is outstanding), the vault is protected by $\text{inApp} = 1$ regardless of the value of *VaultOwner*.

Vault sets and backing. For a ledger $\mathcal{L}_{\text{ETH}}$, define the open,

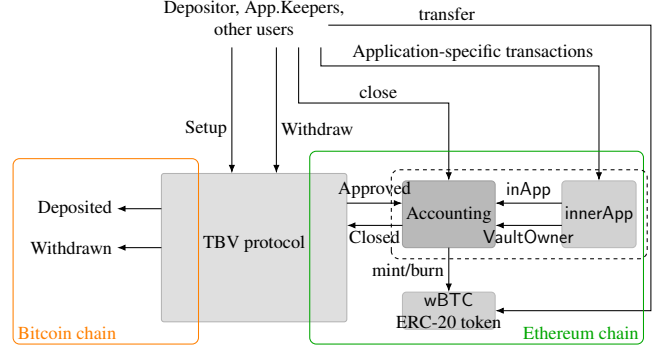


Figure 4: Hybrid TBV-Bridge protocol overview. This is similar to the TBV architecture shown in Fig. 1, except that there is an accounting layer that interacts with *innerApp* and an ERC-20 token.

the committed, and the reserved vaults:

$$\mathcal{U}(\mathcal{L}_{\text{ETH}}) := \{V \in \mathcal{V} : \text{Approved}(V) \in \mathcal{L}_{\text{ETH}} \wedge \forall P: \text{Closed}(V, P) \notin \mathcal{L}_{\text{ETH}}\},$$

$$\mathcal{U}_{\perp}(\mathcal{L}_{\text{ETH}}) := \{V \in \mathcal{U}(\mathcal{L}_{\text{ETH}}) : \text{inApp}(\mathcal{L}_{\text{ETH}}, V) = 1\},$$

$$\mathcal{U}_P(\mathcal{L}_{\text{ETH}}) := \{V \in \mathcal{U}(\mathcal{L}_{\text{ETH}}) : \text{inApp}(\mathcal{L}_{\text{ETH}}, V) = 0 \wedge \text{VaultOwner}(\mathcal{L}_{\text{ETH}}, V) = P\}.$$

Since *inApp* is binary and *VaultOwner* is a total function into the parties, $\mathcal{U}(\mathcal{L}_{\text{ETH}}) = \mathcal{U}_{\perp}(\mathcal{L}_{\text{ETH}}) \cup \bigcup_P \mathcal{U}_P(\mathcal{L}_{\text{ETH}})$. We say P is *backed* in $\mathcal{L}_{\text{ETH}}$ if $\text{tokBal}_{\mathcal{L}_{\text{ETH}}}(P) \geq |\mathcal{U}_P(\mathcal{L}_{\text{ETH}})|$, i.e., every uncommitted vault reserved for P is covered by a token in P ’s balance. Intuitively, P is backed as long as P has *stayed in the application*: every coin minted for P ’s vaults is either committed inside *innerApp* or still in P ’s balance. A party that spends its wBTC elsewhere becomes unbacked and, as formalized below, forfeits the protection of its reserved vaults to exactly that extent. Thus, a party retains exclusive access to its vaults exactly while it retains sufficient token ownership to back them.

Validity of Closed. The hybrid vault contract admits $\text{Closed}(V, P)$ as a valid Ethereum transaction with respect to a prefix $\mathcal{L}$ iff

- 1) $V \in \mathcal{U}(\mathcal{L})$, (vault approved and open)
- 2) $\text{inApp}(\mathcal{L}, V) = 0$, (coin not committed in *innerApp*)
- 3) $\text{tokBal}_{\mathcal{L}}(P) \geq 1$, (claimant holds a token)
- 4) $\text{VaultOwner}(\mathcal{L}, V) = P$, or $\text{tokBal}_{\mathcal{L}}(Q) < |\mathcal{U}_Q(\mathcal{L})|$ for $Q := \text{VaultOwner}(\mathcal{L}, V)$. (claimant owns V , or its owner is unbacked)

Its effect is to burn one wBTC token of P and remove V from $\mathcal{U}$.

Definition 7 (Hybrid application soundness). HybridApp(*innerApp*) is *sound* if:

- 1) (*Owner stability*) If $\text{VaultOwner}(\mathcal{L}_{\text{ETH}}, V) = P$ and P is honest, no PPT adversary can produce a valid $\mathcal{L}'_{\text{ETH}} \succeq \mathcal{L}_{\text{ETH}}$ with $\text{VaultOwner}(\mathcal{L}'_{\text{ETH}}, V) \neq P$.
- 2) (*Token conservation*) For all valid ledgers $\mathcal{L}_{\text{ETH}}$:

$$\sum_P \text{tokBal}_{\mathcal{L}_{\text{ETH}}}(P) + \sum_{V \in \mathcal{V}} \text{inApp}(\mathcal{L}_{\text{ETH}}, V) \leq |\mathcal{U}(\mathcal{L}_{\text{ETH}})|.$$

Remark 3. An arbitrary application could mint unbacked tokens. Our concrete Hybrid Vault application achieves the above soundness property: tokens are minted only upon $\text{Approved}(V)$, which by setup safety (Def. 4) corresponds to a distinct deposited vault; Closed burns a token; committing a coin to innerApp removes it from balances, and inApp counts no coin twice; transfers conserve balances.

Theorem 8.1 (Pegout admissibility). *Suppose $\text{HybridApp}(\text{innerApp})$ is sound. Then for every valid ledger $\mathcal{L}_{\text{ETH}}$:*

- 1) (Existence) *For every operator O with $\text{tokBal}_{\mathcal{L}_{\text{ETH}}}(O) \geq 1$, there exists $V \in \mathcal{U}(\mathcal{L}_{\text{ETH}})$ such that $\text{Closed}(V, O)$ is valid with respect to $\mathcal{L}_{\text{ETH}}$.*
- 2) (Conditional protection) *For every V and every $P \neq \text{VaultOwner}(\mathcal{L}_{\text{ETH}}, V) =: Q$, if Q is backed or $\text{inApp}(\mathcal{L}_{\text{ETH}}, V) = 1$, then $\text{Closed}(V, P)$ is invalid with respect to $\mathcal{L}_{\text{ETH}}$.*

Proof. Conditional protection is immediate from validity conditions 2 and 4 for Closed. For existence, suppose toward a contradiction that $\text{tokBal}_{\mathcal{L}_{\text{ETH}}}(O) \geq 1$ but no $\text{Closed}(V, O)$ is valid. Then $\mathcal{U}_O(\mathcal{L}_{\text{ETH}}) = \emptyset$, since any $V \in \mathcal{U}_O$ satisfies conditions 1–4 with $P = O$ via the owner branch. Moreover, for every $Q \neq O$ with $\mathcal{U}_Q(\mathcal{L}_{\text{ETH}}) \neq \emptyset$, party Q must be backed; otherwise, condition 4 admits $\text{Closed}(V, O)$ for any $V \in \mathcal{U}_Q$, hence $\text{tokBal}_{\mathcal{L}_{\text{ETH}}}(Q) \geq |\mathcal{U}_Q(\mathcal{L}_{\text{ETH}})|$. Since inApp counts no coin twice, $\sum_V \text{inApp}(\mathcal{L}_{\text{ETH}}, V) \geq |\mathcal{U}_{\perp}(\mathcal{L}_{\text{ETH}})|$. Summing and noting that $\mathcal{U}_{\perp}(\mathcal{L}_{\text{ETH}})$ and $\bigcup_{Q \neq O} \mathcal{U}_Q(\mathcal{L}_{\text{ETH}})$ cover $\mathcal{U}(\mathcal{L}_{\text{ETH}})$ since $\mathcal{U}_O(\mathcal{L}_{\text{ETH}}) = \emptyset$:

$$\begin{aligned} \sum_P \text{tokBal}_{\mathcal{L}_{\text{ETH}}}(P) + \sum_V \text{inApp}(\mathcal{L}_{\text{ETH}}, V) &\geq \\ \underbrace{\text{tokBal}_{\mathcal{L}_{\text{ETH}}}(O)}_{\geq 1} + \sum_{Q \neq O} |\mathcal{U}_Q(\mathcal{L}_{\text{ETH}})| + |\mathcal{U}_{\perp}(\mathcal{L}_{\text{ETH}})| &\geq \\ &1 + |\mathcal{U}(\mathcal{L}_{\text{ETH}})| \end{aligned}$$

contradicting token conservation. $\square$

The accounting layer above characterizes exactly when Closed events are emitted: backed parties keep their reserved vaults, while every token holder can still find an open vault to redeem against. Turning this validity statement into ledger inclusion, and composing it with the TBV withdrawal relays, is deferred to Appendix C, where we prove the end-to-end guarantees.

Acknowledgments

SS was supported in part by the AFOSR Award FA9550-24-1-0156, and research grants from the Bakar Fund, Peder Sather Funds, the Stellar Development Foundation, Supra Inc., Category Labs, Input Output Global, and Byzantine Research Inc. SS performed part of this work while employed by Byzantine Research. LA was supported in part by Input|Output (iog.io) through their funding of the Edinburgh Blockchain Technology Lab and by Babylon Labs. DT is a co-founder of Byzantine Research.

References

- [1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” <https://bitcoin.org/bitcoin.pdf>, 2008.
- [2] WBTC, “Wrapped tokens: A multi-institutional framework for tokenizing any asset,” Whitepaper, <https://wbtc.network/assets/wrapped-tokens-whitepaper.pdf>, 2019.
- [3] Threshold Network, “Threshold network powers tbtc,” <https://docs.threshold.network/>.
- [4] J. Nick, A. Poelstra, and G. Sanders, “Liquid: A Bitcoin sidechain,” <https://blockstream.com/assets/downloads/pdf/liquid-whitepaper.pdf>, 2020.
- [5] S. Nakamoto, “Bitcoin open source implementation of p2p currency,” <https://satoshi.nakamotoinstitute.org/posts/p2pfoundation/1/>, February 2009, last accessed: 2026-06.
- [6] Aave Labs, “Aave V4 documentation,” <https://aave.com/docs/aave-v4>, 2025, accessed: 2026-06.
- [7] Z. Pokorny, “The state of crypto leverage – q2 2026: An orderly, measured decline,” Galaxy Research, <https://www.galaxy.com/insights/research/crypto-leverage-report-q2-2026-defi-lending-decline-futures-open-interest>, 2026, accessed: 2026-08-27.
- [8] Morpho, “Morpho network data: Base,” <https://data.morpho.org/chain/base>, 2026, accessed: 2026-08-27.
- [9] Babylon Team, “Babylon staking dashboard,” <https://staking.babylonlabs.io/btc>, 2026, accessed: 2026-08-27.
- [10] Coinbase, “cbBTC: Coinbase wrapped assets,” Project documentation. [Online]. Available: <https://www.coinbase.com/cbbtc>
- [11] R. Linus, “BitVM: Compute anything on Bitcoin,” <https://bitvm.org/bitvm.pdf>, December 2023, last accessed: 2026-06.
- [12] L. Aumayr, Z. Avarikioti, R. Linus, M. Maffei, A. Pelosi, C. Stefo, and A. Zamyatin, “BitVM: Quasi-turing complete computation on Bitcoin,” Cryptology ePrint Archive, Report 2024/1995, 2024. [Online]. Available: <https://eprint.iacr.org/2024/1995>
- [13] R. L. Woll, L. Aumayr, Z. Avarikioti, M. Maffei, A. Pelosi, O. S. T. Litos, C. Stefo, D. Tse, and A. Zamyatin, “Bridging Bitcoin to second layers via BitVM2,” in *35th USENIX Security Symposium (USENIX Security 26)*. Baltimore, MD: USENIX Association, Aug. 2026, pp. 969–988.
- [14] R. Linus, “BitVM 3s - garbled circuits for efficient computation on Bitcoin,” <https://bitvm.org/bitvm3s.pdf>, 2025, last accessed: 2026-06.
- [15] S. Garg, D. Kolonelos, M. Sergeevitch, S. Sridhar, and D. Tse, “BABE: Verifying proofs on bitcoin made 1000x cheaper,” Cryptology ePrint Archive, Paper 2026/065, 2026, to appear at ACM CCS 2026. [Online]. Available: <https://eprint.iacr.org/2026/065>
- [16] L. Aumayr, Z. Avarikioti, R. Linus, M. Maffei, A. Pelosi, C. Stefo, and A. Zamyatin, “BitVM: Quasi-turing complete computation on Bitcoin,” Cryptology ePrint Archive, Paper 2024/1995, 2024. [Online]. Available: <https://eprint.iacr.org/2024/1995>
- [17] E. Bal, L. Aumayr, A. İyidoğan, G. Scaffino, H. Karakuş, C. E. Aslan, and O. S. T. Litos, “Clementine: A collateral-efficient, trust-minimized, and scalable Bitcoin bridge,” Cryptology ePrint Archive, Paper 2025/776, 2025. [Online]. Available: <https://eprint.iacr.org/2025/776>
- [18] S. D. Lerner, R. Amela, S. Mishra, M. Jonas, and J. Álvarez Cid-Fuentes, “BitVMX: A cpu for universal computation on Bitcoin,” 2024. [Online]. Available: <https://arxiv.org/abs/2405.06842>
- [19] L. Aumayr, J. Diaz, D. Jetchev, and A. Kiayias, “Cardinal: Bridging Bitcoin with ownership preservation,” Cryptology ePrint Archive, Paper 2025/2196, 2025. [Online]. Available: <https://eprint.iacr.org/2025/2196>
- [20] R. L. Woll, I. Alexopoulos, L. Aumayr, Z. Avarikioti, M. Maffei, and D. Tse, “BitVM3: Efficient Bitcoin bridges via garbled circuits,” Cryptology ePrint Archive, Paper 2026/933, 2026, to appear at ACM CCS 2026. [Online]. Available: <https://eprint.iacr.org/2026/933>

