Tritone: Pairwise Garbled Disputes for Multi-Party Proof Verification on Bitcoin

Oleksii Avramenko*, Alessandro Baccarini†, and Torben Poguntke‡

Pogun

October 23, 2026

Abstract

Bitcoin Script lacks native pairing operations for efficient direct verification of Groth16 proofs. Conventional federated bridges instead authorize withdrawals through a signing quorum, whose collusion can release funds without valid evidence. We present the Tritone protocol, which lets Bitcoin enforce the outcome of verifying a Groth16 proof for an NP relation fixed in a one-time trusted setup, with the locked funds secure as long as one of a fixed set of N operators is honest. Verification is optimistic and off-chain, and every transaction is standard under current relay policy, so Tritone runs on Bitcoin today without consensus changes.

Tritone generalizes the two-party BABE protocol [GKS+26] to a fixed committee of N operators, with one dispute instance per ordered pair. Cut-and-choose protects against malicious garblers, while Feldman verifiable secret sharing correlates input keys across candidates. Our new Lamport-adaptor gadget forces a challenging verifier to disclose the shares selected by the prover's committed proof element, providing the transaction-level binding left unspecified by BABE. A single set of disclosed shares then unlocks all retained candidates. We formalize the multi-party primitive, prove the protocol secure against up to $N - 1$ statically corrupted operators, and specify its complete pre-signed transaction graph. We implement Tritone in Rust. On Bitcoin mainnet, when one verifier disputes a claim at every stage, the prover-candidate posts 9,607 vB and the verifier 18,160 vB of protocol transactions. In local benchmarks at 40-bit statistical security, the prover-candidate of each dispute instance retains as little as 112.83 MiB. Tritone thus replaces trust in a signing quorum with a single-honest-operator assumption wherever locked bitcoin is released to an operator against a Groth16 proof, with Bitcoin bridges as our showcase.

*oleksii@invisible.boats
†abaccarini@proton.me
‡torben@blocksmithy.io

1 Introduction

In the world of cryptocurrencies, Bitcoin [Nak08] remains dominant in terms of market capitalization and popularity, yet its scripting language is too limited to express the rich applications available on newer chains. Every application that locks bitcoin must answer one question: under what conditions are the locked coins released? Bitcoin Script can release coins against signatures, hash preimages, and timelocks, but not against evidence of an off-chain computation or of an event on another system, even though a succinct non-interactive argument of knowledge (SNARK) captures exactly such evidence. Script cannot natively verify a SNARK: the elliptic-curve pairing arithmetic of pairing-based SNARKs such as Groth16 [Gro16] is far beyond its opcode set, and no covenant or introspection functionality exists to enforce it. Without such verification, applications fall back on trusted federations or multi-signature custody, in which a colluding quorum of signers can steal every locked coin.

Bitcoin bridges are one application that needs such verification. They connect Bitcoin to secondary systems that offer greater functionality and programmability: BTC is locked on Bitcoin and represented on the secondary system as a wrapped asset, unlocking decentralized finance (DeFi) applications for Bitcoin holders. A withdrawal (or "peg-out") is legitimate only against evidence of a corresponding event on the secondary system, such as a proof that the wrapped funds were surrendered. We refer to the parties who facilitate a bridge, custodying the locked coins and servicing peg-outs, as its operators. The goal of this work is to release locked coins in a *trust-minimized* way: the locked funds remain secure as long as a single operator is honest. Throughout, we keep the secondary system and the bridged application abstract.

The central obstacle is therefore on-chain proof verification. A recent line of work moves the verifier off-chain into a garbled circuit [Yao82], so that a dispute is settled on-chain with a compact secret instead of executing the verifier (Section 2). BABE [GKS+26] combines witness encryption [GGSW13] for Groth16 with garbled scalar multiplication based on decomposable randomized encodings (DREs) [Ish13], reducing the off-chain footprint to ∼22 MiB. Its proposed maliciously secure variant uses *cut-and-choose* [Lin16], opening most of many garbled copies and using the M_{CC} unopened ones, and BABE notes that verifiable secret sharing (VSS) [Fel87] and adaptor signatures [AEE+21, GSST24] could together reduce the on-chain footprint of this variant. This leaves two gaps: BABE is specified for two parties only, and it leaves open the transaction design that binds the committed shares to the on-chain dispute logic.

In this work, we close both gaps. We present the Tritone protocol, a maliciously secure N-party generalization of BABE that releases locked coins against a Groth16 proof. The bridge above serves as our showcase, but the protocol applies to any NP statement that can be fixed at setup and proven with Groth16. Operators are paired together (in order) to form a dispute instance, and security holds against an adversary that statically corrupts up to $N-1$ of the N operators. Our construction combines witness encryption for Groth16, garbled scalar multiplication via decomposable randomized encodings, cut-and-choose with Feldman VSS, and a new on-chain gadget that binds the VSS shares to the dispute logic. Every protocol transaction is standard under current relay policy: the protocol runs on Bitcoin today, without consensus changes.

Our Contributions Concretely, we make the following contributions:

- **A multi-party** BitVM-core **primitive.** We generalize the two-party BitVM-core primitive of BABE to an operator set of size N, in which every ordered pair of operators forms one of $N(N-1)$ dispute instances. We provide the protocol syntax and formal definitions of setup safety, completeness, knowledge soundness, and non-triviality under static corruption of up to

$N-1$ operators (Section 4).

- **The Lamport-adaptor gadget.** We introduce a hybrid Lamport-adaptor locking script that branches on the Lamport [Lam79] preimage of a committed bit, such that the spending party can complete a script-accepted spend only via the branch selected by the committed bit, and completing it necessarily discloses a designated secret for that bit value. The gadget realizes exactly the share binding that BABE leaves open, collapsing the on-chain reveal from M_{CC} input-label sets down to a single set of zero-shares, and is of potential independent interest for commit-reveal applications (Section 7).
- **The Tritone protocol.** We present the construction and pseudocode for all setup and proving procedures, and the full transaction graph, which depends on neither the relation nor the statement (Section 8). We prove security of the construction with respect to the primitive's definitions (Section 9).
- **Implementation and evaluation.** We implement the full stack in Rust (script and transaction graph construction, the DRE, garbling with cut-and-choose, witness encryption for Groth16, and the end-to-end operator protocol) (Section 10).

Organization Section 2 discusses related work. Section 3 recalls the required building blocks. Section 4 defines the multi-party BitVM-core primitive and its security notions. Sections 5 and 6 present witness encryption for Groth16 and garbled scalar multiplication on BN254. Section 7 introduces the Lamport-adaptor gadget and proves its security. Section 8 presents the Tritone protocol and its transaction graph, with the full pseudocode alongside. Section 9 contains the security proofs, and Section 10 reports our experimental evaluation.

2 Related Work

On-chain proof verification has progressed through two generations of protocols. Across both, the predominant instantiation of the underlying proof system is Groth16 [Gro16], due to its unmatched proof compactness.

BitVM [Lin23, AAL+24] first demonstrated that arbitrary computation can be verified on Bitcoin without consensus changes. Two parties pre-sign a sequence of challenge–response transactions in which funds move along the default path unchallenged, and a dispute triggers an interactive bisection over the committed computation that narrows the fault down to a single gate, whose on-chain execution lets the honest party claim the cheater's collateral deposit. While each transaction is small, the bisection game requires many sequential transactions (79 in the worst case, spanning weeks) [WAA+26b], significant off-chain communication, and a strictly two-party setting in which only the designated counterparty may challenge.

BitVM2 [WAA+26b] extends this paradigm to a bridge. Its operators form a fixed (permissioned) committee: an operator fronts the peg-out funds to the user and reclaims the locked deposit by proving on-chain, via a light-client proof, that the corresponding burn occurred on the secondary system [WAA+26a]. Challenging, by contrast, is permissionless for the first time. Disputes are settled in three on-chain transactions: an operator asserts the intermediate states of the (Groth16-instantiated) verifier circuit, split into sequential chunks, and any challenger disproves a faulty claim by executing a single incorrect chunk on-chain. The design has been adopted by several industry bridge projects [Cit26, Bit26, BOB26, Bab26], but its assert and disprove transactions are multiple megabytes in size, exceeding the standardness limits of Bitcoin's relay policy (and, thus, unrelayable by unmodified nodes), with a reported worst-case dispute cost of ∼$16,000 [WAA+26a].

The second generation of protocols shifts the verifier circuit entirely off-chain. As first observed

in Delbrag [Rub25] and independently in Glock [Eag25], if a garbled circuit [Yao82] (an encrypted encoding of a circuit that can be evaluated only on encoded inputs) encodes the verifier, then evaluating it on an invalid proof reveals a secret that serves as a compact fraud proof, replacing on-chain execution altogether. This is essentially a conditional disclosure of secrets construction [DGGM19, LVW17, GIKM98]. (Notably, witness encryption [GGSW13], in which a ciphertext formed under an NP statement can be decrypted by anyone holding a corresponding witness, was proposed for trustless Bitcoin bridging prior to the inception of BitVM [Hio22].)

BitVM3 [WAA+26a] realizes this paradigm for the Groth16 verifier: a dispute consists of only two transactions, the second carrying a fraud proof of ∼93 virtual bytes (vB), for a reported total on-chain cost of ∼$9 (under $0.20 for the fraud proof, at 2 sat/vB). This is a reported reduction of more than 1000× over BitVM2's worst-case dispute cost. The trade-off is a substantial off-chain cost: each operator's garbled circuit occupies ∼41 GB, or ∼288 GB per operator under *cut-and-choose* [Lin16], the standard technique for malicious security in which most of many independently garbled copies are opened and verified; the unopened remainder is used for the actual computation. Moreover, the storage requirement grows linearly with each successive peg-in (deposit into the bridge). The authors note that cut-and-choose and verifiable secret sharing (VSS) can be employed to ensure the setup phase was executed correctly, but do not provide a concrete specification. Mosaic [KTA+26] presents exactly such a protocol. It achieves malicious security via cut-and-choose, while using Feldman VSS [Fel87] and Schnorr adaptor signatures [AEE+21, GSST24] (signatures whose completion necessarily reveals a designated secret) to correlate labels across the garbled copies and keep the on-chain footprint independent of their number (a technique first introduced in the Bitcoin setting by Glock). Mosaic provides a complete construction, transaction graph, and implementation, with informal security arguments. However, the protocol is specified only for two parties, and only for BitVM3-style fraud proofs.

This brings us to BABE [GKS+26], which abstracts the dispute mechanism into a two-party primitive termed BitVM-core and combines witness encryption for Groth16 with garbled scalar multiplication based on decomposable randomized encodings (DREs) [Ish13]. This results in an off-chain footprint of ∼22 MiB, a three-orders-of-magnitude improvement over BitVM3, while preserving BitVM3's low on-chain cost. Its dispute flow realizes what we refer to as a "proof of innocence," where the verifier garbles a circuit that maps π_1, one of the three group elements of a Groth16 proof committed on-chain by the prover, to $r\pi_1$ under the verifier's secret scalar r, and a wrongly challenged prover exculpates itself by evaluating the circuit and decrypting a witness-encrypted message. BABE reports a prototype implementation, including its cut-and-choose setup, but mainnet-derived on-chain costs only for the two-party setting under the assumption that setup is executed honestly. In its proposed maliciously secure variant, BABE employs cut-and-choose and posts M_{CC} distinct input-label sets on-chain, where M_{CC} denotes the number of unopened (finalized) candidate circuits, or reduces them to a single set with a zk-SNARK proof of label consistency (*soldering*), at the cost of about 12 minutes of proving at $M_{\mathsf{CC}} = 4$. It notes that VSS and adaptor signatures can be used together to reduce the on-chain footprint, but leaves this unspecified. Moreover, BABE describes multi-verifier support as an easy extension of its construction, but does not define it concretely; see the concurrent work below.

Concurrent Work Concurrently and independently, Aumayr et al. [ADH+26] propose Trustless Bitcoin Vaults, a primitive for BTC-collateralized applications, and instantiate its proof-verification layer with a multi-verifier extension of BABE in which each claimer is the prover of its own instance and a verifier in every other claimer's instance. Tritone differs in its malicious-setup binding, where the Lamport-adaptor gadget ties the Feldman zero-shares of the cut-and-choose to the dispute

logic; in an optimistic withdrawal path without an on-chain proof commitment; and in proofs of the security properties of the multi-party protocol itself, which [ADH+26] assumes in the formal proofs of its vault protocol and argues informally for its multi-verifier extension of BABE.

3 Background

3.1 Notation

We denote vectors as lowercase bold $\boldsymbol{x}$ and matrices as uppercase bold $\boldsymbol{A}$. H is a suitable hash function modeled as a random oracle. The notation $x \xleftarrow{\$} \mathcal{X}$ corresponds to uniformly sampling x from a finite set $\mathcal{X}$. We let $[n]$ denote the set $\{1, 2, \ldots, n\}$. In destructuring assignments, $\cdot$ discards a component the executing algorithm does not use. Throughout, λ denotes the computational security parameter and κ the statistical security parameter.

We consider *bilinear groups* parameterized by $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, G_2, e)$, where $\mathbb{G}_1, \mathbb{G}_2$ ("source") and $\mathbb{G}_T$ ("target") are groups of prime order q, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map ("pairing"), and G_1, G_2 are random generators for $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. As standard, we utilize implicit notation $[x]_s := xG_s$ for $s \in \{1, 2, T\}$ and omit the subscript s when there is no ambiguity about which group the generator belongs to.

We fix the following field conventions throughout: p denotes the BN254 base-field characteristic and q the BN254 group order, equivalently its scalar-field characteristic. Additionally, we work over the elliptic curve secp256k1 with generator G and prime group order q_s, whose scalar field we write $\mathbb{F}_{q_s}$. When the curve is clear from context, we extend the implicit notation as $[x] := xG$.

For $x \in \{0, \ldots, 2^n - 1\}$, we write $\mathsf{bin}(x) = (x_0, \ldots, x_{n-1}) \in \{0,1\}^n$ for its binary decomposition, such that $x = \sum_{j=0}^{n-1} 2^j x_j$.

We also require a symmetric encryption scheme $(\mathsf{Enc}(k, m), \mathsf{Dec}(k, c))$ with key space $\mathcal{K} = \{0,1\}^\lambda$ that satisfies IND-CPA security (see, e.g., [Rog04, Sec. 3] for a definition). We write $\mathsf{Dec}(k, c) = \perp$ for a failed decryption and require that decrypting under a key other than the one used for encryption fails except with negligible probability. Where a key is a field element, it is first mapped into $\mathcal{K}$ by $H_{\mathsf{key}} : \mathbb{F}_{q_s} \to \mathcal{K}$, a domain-separated instance of the random oracle H.