- [21] Aave Labs, “Flash loans,” <https://aave.com/docs/aave-v3/guides/flash-loans>, 2025, accessed: 2026-06.
- [22] a16z, “Helios: A fast, secure, and portable multichain light client for ethereum,” <https://github.com/a16z/helios>, 2022, last accessed: 2026-06.
- [23] J. Poon and T. Dryja, “The Bitcoin lightning network: Scalable off-chain instant payments,” <https://lightning.network/lightning-network-paper.pdf>, 2016, last accessed: 2026-01-12.
- [24] S. Dziembowski, S. Faust, and K. Hostáková, “General state channel networks,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 949–966. [Online]. Available: <https://doi.org/10.1145/3243734.3243856>
- [25] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, “Generalized channels from limited blockchain scripts and adaptor signatures,” in *Advances in Cryptology – ASIACRYPT 2021*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 635–664.
- [26] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, “FastKitten: Practical smart contracts on Bitcoin,” in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 801–818.
- [27] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, “Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019, pp. 185–200.
- [28] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 839–858.
- [29] I. Bentov and R. Kumaresan, “How to use Bitcoin to design fair protocols,” in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 421–439.
- [30] R. Kumaresan and I. Bentov, “How to use bitcoin to incentivize correct computations,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, G. Ahn, M. Yung, and N. Li, Eds. ACM, 2014, pp. 30–41. [Online]. Available: <https://doi.org/10.1145/2660267.2660380>
- [31] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, “Multi-party virtual state channels,” in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds. Cham: Springer International Publishing, 2019, pp. 625–656.
- [32] M. Bartoletti and R. Zunino, “BitML: A calculus for Bitcoin smart contracts,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018.
- [33] F. Badaloni, S. Holler, C. Oikonomou, P. Moreno-Sanchez, and C. Schneidewind, “*BitML*²: Secure cross-chain smart contracts for Bitcoin-style cryptocurrencies,” in *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*, 2025, pp. 316–330.
- [34] Babylon Team, “Bitcoin staking: Unlocking 21m Bitcoins to secure the proof-of-stake economy,” Whitepaper, [https://docs.babylonlabs.io/papers/btc_staking_litepaper\(EN\).pdf](https://docs.babylonlabs.io/papers/btc_staking_litepaper(EN).pdf), 2023.
- [35] X. Dong, O. S. T. Litos, E. N. Tas, D. Tse, R. L. Woll, L. Yang, and M. Yu, “Remote staking with economic safety,” *CoRR*, vol. abs/2408.01896, 2024.
- [36] M. Black, T. Liu, and T. Cai, “Atomic loans: Cryptocurrency debt instruments,” arXiv preprint arXiv:1901.05117, 2019. [Online]. Available: <https://arxiv.org/abs/1901.05117>
- [37] T. Dryja, “Discreet log contracts,” <https://adiabat.github.io/dlc.pdf>, 2017, last accessed: 2026-01-12.
- [38] Lava, “Lava loans protocol v2,” <https://lava.xyz>, 2024.
- [39] Firefish, “Firefish: Bitcoin-backed loans,” Project documentation, 2024. [Online]. Available: <https://firefish.io/>
- [40] Satora, “Satora,” <https://app.satora.io/>, 2026.
- [41] D. Castejon-Molina, V. Madathil, D. Vasilopoulos, S. A. Thyagarajan, and P. Moreno-Sanchez, “Cryptographic collateralized loan without smart contracts,” Cryptology ePrint Archive, Paper 2026/1123, 2026. [Online]. Available: <https://eprint.iacr.org/2026/1123>
- [42] C. Lalor, M. Marshall, and A. Russo, “Bitcoin smart accounts: Trust-minimized native Bitcoin DeFi infrastructure,” 2026. [Online]. Available: <https://arxiv.org/abs/2603.26293>
- [43] L. Aumayr, X. Dong, R. Han, S. Sridhar, D. Tse, and N. Vanjani, “Trustless Bitcoin vaults,” Cryptology ePrint Archive, Paper 2026/2466, 2026, Extended version of this manuscript. [Online]. Available: <https://eprint.iacr.org/2026/2466>
- [44] J. A. Garay, A. Kiayias, and N. Leonardos, “The Bitcoin backbone protocol: Analysis and applications,” in *Advances in Cryptology – EUROCRYPT 2015*, ser. LNCS, vol. 9057. Springer, 2015, pp. 281–310.
- [45] V. Buterin, “A next-generation smart contract and decentralized application platform,” Ethereum White Paper, <https://ethereum.org/en/whitepaper/>, 2014.
- [46] P. Wuille, J. Nick, and A. Towns, “BIP 0341, Taproot: SegWit version 1 spending rules,” https://en.bitcoin.it/wiki/BIP_0341, January 2020, last accessed: 2026-06.
- [47] M. Herlihy, “Atomic cross-chain swaps,” in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, ser. PODC ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 245–254. [Online]. Available: <https://doi.org/10.1145/3212734.3212736>
- [48] T. Nolan, “Atomic swaps using cut and choose,” Bitcointalk Forum, 2016, <https://bitcointalk.org/index.php?topic=1364951>.
- [49] F. Vogelsteller and V. Buterin, “ERC-20: Token standard,” Ethereum Improvement Proposals, <https://eips.ethereum.org/EIPS/eip-20>, 2015.
- [50] J. Groth, “On the size of pairing-based non-interactive arguments,” in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326.
- [51] Boundless, “The Signal: Ethereum (TSEth),” <https://github.com/boundless-xyz/Signal-Ethereum>, 2026, GitHub repository. Accessed: 2026-06-10.
- [52] Bitcoin Optech, “Child pays for parent (CPFP),” <https://bitcoinops.org/en/topics/cpfp/>, 2024.
- [53] G. Zhao, “BIP-431: Topology restrictions for pinning (version 3 / TRUC transactions),” <https://github.com/bitcoin/bips/blob/master/bip-0431.mediawiki>, 2024.
- [54] mempool.space, “mempool.space: An open-source Bitcoin explorer and mempool visualizer,” <https://mempool.space>, 2024.
- [55] L. Lamport, “Constructing digital signatures from a one-way function,” SRI International Computer Science Laboratory, Technical Report SRI-CSL-98, Oct. 1979.
- [56] N. Bitansky, R. Canetti, O. Paneth, and A. Rosen, “On the existence of extractable one-way functions,” in *46th Annual ACM Symposium on Theory of Computing*, D. B. Shmoys, Ed. New York, NY, USA: ACM Press, May 31 – Jun. 3, 2014, pp. 505–514.
- [57] E. Boyle and R. Pass, “Limits of extractability assumptions with distributional auxiliary input,” in *Advances in Cryptology – ASIACRYPT 2015, Part II*, ser. Lecture Notes in Computer Science, T. Iwata and J. H. Cheon, Eds., vol. 9453. Auckland, New Zealand: Springer Berlin Heidelberg, Germany, Nov. 30 – Dec. 3, 2015, pp. 236–261.

Appendix A.

Formal Security Proofs of the TBV Protocol

This appendix gives the formal security analysis of the TBV protocol of Sec. 5.1: we prove that it satisfies the formal security definitions of Defs. 2 to 6, with explicit time bounds.

Standing assumptions. Throughout this appendix we assume (definitions in Appendix D):

- (S1) Bitcoin and Ethereum ledgers are safe (cf. Def. 8) and live (cf. Def. 10); combining liveness with τ -chain growth (cf. Thm. D.1), we write u_{BTC} (resp. u_{ETH}) for the number of rounds within which an unstoppable transaction written by an honest party appears in the views of Bitcoin (resp. Ethereum) of all honest parties;
- (S2) all digital signature schemes are EUF-CMA secure and Hash is a random oracle;
- (S3) $\Pi_{\text{BITVM-CORE}}$ satisfies setup correctness (cf. Def. 15), $u_{\text{core-robustness}}$ (cf. Def. 13), and knowledge soundness (cf. Def. 14);
- (S4) the Ethereum light client LC is safe (cf. Def. 18), live (cf. Def. 19), and $u_{\text{LC-timely}}$ (cf. Def. 20);
- (S5) the application App satisfies application soundness (Def. 1);
- (S6) the off-chain Phases 3–4 of Setup complete within δ_{setup} rounds, and Phase 2 of Withdraw (witness generation) completes within δ_{wit} rounds, whenever the executing parties are honest;
- (S7) for $X \in \{BTC, ETH\}$, with probability $1 - \text{negl}$, every honest party's view of $\mathcal{L}_X$ grows by at most $\beta_X \cdot s$ blocks over any s rounds; writing τ_X for the chain-growth constant of $\mathcal{L}_X$ (cf. Thm. D.1), any n consecutive blocks of $\mathcal{L}_X$ thus span between n/β_X and n/τ_X rounds.

Parameter compatibility. The protocol timeouts must be mutually consistent. Recall that T_{stale} , T_{refund} , k are measured in Bitcoin blocks, T_{activate} in Ethereum blocks, and all remaining parameters in rounds; we convert between them via (S7). We require:

- (C1) the activation deadline outlasts Phases 2–6 of an all-honest Setup even when Ethereum blocks are produced at the maximal rate:

$$T_{\text{activate}}/\beta_{\text{ETH}} \geq u_{\text{BTC}} + \delta_{\text{setup}} + k/\tau_{\text{BTC}} + 3u_{\text{ETH}};$$

- (C2) the refund leaf stays locked past the latest possible confirmation of $\text{tx}_{\text{Deposit}}$ even when Bitcoin blocks are produced at the maximal rate:

$$T_{\text{refund}}/\beta_{\text{BTC}} \geq T_{\text{stale}}/\tau_{\text{BTC}} + T_{\text{activate}}/\tau_{\text{ETH}} + u_{\text{BTC}}.$$

Auxiliary lemmas.

Lemma 1 (Secret secrecy). *Suppose Dep is honest. Until Dep broadcasts his $\text{activateVault}(V, s)$ call in Phase 6 of Setup, no p.p.t. adversary outputs s , except with negligible probability.*

Proof. $s \xleftarrow{\$} \{0, 1\}^\lambda$ is sampled by Dep and the only information about s released before Phase 6 is $h = \text{Hash}(s)$. Producing s from h is a preimage computation against the random oracle (S2). $\square$

Lemma 2 (Unstoppability). *Suppose at least one member of $\{V.\text{Dep}\} \cup \text{App.Keepers}$ is honest. Then (i) $\text{tx}_{\text{PreDeposit}}$ is unstoppable when Dep broadcasts it in Phase 2; and (ii) once s is public and until T_{refund} blocks have elapsed since the inclusion of $\text{tx}_{\text{PreDeposit}}$, $\text{tx}_{\text{Deposit}}$ is unstoppable.*

Proof. (i) $\text{tx}_{\text{PreDeposit}}$ spends only Dep's own inputs; invalidating it requires forging Dep's signature (S2). (ii) Before T_{refund} elapses, the refund leaf is blocked by $\text{RelTimelock}(T_{\text{refund}})$, so the deposit leaf is the only spendable path of the $\text{tx}_{\text{PreDeposit}}$ output. Any spend of the deposit leaf other than $\text{tx}_{\text{Deposit}}$ requires every party's signature on a different skeleton; the honest member signed no such skeleton, so producing one is an EUF-CMA forgery (S2). $\square$

Lemma 3 (Vault spend-restriction). *Suppose at least one member of $\{V.\text{Dep}\} \cup \text{App.Keepers}$ is honest. Then, except with negligible probability, every transaction spending the vault UTXO that appears in an honest party's view of $\mathcal{L}_{\text{BTC}}$ belongs to $\bigcup_P \mathcal{S}_{V,P}$.*

Proof. The vault UTXO carries a single leaf requiring a signature from every member of $\{V.\text{Dep}\} \cup \text{App.Keepers}$ on the spending skeleton. The only multi-party signatures the honest member produced during Setup are those on the transactions of $\bigcup_P \mathcal{S}_{V,P}$ (Phase 3) and on $\text{tx}_{\text{Deposit}}$ itself (Phase 4); the latter spends the $\text{tx}_{\text{PreDeposit}}$ output, not the vault UTXO. Any other spend of the vault UTXO therefore contains an EUF-CMA forgery of the honest member's signature (S2). $\square$

Lemma 4 (Withdrawal attribution). *Suppose $P^\dagger \in \{V.\text{Dep}\} \cup \text{App.Keepers}$ is honest and a transaction from $\mathcal{S}_{V,P'}$ with $P' \neq P^\dagger$ appears in an honest party's view of $\mathcal{L}_{\text{BTC}}$ at round r . Then, except with negligible probability, $\text{Closed}(V, P') \in \mathcal{L}_{\text{ETH}}^{P_2, r'}$ for every honest party P_2 of $\mathcal{L}_{\text{ETH}}$ and every round $r' \geq r + u_{\text{LC}}$.*

Proof. $P^\dagger \in \mathcal{V}_{P'}$, i.e. the honest $P^\dagger$ is one of the verifiers of P' 's BITVM-CORE instance and ran V_{Setup} and V_{Prove} honestly. Since a success transaction of that instance is on $\mathcal{L}_{\text{BTC}}$ at round r , knowledge soundness (S3) yields an extractor that outputs, from $P^\dagger$'s view at round r , a witness $w = (\mathcal{L}_{\text{ETH}}^*, w)$ with $\text{LC}(\text{Closed}(V, P'), \mathcal{L}_{\text{ETH}}^*, w) = 1$. Light-client safety (S4) gives $\text{Closed}(V, P') \in \mathcal{L}_{\text{ETH}}^*$, and $u_{\text{LC-timeliness}}$ (S4) gives $\mathcal{L}_{\text{ETH}}^* \preceq \mathcal{L}_{\text{ETH}}^{P_2, r'}$ for every honest P_2 and every $r' \geq r + u_{\text{LC}}$, whence $\text{Closed}(V, P') \in \mathcal{L}_{\text{ETH}}^{P_2, r'}$. $\square$

Main theorems.

Theorem A.1 (Setup correctness). *Under (S1)–(S3), (S6)–(S7), and (C1)–(C2), the TBV protocol of Sec. 5.1 satisfies T_1 -setup correctness (Def. 2) with $T_1 = 4u_{\text{ETH}} + 2u_{\text{BTC}} + \delta_{\text{setup}} + k/\tau_{\text{BTC}}$.*

Proof. All parties are honest, so we walk forward through the phases of Setup: the registerDeposit call confirms within u_{ETH} (Phase 1); $tx_{PreDeposit}$ confirms within u_{BTC} (Lem. 2(i), Phase 2); the BITVM-CORE setups complete without abort by setup correctness (S3) within δ_{setup} rounds, and no party aborts since all artifacts verify (Phases 3–4, one u_{ETH} for posting signatures); every keeper ACKs once $tx_{PreDeposit}$ reaches depth k , which takes at most k/τ_{BTC} rounds by (S7) (Phase 5, one u_{ETH}); Dep activates within $T_{activate}$ by (C1), so Approved(V) confirms within u_{ETH} (Phase 6); finally any keeper completes $tx_{Deposit}$, which is unstoppable until T_{refund} (Lem. 2(ii), (C2)) and confirms within u_{BTC} (Phase 7). Summing the bounds gives T_1 ; both Deposited(V) and Approved(V) persist at all later rounds by ledger safety (S1). $\square$

Theorem A.2 (Setup liveness). *Under (S1)–(S2), the TBV protocol satisfies T_2 -setup liveness (Def. 3) with $T_2 = u_{ETH}$.*

Proof. Suppose Deposited(V) $\in \mathcal{L}_{BTC}^{P_1, r_1}$. By ledger validity, the witness of $tx_{Deposit}$ contains a preimage of h . By Lem. 1, the preimage is s and it became public no earlier than honest Dep’s broadcast of activateVault(V, s); hence the broadcast happened at some round $t_b \leq r_1$. Honest Dep calls activateVault only after ACKAccepted(V) is on-chain, so the call does not revert; and even if an adversary front-runs it with a copy carrying the same s , that copy also emits Approved(V). In either case a valid activateVault(V, s) transaction is included within u_{ETH} of t_b , so Approved(V) $\in \mathcal{L}_{ETH}^{P_2, r_2}$ for every honest P_2 and every $r_2 \geq r_1 + u_{ETH}$; persistence follows from ledger safety (S1). $\square$

Theorem A.3 (Setup safety). *Under (S1)–(S2) and (C2), if at least one member of $\{V.Dep\} \cup App.Keepers$ is honest, the TBV protocol satisfies T_3 -setup safety (Def. 4) with $T_3 = u_{BTC}$.*

Proof. Suppose Approved(V) $\in \mathcal{L}_{ETH}^{P_2, r_2}$. By the checks of $\mathcal{C}_{VM}$, ACKAccepted(V) and DepositSignaturesPosted(V) precede it on $\mathcal{L}_{ETH}$; moreover the payload of Approved(V) reveals s . We claim the pre-signatures posted in Phase 4 are complete and valid. Completeness is checked by $\mathcal{C}_{VM}$. Validity is not, so it must come from the honest member $P^\dagger \in \{V.Dep\} \cup App.Keepers$, and it does in either case. If $P^\dagger \in App.Keepers$: ACKAccepted(V) requires an ACK from every keeper, hence from $P^\dagger$, and an honest keeper ACKs only after verifying the posted signatures (Phase 5). If $P^\dagger = Dep$: by Lem. 1, s can become public only through Dep’s own activateVault(V, s) call, and honest Dep issues that call only after performing the same verification himself (Phase 6). Either way the signatures on $\mathcal{L}_{ETH}$ are valid. Hence a complete valid witness for $tx_{Deposit}$ is assemblable from on-chain data alone, and the honest member — depositor or keeper — broadcasts $tx_{Deposit}$ per Phase 7. By Lem. 2(ii) and (C2), $tx_{Deposit}$ is unstoppable in this window and confirms in all honest views of $\mathcal{L}_{BTC}$ within u_{BTC} rounds; it persists thereafter by ledger safety (S1). Thus Deposited(V) $\in \mathcal{L}_{BTC}^{P_1, r_1}$ for all $r_1 \geq r_2 + u_{BTC}$. $\square$

Theorem A.4 (Withdrawal liveness). *Under (S1)–(S7) and (C2), if $P \in \{V.Dep\} \cup App.Keepers$ is honest, the TBV protocol satisfies T_4 -withdrawal liveness (Def. 5) with $T_4 = \delta_{wit} + u_{core}$.*

Proof. Suppose Closed(V, P) $\in \mathcal{L}_{ETH}^{P_2, r_2}$. First, by application soundness (S5), Approved(V) $\in \mathcal{L}_{ETH}$, and by Thm. A.3 (with honest member P) the vault UTXO exists on $\mathcal{L}_{BTC}$. Second, the vault UTXO is unspent when P acts: by Lem. 3 any earlier spend lies in some $\mathcal{S}_{V, P'}$; $P' = P$ is excluded since honest P has not yet withdrawn, and $P' \neq P$ would, by Lem. 4, place Closed(V, P') in honest views of $\mathcal{L}_{ETH}$ alongside Closed(V, P), contradicting the at-most-one-Closed clause of application soundness (S5). Third, honest P waits until Closed(V, P) is confirmed in its local view of $\mathcal{L}_{ETH}$ and computes the light-client witness; Phase 2 of Withdraw completes within δ_{wit} rounds (light-client liveness (S4), (S6)), and P runs P_{Prove} against $\mathcal{V}_P$. By u_{core} -robustness (S3) — whose adversary controls all verifiers, matching our setting — a success transaction from $\mathcal{S}_{V, P}$, i.e. Withdrawn(V, P), confirms within u_{core} rounds and persists by ledger safety (S1). Summing gives T_4 . $\square$

Theorem A.5 (Withdrawal safety). *Under (S1)–(S4), if $P \in \{V.Dep\} \cup App.Keepers$ is honest, the TBV protocol satisfies T_5 -withdrawal safety (Def. 6) with $T_5 = u_{LC}$.*

Proof. Suppose Withdrawn(V, P') $\in \mathcal{L}_{BTC}^{P_1, r_1}$. By Lem. 3, this transaction belongs to $\mathcal{S}_{V, P'}$. If $P' = P$: honest P broadcasts its success transaction only after observing Closed(V, P) confirmed in its local view of $\mathcal{L}_{ETH}$, so Closed(V, P) precedes r_1 and persists by ledger safety (S1). If $P' \neq P$: Lem. 4 (with $P^\dagger = P$) gives Closed(V, P') $\in \mathcal{L}_{ETH}^{P_2, r_2}$ for every honest party P_2 and every round $r_2 \geq r_1 + u_{LC}$. $\square$

Realizing the TBV primitive. Taken together, Thms. A.1 to A.5 establish that the construction of Sec. 5.1 satisfies Defs. 2 to 6 for every application App satisfying application soundness (Def. 1).

Appendix B. Implementation Details

Table 5 gives the conditions for each application’s VaultOwner transitions.

B.1. The Deposit and Withdrawal Transaction Graph

Fig. 3 shows the deposit and withdrawal transaction graph of Sec. 6 for one depositor Dep, one prover P, and one verifier Vf. We describe each path below.

Deposit. The deposit realizes Setup on Bitcoin in two transactions. The depositor posts Pre-Deposit, which locks the BTC behind a hashlock h with a timed refund path. If the BTC vault is activated on Ethereum, then anyone can submit Deposit that spends Pre-Deposit and can be

TABLE 5: Example applications, specified by the conditions under which each VaultOwner transition fires. Each application emits $\text{Closed}(V, P)$ at most once, when the current owner P requests to close vault V .

Application	App.Keepers	VaultOwner(V) $\leftarrow V.\text{Dep}$ when	VaultOwner(V) $\leftarrow \perp$ when	VaultOwner(V) $\leftarrow P \in \text{App.Keepers}$ when
Lending	liquidators	Approved(V), or Dep repays the full debt	Dep borrows against the collateral	P repays the full debt and $\text{LTV} > \text{LLTV}$
Stablecoin	auction participants	Approved(V), or Dep burns the full stablecoin debt	Dep mints stablecoins against the collateral	P wins the auction and burns the full stablecoin debt, and $\text{LTV} > \text{LLTV}$
Perp DEX	P&L obligation coverers	Approved(V), or Dep closes the position and settles its P&L	Dep opens a perpetual position on the BTC margin	P closes the position and settles its P&L, and $M < \text{MMR}$

spent only by the depositor together with the keepers. If the BTC vault is not activated on Ethereum within the activation deadline Δ_a , then the depositor can recover the funds in Pre-Deposit once its refund timelock Δ_r expires. The confirmed Deposit transaction creates the vault UTXO, i.e., it is the $\text{Deposited}(V)$ transaction of Sec. 4.

Withdrawal (no challenge). If there is no challenge, a withdrawal realizes Withdraw in three transactions. The prover posts Claim to declare the intent to withdraw, and posts Assert that commits the Groth16 proof encoded in a Lamport signature under its key lpk_P , enforced by the $\text{CheckLampSig}(\text{lpk}_P)$ condition in Fig. 3. Once the challenge period Δ_2 passes, the prover posts Withdraw that spends the Deposit output, i.e., the $\text{Withdrawn}(V, P)$ transaction of Sec. 4 for the withdrawing party P .

Withdrawal (correctly challenged). In order to challenge Claim and Assert from a prover, the verifier submits ChallengeAssertX and ChallengeAssertY together. The two transactions jointly realize BABE’s ChallengeAssert for one prover-verifier pair. They reveal the garbled circuit (GC) input labels that encode the prover’s Groth16 proof: the $\text{CheckLampSigsMatch}(\text{lpk}_P, \text{lpk}_{V_f}^X)$ and $\text{CheckLampSigsMatch}(\text{lpk}_P, \text{lpk}_{V_f}^Y)$ conditions require the verifier to publish Lamport signatures under its one-time keys $\text{lpk}_{V_f}^X$ and $\text{lpk}_{V_f}^Y$ on the same message that the prover signed under lpk_P , and these signatures reveal the input labels. If the Groth16 proof is invalid, the prover cannot recover the encrypted secret and thus cannot submit WronglyChallenged or Withdraw . Once the response period Δ_1 elapses, the verifier posts NoWithdraw to block Withdraw .

Withdrawal (wrongly challenged). If the Groth16 proof is valid but a verifier still posts ChallengeAssertX and ChallengeAssertY , then the prover can evaluate the garbled circuits using the verifier’s input labels and decrypt the secret. The prover can then use the secret as the witness for WronglyChallenged : its $\text{HashLock}(h')$ condition fixes h' to the hash of the secret, whose preimage the prover can recover only by evaluating the GC on a valid proof. It then proceeds to Withdraw as in the no-challenge case.

B.2. Smart Contracts and Aave v4 Integration

This section details the smart contract implementation summarized in Sec. 6. We integrate BTC Vault contracts with Aave v4 [6], a major Ethereum lending protocol, *without modifying Aave*. The BTC Vault core invokes the adapter, and the adapter drives Aave v4 only through its public interface.

A BTC vault is a *non-fungible* collateral unit: a whole Bitcoin UTXO that cannot be split or merged. Aave v4, however, accepts collateral only as an ERC-20 token (Appendix D.5). We bridge this gap with the wBTC token, a synthetic ERC-20 token that the adapter mints 1:1 with each vault. It serves as a translation layer between BTC vaults and Aave’s collateral accounting. Since the underlying vaults are non-fungible, wBTC cannot be allowed to behave like an ordinary fungible token. It must not be tradable on secondary markets or composable with other protocols since users other than Dep and the liquidators cannot redeem the wBTC they obtain.

We codify these spending conditions directly in the ERC-20 token contract (List. 1). The adapter is the only minter and burner of wBTC, with one unit of supply per onboarded vault. Its transfer and transferFrom admit only protocol-controlled destinations: the Aave Spoke, the Aave Hub, the adapter, and the borrower’s isolated position account. They revert on any other recipient.