3.2 Lamport Digital Signature

Let $h : \{0,1\}^* \to \{0,1\}^\lambda$ be a one-way function with security parameter λ and let $m \in \{0,1\}^\ell$ be an ℓ-bit message with $\ell \in \mathbb{N}_{>0}$. A *Lamport one-time digital signature scheme* [Lam79] Lamp is a triple of algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, presented in Scheme 1.

Scheme 1: Lamport Signature Lamp

$\mathsf{Gen}(\ell) \to (\mathsf{lsk}, \mathsf{lpk})$:

1. For $b \in \{0,1\}, i \in \{1, \ldots, \ell\}$, sample $L_i^b \xleftarrow{\$} \{0,1\}^\lambda$ and let

$$\mathsf{lsk} = \begin{pmatrix} L_1^0 & \cdots & L_\ell^0 \\ L_1^1 & \cdots & L_\ell^1 \end{pmatrix}, \quad \mathsf{lpk} = \begin{pmatrix} h(L_1^0) & \cdots & h(L_\ell^0) \\ h(L_1^1) & \cdots & h(L_\ell^1) \end{pmatrix}.$$

2. Output $(\mathsf{lsk}, \mathsf{lpk})$.

$\mathsf{Sign}(m, \mathsf{lsk}) \to \mu$: Output $\mu = (L_1^{m_1}, \cdots, L_\ell^{m_\ell})$.

$\mathsf{Verify}(m, \mu, \mathsf{lpk}) \to \{0,1\}$: Output 1 iff, $\forall i \in \{1, \ldots, \ell\}$, $h(\mu_i) = \mathsf{lpk}_i^{m_i}$.

A Lamport signature constructed with $h = \mathsf{Hash}_{\mathrm{BTC}}$ can be efficiently verified in Bitcoin Script [GKS+26, WAA+26b].

3.3 Verifiable (Shamir) Secret Sharing (VSS)

Secret sharing refers to splitting a secret into *shares*: a trusted dealer distributes a secret s to n participants such that any cooperating subset of $t+1$ participants can recover it, while any t or fewer shares reveal nothing about s. In Shamir's (n,t)-threshold scheme [Sha79], the dealer samples t random coefficients $a_1, \ldots, a_t$ and defines the degree-t polynomial $f(x) = s + \sum_{i=1}^{t} a_i x^i$. The shares are the evaluations $\{s_i = f(i)\}_{i=1}^{n}$, and any $t+1$ of them recover $s = f(0)$ via Lagrange interpolation.

As presented, Shamir's scheme does not guarantee that the dealer's shares are consistent with a single polynomial. Feldman's *verifiable* secret sharing [Fel87], which we instantiate over $\mathbb{F}_{q_s}$, rectifies this by having the dealer additionally publish a commitment $C_f = ([a_0], [a_1], \ldots, [a_t])$ to the coefficients of f, against which every share can be checked. Moreover, the commitments can be evaluated homomorphically. Given C_f and an index α, one can compute the commitment $[f(\alpha)]$ *without* knowing f:

$$C_f(\alpha) = \sum_{k=0}^{t} \alpha^k \cdot [a_k] = \left[\sum_{k=0}^{t} a_k \alpha^k\right] = [f(\alpha)].$$

In our protocol, VSS allows for the efficient verifiable cut-and-choose for garbled circuits, where the verifier secret-shares the garbled circuits' input keys across candidates, and the prover later reconstructs the finalized keys by interpolation and decrypts the corresponding input labels with them.

Scheme 2: Verifiable Secret Sharing VSS

$\mathsf{Gen}(t, n) \to \left(\{x_i, s_i\}_{i\in[n]}, C_f\right)$:

1. Sample $a_i \xleftarrow{\$} \mathbb{F}_{q_s}$ for $i = 0, \ldots t$.
2. Define the univariate polynomial $f(x) = \sum_{i=0}^{t} a_i x^i$.
3. Compute the commitment $C_f = \mathsf{Com}(f)$.
4. Output the set of shares $\{(x_i, s_i := f(x_i))\}_{i\in[n]}$ and C_f.

$\mathsf{Com}(f) \to C_f$: For $f(x) = \sum_{i=0}^{t} a_i x^i$, output the coefficient commitments $C_f = ([a_0], [a_1], \ldots, [a_t])$.

$\mathsf{Verify}((x, s), C_f) \to \{0, 1\}$:

1. Compute $C_f(x) = \sum_{j=0}^{t} x^j [a_j]$.
2. Output 1 iff $[s] = C_f(x)$.

$\mathsf{Interpolate}(\{x_i, s_i\}_{i=1}^{t+1}) \to f$: Output the unique polynomial of degree at most t through the supplied points (Lagrange),

$$f(x) = \sum_{i=1}^{t+1} s_i \prod_{j=1, j\neq i}^{t+1} \frac{x - x_j}{x_i - x_j}.$$

The security properties that VSS provides are:

1. **Secrecy.** For an honest dealer, any set of at most t shares is information-theoretically independent of $f(0)$: t shares of a degree-t polynomial are consistent with every possible value of

$f(0)$. The commitment itself reveals $[f(0)] = C_f(0)$, so once C_f is published, secrecy of $f(0)$ holds only *computationally* under the discrete logarithm assumption.

2. **Binding.** Under the discrete logarithm assumption, it is infeasible to produce a commitment C_f and two sets of $t+1$ valid shares that interpolate to different secrets. The commitment is therefore bound to a unique polynomial and secret $f(0)$.

3.4 Schnorr Adaptor Signatures

Our construction leverages Schnorr adaptor signatures [Fou19, GSST24, AEE+21]. An adaptor signature scheme binds the completion of a signature to the disclosure of a witness. The statement is a point $T = [y]$ for a witness $y \in \mathbb{F}_{q_s}$. In our construction, the prover holds a signing key pair $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{pk} = [\mathsf{sk}]$ and is responsible for generating incomplete adaptors. The verifier holds the prover's public key pk and completes ("adapts") the adaptors. In our construction, we instantiate the scheme over the curve secp256k1.[1]

Scheme 3: Schnorr Adaptor Signature SASig

For a point R other than the identity, let $\varepsilon(R) \in \{\pm 1\}$ denote the sign such that $\varepsilon(R) \cdot R$ has even y-coordinate (well defined, as secp256k1 has prime order and so $R \neq -R$). The challenge hash $H(R, \mathsf{pk}, m)$ depends on R only through $x(R)$, so $H(R, \mathsf{pk}, m) = H(\varepsilon R, \mathsf{pk}, m)$.

$\mathsf{GenAdaptor}(\mathsf{sk}, T, m) \to \tilde{\sigma}$:

1. Sample $r' \xleftarrow{\$} \mathbb{F}_{q_s}^*$ and compute $R' = [r']$.
2. Compute $R = R' + T$, $\varepsilon = \varepsilon(R)$, and $e = H(R, \mathsf{pk}, m)$.
3. Compute $s' = \varepsilon \cdot r' + e \cdot \mathsf{sk}$.
4. Output $\tilde{\sigma} = (s', R', \varepsilon)$.

$\mathsf{VerAdaptor}(\mathsf{pk}, T, m, \tilde{\sigma}) \to \{0, 1\}$:

1. Parse $\tilde{\sigma} = (s', R', \varepsilon)$.
2. Compute $R = R' + T$ and $e = H(R, \mathsf{pk}, m)$. (T is the *verifier's* intended statement)
3. Output 1 iff $\varepsilon = \varepsilon(R)$ and $[s'] = \varepsilon \cdot R' + e \cdot \mathsf{pk}$.

$\mathsf{Adapt}(\tilde{\sigma}, y) \to \sigma$:

1. Parse $\tilde{\sigma} = (s', R', \varepsilon)$.
2. Output $\sigma = (s' + \varepsilon \cdot y,\ \varepsilon \cdot (R' + [y]))$. ($[s' + \varepsilon y] - e \cdot \mathsf{pk} = \varepsilon(R' + T)$ has even y-coordinate)

$\mathsf{Extract}(\tilde{\sigma}, \sigma) \to y$:

1. Parse $\tilde{\sigma} = (s', R', \varepsilon)$ and $\sigma = (s, R)$.
2. Output $y = \varepsilon \cdot (s - s')$. (since $s - s' = (\varepsilon r' + \varepsilon y + e \cdot \mathsf{sk}) - (\varepsilon r' + e \cdot \mathsf{sk}) = \varepsilon y$)

An adaptor signature scheme satisfies the following properties (see [GSST24] for a rigorous treatment)[2]:

[1] Although we write Scheme 3 using full group elements, our implementation follows the BIP-340 [WNR20] specification where both pk and the nonce point R are x-only with implicit even y-coordinate. In key generation, the secret key sk is negated if $[\mathsf{sk}]$ has odd y-coordinate, and the signature (s, R) is serialized as $(x(R), s)$. The signature is verified by recomputing $R = [s] - e \cdot \mathsf{pk}$ and rejected if its y-coordinate is odd. We explicitly incorporate the sign ε into the adaptor, such that the completed nonce $\varepsilon(R' + T)$ has even y-coordinate, and since the signer controls only r', the adapting party applies the same sign to y.

[2] The results of [GSST24] are stated for the textbook scheme, in which $\varepsilon \equiv 1$. Our scheme given a statement T with witness y coincides with the textbook scheme for $(\varepsilon T, \varepsilon y)$, where $\varepsilon = \varepsilon(R' + T)$ is a public function of the pre-signature and the statement, so any adversary against our scheme yields one against the textbook scheme with the same advantage up to a factor 2. This is due to the reduction flipping the signs of statements and witnesses, answering each pre-signing query by querying its own oracle on $\pm T$ until the nonce point has even y-coordinate (two

1. **Adaptability.** If $\mathsf{VerAdaptor}(\mathsf{pk}, T, m, \tilde{\sigma}) = 1$ and $T = [y]$, then $\mathsf{Adapt}(\tilde{\sigma}, y)$ produces a valid signature under pk on m.
2. **Witness extractability.** A malicious verifier cannot use a pre-signature $\tilde{\sigma}$ with respect to a statement T to produce a valid signature σ without revealing a witness y for T: the signer recovers it as $y = \mathsf{Extract}(\tilde{\sigma}, \sigma)$. Formally, it is infeasible to produce a valid σ for which extraction fails to yield a valid witness. We utilize the multi-query *full extractability* notion of [DOY22], shown to hold for Schnorr adaptor signatures in the ROM by [GSST24].
3. **Unforgeability (aEUF-CMA).** No efficient adversary can forge a signature under pk, even with access to signing and pre-signing oracles on messages of its choice and knowledge of a pre-signature on a possibly maliciously chosen message m^*.

3.5 The Groth16 SNARK

We recall the non-zero-knowledge Groth16 proof system [Gro16] below.

Rank-1 Constraint Systems (R1CS) Groth16 is designed for any NP relation $\mathcal{R}$ represented by a rank-1 constraint system (R1CS), which is a set of quadratic equations that encode the same checks done by $\mathcal{R}$. Formally, it corresponds to

$$\mathcal{R} = \left\{(\boldsymbol{x}; \boldsymbol{w}) : (\boldsymbol{A} \times \boldsymbol{z}^{\mathrm{T}}) \circ (\boldsymbol{B} \times \boldsymbol{z}^{\mathrm{T}}) = \boldsymbol{C} \times \boldsymbol{z}^{\mathrm{T}} \wedge \boldsymbol{z} = (\boldsymbol{x} \| \boldsymbol{w})\right\},$$

where the relation is characterized by fixed matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}_q^{n \times m}$, the statement $\boldsymbol{x} \in \mathbb{F}_q^{\ell}$ is an ℓ-vector, the witness $\boldsymbol{w} \in \mathbb{F}_q^{m-\ell}$ is a $(m-\ell)$-vector, and the "extended witness" $\boldsymbol{z} \in \mathbb{F}_q^{m}$. The $\circ$ operator corresponds to the Hadamard (entry-wise) product. The dimensions n (constraints), m (variables), and ℓ (statement length) are local to this subsection and Section 5.

The Groth16 construction For an R1CS instance, the matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are each interpolated into polynomials as

$$a_i(X) = \sum_{j=1}^{n} a_{i,j} \mathscr{L}_j(X), \quad b_i(X) = \sum_{j=1}^{n} b_{i,j} \mathscr{L}_j(X), \quad c_i(X) = \sum_{j=1}^{n} c_{i,j} \mathscr{L}_j(X)$$

for each $i \in [m]$, and where $\mathscr{L}_j(X) = \prod_{\substack{1 \le i \le n: \\ i \ne j}} \frac{X - \omega^i}{\omega^j - \omega^i}$ is the Lagrange polynomial over the interpolation nodes $\omega^1, \ldots, \omega^n$. The R1CS constraints are then equivalent to the following polynomial relation:

$$\left(\sum_{i=1}^{m} z_i a_i(X)\right)\left(\sum_{i=1}^{m} z_i b_i(X)\right) - \left(\sum_{i=1}^{m} z_i c_i(X)\right) = q(X)V(X),$$

where $V(X) = \prod_{i=1}^{n}(X - \omega^i)$ is the vanishing polynomial and $q(X)$ is the quotient polynomial (defined below). This polynomial relation is equivalent to R1CS satisfiability. The full Groth16 SNARK protocol is defined as follows:

queries in expectation), and simulating the single challenge pre-signature of the unforgeability game with probability 1/2. Extraction fails for the adversary exactly when it fails for the reduction, as $\varepsilon^2 = 1$. Hashing $x(R)$ in place of R leaves the random-oracle analysis of [GSST24] unaffected, as the even-y lift fixes a canonical nonce point.