The token alone cannot capture which *specific* vaults back a position, because equal wBTC balances are interchangeable in the ERC-20 contract. The adapter therefore tracks each BTC vault individually. Every vault carries a lifecycle status and an owner authorized to withdraw it on Bitcoin. Each borrower’s position corresponds to an *ordered list of BTC vault IDs*.

The lending market then consumes wBTC as an ordinary collateral reserve through Aave v4’s Hub-and-Spoke architecture. On activation, the adapter mints wBTC into a per-borrower position account and supplies it to the Aave Core Spoke. It then drives the position – borrow, repay, withdraw, and liquidate – through the Spoke’s standard interface, and the Hub provides the borrowed liquidity. No Hub or Spoke contract is modified. From Aave’s side, wBTC is just another collateral reserve.

Listing 1: Spending conditions codified in the wBTC token: it is minted and burned only by the adapter, and may be transferred only to protocol-controlled destinations.

```

1 // wBTC: synthetic ERC-20 (8 decimals), 1:1 with one
  non-fungible BTC vault.
2 contract WBTC is ERC20 {
3   address adapter; // the BTC Vault adapter
4   ...
5   modifier onlyAdapter() { require(msg.sender ==
    adapter); _; }
6
7   // created and destroyed only by the adapter, one
    unit per vault
8   function mint(address to, uint256 amt) external
    onlyAdapter { _mint(to, amt); }
9   function burn(address from, uint256 amt) external
    onlyAdapter { _burn(from, amt); }
10
11  // transfer-restricted: no secondary trading, no
    DeFi composition
12  // (transferFrom is guarded identically)
13  function transfer(address to, uint256 amt) public
    override returns (bool) {
14    require(_authorized(to), "unauthorized
    destination");
15    return super.transfer(to, amt);
16  }
17
18  function _authorized(address to) internal view
    returns (bool) {
19    return to == coreSpoke // Aave Spoke for the wBTC
    reserve
20    || to == hub // Aave Hub
21    || to == adapter // BTC Vault adapter
22    || isUserProxy(to); // borrower's position
    account
23  }
24  ...
25 }

```

B.3. Deployed contract addresses

Table 6 lists the BTC Vault contracts deployed on Ethereum mainnet (chainId 1) for the evaluation in §7, each linked to its Etherscan page for independent verification. All contracts were deployed and administered by a79111C9, with the BtcPrism light client anchored at Bitcoin mainnet block height 952184 and the Aave v4 market identifier set to 1. The contracts split into the BTC Vault protocol core (the registry, protocol parameters, the wBTC token, the Aave v4 integration adapter, and the application and metadata registries), the self-hosted Aave v4 market instance against which wBTC is exercised as collateral, and the mock reserves and price feeds backing that market. We deploy a self-hosted Aave v4 instance and mock reserves because Aave v4 on mainnet is not permissionless (§7); the addresses below are the entry points the cost runs interact with, out of a full deployment of 43 contracts.

Appendix C. Hybrid TBV-Bridge: From Validity to Inclusion

Every vault $V \in \mathcal{V}$ is an instantiation of the TBV protocol for HybridApp(innerApp) with keeper set LocalKeepers $\cup$ Operators, so the setup properties (Defs. 2 to 4) and the withdrawal relays (Defs. 5 and 6) are inherited per vault,

TABLE 6: BTC Vault contract addresses on Ethereum mainnet. Each address links to Etherscan.

Contract	Address
<i>BTC Vault protocol core</i>	
BTCVaultRegistry	1a2201f1
ProtocolParams	f892d652
wBTC token	9403c2f6
AaveAdapter	9c804fa4
AaveAdapterLens	dfe7444c
ApplicationRegistry	19dca9fa
MetadataRegistry	cd8b63c8
<i>Self-hosted Aave v4 market</i>	
Aave Hub	c2349a62
Aave Spoke	30e260e2
Aave Oracle	f7f77d07
<i>Mock reserves and price feed</i>	
BTC price feed (mock)	7e8f8d15
USDC debt reserve (mock)	360deb27
WBTC (mock)	f68e6aca

unchanged. The relays convert Closed on Ethereum into Withdrawn on Bitcoin and back; the accounting layer determines which Closed are *valid*. The one remaining obligation is the step in between: validity does not imply *inclusion*, since a competing transaction may land first and invalidate a pending one. This subsection closes that gap. Let u_E denote the liveness parameter of $\mathcal{L}_{ETH}$ (cf. Def. 10).

Proposition 1 (Unstoppable closing). *Suppose HybridApp(innerApp) is sound. Let P be honest with $V \in \mathcal{U}_P(\mathcal{L}_{ETH}^{P,r})$ and P backed in $\mathcal{L}_{ETH}^{P,r}$, and suppose P posts Closed(V, P) at round r . Then for all honest parties P_2 of $\mathcal{L}_{ETH}$ and all rounds $r' \geq r + u_E$: Closed(V, P) $\in \mathcal{L}_{ETH}^{P_2,r'}$. This holds with no trust assumption on Operators or LocalKeepers $\setminus \{P\}$.*

Proof sketch. Closed(V, P) is valid (owner branch of condition 4). In any extension of the ledger not containing P 's own transactions, P remains backed (only P can spend or commit P 's tokens) and remains the owner (owner stability), so by conditional protection (Thm. 8.1) every competing Closed(V, P') is invalid and Closed(V, P) stays valid. Hence Closed(V, P) is u_E -unstoppable (cf. Def. 9) and Ethereum liveness includes it. $\square$

Proposition 2 (Exclusion). *Suppose HybridApp(innerApp) is sound. Let P be honest and backed in every honest view throughout the execution, and $V \in \mathcal{U}_P(\mathcal{L}_{ETH}^{P,r})$ at some round r . Then for all $P' \neq P$, no honest party of $\mathcal{L}_{ETH}$ ever holds Closed(V, P') in its view. This holds with no trust assumption on Operators or LocalKeepers $\setminus \{P\}$.*

Proof sketch. By conditional protection and owner stability, Closed(V, P') is invalid with respect to every prefix of every valid Ethereum ledger while P is backed; by ledger validity (cf. Def. 17), invalid transactions never appear in honest views. $\square$

Proposition 3 (Operator inclusion). *Suppose HybridApp(innerApp) is sound. Let O be honest with $\text{tokBal}_{\mathcal{L}_{ETH}^{O,r}}(O) \geq 1$, posting a valid $\text{Closed}(\cdot, O)$ from round r on and re-selecting the vault against its current view after each failure. Then there exists a vault V such that for all honest parties P_2 of $\mathcal{L}_{ETH}$ and all rounds $r' \geq r + (k+1)u_E$: $\text{Closed}(V, O) \in \mathcal{L}_{ETH}^{P_2,r'}$, where k is the number of open vaults at round r .*

Proof sketch. By existence (Thm. 8.1), O always has a valid candidate. O 's transaction need not be unstoppable: a competing valid $\text{Closed}(V', P')$ may consume the selected vault. But each interference burns a token and removes a vault from $\mathcal{U}$ while O retains its token, so existence re-applies to the updated ledger; at most k interferences can occur, and between interferences inclusion takes at most u_E rounds. $\square$

Corollary 3 (End-to-end guarantees). *Suppose HybridApp(innerApp) is sound. Composing Props. 1 to 3 with the per-vault withdrawal properties (Defs. 5 and 6) yields the following guarantees:*

- 1) (Trustless withdrawal) *An honest, backed P with $V \in \mathcal{U}_P$ obtains $\text{Withdrawn}(V, P) \in \mathcal{L}_{BTC}^{P_1,r'}$ for all honest P_1 and $r' \geq r + u_E + T_4$, trusting no other party.*
- 2) (Trustless protection) *For an honest P that remains backed, no honest party of $\mathcal{L}_{BTC}$ ever holds $\text{Withdrawn}(V, P')$ for $V \in \mathcal{U}_P$ and $P' \neq P$: otherwise T_5 -withdrawal safety would place $\text{Closed}(V, P')$ in honest Ethereum views, contradicting Prop. 2.*
- 3) (Bridge-style redemption) *Any honest wBTC holder P , given one honest operator $O \in \text{Operators}$, obtains $\text{Swap}_{BTC}(P)$ on Bitcoin via swap uniformity (cf. Def. 16); O is made whole by Prop. 3 and T_4 -withdrawal liveness, within $T_{\text{swap}} + (k+1)u_E + T_4$ rounds. Conditional protection guarantees the consumed vault was reserved for O itself or for an unbacked owner; backed parties' reservations are never consumed.*

In summary, a hybrid vault is trustless for backed parties: for each depositor or local keeper P , setup safety, withdrawal liveness, and withdrawal protection hold as long as just P is honest and P has stayed in the application. In addition, and beyond what a pure TBV offers, any wBTC holder can redeem BTC under a 1-of- m honest-operator assumption, exactly as in a bridge.

Appendix D. Preliminaries

D.1. Distributed Ledgers

Ledger Model. We model Bitcoin and Ethereum as distributed ledger functionalities $\mathcal{F}_{BTC}$ and $\mathcal{F}_{ETH}$ maintaining append-only sequences of confirmed transactions (*ledgers*). We assume a global clock which counts time in rounds $r \in \mathbb{N}$ and $r \leq \text{poly}$. Any party can submit a transaction tx to a ledger at any round r by calling $\mathcal{F}.\text{WRITE}(\text{tx})$ where

$\mathcal{F} \in \{\mathcal{F}_{BTC}, \mathcal{F}_{ETH}\}$. For a party P and round $r \in \mathbb{N}$, the notation $\mathcal{L}_{BTC}^{P,r}$ denotes party P 's local view of the Bitcoin ledger at round r and $\mathcal{L}_{ETH}^{P,r}$ denotes party P 's local view of the Ethereum ledger at round r .

The height of a ledger $\mathcal{L}$ is denoted $h(\mathcal{L})$. A ledger of height h is a sequence of blocks $\mathcal{L}[0], \dots, \mathcal{L}[h-1]$, and $\mathcal{L}[:h]$ denotes the sequence of all blocks in $\mathcal{L}$ at heights below h . $\mathcal{L}||B$ denotes the ledger formed by appending the block B to the ledger $\mathcal{L}$. Each block $\mathcal{L}[i]$ is a sequence of transactions. By flattening the blocks, the ledger itself can be viewed as a sequence of transactions. We denote by $\mathcal{L}^1 \preceq \mathcal{L}^2$ that $\mathcal{L}^1$ is a prefix of $\mathcal{L}^2$.

We assume the Bitcoin and Ethereum ledgers satisfy the two fundamental properties safety and liveness. Informally, safety ensures that the ledgers of honest parties are consistent with each other, and liveness ensures that new valid transactions are eventually included in the ledgers of honest parties. Safety is defined as in [44].

Definition 8 (Ledger Safety). A ledger functionality $\mathcal{F}$ is *safe* if with probability $1 - \text{negl}$: (i) for any honest party P and rounds $r_1 \leq r_2$, $\mathcal{L}^{P,r_1} \preceq \mathcal{L}^{P,r_2}$ (self-consistency); and (ii) for any honest parties P_1, P_2 and any round r , $\mathcal{L}^{P_1,r} \preceq \mathcal{L}^{P_2,r}$ or $\mathcal{L}^{P_2,r} \preceq \mathcal{L}^{P_1,r}$ (view-consistency).

To define liveness, we note that only valid transactions may be included in the ledger. Validity will be defined using a predicate $\text{Valid}(\mathcal{L})$ which operates differently for Bitcoin and Ethereum ledgers, as described in Appendix D.2 and Appendix D.5, respectively. However, the adversary may delay the inclusion of a valid transaction. The adversary may also include his own transactions in the ledger which may cause the honest party's transaction to become invalid thereafter, e.g., the adversary's transaction may use the same input as the honest party's transaction. Thus, liveness is guaranteed when the adversary is (computationally) unable to exclude a transaction for too many blocks. To make this precise, we extend the definition of unambiguous transactions [44] to *unstoppable transactions*. Informally, a transaction is unstoppable if inserting this transaction in any sequence of u valid blocks preserves their validity. For example, a transaction that requires the honest party's signature will be unstoppable since the adversary cannot forge the honest party's signature; hence, the inputs for the transaction cannot be spent by the adversary before the honest party's transaction is included in the ledger. We assume that for large enough u , Bitcoin satisfies liveness for u -unstoppable transactions. In general, the adversary may have access to some state st , which may include certain signatures, ciphertexts, etc. that the adversary has access to.

Definition 9 (u -Unstoppable Transactions). A transaction tx is *u -unstoppable* with respect to a ledger $\mathcal{L}$ and adversarial state st , if for all p.p.t. adversaries $\mathcal{A}$:

$$\Pr \left[\begin{array}{l} \text{Valid}(\mathcal{L}||B_1||\dots||B'_i||\dots||B_u) = 1 : \\ B_1, \dots, B_u, i, B'_i \leftarrow \mathcal{A}(\mathcal{L}, \text{st}, \text{tx}) \\ \text{Valid}(\mathcal{L}||B_1||\dots||B_u) = 1 \\ B'_i \setminus B_i = \{\text{tx}\} \end{array} \right] \geq 1 - \text{negl}(1)$$

Here, $B'_i \setminus B_i = \{\text{tx}\}$ means that the block B'_i contains the same sequence of transactions as B_i except that tx is inserted at some location in the sequence.