Construction 1: Groth16 (non-zero-knowledge)

$\mathsf{Gen}(\mathcal{R}) \to \mathsf{crs}$

1. Sample $(\tau, \alpha, \beta, \gamma, \delta) \xleftarrow{\$} \mathbb{F}_q^*$.
2. Output

$$\mathsf{crs} = \left\{ \{[\alpha]_1, [\beta]_2, [\gamma]_2, [\delta]_1, [\delta]_2\}, \left\{ \left[\tau^i\right]_1, \left[\tau^i\right]_2 \right\}_{i=0}^{n-1}, \left\{ \left[\frac{V(\tau)\tau^i}{\delta} \right]_1 \right\}_{i=0}^{n-2}, \right.$$
$$\{[a_i(\tau)]_1, [b_i(\tau)]_1, [b_i(\tau)]_2\}_{i=1}^{m}, \left\{ \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma} \right]_1 \right\}_{i=1}^{\ell},$$
$$\left. \left\{ \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta} \right]_1 \right\}_{i=\ell+1}^{m} \right\}.$$

$\mathsf{Prove}(\mathsf{crs}, \boldsymbol{x}, \boldsymbol{w}) \to \pi$

1. Set $\boldsymbol{z} = (\boldsymbol{x} \| \boldsymbol{w})$.
2. Compute $q(X) = \sum_{i=0}^{n-2} q_i X^i = \frac{(\sum_{i=1}^m z_i a_i(X))(\sum_{i=1}^m z_i b_i(X)) - \sum_{i=1}^m z_i c_i(X)}{V(X)}$.
3. Output $\pi = (\pi_1, \pi_2, \pi_3)$, where

$$\pi_1 = [\alpha]_1 + \sum_{i=1}^{m} z_i \left[a_i(\tau)\right]_1, \qquad \pi_2 = [\beta]_2 + \sum_{i=1}^{m} z_i \left[b_i(\tau)\right]_2,$$
$$\pi_3 = \sum_{i=\ell+1}^{m} z_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\delta} \right]_1 + \sum_{i=0}^{n-2} q_i \left[\frac{V(\tau)\tau^i}{\delta} \right]_1.$$

$\mathsf{Verify}(\mathsf{crs}, \boldsymbol{x}, \pi) \to \{0,1\}$: Output 1 (accept) iff

$$e(\pi_1, \pi_2) \stackrel{?}{=} e([\alpha]_1, [\beta]_2) + e\left(\sum_{i=1}^{\ell} x_i \left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma} \right]_1, [\gamma]_2 \right) + e(\pi_3, [\delta]_2).$$

Since we do not consider the zero-knowledge property, Prove is **deterministic**, i.e., $\pi = \mathsf{Prove}(\mathsf{crs}, \boldsymbol{x}, \boldsymbol{w})$ is a unique proof for $(\boldsymbol{x}, \boldsymbol{w})$.

3.6 Extractable Witness Encryption

We reproduce the security definition of extractable WE from [GKS+26, GGSW13]. An (extractable) witness encryption scheme for a relation $\mathcal{R}$ allows one to encrypt a message msg under an NP statement $\mathbf{x} \in \mathcal{R}$, such that anyone who knows the corresponding witness $\mathbf{w}$ for $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ can decrypt the ciphertext.[3]

Definition 1. *An* extractable witness encryption (WE) scheme *for a relation $\mathcal{R}$ and message space $\mathcal{M}$ is the tuple of three algorithms* $\mathsf{WE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $\mathsf{Gen}(\mathcal{R}) \to \mathsf{crs}$*: takes as input the relation $\mathcal{R}$ and outputs a common reference string* crs*.*
- $\mathsf{Enc}(\mathsf{crs}, \mathbf{x}, \mathsf{msg}) \to \mathsf{ct}$*: takes as input a common reference string* crs*, a statement $\mathbf{x}$, and message $\mathsf{msg} \in \mathcal{M}$ and outputs a ciphertext* ct*.*
- $\mathsf{Dec}(\mathsf{crs}, \mathsf{ct}, \mathbf{w}) \to \mathsf{msg}$*: takes as input a common reference string* crs*, a witness $\mathbf{w}$, and ciphertext* ct *and outputs a message* msg*.*

The WE scheme should satisfy the following properties:

[3] As in BABE, we distinguish between the statement and witness pair used in a SNARK with $(\boldsymbol{x}, \boldsymbol{w})$, and that used in witness encryption with $(\mathbf{x}, \mathbf{w})$.

- **Correctness.** For every statement-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ and message msg:

$$\Pr\left[\ \mathsf{Dec}(\mathsf{crs}, \mathsf{ct}, \mathbf{w}) = \mathsf{msg} \ \ : \ \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(\mathcal{R}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathbf{x}, \mathsf{msg}) \end{array} \right] = 1$$

- **Security.** An extractable witness encryption for a relation $\mathcal{R}$ is secure if for every PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$ such that for all benign auxiliary inputs $\mathsf{aux} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, if

$$\Pr\left[\ b = b' \ \ : \ \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Gen}(\mathcal{R}) \\ (\mathbf{x}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \xleftarrow{\$} \{0,1\}, \\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathbf{x}, \mathsf{msg}_b) \\ b' \leftarrow \mathcal{A}(\mathsf{ct}_b, \mathsf{aux}) \end{array} \right] = \frac{1}{2} + \varepsilon$$

  then

$$\Pr\left[(\mathbf{x}, \mathbf{w}) \in \mathcal{R} : \mathbf{w} \leftarrow \mathcal{E}(\mathsf{crs}, \mathbf{x}, \mathsf{aux})\right] \geq \varepsilon - \mathsf{negl}(\lambda).$$

3.7 Privacy-Free Garbled Circuits

Similar to many recent BitVM works, we utilize *privacy-free* garbled circuits [FNO15], where the evaluator knows some of the input to the garbled program in cleartext. We adapt the (non-standard) seeded garbling definition of [KTA+26], where the input labels are derived from the seed rather than supplied externally; in our construction the Feldman shares do not serve as labels but as keys of a table wrapped around them (Scheme 4).

Definition 2 (Garbling Scheme)**.** *Let $\mathcal{C}_{\mathsf{fam}}$ be a family of binary circuits. A* seeded garbling scheme *is a tuple of algorithms:*

- $\mathsf{Garble}(1^\lambda, C, \mathsf{seed}) \to \left(\mathsf{ct}_{\mathsf{GC}}, \{L_{j,b}\}_{j\in[\ell], b\in\{0,1\}}\right)$*: takes as input a circuit $C\colon \{0,1\}^\ell \to \{0,1\}^k \in \mathcal{C}_{\mathsf{fam}}$ and a seed $\mathsf{seed} \in \{0,1\}^\lambda$, and outputs a garbled circuit description $\mathsf{ct}_{\mathsf{GC}}$ together with the input labels $L_{j,b} \in \{0,1\}^\lambda$ of each input wire j and binary value b. Both outputs are deterministic functions of (C, seed).*
- $\mathsf{Eval}(\mathsf{ct}_{\mathsf{GC}}, L_x)$*: given a garbled circuit $\mathsf{ct}_{\mathsf{GC}}$ and the encoded input $L_x = (L_{1,x_1}, \ldots, L_{\ell,x_\ell})$, one label per input wire, outputs the circuit output $y \in \{0,1\}^k$.*
- $\mathsf{CommitTable}(\mathsf{ct}_{\mathsf{GC}}) \to H_{\mathsf{GC}}$*: returns $H_{\mathsf{GC}} := H(\mathsf{ct}_{\mathsf{GC}})$, where H is a random oracle.*

The garbling scheme should satisfy the following properties.

Correctness. *For any boolean circuit $C \in \mathcal{C}_{\mathsf{fam}}$, any input $x \in \{0,1\}^\ell$, and any seed $\mathsf{seed} \in \{0,1\}^\lambda$,*

$$\Pr\left[C(x) = \mathsf{Eval}(\mathsf{ct}_{\mathsf{GC}}, L_x) : \ \begin{array}{c} \left(\mathsf{ct}_{\mathsf{GC}}, \{L_{j,b}\}_{j\in[\ell], b\in\{0,1\}}\right) \leftarrow \mathsf{Garble}\left(1^\lambda, C, \mathsf{seed}\right) \\ L_x := (L_{1,x_1}, \ldots, L_{\ell,x_\ell}) \end{array} \right] = 1.$$

Adaptive privacy. *There exists a PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for every PPT*

adversary $\mathcal{A}$, every circuit $C \in \mathcal{C}_{\mathsf{fam}}$, and every auxiliary input $\mathsf{aux} \in \{0,1\}^{\mathsf{poly}(\lambda)}$, we have

$$\left| \Pr\left[\mathcal{A}(\mathsf{ct_{GC}}, L_x, \mathsf{aux}) = 1 : \begin{array}{c} \mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda \\ \left(\mathsf{ct_{GC}}, \{L_{j,b}\}_{j\in[\ell],b\in\{0,1\}}\right) \leftarrow \mathsf{Garble}\left(1^\lambda, C, \mathsf{seed}\right) \\ x \leftarrow \mathcal{A}(\mathsf{ct_{GC}}, \mathsf{aux}) \\ L_x := (L_{1,x_1}, \ldots, L_{\ell,x_\ell}) \end{array} \right] - \Pr\left[\mathcal{A}(\mathsf{ct_{GC}}, L_x, \mathsf{aux}) = 1 : \begin{array}{c} (\mathsf{ct_{GC}}, \mathsf{st}) \leftarrow \mathsf{Sim}_1(\mathsf{topo}(C)) \\ x \leftarrow \mathcal{A}(\mathsf{ct_{GC}}, \mathsf{aux}) \\ L_x \leftarrow \mathsf{Sim}_2(\mathsf{st}, x, C(x)) \end{array} \right] \right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{topo}(C)$ *denotes the circuit topology of* C*. The garbling scheme is referred to as* privacy-free *if it only satisfies correctness.*

Note that the garbling scheme as presented is "seeded" since $\mathsf{ct_{GC}}$ and the input labels are deterministically generated from seed alone. This is useful for our setting because it enables a challenger to verify any garbled circuit, input labels included, by re-deriving it from the opened seed, thus eliminating the need to transmit the circuit itself. The commitment is *binding* by collision resistance of H, which is what cut-and-choose needs (Lemma 16), and in the random oracle model it is *equivocable*, i.e., a simulator may fix H_{GC} before the table and program H accordingly, which the knowledge-soundness proof uses (Lemma 18). Note, we do not require the commitment to be hiding, as finalized tables are delivered and opened ones are re-derived from the seed.

Cut-and-Choose As presented, the garbling scheme is not secure against a malicious garbler, i.e., one who can arbitrarily deviate from the protocol description. This is remedied by utilizing *cut-and-choose* [Lin16]. The idea behind cut-and-choose is that the garbler generates N_{CC} garbled circuit *candidates*, commits to them, and sends the commitments to the evaluator. The evaluator then uniformly at random chooses a *finalized set* $\mathcal{I} \subseteq [N_{\mathsf{CC}}]$ of size M_{CC} and challenges its complement: for each opened candidate $i \in [N_{\mathsf{CC}}] \setminus \mathcal{I}$, the garbler "opens" the candidate by revealing the garbling seed, together with any garbler-supplied inputs, to the evaluator. The evaluator re-derives the garbled circuit to verify consistency against the commitment, aborting if any check fails. The finalized candidates in $\mathcal{I}$ are subsequently used for actual evaluation. This approach ensures a cheating garbler would be caught with very high probability: it goes undetected only if every corrupted candidate lies in $\mathcal{I}$, and corrupts *all* finalized candidates only if the corrupted set is exactly $\mathcal{I}$, occurring with probability at most $\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}^{-1}$ (at most $2^{-\kappa} = 2^{-40}$ for our parameter choices). We reserve the term *instance* for the dispute instances of Section 4.

3.8 The Bitcoin Ledger

We model Bitcoin as a standard distributed *ledger* $\mathcal{F}_{\mathrm{BTC}}$: an append-only sequence of confirmed transactions. We assume the existence of a global clock that measures time in rounds $r \in \mathcal{N}$, for $r \leq \mathsf{poly}(\lambda)$. A transaction tx can be submitted by any party at any round r by invoking $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(tx)$. A party P's local view of the ledger at round r is denoted by $\mathcal{L}_P^r$; we drop the round superscript and write $\mathcal{L}_P$ (respectively, $\mathcal{L}_V$) for a party's current view when the round is immaterial. We denote the height of a ledger $\mathcal{L}$ as $h(\mathcal{L})$, and that a ledger of height h corresponds to a sequence of *blocks* $\mathcal{L}[0], \ldots, \mathcal{L}[h-1]$. A block $\mathcal{L}[i]$ is a sequence of transactions, such that the ledger itself can also be viewed as an ordered sequence of transactions. Appending (concatenating) a block blk to a ledger $\mathcal{L}$ forms a new ledger $\mathcal{L}||\mathsf{blk}$.

As standard, we assume a ledger satisfies the standard properties of *safety* (once a transaction is confirmed, it is immutable) and *liveness* (each valid transaction is included in the ledger within a known bound of Δ_{conf} blocks) [GKL15]. The confirmation bound Δ_{conf} enters the timelock conditions of Section 9.

UTXO Model and Taproot Scripts Bitcoin uses the UTXO (unspent transaction output) model to represent coins, where each UTXO is a pair $\mathsf{out} = (a, \mathsf{lockScript})$, in which $a \in \mathbb{R}_{\geq 0}$ is the amount of coins (in BTC) in the UTXO and $\mathsf{lockScript}$ is a program that specifies under which condition(s) the UTXO can be spent. We leverage Taproot Trees [WNT20a] (equivalently "Taptrees"), which make a UTXO spendable by satisfying one of multiple locking scripts. The spending conditions are encoded as (Tap)leaves of a Merkle tree.

We denote the locking script of a UTXO as $\mathsf{lockScript} = \langle \mathsf{leaf}_0, \ldots, \mathsf{leaf}_{k-1} \rangle$, where leaf_i is the ith leaf of a taproot tree. Spending a UTXO through a Tapleaf of a Taptree implies that the user provided a valid Merkle inclusion proof for that Tapleaf.

Transactions A *transaction* X is a triple $\mathsf{X} = (\mathsf{inputs}, \mathsf{outputs}, \mathsf{witness})$, where:

- $\mathsf{inputs} = \langle \mathsf{in}_0, \ldots, \mathsf{in}_{n-1} \rangle$. Each input references an output of a previous transaction.
- $\mathsf{outputs} = \langle \mathsf{out}_0, \ldots, \mathsf{out}_{m-1} \rangle$. Each output is a pair $(a, \mathsf{lockScript})$ (defined above).
- $\mathsf{witness} = \langle w_0, \ldots, w_{n-1} \rangle$. The ith witness w_i satisfies the corresponding input leaf script.

For a transaction to be valid, the following conditions must be met: every witness must satisfy the locking condition of its corresponding input, every input must be unspent, and the total value of the outputs must be at most the total value of the inputs. We define the transaction *skeleton* to be $(\mathsf{inputs}, \mathsf{outputs})$, i.e., X without its witness, written $\overline{\mathsf{tx}}$ where the distinction matters; a named transaction X denotes its skeleton until the witness field is assigned via $\mathsf{X}.\mathsf{witness} \leftarrow \cdots$, one tuple per input (";"-separated when X has several inputs). Transaction IDs and sighashes depend only on the skeleton, but never on the witness.

A *sighash* (signature message) additionally depends on which input is signed and on the executed script path. We write $\mathsf{SigMsg}(\overline{\mathsf{tx}}, i, c)$ for the BIP-341/342 tapscript signature message [WNT20a, WNT20b] for spending input i of $\overline{\mathsf{tx}}$ under $\mathsf{SIGHASH_DEFAULT}$. This quantity commits to the entire skeleton, the input index i, the executed tapleaf, and the opcode position c of the last-executed $\mathsf{OP_CODESEPARATOR}$ (where $c = \texttt{0xffffffff}$ if none is executed; the tapleaf is left implicit in the notation).

Covenant Emulation A *covenant* restricts how a UTXO may be spent in the future, beyond what Bitcoin's standard scripts can express [WAA+26a]. Proposed covenant opcodes [MEGS16, OP17] would express such constraints directly, but they do not exist on Bitcoin today. We therefore emulate covenants with a signing committee: the output to be constrained is locked under an N-of-N aggregate key, and at setup the committee signs only the transactions that respect the intended transaction graph. Since each signature commits to the spending transaction's exact inputs and outputs under $\mathsf{SIGHASH_DEFAULT}$, substituting any other transaction invalidates it. Once the committee deletes its key shares, no further covenant signatures can ever be created, making the pre-signed set exhaustive (Section 8.1.1).

Connectors A *connector output* is an output of a transaction that can potentially be spent by more than one subsequent transaction. For example, suppose transaction A has one connector output that can be spent by either transactions B or C under different conditions. If a user posts transaction B (by satisfying the conditions that dictate how the connector output is spent), then

they prevent any other party from posting C because one of the inputs of C is no longer available. We reference connectors according to their position within a transaction: $\mathsf{X}[i]$ denotes the ith (0-based) output of a transaction X, so the connector output of A above is $\mathsf{A}[0]$. We extensively use connectors throughout our construction.

Ledger Interface Parties append a transaction X by invoking $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{X})$, which broadcasts the fully witnessed X, and observe the chain through their view (e.g., "upon observing X in $\mathcal{L}_P$"), where confirmations are taken at a reorg-safe depth.

Cryptographic Objects Our construction requires the existence of several cryptographic objects that the ledger functionality utilizes:

- *Signature Scheme.* EUF-CMA secure signature scheme $\mathsf{Sig}_{\mathrm{BTC}} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ with algorithms:

$$\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$$
$$\mathsf{Sign}(\mathsf{sk}, \mathsf{X}) \to \sigma$$
$$\mathsf{Verify}(\mathsf{pk}, \mathsf{X}, \sigma) \to \{0, 1\}$$

  When the message is a transaction X, signing and verification operate on the corresponding sighash $\mathsf{SigMsg}(\overline{\mathsf{tx}}, i, c)$ of the input being spent; we pass an explicit sighash only where the codeseparator position is load-bearing (Section 7). $\mathsf{Sig}_{\mathrm{BTC}}$ is instantiated as a BIP-340 Schnorr signature over secp256k1 [WNR20].
  For a transaction X and role $\rho \in \{P, V\}$, a role in the *superscript* $(\sigma^P_{\mathsf{X}}, \sigma^V_{\mathsf{X}}, \sigma^{\mathsf{id}}_{\mathsf{X}}, \sigma^{\mathsf{cov}}_{\mathsf{X}})$ denotes a pre-signature produced at setup, while a role in the *subscript* $(\sigma_{P,\mathsf{X}}, \sigma_{V,\mathsf{X}})$ denotes a runtime signature produced by an explicit $\mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}$ call at broadcast time. Per-input copies are distinguished by tuple position or a $^{(j)}$ index, and transaction names may be abbreviated in subscripts (e.g., $\mathsf{WithdrawOpt}$).
- *Hash Function.* $\mathsf{Hash}_{\mathrm{BTC}}(m) = h$, modeled as a random oracle; it is instantiated as SHA256 for message hashlocks and as HASH160 for Lamport commitments.

Locking Scripts Our construction utilizes the following locking scripts:

- *Signature Check.* $\mathsf{CheckSig}(\mathsf{pk})$ requires the spending transaction X's witness w to contain a signature σ such that $\mathsf{Verify}(\mathsf{pk}, \mathsf{X}, \sigma) = 1$.
- *Relative Timelock.* $\mathsf{RelTimelock}(\Delta)$ requires that *at least* Δ blocks have elapsed since the output referenced by the input was created. If a transaction with an input of the form $\mathsf{in} = (\mathsf{X}_{\mathsf{prev}}[\mathsf{outIdx}], \mathsf{RelTimelock}(\Delta))$ is included at block height h, then $\mathsf{X}_{\mathsf{prev}}$ must have been included in a block at height at most $h - \Delta$.
- *Hash Lock.* $\mathsf{HashLock}(h)$ requires the transaction witness w contains a preimage of a hash h, i.e., $\mathsf{Hash}_{\mathrm{BTC}}(w) = h$.

Multiple scripts can be chained together with logical AND ($\wedge$) and OR ($\vee$) to create more complex locking scripts.

- *Lamport Signature.* $\mathsf{CheckLampSig}(\mathsf{lpk})$ requires that the transaction witness contains a valid Lamport signature for a given public key lpk on some message, with each preimage of length λ. In our construction, this enforces a binding commitment from the prover to some proof element is included in a transaction.

$$\mathsf{CheckLampSig}(\mathsf{lpk}) := \bigwedge_{i=1}^{\ell} (\mathsf{HashLock}(\mathsf{lpk}_i^0) \vee \mathsf{HashLock}(\mathsf{lpk}_i^1))$$

4 The Multi-Party BitVM-core Primitive

We define a modified multi-party variant of the BitVM-core primitive from [GKS+26] (which itself is a simplified version of BitVM2-core from [WAA+26b]).

A BitVM-core protocol is an interactive protocol run by the *operator set* $\mathcal{O} := \{O_1, \ldots, O_N\}$ together with a user U who supplies a deposit, all of whom interact with the Bitcoin ledger functionality $\mathcal{F}_{\text{BTC}}$. Every ordered pair of operators forms a *dispute instance*. We define the instance set $\mathcal{D} := \{(\mathsf{p}, \mathsf{v}) \in [N]^2 : \mathsf{p} \neq \mathsf{v}\}$ with $|\mathcal{D}| = N(N-1)$, where O_{p} acts as the *prover-candidate* and O_{v} as the *verifier* of instance (p, v). Each operator is thus the prover-candidate of $N-1$ instances and the verifier of $N-1$ others. Committee indices are set in upright sans-serif. We use i, j for operator indices and p, v for the prover-candidate and verifier indices in a particular instance.

We consider a PPT adversary $\mathcal{A}$ that *statically* corrupts a set $C \subset \mathcal{O}$ of at most $N-1$ operators, fixed before the execution begins. Corrupted parties may deviate arbitrarily from the protocol. Honest parties follow the protocol and monitor the ledger continuously. The resulting 1-of-N assumption, that at least one operator remains honest throughout both phases, is what makes the protocol *trust-minimized*. Honesty during setup matters in particular for covenant emulation, as the pre-signed spender set is exhaustive only if at least one operator deletes its covenant key share (Section 3.8).

Parties communicate over authenticated pairwise channels and an authenticated broadcast channel. The network is synchronous, and a party that fails to send an expected message by its deadline is treated as having aborted. The ledger satisfies safety and liveness with confirmation bound Δ_{conf}. The common reference string is generated in a one-time trusted setup, and the MuSig2 multi-signature scheme [NRS21] used for covenant emulation is secure.

A BitVM-core protocol is divided into the *setup* and *proving* phases. In the setup phase, the parties establish the *transaction set* $\mathcal{T}$, the set of transaction skeletons comprising the pre-agreed transaction graph. Each operator O_{i} generates and broadcasts the public statement $\boldsymbol{x}_{\mathsf{i}}$ for the instances in which it acts as prover-candidate, produced by an application-specific procedure $\mathsf{GenStmt}$ whose semantics we treat as an application-level concern. The user's deposit of value B is locked on $\mathcal{F}_{\text{BTC}}$ during setup, spendable only through the pre-agreed graph. To prevent malicious behavior, parties validate all messages received from each other and abort the setup upon detecting invalid information.

We denote the subset $\mathcal{S} \subseteq \mathcal{T}$ of transactions permitted to spend the deposit the *spender set*, and write $\mathcal{S}_{\mathsf{p}} \subseteq \mathcal{S}$ for the spenders paying B to operator O_{p}. The objective of the protocol is that O_{p} can post a transaction from $\mathcal{S}_{\mathsf{p}}$ if and only if it knows a valid witness for its statement $\boldsymbol{x}_{\mathsf{p}}$. Definitions 4 and 5 formalize the two directions of this guarantee.

In the proving phase, the operators interact with the ledger functionality. A prover-candidate that claims to know a witness $\boldsymbol{w}$ for its statement initiates a withdrawal, and the operators post transactions from $\mathcal{T}$ in response to on-chain events. The prover wins the instance by posting a transaction from $\mathcal{S}_{\mathsf{p}}$, and a verifier wins by permanently preventing every transaction in $\mathcal{S}_{\mathsf{p}}$ from being posted. Exactly one of the two outcomes occurs in any execution.

Protocol Syntax Our BitVM-core construction consists of several interfaces. We first have a one-time initialization algorithm, which must occur prior to any setup procedures:

- *Initialization*: $\mathsf{Gen}(\mathcal{R}) \to \mathsf{crs}$ takes as input an NP relation $\mathcal{R}$ and outputs a common reference string crs. It is run in a one-time trusted setup, and all parties have access to its output.

The setup phase consists of the committee setup:

- *Committee setup*: $\mathsf{Setup}_{\mathcal{O}}(\mathsf{crs}) \to \left(\{\boldsymbol{x}_{\mathsf{j}}\}_{\mathsf{j}\in[N]}, \mathcal{T}, \mathcal{S}, \mathsf{st}_{O_{\mathsf{i}}}\right)$ takes as input a common reference

string crs. It outputs the statements $\{\boldsymbol{x}_{\mathsf{j}}\}_{\mathsf{j}\in[N]}$ of all parties, the transaction set $\mathcal{T}$, the spender set $\mathcal{S}$, and the executing party's local state $\mathsf{st}_{O_{\mathsf{i}}}$. Alternatively, a party may output $\perp$ to indicate that the committee setup failed.

Within the committee setup, every instance $(\mathsf{p},\mathsf{v}) \in \mathcal{D}$ performs four pairwise *interactive* sub-protocols between O_{p} and O_{v}. First, parties engage in the cut-and-choose sub-protocols. Then, the transaction set is assembled by $\mathsf{CreateTxSet}_{\mathsf{mp}}$. Finally, the parties engage in the pre-signing sub-protocols, as the pre-signatures are computed over sighashes of $\mathcal{T}$. In each sub-protocol, either party may output $\perp$ to indicate that the setup failed.

- *Prover cut-and-choose*: $P^{\mathsf{cc}}_{\mathsf{Setup}}(\mathsf{crs},\boldsymbol{x}_{\mathsf{p}}) \to \mathsf{st}^P$ takes as input crs and the prover-candidate's statement $\boldsymbol{x}_{\mathsf{p}}$ and outputs the prover's instance state st^P.
- *Verifier cut-and-choose*: $V^{\mathsf{cc}}_{\mathsf{Setup}}(\mathsf{crs},\boldsymbol{x}_{\mathsf{p}}) \to \mathsf{st}^V$ takes the same inputs and outputs the verifier's instance state st^V.
- *Prover pre-signing*: $P^{\mathsf{ps}}_{\mathsf{Setup}}(\mathcal{T},\mathsf{st}^P) \to \mathsf{st}^P$ takes as input the transaction set $\mathcal{T}$ and the prover's instance state and outputs an updated st^P.
- *Verifier pre-signing*: $V^{\mathsf{ps}}_{\mathsf{Setup}}(\mathcal{T},\mathsf{st}^V) \to \mathsf{st}^V$ takes as input $\mathcal{T}$ and the verifier's instance state and outputs an updated st^V.

The proving phase consists of the following algorithms:

- *Prover*: $P_{\mathsf{Prove}}(\mathsf{crs},\boldsymbol{x}_{\mathsf{p}},\mathcal{T},\mathcal{S},\mathsf{st}_{O_{\mathsf{p}}},\boldsymbol{w})$ takes as input crs, the party's own statement $\boldsymbol{x}_{\mathsf{p}}$, the transaction set $\mathcal{T}$, the spender set $\mathcal{S}$, its local state $\mathsf{st}_{O_{\mathsf{p}}}$, and a witness $\boldsymbol{w}$. It returns no value, as its outcome is realized on the ledger.
- *Verifier*: $V_{\mathsf{Prove}}(\mathsf{crs},\boldsymbol{x}_{\mathsf{p}},\mathcal{T},\mathcal{S},\mathsf{st}_{O_{\mathsf{v}}})$ takes as input crs, the prover-candidate's statement $\boldsymbol{x}_{\mathsf{p}}$, $\mathcal{T}$, $\mathcal{S}$, and its local state $\mathsf{st}_{O_{\mathsf{v}}}$. It returns no value, as its outcome is realized on the ledger.

We say the committee setup *completes* if every honest party obtains an output other than $\perp$ and the deposit is forwarded under the pre-agreed graph. Otherwise, the setup *aborts*. During setup, the worst a malicious party can do is force an abort, and setup safety guarantees that no funds are lost in this case.

Definition 3 (Setup safety)**.** *If the committee setup aborts, then an honest user recovers the full deposit B after a fixed timeout, and no honest operator loses funds (beyond transaction fees).*

Completeness (equivalently, "robustness" in BABE) guarantees that malicious verifiers cannot prevent an honest prover from withdrawing.