Definition 10 (Ledger Liveness). A ledger functionality $\mathcal{F}$ is u -live ($u \in \mathbb{N}$) if for all adversarial states st , with probability $1 - \text{negl}$, for any party P calling $\mathcal{F}.\text{WRITE}(\text{tx})$ at any round r such that tx is u -unstoppable with respect to $\mathcal{L}^{P,r}$ and st , for all honest parties H and rounds r' with $h(\mathcal{L}^{H,r'}) \geq h(\mathcal{L}^{P,r}) + u$, $\text{tx} \in \mathcal{L}^{H,r'}[h(\mathcal{L}^{P,r}) + u]$.

While the above definition of liveness guarantees inclusion of a transaction within a certain number of blocks, we use the *chain growth* property [44] to guarantee that the height of every party's ledger grows.

Theorem D.1 (τ -Chain Growth). *For all $s > \lambda$, for all honest parties P , $\Pr[\forall r : h(\mathcal{L}^{P,r+s}) < h(\mathcal{L}^{P,r}) + \tau s] \leq \text{negl}$.*

Throughout the paper, we assume that the Bitcoin and Ethereum ledgers satisfy safety, liveness, and chain growth.

D.2. The Bitcoin Ledger

UTXO Model and Taproot Scripts. The ledger's state is represented as a set of unspent transaction outputs (UTXOs). Each UTXO is a pair $\text{out} = (a, \text{lockScript})$ where $a \in \mathbb{R}_{\geq 0}$ is the amount of coins (in BTC) in that UTXO, and lockScript is a program (the *locking script*) that determines under which conditions the UTXO can be spent.

In BABE, we widely use Taproot Trees [46], or Taptrees, which make a UTXO spendable by satisfying one among multiple locking scripts. We will represent the locking script of such a UTXO as $\text{lockScript} = \langle \text{leaf}_0, \dots, \text{leaf}_{k-1} \rangle$.

Transactions. A transaction is a triple $\text{tx} = (\text{inputs}, \text{tx_witnesses}, \text{outputs})$ where:

- $\text{inputs} = [\text{in}_1, \dots, \text{in}_n]$, where each input references an output of a previous transaction, indexed as $\text{in} = (\text{PrevTx}, \text{outIndex}, \text{leaf})$ where PrevTx is the previous transaction, outIndex is the index of the output in the previous transaction, and leaf is the leaf of the Taproot tree to be satisfied.
- $\text{outputs} = [\text{out}_1, \dots, \text{out}_m]$ where each output is a pair $(a, \text{lockScript})$ as above;
- $\text{tx_witnesses} = [w_1, \dots, w_n]$, where w_i is a transaction witness (e.g., signature, data, etc.) intended to satisfy the corresponding input's leaf script.

The pair $\bar{\text{tx}} = (\text{inputs}, \text{outputs})$ is called the *transaction skeleton*.

Cryptographic Primitives. We assume the existence of the following cryptographic primitives that are used by the ledger functionality:

- **Signature Scheme:** Sig_{BTC} with algorithms $\text{Sig}_{BTC}.\text{Gen}(1^\lambda) \rightarrow (\text{sk}, \text{pk})$, $\text{Sig}_{BTC}.\text{Sign}(\text{sk}, \text{tx}) \rightarrow \sigma$, and $\text{Sig}_{BTC}.\text{Verify}(\text{pk}, \bar{\text{tx}}, \sigma) \rightarrow 0/1$ which is EUF-CMA secure.
- **Hash Function:** $\text{Hash}_{BTC}(m) \rightarrow h$ modeled as a random oracle.

- **Lamport One-Time Signature Scheme [55]:** LampSig with algorithms $\text{LampSig}.\text{Gen}(1^\lambda, \ell) \rightarrow (\text{lsk}, \text{lpk})$, $\text{LampSig}.\text{Sign}(\text{lsk}, m) \rightarrow \mu$, and $\text{LampSig}.\text{Verify}(\text{lpk}, m, \mu) \rightarrow 0/1$ where ℓ is the number of bits in the message m . A Lamport signature scheme constructed using Hash_{BTC} can be efficiently verified in Bitcoin script.

$$\begin{aligned} \text{lsk} &:= \begin{pmatrix} L_0^0 & \dots & L_{\ell-1}^0 \\ L_0^1 & \dots & L_{\ell-1}^1 \end{pmatrix} \xleftarrow{\$} (\{0, 1\}^\lambda)^{2 \times \ell} \\ \text{lpk} &:= \begin{pmatrix} \text{Hash}_{BTC}(L_0^0) & \dots & \text{Hash}_{BTC}(L_{\ell-1}^0) \\ \text{Hash}_{BTC}(L_0^1) & \dots & \text{Hash}_{BTC}(L_{\ell-1}^1) \end{pmatrix} \\ \mu &:= (L_0^{m_0} \dots L_{\ell-1}^{m_{\ell-1}}) \\ \text{LampSig}.\text{Verify}(\text{lpk}, m, \mu) &= 1 \iff \\ \forall i \in \{0, \dots, \ell-1\}, \text{Hash}_{BTC}(\mu_i) &= \text{lpk}_i^{m_i} \end{aligned}$$

Locking Scripts. The locking scripts of Bitcoin that are used by BABE are described below:

- **Signature Check:** $\text{CheckSig}(\text{pk})$ — Requires w to contain a digital signature σ such that $\text{Sig}_{BTC}.\text{Verify}(\text{pk}, \bar{\text{tx}}, \sigma) = 1$.
- **Relative Timelock:** $\text{RelTimelock}(\tau)$ — Requires that at least τ blocks have elapsed since the output referenced by the input was created. That is, if a transaction with an input of the form $\text{in} = (\text{PrevTx}, \text{outIndex}, \langle \text{RelTimelock}(\tau) \rangle)$ is included in a block at height h , then PrevTx must have been included in a block at height at most $h - \tau$.
- **Hash Lock:** $\text{HashLock}(h)$ — Requires that the transaction witness contains a hash pre-image of h , i.e. w such that $\text{Hash}_{BTC}(w) = h$.
- **Logical Operations:** The above scripts can be combined using logical operators $\wedge$ (AND) and $\vee$ (OR) to form more complex locking conditions.

Using the above scripts and logical operators $\wedge$ (AND) and $\vee$ (OR), we can define more complex locking scripts, for example:

- **Lamport Signature:** $\text{CheckLampSig}(\text{lpk})$ — Requires that the transaction witness contains a Lamport signature for the public key lpk on some message. In BABE, we will use this script to ensure that the Prover, while posting this transaction, makes a binding commitment to some proof.

$$\text{CheckLampSig}(\text{lpk}) := \bigwedge_{i=0}^{\ell-1} \left(\text{HashLock}(\text{lpk}_i^0) \vee \text{HashLock}(\text{lpk}_i^1) \right) \quad (2)$$

- **Check Lamport Signatures from Multiple Parties on the Same Message:** $\text{CheckLampSigsMatch}(\text{lpk}_A, \text{lpk}_B)$ — Requires that the transaction witness contains a Lamport signature for each public key $\text{lpk}_A, \text{lpk}_B$, both on the same message. In BABE, we will use this script to ensure that the Verifier (who holds only lsk_B) commits to the same proof that the Prover (who holds lsk_A) committed to. This is guaranteed because the Verifier, who does not know lsk_A , cannot create a valid Lamport

signature under lpk_A for any message other than the one committed to by the Prover. The Prover can then use the Verifier's commitment to evaluate the garbled circuit.

$$\begin{aligned} \text{CheckLampSigsMatch}(\text{lpk}_A, \text{lpk}_B) &:= \\ \bigwedge_{i=0}^{\ell-1} &\left[\left(\text{HashLock}((\text{lpk}_A)_i^0) \wedge \text{HashLock}((\text{lpk}_B)_i^0) \right) \right. \\ &\left. \vee \left(\text{HashLock}((\text{lpk}_A)_i^1) \wedge \text{HashLock}((\text{lpk}_B)_i^1) \right) \right] \quad (3) \end{aligned}$$

Definition 11 (Transaction Validity). A transaction $\text{tx} = (\text{inputs}, \text{tx_witnesses}, \text{outputs})$ is *valid* with respect to a ledger $\mathcal{L}$, denoted $\text{Valid}_{\mathcal{L}}(\text{tx})$, if:

- 1) *All Inputs Unspent*: For each $\text{in}_i = (\text{PrevTx}, \text{outIndex}, \text{leaf})$ in inputs, the transaction PrevTx exists in $\mathcal{L}$. The referenced output must exist and must not have been spent by any earlier transaction in $\mathcal{L}$, and no two inputs of tx may reference the same output.
- 2) *All Locking Scripts Satisfied*: Each input's specified leaf script is one of the leaves of the tap-tree for that input. That is, for each $\text{in}_i = (\text{PrevTx}, \text{outIndex}, \text{leaf})$, $\text{leaf} \in \langle \text{leaf}_0, \dots, \text{leaf}_{k-1} \rangle = \text{PrevTx.outputs}[\text{outIndex}].\text{lockScript}$. Moreover, the transaction witness $\text{tx_witnesses}[i]$ must satisfy the script leaf.
- 3) *Value Preservation*: The total amount of coins in the outputs is less than or equal to the total amount of coins in the inputs.

$$\sum_{j=1}^m \text{outputs}[j].a \leq \sum_{i=1}^n \text{in}_i.\text{PrevTx.outputs}[\text{in}_i.\text{outIndex}].a$$

A ledger $\mathcal{L}$ is *valid*, denoted $\text{Valid}(\mathcal{L})$, if for all transactions tx in $\mathcal{L}$, $\text{Valid}_{\mathcal{L}[:\text{tx}]}(\text{tx}) = 1$ ($\mathcal{L}[:\text{tx}]$ is the ledger containing all transactions in $\mathcal{L}^{P,r}$ before tx).

Definition 12 (Ledger Validity). If a transaction tx appears in the ledger view $\mathcal{L}^{P,r}$ of any party P at any round r , then tx is valid with respect to the state $\mathcal{L}^{P,r}[:\text{tx}]$ (the ledger containing all transactions in $\mathcal{L}^{P,r}$ before tx).

D.3. BITVM-CORE with Multiple Verifiers

This section extends the definition of BITVM-CORE [15] to the setting of multiple verifiers. Appendix E describes a protocol that extends BABE [15] to realize BITVM-CORE with multiple verifiers.

A BITVM-CORE protocol is an interactive protocol run between a Prover P and a set of Verifiers $\mathcal{V}$ interacting with the Bitcoin ledger functionality $\mathcal{F}_{\text{BTC}}$. The protocol consists of two phases: a setup phase and a proving phase. In the setup phase, the Prover and Verifiers are given a common reference string crs and a statement $\mathbf{x}$ to be proven, and interact off-chain to agree on a set of Bitcoin transactions $\mathcal{S}$. These transactions are called the *success transactions* because they can be posted on Bitcoin in the proving phase if and only if he has a valid witness for the statement. During

setup, the Prover and Verifiers also obtain and store local state (e.g., transactions, ciphertexts, private keys) to be used in the proving phase. To defend against malicious behavior, both the Prover and Verifiers validate information received from the other parties and may abort the setup if they detect invalid information.

During the proving phase, the Prover and Verifiers interact with the Bitcoin ledger. The Prover, who now knows a witness for the statement to be proven, and the Verifiers take turns posting transactions to the Bitcoin ledger. In the end, the Prover wins by posting a success transaction from $\mathcal{S}$. Security means that the Prover must win if he has a valid witness (even if all the Verifiers are malicious) and any one honest Verifier can prevent the Prover from winning otherwise (even if the Prover and all other Verifiers are malicious).

Protocol Syntax. A BITVM-CORE protocol for a relation $\mathcal{R}$ consists of the following *p.p.t.* algorithms:

- $\text{Gen}(\mathcal{R}) \rightarrow \text{crs}$: Takes as input the relation and outputs a common reference string crs .
- $P_{\text{Setup}}(\text{crs}, \mathbf{x})$: Run by the Prover during the setup phase, interacts with the Verifiers and the ledger functionality $\mathcal{F}_{\text{BTC}}$, and outputs the Prover's state st_P and a set of transactions $\mathcal{S}$. Alternatively, the Prover may output $\perp$ to indicate that the setup failed.
- $V_{\text{Setup}}(\text{crs}, \mathbf{x})$: Run by each Verifier during the setup phase, interacts with the Prover and the ledger functionality $\mathcal{F}_{\text{BTC}}$, and outputs the Verifier's state st_V and a set of transactions $\mathcal{S}$. Alternatively, the Verifier may output $\perp$ to indicate that the setup failed.
- $P_{\text{Prove}}(\text{st}_P, \mathbf{w})$: Run by the Prover knowing a witness $\mathbf{w}$, interacts with the ledger functionality $\mathcal{F}_{\text{BTC}}$, and outputs a bit $b \in \{0, 1\}$ ($b = 1$ indicates success of the proving phase).
- $V_{\text{Prove}}(\text{st}_V)$: Run by each Verifier during the proving phase, interacts with the ledger functionality $\mathcal{F}_{\text{BTC}}$.