Definition 4 (Completeness)**.** *Suppose the committee setup completes, and let O_{p} be an honest prover-candidate that knows a witness $\boldsymbol{w}$ with $(\boldsymbol{x}_{\mathsf{p}},\boldsymbol{w}) \in \mathcal{R}$. For every PPT adversary $\mathcal{A}$ statically corrupting any subset of $\mathcal{O}\backslash\{O_{\mathsf{p}}\}$, when O_{p} executes $P_{\mathsf{Prove}}(\mathsf{crs},\boldsymbol{x}_{\mathsf{p}},\mathcal{T},\mathcal{S},\mathsf{st}_{O_{\mathsf{p}}},\boldsymbol{w})$, a transaction from $\mathcal{S}_{\mathsf{p}}$ is included in the ledger within a bounded number of rounds (a fixed function of the protocol timelocks and Δ_{conf}), unless a transaction from $\mathcal{S}_{\mathsf{j}}$ for a corrupt O_{j} with $\mathsf{j} \neq \mathsf{p}$ is included first, except with probability $(N-1)\cdot 2^{-\kappa} + \mathsf{negl}(\lambda)$.*

This exception is necessary because $\mathsf{Deposit}[0]$ can be spent only once: if a corrupt operator withdraws first, the honest operator can no longer withdraw. Knowledge soundness (Definition 5) applies to the corrupt operator's withdrawal. If $\mathcal{R}$ and $\mathsf{GenStmt}$ allow a valid witness for at most one operator's statement per deposit, this competing withdrawal cannot succeed except with the soundness error probability.

Knowledge soundness guarantees that a prover-candidate that withdraws knows a valid witness, provided a single verifier of the instance is honest. Knowledge is formalized by an extractor that recovers the witness from the adversary's inputs and the transcript of its interactions with the honest verifier and $\mathcal{F}_{\mathrm{BTC}}$. Note that $\mathcal{S}_{\mathsf{p}}$ contains the optimistic spend, so soundness in particular rules out an unchallenged optimistic withdrawal by a prover without a witness.

Definition 5 (Knowledge soundness)**.** *Suppose the committee setup completes, and let O_{v} be an honest verifier of instance (p, v) that executes $V_{\mathsf{Prove}}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \mathcal{T}, \mathcal{S}, \mathsf{st}_{O_{\mathsf{v}}})$. For every PPT adversary $\mathcal{A}$ statically corrupting O_{p} (and possibly all other parties except O_{v}), there exists a PPT extractor $\mathcal{E}$ such that, if a transaction from $\mathcal{S}_{\mathsf{p}}$ is included in the ledger, then $\mathcal{E}$, given $\mathcal{A}$'s inputs and the transcript of its interactions with O_{v} and $\mathcal{F}_{\mathrm{BTC}}$, outputs $\boldsymbol{w}$ with $(\boldsymbol{x}_{\mathsf{p}}, \boldsymbol{w}) \in \mathcal{R}$, except with probability $2^{-\kappa} + \mathsf{negl}(\lambda)$.*

Both properties bound the probability that the setup completes and the property fails. They are therefore trivially satisfied by a protocol in which some party always aborts during setup. The final property rules this out.

Definition 6 (Non-triviality)**.** *If all parties are honest, then the committee setup completes.*

Remark 1 (Composition)**.** *Knowledge soundness is a per-instance property, while completeness covers the $N-1$ instances of an honest prover-candidate, hence the factor $N-1$ in Definition 4. The system-level guarantees follow by a union bound over all $N(N-1)$ instances with $N = \mathsf{poly}(\lambda)$, which we prove in Section 9. The instances are not independent, as a prover-candidate reuses its Lamport and reveal keys across all of its instances, but a union bound requires no independence.*

5 Witness Encryption for Groth16

Witness encryption for Groth16 is the foundational cryptographic technique for BABE. It relies upon the intermediate assumption that one proof element (π_1) is known to the encryptor. This is an unnatural assumption but is easily remedied when combined with a suitable garbling scheme.

Let $\mathcal{R}$ be an arbitrary R1CS relation. Define the following relation:

$$\mathcal{R}' = \{((\mathsf{crs}, \boldsymbol{x}, \pi_1); \boldsymbol{w}) : (\pi_1, \pi_2, \pi_3) = \mathsf{Groth16.Prove}(\mathsf{crs}, \boldsymbol{x}, \boldsymbol{w}) \wedge (\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}\}\,.$$

The proof element π_1 is known to the encryptor, i.e., it is part of the statement $(\mathsf{crs}, \boldsymbol{x}, \pi_1)$. Our witness encryption scheme is formally specified as a triple of algorithms $\mathsf{WE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Construction 2: Groth16 WE [GKS+26] with known π_1

$\mathsf{Gen}(\mathcal{R}') \to \mathsf{crs}$: Output $\mathsf{crs} = \mathsf{Groth16.Gen}(\mathcal{R})$.

$\mathsf{Enc}(\mathsf{crs}, \boldsymbol{x}, \pi_1, \mathsf{msg}) \to (\mathsf{ct}_{\mathsf{setup}}, \mathsf{ct}_{\mathsf{prove}})$:

1. Sample $r \xleftarrow{\$} \mathbb{F}_q^*$ and execute the two subroutines:
2. $\mathsf{Enc}_{\mathsf{setup}}(\mathsf{crs}, \boldsymbol{x}, \mathsf{msg}; r) \to \mathsf{ct}_{\mathsf{setup}}$:
   2.1. Compute $X = \sum_{i=1}^{\ell} x_i \left(\left[\frac{\beta a_i(\tau) + \alpha b_i(\tau) + c_i(\tau)}{\gamma} \right]_1 \right)$, where ℓ is the statement length (Section 3.5).
   2.2. Compute $Y = e([\alpha]_1, [\beta]_2) + e(X, [\gamma]_2)$.
   2.3. Compute $\mathsf{ct}_{\mathsf{rd}} = r[\delta]_2$ and $\mathsf{ct}_{\mathsf{msg}} = H(rY) + \mathsf{msg}$.
   2.4. Output $\mathsf{ct}_{\mathsf{setup}} := (\mathsf{ct}_{\mathsf{rd}}, \mathsf{ct}_{\mathsf{msg}})$.
3. $\mathsf{Enc}_{\mathsf{prove}}(\pi_1; r) \to \mathsf{ct}_{\mathsf{prove}}$: Output $\mathsf{ct}_{\mathsf{prove}} = r\pi_1$.
4. Output $(\mathsf{ct}_{\mathsf{setup}}, \mathsf{ct}_{\mathsf{prove}})$.

$\mathsf{Dec}((\mathsf{ct}_{\mathsf{setup}}, \mathsf{ct}_{\mathsf{prove}}), (\pi_2, \pi_3)) \to \mathsf{msg}$:

1. Compute $\mathsf{msg} = \mathsf{ct}_{\mathsf{msg}} - H(e(\mathsf{ct}_{\mathsf{prove}}, \pi_2) - e(\pi_3, \mathsf{ct}_{\mathsf{rd}}))$.
2. Output msg.

This is a WE scheme for the relation $\mathcal{R}$, such that the decryptor cannot learn any information about the message msg unless they know a valid witness $\boldsymbol{w}$ for the R1CS relation $\mathcal{R}$. The scheme

is secure in the context where an adversary can adaptively choose part of the statement π_1 after receiving a partial ciphertext $\mathsf{ct}_{\mathsf{setup}}$ corresponding to $\boldsymbol{x}$.

We reproduce the security definition of extractable WE for a relation $\mathcal{R}'$ from [GKS+26].

Definition 7 (Adaptive Security of WE for $\mathcal{R}'$ [GKS+26])**.** *A witness encryption scheme* $\mathsf{WE} = (\mathsf{Gen}, (\mathsf{Enc}_{\mathsf{setup}}, \mathsf{Enc}_{\mathsf{prove}}), \mathsf{Dec})$ *for the relation* $\mathcal{R}'$ *is* adaptively secure *iff for every PPT adversary* $\mathcal{A}$*, there exists a PPT extractor* $\mathcal{E}$ *such that for all benign auxiliary input* $\mathsf{aux} \in \{0,1\}^{\mathsf{poly}(\lambda)}$*, if*

$$\Pr\left[\, b = b' \;:\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Gen}(\mathcal{R}') \\ (\boldsymbol{x}, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \mathbb{F}_q^* \\ \mathsf{ct}^b_{\mathsf{setup}} \leftarrow \mathsf{Enc}_{\mathsf{setup}}(\mathsf{crs}, \boldsymbol{x}, \mathsf{msg}_b; r) \\ \pi_1 \leftarrow \mathcal{A}(\mathsf{ct}^b_{\mathsf{setup}}) \\ \mathsf{ct}_{\mathsf{prove}} \leftarrow \mathsf{Enc}_{\mathsf{prove}}(\pi_1; r) \\ b' \leftarrow \mathcal{A}(\mathsf{ct}_{\mathsf{prove}}, \mathsf{aux}) \end{array} \right] = \frac{1}{2} + \varepsilon$$

then

$$\Pr\left[(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R} : \boldsymbol{w} \leftarrow \mathcal{E}(\mathsf{crs}, \boldsymbol{x}, \mathsf{aux})\right] \geq \varepsilon - \mathsf{negl}(\lambda).$$

The construction is secure in the generic bilinear group model and random oracle model, i.e., the extractor can access the adversary's queries to both oracles. The RO ensures the extractor can access the group element that randomizes the message (rY). Otherwise, from the POV of the adversary, the ciphertext perfectly hides msg. The only way for the adversary to learn anything from the ciphertext is to query rY and learn the masking term, but then the extractor would be given access to rY in the process.

Lemma 1 ([GKS+26])**.** *Construction 2 is adaptively secure according to Definition 7 in the generic bilinear group and random oracle models.*

Corollary 1 ([GKS+26])**.** *Construction 2 is a secure witness encryption scheme for* $\mathcal{R}$ *in the generic bilinear group and random oracle models.*

6 Garbled Scalar Multiplication on BN254

We summarize the core components of the GC computation here from [GKS+26]. The prover supplies a public, authenticated input point π_1. The garbled circuit embeds the verifier's secret scalar r. The circuit evaluation outputs $f_r(\pi) = r\pi$. Following [GKS+26], throughout this section π denotes a generic curve point, which our construction instantiates with the proof element π_1.

6.1 Notation and Addition on BN254

Let $E/\mathbb{F}_p$ be the BN254 curve in short Weierstrass form

$$E : y^2 \equiv x^3 + 3 \pmod{p},$$

where p is the base-field prime. We denote $n = \lceil \log_2 p \rceil$ as the bit length of elements in $\mathbb{F}_p$ (i.e., the x- and y-coordinates of a point are n bits each). For BN254, the bit length of the scalar field $\mathbb{F}_q^*$ is also $n = \lceil \log_2 q \rceil$.

Let π be a point on the curve, represented in affine coordinates $\pi := (x(\pi), y(\pi)) \in \mathbb{F}_p^2$. Define a vector $\boldsymbol{u}(\pi) \in \mathbb{F}_p^6$ as $\boldsymbol{u}(\pi) := (1, x(\pi), y(\pi), x(\pi)^2, y(\pi)^2, x(\pi)y(\pi))$, and let $\overline{\boldsymbol{u}} \in \{0,1\}^\ell$ be $\boldsymbol{u}$'s binary decomposition, where $\ell = 5n + 1$ throughout this section (Section 8.1.3 reuses ℓ for the $2n$-bit committed message). Concretely, if $\boldsymbol{u}(\pi) = (u_0, u_1, u_2, u_3, u_4, u_5)$ where $u_0 = 1, u_1 = x(\pi), u_2 = y(\pi), u_3 = x(\pi)^2, u_4 = y(\pi)^2, u_5 = x(\pi)y(\pi)$, then

$$\overline{\boldsymbol{u}}(\pi) = (1, u_{1,0}, \ldots, u_{1,n-1}, u_{2,0}, \ldots, u_{2,n-1}, u_{3,0}, \ldots, u_{3,n-1}, u_{4,0}, \ldots, u_{4,n-1}, u_{5,0}, \ldots, u_{5,n-1}),$$

where $(u_{i,0}, \ldots, u_{i,n-1}) = \mathsf{bin}(u_i)$. The notation $u_i(\pi)$ and $\overline{u_i}(\pi)$ denotes the ith component of $\boldsymbol{u}(\pi)$ and $\overline{\boldsymbol{u}}(\pi)$, respectively.

Let $\boldsymbol{G} \in \mathbb{F}_p^{6\times(1+5n)}$ be the matrix

$$\boldsymbol{G} = \left(\begin{array}{c|ccccc} 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline 0 & \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & \mathbf{0} & \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{0} & \mathbf{0} \\ 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{0} \\ 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{2} \end{array}\right),$$

such that $\boldsymbol{G} \times \overline{\boldsymbol{u}}^{\mathrm{T}}(\pi) = \boldsymbol{u}^{\mathrm{T}}(\pi)$. The notation $\mathbf{0} = (0, \ldots, 0) \in \mathbb{F}_p^n$ is a row of n zeroes and $\mathbf{2} = (2^0, 2^1, \ldots, 2^{n-1}) \in \mathbb{F}_p^n$ is a row of powers of two.

We state the lemmas from [GKS+26] that formally define elliptic curve addition on BN254.

Lemma 2 ([GKS+26, Lemma 3])**.** *Let $\pi = (x(\pi), y(\pi)) \in E(\mathbb{F}_p)\setminus\{\mathcal{O}\}$ be given in affine coordinates and let $\delta \in \{0,1\}$. Define $\boldsymbol{M}(\delta) \in \mathbb{F}_p^{3\times 6}$ by*

$$\boldsymbol{M}(\delta) = \begin{pmatrix} 1-\delta & \delta & 0 & 0 & 0 & 0 \\ 1-\delta & 0 & \delta & 0 & 0 & 0 \\ \delta & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Then $\boldsymbol{M}(\delta) \times \boldsymbol{u}^T(\pi)$ is a Jacobian representation of the point $\delta\pi$. Alternatively, $\boldsymbol{N}(\delta) \times \overline{\boldsymbol{u}}^T(\pi)$ is equivalent to $\delta\pi$, where $\boldsymbol{N}(\delta) = \boldsymbol{M}(\delta) \times \boldsymbol{G}$.

Lemma 3 ([GKS+26, Lemma 4])**.** *Let $\phi, \pi \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ be two affine points with $x(\phi) \neq x(\pi)$. Then a Jacobian representation of their sum $\phi + \pi$ is given by $\boldsymbol{A}(\phi) \times \boldsymbol{u}^T(\pi)$, where $\boldsymbol{A}(\phi) \in \mathbb{F}_p^{3\times 6}$ is the matrix defined as*

$$\boldsymbol{A}(\phi) = \begin{pmatrix} 6 & x(\phi)^2 & -2y(\phi) & x(\phi) & 0 & 0 \\ 9y(\phi) & 0 & -(y(\phi)^2 + 9) & 3x(\phi)y(\phi) & y(\phi) & -3x(\phi)^2 \\ -x(\phi) & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Equivalently, the Jacobian coordinates of the sum $\phi + \pi$ are given by $\boldsymbol{B}(\phi) \times \overline{\boldsymbol{u}}^T(\pi)$, where $\boldsymbol{B}(\phi) \in \mathbb{F}_p^{3\times(1+5n)}$ is the matrix defined as $\boldsymbol{B}(\phi) = \boldsymbol{A}(\phi) \times \boldsymbol{G}$.