We denote by $\langle P_{\text{Setup}}(\text{crs}, \mathbf{x}), V_{\text{Setup}}(\text{crs}, \mathbf{x}) \rangle$ the random variable representing the transcript of the setup phase run interactively by the Prover and the Verifiers. The outputs of the Prover P and Verifier $V \in \mathcal{V}$ during setup are denoted respectively by

$$\text{out}_P(\langle P_{\text{Setup}}(\text{crs}, \mathbf{x}), V_{\text{Setup}}(\text{crs}, \mathbf{x}) \rangle) = (\mathcal{S}, \text{st}_P) \text{ or } \perp \quad (4)$$

$$\text{out}_V(\langle P_{\text{Setup}}(\text{crs}, \mathbf{x}), V_{\text{Setup}}(\text{crs}, \mathbf{x}) \rangle) = (\mathcal{S}, \text{st}_V) \text{ or } \perp. \quad (5)$$

Similarly, the outputs of each party during the proving phase are denoted respectively by $\text{out}_P(\langle P_{\text{Prove}}(\cdot), V_{\text{Prove}}(\cdot) \rangle)$ and $\text{out}_V(\langle P_{\text{Prove}}(\cdot), V_{\text{Prove}}(\cdot) \rangle)$ where $(\cdot)$ is replaced by the inputs of the corresponding algorithm. We use $\langle P_{\text{Prove}}(\cdot), V_{\text{Prove}}(\cdot) \rangle_r$ to denote that the proving phase began at round r whenever the starting round is relevant.

Robustness ensures that as long as the Prover is honest, has accepted that the setup was performed correctly, and has a valid witness, malicious Verifiers cannot prevent the Prover from succeeding (i.e., the required success transaction appears on the ledger) except with negligible probability. Thus, the worst thing a malicious Verifier can do is to cause

the Prover to abort the setup, in which case no Bitcoin is ever lost by any party.

Definition 13 (*u*-Robustness). For all NP relations $\mathcal{R}$, for all statements x and witnesses w such that $(x, w) \in \mathcal{R}$, for all *p.p.t.* adversaries $\mathcal{V}^*$ controlling all the verifiers, and all rounds $r \in \mathbb{N}$, the following holds:

$$\Pr \left[\begin{array}{l} (\text{out}_P \neq \perp) \wedge ((b \neq 1) \vee \\ (\forall \text{tx} \in \mathcal{S}, \text{tx} \notin \mathcal{L}^{P,r+u})) : \\ \text{crs} \leftarrow \text{Gen}(\mathcal{R}) \\ \text{out}_P \leftarrow \text{out}_P(\langle P_{\text{Setup}}(\text{crs}, x), \mathcal{V}^*(\text{crs}, x) \rangle) \\ \text{If } \text{out}_P \neq \perp : \\ (\mathcal{S}, \text{st}_P) \leftarrow \text{out}_P \\ b \leftarrow \text{out}_P(\langle P_{\text{Prove}}(\text{st}_P, w), \mathcal{V}^*(\text{crs}, x) \rangle_r) \end{array} \right] \leq 2^{-\kappa} + \text{negl} \quad (6)$$

Knowledge soundness ensures that if any honest Verifier accepts the setup, then an adversary controlling the Prover and all other verifiers cannot successfully include the success transaction on Bitcoin unless he knows a valid witness for the chosen relation. As usual in knowledge soundness, the Prover's knowledge of the witness is defined by the existence of an extractor that can extract the witness using the adversary's inputs and its view View_{P^*} of its interactions with the honest Verifier and Bitcoin. Every algorithm is stateful. For an execution of an algorithm $\mathcal{A}$, $\text{View}_{\mathcal{A}}$ consists of all inputs received throughout the execution together with its random coins. Below, assigning the proving interaction to τ_{Prove} denotes executing that interaction, including its ledger effects; its parties' outputs are not used.

Definition 14 (Knowledge Soundness). For all NP relations $\mathcal{R}$, for every honest $V \in \mathcal{V}$ and every *p.p.t.* adversary P^* controlling the Prover and all other verifiers, there exists a *p.p.t.* extractor $\mathcal{E}_{P^*}$ such that for every benign ⁴ auxiliary input $\text{aux} \in \{0, 1\}^{\text{poly}}$:

$$\Pr \left[\begin{array}{l} (\text{out}_V \neq \perp) \wedge (\exists r \in \mathbb{N}, \text{tx} \in \mathcal{S} : \text{tx} \in \mathcal{L}^{V,r}) \\ \wedge (x, w) \notin \mathcal{R} : \\ \text{crs} \leftarrow \text{Gen}(\mathcal{R}) \\ x \leftarrow P^*(\text{crs}, \text{aux}) \\ \text{out}_V \leftarrow \text{out}_V(\langle P^*(\text{crs}, \text{aux}), V_{\text{Setup}}(\text{crs}, x) \rangle) \\ \text{If } \text{out}_V \neq \perp : \\ (\mathcal{S}, \text{st}_V) \leftarrow \text{out}_V \\ \tau_{\text{Prove}} \leftarrow \langle P^*(\text{crs}, \text{aux}), V_{\text{Prove}}(\text{st}_V) \rangle \\ w \leftarrow \mathcal{E}_{P^*}(\text{View}_{P^*}) \end{array} \right] \leq 2^{-\kappa} + \text{negl} \quad (7)$$

Remark 4. Assuming the existence of strong cryptographic primitives, there are auxiliary information distributions for which not all SNARKs admit an extractor [56], [57]. Following the SNARK literature, we therefore formally assume that the adversary should have access only to “benign” auxiliary

4. Assuming the existence of strong cryptographic primitives, there are auxiliary information distributions for which not all SNARKs admit an extractor [56], [57]. Following the SNARK literature, we therefore formally assume that the adversary should have access only to “benign” auxiliary inputs. Notably, these results are highly theoretical and do not affect SNARKs' security in practice.

inputs. Notably, these results are highly theoretical and do not affect SNARKs' security in practice.

Together, robustness and knowledge soundness make a BITVM-CORE protocol *trustless*. It is trustless because once the Prover and Verifiers have mutually accepted that setup was performed correctly, an honest Prover can withdraw Bitcoin even if all Verifiers are malicious, and any one honest Verifier can prevent a malicious Prover from withdrawing Bitcoin.

Since a trivial protocol where the setup always aborts satisfies the above two properties, we also require that if all parties are honest, then no party aborts during setup.

Definition 15 (Setup Correctness). For all NP relations $\mathcal{R}$, all statements x ,

$$\Pr \left[\begin{array}{l} \text{out}_P(\langle P_{\text{Setup}}(\text{crs}, x), \mathcal{V}_{\text{Setup}}(\text{crs}, x) \rangle) = \perp \\ \vee \exists V \in \mathcal{V} \text{ s.t.} \\ \text{out}_V(\langle P_{\text{Setup}}(\text{crs}, x), \mathcal{V}_{\text{Setup}}(\text{crs}, x) \rangle) = \perp : \\ \text{crs} \leftarrow \text{Gen}(\mathcal{R}) \end{array} \right] \leq \text{negl}$$

D.4. Atomic Swaps

An atomic cross-chain swap is a protocol by which two or more parties exchange assets across distinct blockchains without a trusted intermediary. The canonical construction uses Hash Time-Locked Contracts (HTLCs): each party locks funds under a hash commitment $\text{Hash}(s)$ with a timelock, revealing the preimage s triggers the transfer, and expiry of the timelock enables a refund [47], [48].

Following [47], each party's outcome falls into one of five states. A party ends with DEAL if all transfers involving it complete as expected, and with NODEAL if no assets change hands. A party achieves FREERIDE if it receives assets without transferring any, or DISCOUNT if it receives assets while transferring fewer than expected. The unacceptable outcome is UNDERWATER: the party transfers assets but does not receive all expected assets in return.

Definition 16 (Swap Uniformity). A swap protocol $\mathbb{P}$ is *uniform* if it satisfies:

- If all parties follow $\mathbb{P}$, they all finish with DEAL.
- If any coalition cooperatively deviates from $\mathbb{P}$, no honest party finishes with UNDERWATER.

Note that if one party is in the FREERIDE or DISCOUNT state, then another party must be in the UNDERWATER state. Then by swap uniformity, no party is in the FREERIDE or DISCOUNT state unless all parties are dishonest. In this paper, we will use swaps for the vault setup.

D.5. The Ethereum Ledger

Smart Contracts. A *smart contract* $\mathcal{C}$ is a stateful program deployed at a fixed address $a \in \{0, 1\}^{160}$. Its persistent state is a key-value store $\text{store} : \{0, 1\}^{256} \rightarrow \{0, 1\}^{256}$. A contract exposes an interface of typed *functions*; each function may read or update store, transfer Ether, invoke functions of other contracts (*internal calls*, which complete before the caller

resumes), and *emit events*. An emitted event $E(\dots)$ is a log entry appended to the transaction receipt: permanently recorded on the blockchain and publicly observable, but not stored in store. We say $E(\dots) \in \mathcal{L}_{\text{ETH}}$ if the transaction that emitted the event is in $\mathcal{L}_{\text{ETH}}$. For a ledger $\mathcal{L}$, we write $\text{store}_{\mathcal{C}}(\mathcal{L})$ for the storage of $\mathcal{C}$ after executing all transactions in $\mathcal{L}$ in order.

Transactions. An Ethereum transaction $\text{tx} = (\text{from}, \text{to}, f, \text{args}, \sigma)$ specifies: the *sender* $\text{from} \in \{0, 1\}^{160}$, the address that signed and submitted the transaction (acting as the immediate caller within its execution); the recipient $\text{to} \in \{0, 1\}^{160}$; a function f of to to invoke with arguments args and a digital signature σ .

ERC-20 Tokens. ERC-20 [49] is the standard interface for *fungible tokens* on Ethereum. An *ERC-20 token contract* $\mathcal{C}_{\text{tok}}$ is a smart contract whose storage encodes:

- $\text{bal} : \{0, 1\}^{160} \rightarrow \mathbb{N}$ — token balance of each address;
- $\text{allow} : \{0, 1\}^{160} \times \{0, 1\}^{160} \rightarrow \mathbb{N}$ — spending allowance from owner to spender;
- $\text{totalSupply} \in \mathbb{N}$ — total token supply.

The standard ERC-20 interface consists of:

- $\text{balanceOf}(\text{owner}) \rightarrow v$: Returns $\text{bal}[\text{owner}]$.
- $\text{transfer}(\text{to}, v)$, called by from : **Requires** $\text{bal}[\text{from}] \geq v$. **Effects:** $\text{bal}[\text{from}] -= v$, $\text{bal}[\text{to}] += v$. **Emits** $\text{Transfer}(\text{from}, \text{to}, v)$.
- $\text{approve}(\text{spender}, v)$, called by owner : **Effects:** $\text{allow}[\text{owner}][\text{spender}] := v$. **Emits** $\text{Approval}(\text{owner}, \text{spender}, v)$.
- $\text{transferFrom}(\text{owner}, \text{to}, v)$, called by spender : **Requires** $\text{bal}[\text{owner}] \geq v$ and $\text{allow}[\text{owner}][\text{spender}] \geq v$. **Effects:** $\text{bal}[\text{owner}] -= v$, $\text{bal}[\text{to}] += v$, $\text{allow}[\text{owner}][\text{spender}] -= v$. **Emits** $\text{Transfer}(\text{owner}, \text{to}, v)$.

Although not part of the ERC-20 standard, token contracts commonly designate a *minter* address $\text{minter} \in \{0, 1\}^{160}$ authorized to call:

- $\text{mint}(a, v)$, called by minter : **Effects:** $\text{bal}[a] += v$, $\text{totalSupply} += v$. **Emits** $\text{Transfer}(\mathbf{0}, a, v)$.
- $\text{burn}(a, v)$, called by minter : **Requires** $\text{bal}[a] \geq v$. **Effects:** $\text{bal}[a] -= v$, $\text{totalSupply} -= v$. **Emits** $\text{Transfer}(a, \mathbf{0}, v)$.

Here $\mathbf{0} := 0^{160}$ is the zero address; by ERC-20 convention, Transfer events from/to $\mathbf{0}$ signal token creation/destruction. Every well-formed ERC-20 contract maintains the invariant $\text{totalSupply} = \sum_a \text{bal}[a]$. For a party P at ledger view $\mathcal{L}_{\text{ETH}}^{P,r}$, we write $\text{tokBal}_{\mathcal{L}_{\text{ETH}}^{P,r}}(P)$ for P 's token balance in $\text{store}_{\mathcal{C}_{\text{tok}}}(\mathcal{L}_{\text{ETH}}^{P,r})$.

Definition 17 (Ethereum Transaction Validity). A transaction $\text{tx} = (\text{from}, \text{to}, f, \text{args}, \sigma)$ is *valid* with respect to a ledger $\mathcal{L}$, denoted $\text{Valid}_{\mathcal{L}}(\text{tx})$, if:

- 1) *Valid signature:* $\text{Sig}_{\text{ETH}}.\text{Verify}(\text{pk}_{\text{from}}, \bar{\text{tx}}, \sigma) = 1$, where pk_{from} is the public key with address from.
- 2) *Sufficient balance:* The sender's Ether balance in $\mathcal{L}$ covers transaction fees.
- 3) *Successful execution:* Executing $f(\text{args})$ at to against the state induced by $\mathcal{L}$ does not revert.

A ledger $\mathcal{L}$ is *valid*, denoted $\text{Valid}(\mathcal{L})$, if every $\text{tx} \in \mathcal{L}$ satisfies $\text{Valid}_{\mathcal{L}[\text{tx}]}(\text{tx}) = 1$.