The next lemma specifies conditional point addition.

Lemma 4 ([GKS+26, Lemma 5])**.** *Let $\phi, \pi \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ be two affine points with $x(\phi) \neq x(\pi)$. Then a Jacobian representation of their sum $\phi + \delta\pi$, where $\delta \in \{0,1\}$ is given by $\boldsymbol{C}(\delta, \phi) \times \boldsymbol{u}^T(\pi)$, where $\boldsymbol{C}(\delta, \phi) \in \mathbb{F}_p^{3\times 6}$ is the matrix defined as*

$$\boldsymbol{C}(\delta, \phi) = \begin{pmatrix} 6\delta + (1-\delta)x(\phi) & \delta x(\phi)^2 & -2\delta y(\phi) & \delta x(\phi) & 0 & 0 \\ 9\delta y(\phi) + (1-\delta)y(\phi) & 0 & -\delta(y(\phi)^2 + 9) & 3\delta x(\phi)y(\phi) & \delta y(\phi) & -3\delta x(\phi)^2 \\ -\delta x(\phi) + (1-\delta) & \delta & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Equivalently, the Jacobian coordinates of the sum $\phi + \delta\pi$ are given by $\boldsymbol{D}(\delta, \phi) \times \overline{\boldsymbol{u}}^T(\pi)$, where $\boldsymbol{D}(\delta, \phi) \in \mathbb{F}_p^{3\times(1+5n)}$ is the matrix defined as $\boldsymbol{D}(\delta, \phi) = \boldsymbol{C}(\delta, \phi) \times \boldsymbol{G}$.

The last lemma states how to construct a uniform representation of a curve point in Jacobian coordinates.

Lemma 5 ([GKS+26, Lemma 6])**.** *Let $\pi \in E(\mathbb{F}_p)$ be a point in Jacobian coordinates, i.e., $\pi = (X, Y, Z)$. Let λ be sampled uniformly from $\mathbb{F}_p^*$ (this λ, a Jacobian randomizer following [GKS+26], is unrelated to the security parameter). Then $\mathsf{diag}(\lambda) \times (X, Y, Z)^T$ is a uniform representation of the same point π, with the diagonal matrix $\mathsf{diag}(\lambda) \in \mathbb{F}_p^{3\times3}$ defined as*

$$\mathsf{diag}(\lambda) = \begin{pmatrix} \lambda^2 & 0 & 0 \\ 0 & \lambda^3 & 0 \\ 0 & 0 & \lambda \end{pmatrix}.$$

6.2 Decomposable Randomized Encodings

A randomized encoding of a function $f(x)$ is a randomized mapping $\hat{f}(x, \rho)$ (called the *encoding*) whose output distribution depends only on the output of f [AIK14]. The encoding is parameterized by some randomness ρ that is independent of the input x, and computes $f(x)$ without revealing any information about x beyond what is leaked by $f(x)$ itself. We restate the DRE from [GKS+26], which in turn follows the definitions from [Ish13]. We refer the reader to [GKS+26] for additional details and proofs.

Definition 8 (Randomized encoding [Ish13, Definition 3.1])**.** *Let $X, Y, \hat{Y}, R$ be finite sets and let $f : X \to Y$. A function $\hat{f} : X \times R \to \hat{Y}$ is a* randomized encoding of f *if it satisfies the following requirements:*

- ***δ-correctness.*** *There exists a* decoder function $\mathsf{Decode} : \hat{Y} \to Y$*, such that for all $x \in X$ and $\rho \in R$ we have*

$$\Pr\left[\mathsf{Decode}(\hat{f}(x, \rho)) \neq f(x)\right] \leq \delta,$$

  where the probability is taken over a uniform choice of ρ from R and the randomness of Decode*.*
- ***ε-privacy.*** *There exists a randomized simulator $\mathsf{Sim} : Y \to \hat{Y}$ such that for all $x \in X$,*

$$\mathsf{Sim}(f(x)) \approx_\varepsilon \hat{f}(x, \rho),$$

  where $\rho \leftarrow R$ is uniform and $\approx_\varepsilon$ denotes distributions that are statistically close.

We omit δ and ε when both are negligible with respect to the (implied) security parameter.

A randomized encoding is said to be *efficient* if the encoding/decoding complexities are polynomial in the size of the input.

We can now define decomposability and some useful properties of randomized encodings.

Definition 9 (Decomposable randomized encoding (DRE) [Ish13, Definition 4.1])**.** *For $f : X_1 \times \cdots \times X_n \to Y$, a* decomposable randomized encoding of f *is one of the form*

$$\hat{f}((x_1, \ldots, x_n), \rho) = (\hat{f}_1(x_1, \rho), \ldots, \hat{f}_n(x_n, \rho))$$

for some functions $\hat{f}_i : X_i \to \hat{Y}_i$.

Lemma 6 (Concatenation [Ish13, Lemma 3.3]). *Suppose $\hat{f}_i(x, \rho_i)$ is a randomized encoding of $f_i(x)$ for $i = 1, \ldots, k$. Then the function $\hat{f}(x, (\rho_1, \ldots, \rho_k)) := (\hat{f}_1(x, \rho_1), \ldots, \hat{f}_k(x, \rho_k))$ is a randomized encoding of $f(x) := (f_1(x), \ldots, f_k(x))$.*

Lemma 7 (Composition [Ish13, Lemma 3.4]). *Suppose $\hat{f}(x, \rho)$ is a randomized encoding of $f(x)$ and $\hat{f}'((x, \rho), \rho')$ is a randomized encoding of $\hat{f}(x, \rho)$, where the latter is viewed as a deterministic function of (x, ρ). Then $\hat{f}''(x, (\rho, \rho')) := \hat{f}'((x, \rho), \rho')$ is a randomized encoding of $f(x)$.*

Lemma 8 (DRE for weighted group summation [Ish13, Claim 4.2]). *Let $\mathcal{G}$ be a finite abelian group and let $f_{a_1,\ldots,a_n} : \mathcal{G}^n \to \mathcal{G}$ be the group summation function $f_{a_1,\ldots,a_n}(x_1, \ldots, x_n) = \sum_{i=1}^n a_i x_i$. Let $R = \{(\rho_1, \ldots, \rho_n) \in \mathcal{G}^n : \sum_{i=1}^n a_i \rho_i = 0\}$. Then the function $\hat{f}_{a_1,\ldots,a_n} : \mathcal{G}^n \times R \to \mathcal{G}^n$ defined by $\hat{f}_{a_1,\ldots,a_n}((x_1, \ldots, x_n), (\rho_1, \ldots, \rho_n)) = (x_1 + \rho_1, \ldots, x_n + \rho_n)$ is a DRE of $f_{a_1,\ldots,a_n}$.*

We also need a DRE for private weights subject to the restriction that inputs are binary. This keeps the DRE encoding function efficient.

Lemma 9 (DRE for private affine functions [GKS+26, Lemma 10]). *Let $\mathcal{G}$ be a finite abelian group and let $f : (\mathcal{G} \times \{0,1\})^n \to \mathcal{G}$ be the group summation function with private weights, $f((a_1, x_1), \ldots, (a_n, x_n)) = \sum_{i=1}^n a_i x_i$. Let $R = \{(\rho_1, \ldots, \rho_n) \in \mathcal{G}^n : \sum_{i=1}^n \rho_i = 0\}$. Then the function $\hat{f} : (\mathcal{G} \times \{0,1\})^n \times R \to \mathcal{G}^n$ defined by $\hat{f}(((a_1, x_1), \ldots, (a_n, x_n)), (\rho_1, \ldots, \rho_n)) = (x_1 a_1 + \rho_1, \ldots, x_n a_n + \rho_n)$ is a DRE of f.*

6.3 DRE for Scalar Multiplication

Definition 10 (BN254 scalar multiplication [GKS+26, Definition 14]). *Let $h : (\mathbb{F}_q^* \times \{0,1\})^\ell \to \mathbb{G}$ be the function that, given $r \in \mathbb{F}_q^*$ and $\overline{\boldsymbol{u}} \in \{0,1\}^\ell$, computes the product $r\pi$:*

$$h((r, \overline{u}_1(\pi)), \ldots, (r, \overline{u}_\ell(\pi))) \longmapsto r\pi.$$

The function h can be expressed as a matrix multiplication, with a transition from field to group operations:

1. ***Field operations.*** *For each $i \in \{0, \ldots, n-1\}$, compute the three Jacobian coordinates of $r_i\pi$ using matrix multiplication over $\mathbb{F}_p$:*

$$\boldsymbol{N}(r_i) \times \overline{\boldsymbol{u}}^T(\pi),$$

   where $\boldsymbol{N}(r_i) \in \mathbb{F}_p^{3\times\ell}$ is the matrix defined in Lemma 2 and $r_0, \ldots, r_{n-1}$ is the bit decomposition of r, i.e., $\sum_{i=0}^{n-1} 2^i r_i = r$.
2. ***Group operations.*** *Apply the weighted group sum using "$+$" as the group operation*

$$\sum_{i=0}^{n-1} 2^i (r_i\pi) = r\pi,$$

   where each $r_i\pi$ is represented by its Jacobian coordinates (X, Y, Z) computed in Step 1 and the operation $\sum_{i=0}^{n-1} 2^i(\cdot)$ uses group operations (elliptic curve addition and scalar multiplication in $\mathbb{G}$).

For each i, the computation $\boldsymbol{N}(r_i) \times \overline{\boldsymbol{u}}^{\mathrm{T}}(\pi)$ yields the Jacobian coordinates of the group element $r_i\pi$. We obtain $r\pi$ from the Jacobian representations by combining them via the weighted sum $\sum_{i=0}^{n-1} 2^i(r_i\pi)$ using elliptic curve addition.

Theorem 1 (DRE of h [GKS+26])**.** *Let the function* $\hat{h} : (\mathbb{F}_q^* \times \{0,1\})^\ell \times (R, \Lambda, S) \to \mathbb{F}_p^{3n\ell}$ *be defined as*

$$\hat{h}((r, \overline{u}_1(\pi)), (r, \overline{u}_2(\pi)), \ldots, (r, \overline{u}_\ell(\pi)), R, \Lambda, S) = (\hat{h}_1(r, \overline{u}_1(\pi), R, \Lambda, S), \ldots, \hat{h}_\ell(r, \overline{u}_\ell(\pi), R, \Lambda, S)),$$

where $\hat{h}_k(r, \overline{u}_k(\pi), R, \Lambda, S) = \{D_{i,j,k} \cdot \overline{u}_k(\pi) + s_{i,j,k}\}$ *and* $\boldsymbol{D}_i := \mathsf{diag}(\lambda_i) \times \boldsymbol{D}(r_i, \rho_i)$*, for* $i \in \{0, \ldots, n-1\}$*,* $j \in \{1,2,3\}$*,* $k \in \{1, \ldots, 5n+1\}$*, and where* $D_{i,j,k}$ *denotes the* $(j,k)^{th}$ *entry in* $\boldsymbol{D}_i$*. The randomness sets* R, Λ, S *are generated subject to the following constraints:*

$$\begin{aligned} R &= \left\{(\rho_0, \ldots, \rho_{n-1}) \in (\mathbb{G} \setminus \{\mathcal{O}\})^n : \sum\nolimits_{i=0}^{n-1} 2^i \rho_i = 0\right\} \\ \Lambda &= \{(\lambda_0, \ldots, \lambda_{n-1}) \in (\mathbb{F}_p \setminus \{0\})^n\} \\ S &= \left\{(s_{i,j,k})_{i=0,j=1,k=1}^{n-1,3,\ell} \in \mathbb{F}_p^{3n\ell} : \forall (i,j), \sum\nolimits_{k=1}^{\ell} s_{i,j,k} = 0\right\} \end{aligned}$$

Then, $\hat{h}$ *is a DRE for* h*.*

The decoding function $\mathsf{Decode} : \mathbb{F}_p^{3n\ell} \to \mathbb{G}$ for $\hat{h}$ takes as input the values $\{t_{i,j,k}\}_{i,j,k}$, where $\{t_{i,j,k}\}_{i,j} := \hat{h}_k(r, \overline{u}_k(\pi), R, \Lambda, S)$ is either $\hat{h}_k(r, 0, R, \Lambda, S)$ or $\hat{h}_k(r, 1, R, \Lambda, S)$ (since $\overline{u}_k(\pi)$ is a bit), and is defined as follows:

1. For each $i \in \{0, \ldots, n-1\}$ and $j \in \{1,2,3\}$, compute the jth coordinate of the randomized Jacobian representation:

$$\hat{t}_{i,j} = \sum_{k=1}^{\ell} t_{i,j,k}.$$

   This quantity corresponds to the jth coordinate since $\sum_{k=1}^{\ell} s_{i,j,k} = 0$.
2. For each $i \in \{0, \ldots, n-1\}$, construct the point $Q_i = (\hat{t}_{i,1}, \hat{t}_{i,2}, \hat{t}_{i,3}) \in \mathbb{F}_p^3$. This represents the group element $\mathsf{diag}(\lambda_i) \times (r_i \pi + \rho_i)$.
3. Apply the weighted group sum to recover $r\pi$:

$$\sum_{i=0}^{n-1} 2^i Q_i = r\pi.$$

   This correctly computes $r\pi$ since $\sum_{i=0}^{n-1} 2^i \rho_i = 0$.

6.4 Garbled Circuit for Scalar Multiplication

Scheme 4 gives the complete construction, which largely follows [GKS+26]. Our construction diverges with respect to how the evaluator obtains its input labels. In BABE, the verifier posts the labels on-chain; in Mosaic [KTA+26], the Feldman shares are the labels. In our construction, we keep the labels seed-derived according to Definition 2 and use the shares as *keys* to them. The garbler supplies one *input key* $s_{j,b} \in \mathbb{F}_{q_s}$ per input wire j and polarity b to GSM.Garble. In step 2, the garbler encrypts $L_{j,b}$ under $s_{j,b}$ so every input wire maps to a two-row *input table*. An evaluator learns a key to exactly one entry of the input table, corresponding to the polarity of the input itself. They subsequently open that row to learn the underlying label, while the other entry remains hidden.

In our full protocol, these keys are Feldman shares (Section 3.3): the evaluator reconstructs them by interpolation, rather than receiving the input encoding directly from the garbler. Encrypting the labels under the shares (rather than using the shares directly as labels) is what allows the keys to be shares in $\mathbb{F}_{q_s}$ while the labels remain seed-derived. This is necessary since a label is a λ-bit string tied to its partner by the free-XOR offset, which a share cannot be (see footnote 5). Accordingly, Encode outputs active polarities and keys where BABE's Encode outputs labels. We include it for completeness only, as in our full protocol the prover assembles K_π from the shares revealed on-chain and subsequent interpolation (as in Mosaic).

As identified by [GKS+26, KBH26], the DRE does not guarantee that r remains hidden if the evaluator supplies a point that is not on E. The solution is to condition the output binary decomposition $\overline{\boldsymbol{u}}(\pi)$ of the boolean garbled circuit on whether π satisfies the curve equation, i.e., the circuit checks that $y(\pi)^2 \equiv x(\pi)^3 + 3 \pmod{p}$. The recent work of Khambhati et al. [KBH26] suggests incorporating the input validation into the DRE instead of using Yao-style garbling, which theoretically yields a lower ciphertext cost.

Scheme 4: Garbled Scalar Multiplication GSM

$\mathsf{Garble}(r, \mathsf{seed}, \{s_{j,b}\}_{j=0,b\in\{0,1\}}^{2n-1}) \to \mathsf{ct_{GC}}$:

1. **Privacy-free binary garbling.**
   1.1. Define the circuit C that embeds the following computation:
   - Compute $b := \left(y(\pi)^2 \stackrel{?}{\equiv} x(\pi)^3 + 3 \pmod p\right)$.
   - Construct $\boldsymbol{u}(\pi) = (1, x(\pi), y(\pi), x(\pi)^2, y(\pi)^2, x(\pi)y(\pi))$.
   - If $b = 1$, output $\overline{\boldsymbol{u}}(\pi) = \mathsf{bin}(\boldsymbol{u}(\pi))$. Otherwise, output $\perp$.[a]

   1.2. $\left(\mathsf{ct_C}, \{L_{j,b}\}_{j=0,b\in\{0,1\}}^{2n-1}\right) \leftarrow \mathsf{Garble}(1^\lambda, C, \mathsf{seed})$

   1.3. The output labels of the garbled circuit $\mathsf{ct_C}$ will be $L_{\overline{u},k}^{\overline{u}_k(\pi)}$ for $k \in \{1, \ldots, \ell\}$.
2. **Input table.** Encrypt each input label under its key, one row per polarity:[b]

$$\mathsf{ct_{in}} := \{\mathsf{ct}_{\mathsf{in},j,b}\}_{j=0,b\in\{0,1\}}^{2n-1} \leftarrow \{\mathsf{Enc}\left(H_{\mathsf{key}}(s_{j,b}),\, L_{j,b}\right)\}_{j=0,b\in\{0,1\}}^{2n-1}.$$

3. **DRE encryption.**
   3.1. For each bit $k \in \{1, \ldots, \ell\}$, the garbler precomputes $\hat{h}_k(r, 0, R, \Lambda, S)$ and $\hat{h}_k(r, 1, R, \Lambda, S)$
   3.2. Encrypt the DRE under the output labels, with respect to the polarity of $\overline{u}_k(\pi)$:

$$\mathsf{ct_{DRE}} := \left\{\mathsf{ct}_k^0, \mathsf{ct}_k^1\right\}_{k=1}^{\ell} \leftarrow \left\{\mathsf{Enc}(L_{\overline{u},k}^0, \hat{h}_k(r, 0, R, \Lambda, S)), \mathsf{Enc}(L_{\overline{u},k}^1, \hat{h}_k(r, 1, R, \Lambda, S))\right\}_{k=1}^{\ell}.$$

4. Output $\mathsf{ct_{GC}} := (\mathsf{ct_C}, \mathsf{ct_{in}}, \mathsf{ct_{DRE}})$.

$\mathsf{Encode}(\pi, \{s_{j,b}\}_{j=0,b\in\{0,1\}}^{2n-1}) \to K_\pi$: Output the active polarities and keys,

$$K_\pi = \big((x_0(\pi), s_{0,x_0(\pi)}), \ldots, (x_{n-1}(\pi), s_{n-1,x_{n-1}(\pi)}), (y_0(\pi), s_{n,y_0(\pi)}), \ldots, (y_{n-1}(\pi), s_{2n-1,y_{n-1}(\pi)})\big).$$

$\mathsf{Eval}(\mathsf{ct_{GC}}, K_\pi) \to r\pi$: write $K_\pi = \big((\pi_j, s_{j,\pi_j})\big)_{j=0}^{2n-1}$, where π_j denotes the j-th input bit of π in the order of Encode.

1. Open the active row of every input table, $L_{j,\pi_j} \leftarrow \mathsf{Dec}\left(H_{\mathsf{key}}(s_{j,\pi_j}), \mathsf{ct}_{\mathsf{in},j,\pi_j}\right)$ for $j = 0, \ldots, 2n-1$, and output $\perp$ if any decryption fails. The row is addressed by the polarity π_j. Let $L_\pi := (L_{0,\pi_0}, \ldots, L_{2n-1,\pi_{2n-1}})$.
2. $L_{\overline{\boldsymbol{u}}(\pi)} \leftarrow \mathsf{Eval}(\mathsf{ct_C}, L_\pi) = \left(L_{\overline{u},1}^{\overline{u}_1(\pi)}, \ldots, L_{\overline{u},\ell}^{\overline{u}_\ell(\pi)}\right)$.
3. $\hat{h}_k(r, \overline{u}_k(\pi), R, \Lambda, S) \leftarrow \mathsf{Dec}(L_{\overline{u},k}^{\overline{u}_k(\pi)}, \mathsf{ct}_k^{\overline{u}_k(\pi)})$
4. $r\pi \leftarrow \mathsf{Decode}(\left\{\hat{h}_k(r, \overline{u}_k(\pi), R, \Lambda, S)\right\})$.
5. Output $r\pi$.

[a] In BABE, if the inputted point π is not on E, then the circuit outputs the binary decomposition of $\boldsymbol{u}(g) = (1, x(g), y(g), x(g)^2, y(g)^2, x(g)y(g))$, where $g \in \mathbb{G}_1$ is some fixed point on the curve. Alternatively, we can explicitly output the bit b from the circuit and have the garbler encrypt the binary decomposition $\overline{\boldsymbol{u}}(\pi) = \mathsf{bin}(\boldsymbol{u}(\pi))$ under the "1" label L_b^1, i.e., $\mathsf{Enc}(L_b^1, \mathsf{ct_{DRE}})$. Therefore, the evaluator can successfully decrypt the remaining output labels if and only if π lies on E.

[b] Instantiated as a one-time pad with a zero tag: row (j, b) is $(L_{j,b} \,\|\, 0^\lambda) \oplus \mathsf{KS}(s_{j,b})$, where KS is an extendable-output hash of the share that plays the roles of H_{key} and of the keystream at once, and Dec accepts a row if and only if the unmasked tag equals 0^λ.

This scheme pushes heavy computation (group operations and algebra) to the prover and the verifier's offline preprocessing. Consequently, the online component is lightweight, consisting of one input-table decryption per input wire, basic arithmetic for binary decomposition and symmetric-key decryptions for recovering the corresponding DRE components.

6.5 DRE Optimizations

As identified in [GKS+26], several optimizations exist for the DRE with respect to the ciphertext size. We concretely specify them below.

Matrix sparsity The first optimization relates to exploiting the sparsity of the matrix $\boldsymbol{D}$, i.e., several entries are zero. The number of inner products need only to grow according to the number of nonzero entries. Eliminating these entries from the DRE shrinks the ciphertext by a factor of 1.87.

Random oracle model The second major optimization applies the random oracle (RO) assumption to the DRE, further shrinking the ciphertext footprint. For completeness, we describe how the DRE randomness set S is generated under the constraint that values across the k index sum to zero ("closure"). In other words, for all (i, j), $\sum_{k=1}^{\ell} s_{i,j,k} = 0$. This is trivially enforced by sampling the first $\ell - 1$ values $s_{i,j,k} \xleftarrow{\$} \mathbb{F}_p$ via a pseudorandom generator (PRG), and computing the final ℓ^{th} term directly from the sum, such that the constraint is satisfied:

$$s_{i,j,\ell} = -\sum_{k=1}^{\ell-1} s_{i,j,k}$$

Despite $s_{i,j,\ell}$ being computed deterministically (as opposed to generated pseudorandomly), we maintain computational indistinguishability since arithmetic is over $\mathbb{F}_p$.

Instead of encrypting and transmitting two ciphertexts $\hat{h}_k(r, 0, R, \Lambda, S)$ and $\hat{h}_k(r, 1, R, \Lambda, S)$, we encrypt a single $\hat{h}_k$ and define the other to be the output of a pseudorandom function on the corresponding label. Let $H : \{0,1\}^* \to \mathbb{F}_p$ be a domain-separated instance of the random oracle H (Section 3.1), mapping into $\mathbb{F}_p$. Observe, in the definition of the DRE, the $\overline{u}_k(\pi) = 0$ case directly maps to the randomness $s_{i,j,k}$ for all (i, j). Since these values are pseudorandom, they can be derived non-interactively by the prover (evaluator) according to the relation

$$s_{i,j,k} := H(L^0_{\overline{u},k}, i, j, k).$$

The question that remains is how to enforce the constraint that the summation over k of all $s_{i,j,k}$ is closed. Clearly, if all $s_{i,j,k}$ values are RO-derived, it is impossible to enforce closure. Therefore, we once again "reserve" a single s value at some index $k = \mathsf{idx} \in \{1, \ldots, \ell\}$ that is deterministically computed as

$$s_{i,j,\mathsf{idx}} := -\sum_{k \in \{1,\ldots,\ell\} \setminus \mathsf{idx}} H(L^0_{\overline{u},k}, i, j, k).$$

We can freely choose the idx as any of the $k \in \{1, \ldots, \ell\}$ values. However, it would require transmitting two ciphertexts at $k = \mathsf{idx}$ since $\hat{h}_{\mathsf{idx}}(r, 0, R, \Lambda, S) = s_{i,j,\mathsf{idx}}$ would be computed deterministically and not from the random oracle. This reduces the ciphertext size transmitted by the garbler by a factor of strictly less than 2.

We can circumvent this restriction and achieve an exact 2-fold improvement by exploiting the structure of the binary decomposition garbled circuit and its corresponding output labels $L^0_{\overline{u},k}, L^1_{\overline{u},k}$. Observe that for $k = 1$, the garbled circuit will *always* evaluate to the "1" label $L^1_{\overline{u},1}$, since we define $\overline{u}_1 = 1$. Therefore, we need not transmit ct^0_1 and set $\mathsf{idx} = 1$, enabling us to achieve an exact $2\times$ improvement.

Complexity Utilizing the matrix sparsity and RO optimizations above, the construction yields a theoretical DRE ciphertext size of 15.65 MiB [GKS+26]. In our implementation, we obtain a DRE ciphertext size of 15.8 MiB, which closely aligns with [GKS+26]'s implementation's concrete value of 15.77 MiB.

Remark 2 (Alternate curves)**.** *The DRE is concretely specified for points on BN254, however it is easy to see how to generalize this construction to an arbitrary pairing-friendly elliptic curve, such as one from the BLS family of curves. One would need to re-derive the matrix* $\boldsymbol{D}$ *from the curve equation. This would naturally impact our parameter selections in our main construction, but would affect neither the underlying functionality nor the correctness.*

7 The Lamport-Adaptor Gadget

In BABE [GKS+26], a verifier is only able to post a Lamport signature in ChallengeAssert that is on the same message (π_1) that the prover posted in Assert. This ensures that an honest prover can successfully evaluate the GC/DRE and decrypt the witness-encryption ciphertexts to prove its innocence (and thus post WronglyChallenged). In the malicious variant of BABE, the verifier must reveal the input labels selected by π_1 for each of the M_{CC} finalized candidates on-chain, undesirably increasing the cost by a factor of M_{CC}. BABE suggests using VSS as a remedy to reduce the number of distinct on-chain input-label sets from M_{CC} down to one, but does not provide a concrete transaction design that binds the committed shares to the on-chain protocol logic, suggesting it can be done via adaptor signatures and nonce commitments.

We introduce an on-chain gadget that realizes exactly this binding, enforcing honest verifier behavior during the proving phase. The technique is of potential independent interest for commit-reveal applications. The core idea is a *hybrid* Lamport-adaptor locking script that *branches* on the Lamport preimage of a committed bit, such that the spending party can complete a script-accepted spend only via the branch selected by the committed bit. Any confirmed spend thereby discloses a designated secret for that polarity.

We first present the construction for a single committed bit and an arbitrary spending transaction, then generalize it to a multi-bit commitment. The concrete instantiation in Tritone is presented alongside the protocol (Section 8.1.3).

7.1 Single-Bit Construction

Suppose we have two parties, a prover and verifier.[4] The prover holds a one-bit Lamport key pair $(\mathsf{lsk}_P, \mathsf{lpk}_P)$ (Scheme 1), whose per-polarity entries we write $(\mathsf{lsk}_P^\beta, \mathsf{lpk}_P^\beta)_{\beta\in\{0,1\}}$, together with a dedicated *reveal* key pair $(\mathsf{sk}_R, \mathsf{pk}_R = [\mathsf{sk}_R])$. The verifier holds an ordinary signing key pair $(\mathsf{sk}_V, \mathsf{pk}_V)$ and two secret scalars $(y_0, y_1) \in \mathbb{F}_{q_s}^2$. The prover holds commitments to the verifier's secret scalars $T_\beta = [y_\beta]$ (i.e., adaptor points). Our objective is to derive a mechanism ensuring that, given an on-chain commitment to a bit b by the prover, the verifier can post *only* a follow-up transaction that discloses y_b (and only y_b) to the prover.

The first component (the binding commitment to b) is realized by the prover Lamport-signing b on-chain (as in [GKS+26]), i.e., revealing the preimage $\mu = \mathsf{lsk}_P^b$ in some prior transaction. The second component consists of: (a) a transaction output whose leaf carries the predicate Φ defined below, and (b) an arbitrary, pre-agreed transaction tx that spends this output (without loss of generality as its single input, with index 0). The leaf contains one branch per polarity β, and the

[4] We use this naming for consistency with our scheme, though the roles need not correspond to a conventional prover–verifier relationship.

signatures checked in the two branches are over *distinct* messages for the same spending transaction. By BIP-342 [WNT20b], the signature message commits to the executed branch of the leaf, which lets us define the following per-polarity sighashes

$$\overline{\mathsf{tx}}^{\beta} := \mathsf{SigMsg}(\overline{\mathsf{tx}},\, 0,\, c_\beta),$$

where c_β is the position identifying the β branch within the leaf and SigMsg is as defined in Section 3.8. Beyond c_β, the message commits to the executed tapleaf and to the entire transaction. Therefore, a signature on $\overline{\mathsf{tx}}^{\beta}$ is consent to exactly the spend tx via branch β.