D.6. Ethereum Light Client

An *Ethereum light client* is a p.p.t. algorithm $\text{LC}(\text{tx}, \mathcal{L}, w) \rightarrow \{0, 1\}$ that takes a transaction tx , a ledger $\mathcal{L}$, and a witness w , and outputs 1 iff it accepts tx as confirmed in $\mathcal{L}$.

Definition 18 (Light Client Safety). A light client LC is *safe* if with probability $1 - \text{negl}$, for all tx , $\mathcal{L}$, w with $\text{LC}(\text{tx}, \mathcal{L}, w) = 1$:

- 1) $\text{tx} \in \mathcal{L}$; and
- 2) for all honest parties P and rounds r , $\mathcal{L} \preceq \mathcal{L}_{\text{ETH}}^{P,r}$ or $\mathcal{L}_{\text{ETH}}^{P,r} \preceq \mathcal{L}$.

Definition 19 (Light Client Liveness). A light client LC is *live* if with probability $1 - \text{negl}$, for every honest party P , round r , and transaction tx with $\text{tx} \in \mathcal{L}_{\text{ETH}}^{P,r}$, there exists a witness w computable from P 's local state at round r such that $\text{LC}(\text{tx}, \mathcal{L}_{\text{ETH}}^{P,r}, w) = 1$.

Definition 20 (Light Client Timeliness). A light client LC is u_{LC} -*timely* ($u_{\text{LC}} \in \mathbb{N}$) if with probability $1 - \text{negl}$, for all tx , $\mathcal{L}$, w with $\text{LC}(\text{tx}, \mathcal{L}, w) = 1$: if the tuple $(\text{tx}, \mathcal{L}, w)$ appears in the view of an honest party at round r , then $\mathcal{L} \preceq \mathcal{L}_{\text{ETH}}^{P,r'}$ for every honest party P and every round $r' \geq r + u_{\text{LC}}$.

Timeliness strengthens the consistency clause of safety in one direction: an accepted ledger does not merely remain prefix-consistent with honest views, but becomes a prefix of every honest view within u_{LC} rounds of the accepted tuple reaching an honest party.

Implementation. In our implementation we use Helios [22], an Ethereum light client in which the ledger is checkpointed at periodic intervals and the witness w consists of signatures for each checkpoint from the *sync committee*, a dynamically evolving subset of the Ethereum validators. To integrate this into the BABE protocol, we compile the Helios verification logic into a SNARK proof with statement $x = \text{tx}$ and witness $w = (\mathcal{L}, w)$, which is then verified on Bitcoin by the BABE protocol. Helios accepts only ledgers ending at a finalized checkpoint; since finalized prefixes are shared by all honest views, Helios satisfies timeliness with u_{LC} equal to Ethereum's finality delay. While Helios provides an efficient implementation, its security rests on an assumption about a subset of validators rather than the full validator set. Signal proofs [51] provide an alternative light-client implementation that achieves the same security as a full node and can serve as a drop-in replacement.

D.7. Bridge

Consider two ledgers $\mathcal{L}_A$ and $\mathcal{L}_B$ and consider a cross-chain application App . For $X \in \{A, B\}$, let $\mathbb{V}_{L_X}(\text{App})$ be the set of syntactically valid transactions w.r.t. an application App on ledger L_X . Let $\varrho(\mathbb{V})$ denote the set of all subsets of $\mathbb{V}$.

Definition 21 (Unidirectional Bridge [13], [20]). Parameterized by a binary relation $R(\text{App}) \subseteq \varrho(\mathbb{V}_{L_A}(\text{App})) \times \varrho(\mathbb{V}_{L_B}(\text{App}))$, a bridge $\Lambda_{R(\text{App})}(\Pi_{L_A}, \Pi_{L_B})$ is a protocol that relays requests issued on the source ledger $\mathcal{L}_A$ to the destination ledger $\mathcal{L}_B$.

A bridge protocol must satisfy bridge liveness and bridge safety. Both properties ensure atomicity of transactions across the two ledgers. Bridge liveness ensures that for a transaction on the source ledger, the corresponding transaction on the destination ledger will eventually be included, while bridge safety is the converse.

Definition 22 (Bridge liveness). A bridge protocol $\Lambda_{R(\text{id})}(\Pi_{L_A}, \Pi_{L_B})$ is *live* with parameter $u_\ell \in \mathbb{N}$ during execution I if, for all honest parties P_1 of $\mathcal{L}_A$ and P_2 of $\mathcal{L}_B$, and for all rounds $r_1, r_2 \in I$ with $r_2 \geq r_1 + u_\ell$, the following holds: for all $(\mathcal{A}, \mathcal{B}) \in R(\text{id})$, if $\text{tx} \in \mathcal{A}$ is included in $\mathcal{L}_A^{P_1, r_1}$, then there exists $\text{tx}' \in \mathcal{B}$ included in $\mathcal{L}_B^{P_2, r_2}$ with overwhelming probability.

Definition 23 (Bridge safety). A bridge protocol $\Lambda_{R(\text{id})}(\Pi_{L_A}, \Pi_{L_B})$ is *safe* with parameter $u_s \in \mathbb{N}$ during execution I if, for all honest parties P_1 of $\mathcal{L}_A$ and P_2 of $\mathcal{L}_B$, and for all rounds $r_1, r_2 \in I$ with $r_2 \geq r_1 + u_s$, the following holds: for all $(\mathcal{A}, \mathcal{B}) \in R(\text{id})$, if $\text{tx}' \in \mathcal{B}$ is included in $\mathcal{L}_B^{P_2, r_2}$, then there exists $\text{tx} \in \mathcal{A}$ included in $\mathcal{L}_A^{P_1, r_1}$ with overwhelming probability.

Definition 24 (Bidirectional Bridge⁵). A *bidirectional bridge* between ledgers $\mathcal{L}_A$ and $\mathcal{L}_B$ is defined by two unidirectional bridge instances both associated with a common identifier id :

$$\Lambda_{R_1}^1(\Pi_{L_A}, \Pi_{L_B}) \quad \text{and} \quad \Lambda_{R_2}^2(\Pi_{L_B}, \Pi_{L_A}),$$

Appendix E. Multiple-Verifier BABE Protocol

This appendix describes the protocol that realizes multi-verifier BITVM-CORE as defined in Appendix D.3, for a Prover P and a verifier set $\mathcal{V} = \{V_1, \dots, V_n\}$. The protocol generalizes BABE [15], which is defined for a single verifier. We stress that the multi-verifier protocol is *not* a composition of n independent single-verifier BABE executions: the deposit and withdrawal transactions (PreDeposit, Deposit, Refund, Claim, Assert, Withdraw) are *shared* across all verifiers, and only the challenge-response path (ChallengeAssert _{j} , WronglyChallenged _{j} , NoWithdraw _{j}) is replicated, once per verifier. This shared structure is precisely what gives the protocol its security profile: there is a single deposit that all parties co-sign, the Prover withdraws it by answering every challenge — which he can always do when he knows a valid witness, even if all verifiers are malicious — and any *single* honest verifier can post its own challenge and, if it goes unanswered, block the withdrawal without

cooperation from any other party. We first recall the single-verifier protocol (Fig. 5), then present the multi-verifier protocol (Fig. 6), and finally argue informally why it satisfies the multi-verifier BITVM-CORE properties (Defs. 13 to 15).

E.1. Recap: Single-Verifier BABE

Setup phase. In BABE [15], during the setup phase the Verifier samples a secret msg , encrypts it under a witness encryption scheme for the Groth16 relation of the statement x (producing ct_{setup}), and garbles the circuit that computes the proof-dependent part of the ciphertext (producing ct_{GC}). The hash $h' = \text{Hash}_{BTC}(\text{msg})$ and the hashes of the garbled circuit's input labels — which function as a Lamport public key lpk_V for the Verifier — are embedded into the locking scripts of the transaction skeletons of Fig. 5, which both parties construct deterministically from each other's public keys. Each party pre-signs the skeletons that the *other* party may post during the proving phase: the Verifier pre-signs Withdraw, and the Prover pre-signs ChallengeAssert and NoWithdraw. Because a Bitcoin signature commits to the transaction skeleton — including all of its inputs — these pre-signatures restrict each party to posting exactly the transactions of the graph, in the order the graph prescribes. To protect against a malicious Verifier, the Verifier proves that the ciphertexts are well formed by a cut-and-choose: he prepares N_{CC} independent instances, the Prover opens and verifies all but a random subset of size M_{CC} , and the M_{CC} unopened instances are finalized; all finalized instances are then faulty with probability at most $p_{\text{err}} = \binom{N_{\text{CC}}}{M_{\text{CC}}}^{-1}$. Either party aborts the setup if any verification fails — before any Bitcoin is locked.

The deposit itself is established in two steps, PreDeposit followed by Deposit, with a hashlock $\text{HashLock}(h)$ on the connecting edge and a timelocked Refund path ($\text{CheckSig}(\text{pk}_{\text{Dep}}) \wedge \text{RelTimelock}(\Delta_r)$) through which the depositor recovers the funds if the setup stalls. This is an atomic-swap-style peg-in, matching the rest of this paper (Sec. 5); BABE [15] locks the deposit in a single transaction instead. The difference is a small design choice orthogonal to the number of verifiers — either deposit structure works in both settings.

Proving phase. The Prover computes a Groth16 proof (π_1, π_2, π_3) , posts Claim, and then posts Assert, whose input script $\text{CheckLampSig}(\text{lpk}_P)$ makes the transaction witness a binding Lamport commitment to π_1 . The Verifier may respond with ChallengeAssert, whose script $\text{CheckLampSigsMatch}(\text{lpk}_P, \text{lpk}_V)$ forces him to reveal the garbled circuit input labels *for the same* π_1 : he cannot forge the Prover's Lamport signature on any other message. A Prover holding a valid proof evaluates the garbled circuit on these labels, decrypts msg , and posts WronglyChallenged, which requires the preimage of h' . If the Prover fails to do so within Δ_1 blocks of ChallengeAssert, the Verifier posts NoWithdraw, which consumes the connector output of Assert that Withdraw also needs, preventing the Prover from ever withdrawing. If no NoWithdraw appears, the Prover

5. This definition was called a wrapped asset bridge in prior work [13], [20], but is in fact more general since the relation may not pertain to minting/burning a wrapped asset.

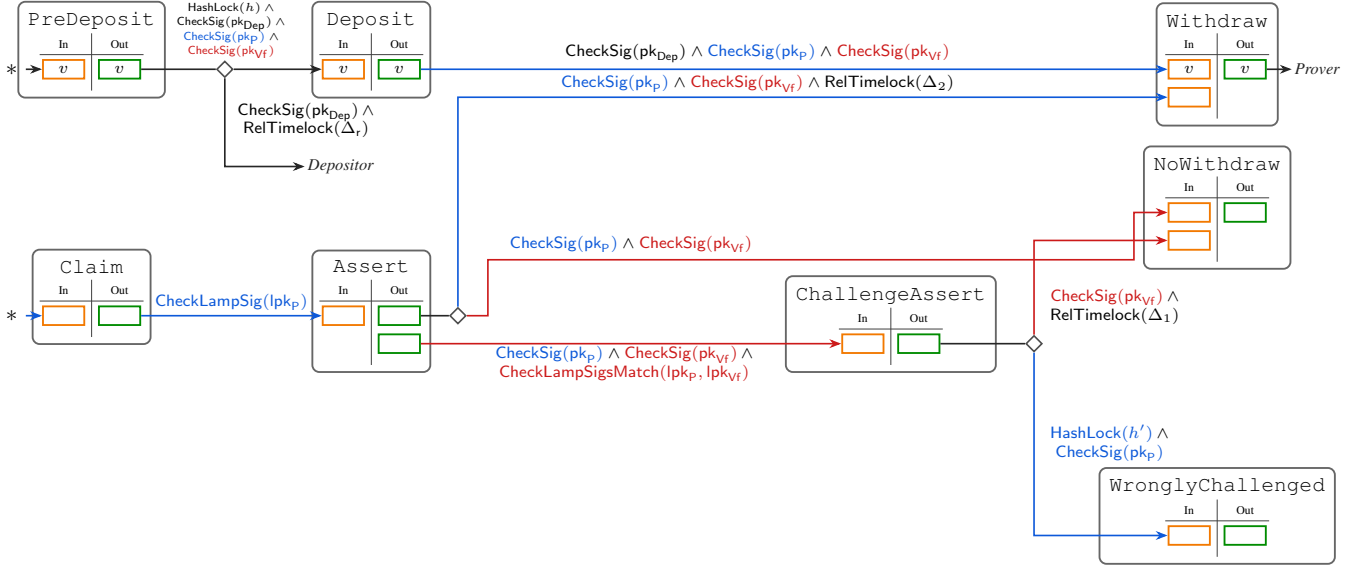


Figure 5: Single-verifier BABE transaction graph (adapted from [15] to the atomic-swap deposit of Fig. 3), shown for one depositor Dep, one prover P, and one verifier Vf. Each box is a Bitcoin transaction with its inputs (orange) and outputs (green); an edge from an output to an input is a spending path, annotated with its script condition (Appendix D.2); a diamond marks an output with alternative spending paths. A condition is **blue** (**red**) if the prover (verifier) provides the witness that satisfies it, and an edge is **blue** (**red**) if the prover (verifier) submits the spending transaction; timelocks and the depositor's conditions and spends are black.

posts Withdraw once Δ_2 blocks have elapsed since Assert; the set of success transactions is $\mathcal{S} = \{\text{Withdraw}\}$. The timelocks satisfy $\Delta_1 > 2u_{BTC}$ and $\Delta_2 > \Delta_1 + 2u_{BTC}$, where u_{BTC} is the liveness parameter of $\mathcal{F}_{BTC}$, and the robustness parameter is $u = \tau^{-1}(\Delta_2 + u_{BTC})$ where τ is the chain-growth parameter.

E.2. The Multi-Verifier Protocol

The multi-verifier transaction graph is shown in Fig. 6. We describe each phase as it differs from the single-verifier protocol.

Setup phase. The Prover sends his Bitcoin public key pk_P and a single Lamport public key lpk_P to all verifiers. A single key suffices despite Lamport signatures being one-time secure: there is one shared Assert transaction, hence one Claim $\rightarrow$ Assert spending path, and the Prover signs exactly one message (π_1) under lpk_P in the entire protocol. Each verifier V_j then independently samples his own secrets, witness encryption ciphertexts, and garbled circuits, and runs his own cut-and-choose with the Prover (with an independent nonce and an independently sampled opening subset), exactly as in the single-verifier setup. This yields, per verifier, the finalized ciphertexts $\{\text{ct}_{\text{setup}}, \text{ct}_{GC}\}$ held by the Prover, the label-hash key lpk_{V_j} , and the secret hash h'_j . The transaction skeletons of Fig. 6 are constructed deterministically from pk_{Dep} , pk_P , lpk_P , and all verifiers' pk_{V_j} , lpk_{V_j} , and h'_j , so every party computes them locally. Pre-signing follows the single-verifier pattern per verifier: each V_j pre-signs Withdraw — which now carries all n verifier signatures

alongside the Prover's and the depositor's — and the Prover pre-signs ChallengeAssert $_j$ and NoWithdraw $_j$ for each j , sending those pre-signatures to V_j . The Prover aborts if any of the n cut-and-choose verifications or pre-signature checks fails, and V_j aborts if his own checks fail; all of this happens off-chain before Deposit, so an abort loses no money and the depositor recovers the funds through the Refund path.

Transaction graph. Relative to Fig. 5, the graph changes as follows (Fig. 6):

- Deposit's output, and the PreDeposit $\rightarrow$ Deposit hashlock edge, require the signatures of the depositor, the Prover, and *all* n verifiers, so no transaction can spend the deposit without every party's (pre-)signature.
- Assert has $n + 1$ outputs instead of two. Its first output is the shared connector, spendable in $n + 1$ ways: by Withdraw, requiring all $n + 1$ signatures of the Prover and the verifiers together with $\text{RelTimelock}(\Delta_2)$ — the only timelock on the withdrawal path — or by any one of the n transactions NoWithdraw $_j$, requiring $\text{CheckSig}(pk_P) \wedge \text{CheckSig}(pk_{V_j})$ as in the single-verifier graph. The remaining n outputs feed the per-verifier challenges: the j -th carries $\text{CheckSig}(pk_P) \wedge \text{CheckSig}(pk_{V_j}) \wedge \text{CheckLampSigsMatch}(lpk_P, lpk_{V_j})$, spendable only by ChallengeAssert $_j$.
- For each j , the transactions ChallengeAssert $_j$, WronglyChallenged $_j$, and NoWithdraw $_j$ are exact counterparts of their single-verifier namesakes: ChallengeAssert $_j$'s output is spendable by WronglyChallenged $_j$ ($\text{HashLock}(h'_j) \wedge \text{CheckSig}(pk_P)$) or, after Δ_1 blocks, by NoWithdraw $_j$

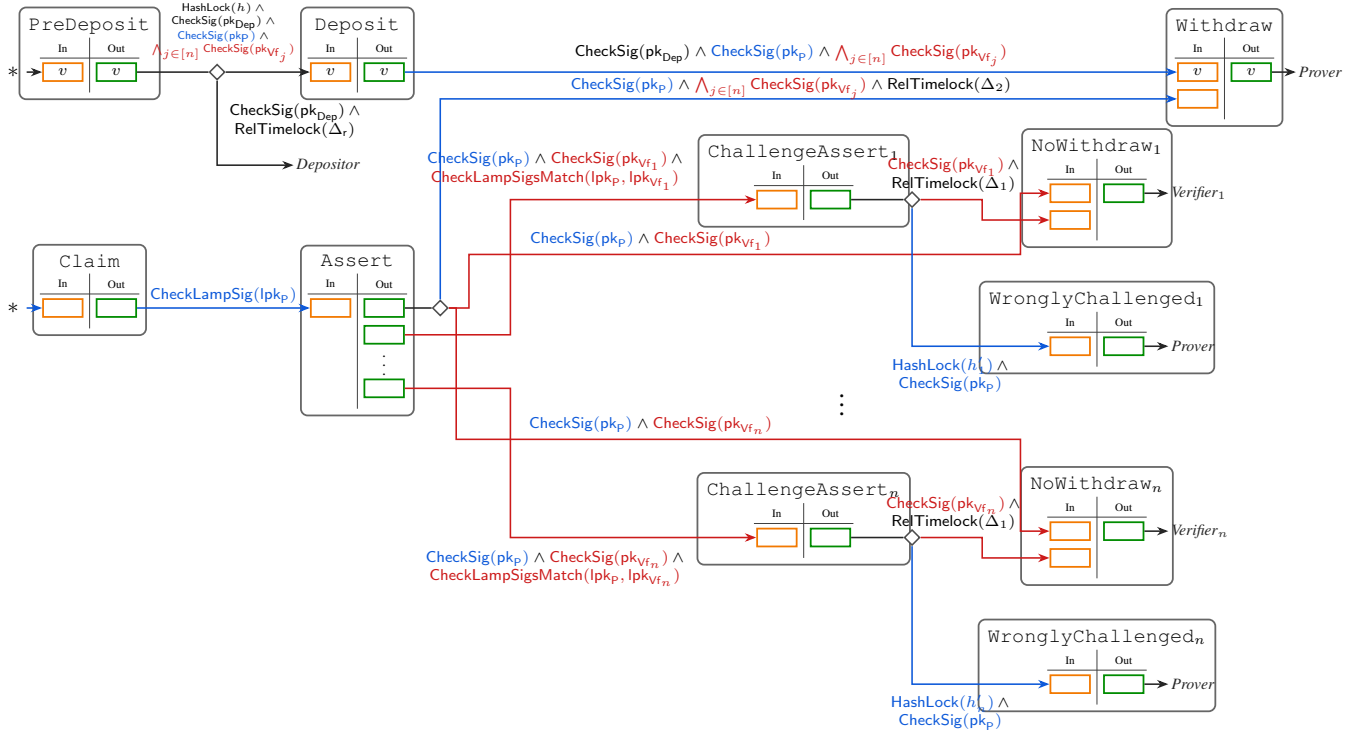


Figure 6: Multi-verifier BABE transaction graph for one depositor Dep, one prover P, and n verifiers $Vf_1, \dots, Vf_n$, in the notation of Fig. 5. The transactions PreDeposit, Deposit, Claim, Assert, and Withdraw are shared across all verifiers, while the challenge–response path (ChallengeAssert_j, WronglyChallenged_j, NoWithdraw_j) is replicated once per verifier Vf_j (branches 1 and n shown). The first output of Assert can be spent by Withdraw (all $n+1$ signatures, after Δ_2 blocks) or by any single NoWithdraw_j; the remaining n outputs feed the per-verifier ChallengeAssert_j transactions.

$(\text{CheckSig}(\text{pk}_{V_j}) \wedge \text{RelTimelock}(\Delta_1))$, and NoWithdraw_j also consumes the shared connector of Assert.

One structural point deserves emphasis: every script on V_j 's challenge path involves only pk_P and pk_{V_j} , and the only pre-signatures the path consumes are the Prover's, which V_j already holds from setup. Hence V_j posts ChallengeAssert_j and NoWithdraw_j unilaterally: no coalition of the other parties can stop him, which is what makes one honest verifier sufficient.

Proving phase. The Prover posts Claim and then Assert, whose witness is his Lamport signature on π_1 — one commitment, shared by all n challenge paths. Each verifier V_j independently decides whether to challenge: upon seeing Assert, he extracts π_1 , encodes it under his own encoding keys to obtain the input labels, and posts ChallengeAssert_j, whose script guarantees the labels encode the same π_1 . Upon seeing ChallengeAssert_j, the Prover evaluates V_j 's garbled circuits, decrypts a secret of V_j , and posts WronglyChallenged_j. Distinct challenges spend distinct outputs of Assert, so the Prover can answer all of them concurrently, with no interaction between the paths. If WronglyChallenged_j does not appear within Δ_1 blocks of ChallengeAssert_j, verifier V_j posts NoWithdraw_j, consuming the shared connector

and thereby preventing Withdraw forever. If the shared connector is unspent Δ_2 blocks after Assert, the Prover posts Withdraw.

Timelocks and costs. Because the n challenge–response paths proceed in parallel rather than sequentially, the timelock constraints are unchanged from the single-verifier protocol: $\Delta_1 > 2u_{BTC}$ and $\Delta_2 > \Delta_1 + 2u_{BTC}$ suffice, and the robustness parameter remains $u = \tau^{-1}(\Delta_2 + u_{BTC})$, independent of n . The costs scale as follows. Off-chain, setup consists of n independent cut-and-choose executions, so its cost is linear in n . On-chain, the optimistic path (no verifier challenges) consists of Claim, Assert, and Withdraw alone, as in the single-verifier protocol, although Assert now carries $n+1$ outputs and Withdraw carries $n+1$ signatures, both linear in n with small constants. In the worst case all n verifiers challenge and the Prover answers each, adding at most one (ChallengeAssert_j, WronglyChallenged_j) pair per verifier.

E.3. Security

We argue informally that the protocol satisfies the three multi-verifier BITVM-CORE properties of Appendix D.3; the formal arguments adapt the single-verifier proofs of [15] to the shared graph.

Setup correctness (Def. 15). When all parties are honest, every cut-and-choose verification and every pre-signature check passes except with negligible probability, so no party aborts; a union bound over the $n = \text{poly}$ verifiers keeps the overall abort probability negligible.

Robustness (Def. 13). Let the adversary control *all* verifiers, and let the honest Prover hold a valid witness and have accepted the setup. Once Deposit is on chain, the only way to prevent Withdraw is to spend the shared connector of Assert via some NoWithdraw_{*j*}, and by pre-signature binding this requires a ChallengeAssert_{*j*} that goes unanswered for Δ_1 blocks. Consider any challenge ChallengeAssert_{*j*} the adversary posts. Its script forces input labels for the very π_1 the Prover committed to (forging the Prover’s Lamport signature on another message is infeasible), and the cut-and-choose with V_j guarantees, except with probability $p_{\text{err}} + \text{negl}$, that at least one of V_j ’s finalized ciphertext pairs is well formed. Hence the Prover — holding a valid Groth16 proof — evaluates that garbled circuit, decrypts the secret, and posts WronglyChallenged_{*j*}; since $\Delta_1 > 2u_{BTC}$, it is included before NoWithdraw_{*j*} becomes valid. A union bound over the at most n challenges degrades the statistical error from $2^{-\kappa}$ to $n \cdot 2^{-\kappa}$, i.e., by $\log_2 n$ bits, which is absorbed by choosing the cut-and-choose parameters for $\kappa + \log_2 n$ bits of security. Finally, by chain growth, the Prover’s view reaches height $\Delta_2 + u_{BTC}$ past Assert within $u = \tau^{-1}(\Delta_2 + u_{BTC})$ rounds, at which point Withdraw is included.

Knowledge soundness (Def. 14). Let the adversary control the Prover and all verifiers except some honest V_j . As observed above, V_j ’s challenge path is unilateral: he holds the Prover’s pre-signatures on ChallengeAssert_{*j*} and NoWithdraw_{*j*} (delivered during setup, before Deposit — otherwise he would have aborted) and needs no other party to post either transaction. Suppose a Withdraw transaction appears in some honest view. Then no NoWithdraw_{*j*} was ever included, and since $\Delta_2 > \Delta_1 + 2u_{BTC}$, the honest V_j — who posts ChallengeAssert_{*j*} as soon as he sees Assert and NoWithdraw_{*j*} as soon as its timelock expires — would have blocked the withdrawal unless WronglyChallenged_{*j*} appeared in time, revealing a preimage of one of V_j ’s secret hashes. The secrets and ciphertexts of the colluding verifiers are sampled independently of V_j ’s, so they give the adversary no advantage in recovering V_j ’s secret. The single-verifier analysis [15] — relying on the adaptive security of the witness encryption scheme, the adaptive privacy of the garbling scheme, and the second-preimage resistance of the hash function — therefore yields an extractor that, given the adversary’s view of the interaction with V_j and Bitcoin, outputs a valid Groth16 witness for x .

In summary, the protocol satisfies setup correctness, u -robustness with $u = \tau^{-1}(\Delta_2 + u_{BTC})$, and knowledge soundness in the multi-verifier sense of Appendix D.3, as assumed of $\Pi_{\text{BITVM-CORE}}$ in Sec. 5.