During Setup, the prover generates one incomplete adaptor signature per polarity and sends both to the verifier:

$$\tilde{\sigma}_\beta \leftarrow \mathsf{GenAdaptor}\big(\mathsf{sk}_R,\, T_\beta,\, \overline{\mathsf{tx}}^{\beta}\big), \qquad \beta \in \{0,1\}\,.$$

The verifier validates $\mathsf{VerAdaptor}(\mathsf{pk}_R, T_\beta, \overline{\mathsf{tx}}^{\beta}, \tilde{\sigma}_\beta) = 1$ for both polarities and aborts on failure. Since the check is against the points T_β (whose witnesses y_β it embeds), adaptability guarantees that each accepted pre-signature is completable with the corresponding y_β.

We are now equipped to define the predicate Φ that binds the verifier's completed signature to the polarity of the prover's Lamport-signed bit:

$$\Phi := \bigvee_{\beta\in\{0,1\}} \Big(\mathsf{HashLock}\big(\mathsf{lpk}_P^{\beta}\big) \wedge \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Verify}\big(\mathsf{pk}_R,\, \overline{\mathsf{tx}}^{\beta},\, \sigma\big)\Big) \wedge \mathsf{CheckSig}(\mathsf{pk}_V),$$

where $\mathsf{HashLock}\big(\mathsf{lpk}_P^{\beta}\big)$ functions as a *polarity gate*, $\mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Verify}\big(\mathsf{pk}_R,\, \overline{\mathsf{tx}}^{\beta},\, \sigma\big)$ is the *reveal check*, and we denote $\mathsf{CheckSig}(\mathsf{pk}_V)$ as the verifier's *tail signature*. The spending witness supplies (σ_V, σ, μ). The leaf is realized as a single script: it branches on $\mathsf{Hash}_{\mathrm{BTC}}(\mu)$ to select the polarity β, marks the branch position c_β (via $\mathsf{OP_CODESEPARATOR}$), verifies σ under pk_R (aborting the spend on failure), and concludes with the verifier tail.

Once the prover has committed b, the verifier reads μ from the prover's commitment transaction (fixing b via $\mathsf{Hash}_{\mathrm{BTC}}(\mu) = \mathsf{lpk}_P^{b}$), completes the corresponding pre-signature $\sigma = \mathsf{Adapt}(\tilde{\sigma}_b, y_b)$, and confirms tx with witness (σ_V, σ, μ). Therefore, an honest verifier can *always* assemble a valid spending witness regardless of the prover's behavior; confirmation further relies on the gadget output being unspent and on ledger liveness (made precise in Section 7.3). The tail signature σ_V is produced dynamically at spend time and is likewise on the executed branch's sighash $\overline{\mathsf{tx}}^{b}$. From the confirmed witness, the prover recovers $y_b = \mathsf{Extract}\big(\tilde{\sigma}_b, \sigma\big)$ as required.

The verifier tail signature is required because every other object in Φ is *prover-known*: lpk_P^{β} is the prover's own Lamport key, so the prover knows both preimages $(\mathsf{lsk}_P^0, \mathsf{lsk}_P^1)$ and holds the reveal key sk_R. Hence, a signature valid under pk_R need not be an adaptor completion: the prover can produce one by signing directly. Without the σ_V conjunct, Φ would be satisfiable by the prover alone over an *arbitrary* transaction, e.g., spending the gadget output immediately and rendering the pre-agreed tx un-confirmable.

Conversely, no prover-side tail signature is necessary. In any verifier-built witness, σ is a completion of a prover pre-signature over $\overline{\mathsf{tx}}^{\beta}$, which commits to the entire transaction and is therefore the prover's setup-time consent to exactly this transaction.

Scheme 5 formally specifies the construction, split into the setup and online phases.

Correctness. Suppose both parties are honest and the verifier accepted the setup. We check that the verifier's spend is accepted and that it hands the prover exactly y_b. In the online

phase, the prover reveals $\mu = \mathsf{lsk}_P^b$. The verifier recovers b from $\mathsf{Hash}_{\mathrm{BTC}}(\mu) = \mathsf{lpk}_P^b$, computes $\sigma = \mathsf{Adapt}(\tilde{\sigma}_b, y_b)$ and $\sigma_V = \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_V, \overline{\mathsf{tx}}^b)$, and posts tx with witness (σ_V, σ, μ). The witness passes every check of Φ through branch b: (a) the polarity gate of branch b opens, as $\mathsf{Hash}_{\mathrm{BTC}}(\mu) = \mathsf{lpk}_P^b$; (b) σ is a valid signature under pk_R on $\overline{\mathsf{tx}}^b$ by adaptability, since the verifier checked $\tilde{\sigma}_b$ against $T_b = [y_b]$ at setup; and (c) σ_V is the verifier's own signature under pk_V on the same sighash $\overline{\mathsf{tx}}^b$. Once tx confirms, the prover reads σ from its witness and computes $\mathsf{Extract}(\tilde{\sigma}_b, \sigma)$. Since $\sigma = \mathsf{Adapt}(\tilde{\sigma}_b, y_b)$, the output is exactly y_b by construction. Conversely, the other secret y_{1-b} is never revealed since the verifier can only complete $\tilde{\sigma}_b$. No message the verifier posts or sends depends on y_{1-b}, therefore the prover learns nothing about it beyond the point T_{1-b} it has held since setup.

We address the security of the construction against parties deviating from the protocol description separately in Section 7.3.

Scheme 5: Single-Bit Lamport-Adaptor Gadget

Setup:

Both: form the leaf Φ from $(\mathsf{lpk}_P, \mathsf{pk}_R, \mathsf{pk}_V)$ and agree on the skeletons of the commitment transaction $\overline{\mathsf{tx}}_{\mathsf{cmt}}$ and the spending transaction $\overline{\mathsf{tx}}$ (single input, index 0, leaf Φ), fixing the branch sighashes $\overline{\mathsf{tx}}^\beta = \mathsf{SigMsg}(\overline{\mathsf{tx}}, 0, c_\beta)$.

Prover$(\{T_\beta\}_{\beta\in\{0,1\}}, \mathsf{lsk}_P, \mathsf{lpk}_P, \mathsf{sk}_R, \mathsf{pk}_R)$:

1. For $\beta \in \{0,1\}$, compute $\tilde{\sigma}_\beta \leftarrow \mathsf{SASig}.\mathsf{GenAdaptor}(\mathsf{sk}_R, T_\beta, \overline{\mathsf{tx}}^\beta)$.
2. Send $(\tilde{\sigma}_0, \tilde{\sigma}_1)$ to the verifier.

Verifier$(\{y_\beta\}_{\beta\in\{0,1\}}, \mathsf{lpk}_P, \mathsf{pk}_R, \mathsf{sk}_V, \mathsf{pk}_V)$:

1. Upon receiving $(\tilde{\sigma}_0, \tilde{\sigma}_1)$: for $\beta \in \{0,1\}$, **abort** if $\mathsf{SASig}.\mathsf{VerAdaptor}(\mathsf{pk}_R, T_\beta, \overline{\mathsf{tx}}^\beta, \tilde{\sigma}_\beta) \neq 1$; otherwise store $(\tilde{\sigma}_0, \tilde{\sigma}_1)$.

Online:

Prover$(b, \mathsf{lsk}_P, \{\tilde{\sigma}_\beta\})$:

1. Compute $\mu \leftarrow \mathsf{Lamp}.\mathsf{Sign}(b, \mathsf{lsk}_P) = \mathsf{lsk}_P^b$.
2. Post $\mathsf{tx}_{\mathsf{cmt}}$ with μ in its witness.
3. Upon confirmation of tx: read σ from its witness and output $y_b \leftarrow \mathsf{SASig}.\mathsf{Extract}(\tilde{\sigma}_b, \sigma)$.

Verifier$(\{y_\beta\}, \mathsf{sk}_V, \mathsf{lpk}_P, \{\tilde{\sigma}_\beta\})$:

1. Upon confirmation of $\mathsf{tx}_{\mathsf{cmt}}$: read μ from its witness and recover b via $\mathsf{Hash}_{\mathrm{BTC}}(\mu) = \mathsf{lpk}_P^b$.
2. Compute $\sigma \leftarrow \mathsf{SASig}.\mathsf{Adapt}(\tilde{\sigma}_b, y_b)$ and $\sigma_V \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_V, \overline{\mathsf{tx}}^b)$.
3. Post tx with witness (σ_V, σ, μ).

7.2 Generalization to Multi-Bit Messages

Now, suppose the prover commits to an ℓ-bit message $m = (m_1, \ldots, m_\ell) \in \{0,1\}^\ell$ under a Lamport key $\mathsf{lpk}_P = \left\{\mathsf{lpk}_t^\beta\right\}_{t\in[\ell],\,\beta\in\{0,1\}}$, and let the verifier hold one secret per bit and polarity, $y_{t,\beta}$, with corresponding adaptor points $T_{t,\beta} = [y_{t,\beta}]$. The leaf simply concatenates ℓ single-bit blocks: the (t,β) branch is marked by its own position $c_{t,\beta}$, yielding per-bit, per-polarity sighashes

$$\overline{\mathsf{tx}}^{t,\beta} := \mathsf{SigMsg}(\overline{\mathsf{tx}},\, 0,\, c_{t,\beta}).$$

During Setup, the prover accordingly pre-signs all 2ℓ messages, $\tilde{\sigma}_{t,\beta} \leftarrow \mathsf{GenAdaptor}(\mathsf{sk}_R, T_{t,\beta}, \overline{\mathsf{tx}}^{t,\beta})$. With the per-bit predicate

$$\phi_t := \bigvee_{\beta\in\{0,1\}} \Big(\mathsf{HashLock}\big(\mathsf{lpk}_t^{\beta}\big) \wedge \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Verify}\big(\mathsf{pk}_R,\, \overline{\mathsf{tx}}^{t,\beta},\, \sigma_t\big)\Big),$$

we define the locking script

$$\mathsf{CheckLampAdaptorMatch}\big(\mathsf{lpk}_P,\, \{T_{t,\beta}\}_{t,\beta}\,,\, \mathsf{pk}_R\big) := \bigwedge_{t=1}^{\ell} \phi_t.$$

The points $T_{t,\beta}$ never appear in the script itself, rather they parameterize the accompanying pre-signatures $\tilde{\sigma}_{t,\beta}$, which are what bind the spend to them. The full leaf is

$$\Phi := \mathsf{CheckLampAdaptorMatch}\big(\mathsf{lpk}_P,\, \{T_{t,\beta}\}_{t,\beta}\,,\, \mathsf{pk}_R\big) \wedge \mathsf{CheckSig}(\mathsf{pk}_V).$$

A *single* verifier tail per transaction suffices to enforce verifier authorization of the transaction. Its sighash also commits to the last executed branch position, but does not commit to the interior branch positions or authenticate all witness components. The signature checked by the tail is over the last executed branch position, $\overline{\mathsf{tx}}^{\mathsf{last}} := \mathsf{SigMsg}(\overline{\mathsf{tx}},\, 0,\, c_{\ell,\beta_\ell})$, where β_t denotes the polarity of the branch executed for bit t. Note, the tail binds the transaction and its last executed branch, but not the interior branch positions: only the prover (which holds sk_R and both preimages of every bit) could substitute the per-bit components of a broadcast witness at unchanged transaction ID. Such a substitution changes neither the transaction nor the verifier's authorization of it, and publishes both preimages of every bit whose executed polarity is altered; extraction, a guarantee for the honest prover (Section 7.3), is unaffected.

Once the prover has published $(\mu_1, \ldots, \mu_\ell) = \mathsf{Lamp}.\mathsf{Sign}(m, \mathsf{lsk}_P)$, the verifier completes $\sigma_t = \mathsf{Adapt}(\tilde{\sigma}_{t,m_t}, y_{t,m_t})$ for every t and spends with witness $(\sigma_V, (\sigma_t, \mu_t)_{t\in[\ell]})$. The spend is rejected unless every per-bit signature verifies, and the prover extracts the disclosed secrets $\{y_{t,m_t}\}_t$ via $\mathsf{Extract}$.

Note, a single transaction may not accommodate all ℓ bits (e.g., due to transaction size and standardness limits). In this case, we partition $[\ell] = \bigcup_{j=0}^{K-1} R_j$ into K contiguous ranges and dedicate one gadget output and pre-agreed spending transaction tx_j per range. The jth output is

$$\Phi_j := \mathsf{CheckLampAdaptorMatch}\big(\mathsf{lpk}_P^{(R_j)},\, \{T_{t,\beta}\}_{t\in R_j,\,\beta\in\{0,1\}}\,,\, \mathsf{pk}_R\big) \wedge \mathsf{CheckSig}(\mathsf{pk}_V),$$

where $\mathsf{lpk}_P^{(R_j)}$ is the Lamport key restricted to R_j and the sighashes in ϕ_t, $t \in R_j$, are on $\overline{\mathsf{tx}}_j$. The full message is then bound by the conjunction across the K transactions,

$$\Phi_m := \bigwedge_{j=0}^{K-1} \Phi_j.$$

Correctness. The single-bit argument applies to each bit on its own. Suppose both parties are honest and the verifier accepted all 2ℓ pre-signatures. After the prover reveals $(\mu_1, \ldots, \mu_\ell)$, the verifier recovers every m_t from $\mathsf{Hash}_{\mathrm{BTC}}(\mu_t) = \mathsf{lpk}_t^{m_t}$ and, for each range j, posts tx_j with witness $(\sigma_V, (\sigma_t, \mu_t)_{t\in R_j})$, where $\sigma_t = \mathsf{Adapt}(\tilde{\sigma}_{t,m_t}, y_{t,m_t})$ and σ_V is its signature on $\overline{\mathsf{tx}}_j^{\mathsf{last}}$. Every block ϕ_t, $t \in R_j$, accepts through branch m_t: μ_t opens its polarity gate, and σ_t is a valid signature under pk_R on the sighash of branch (t, m_t) by adaptability. The tail accepts as in the single-bit case. From the confirmed tx_j, the prover extracts $y_{t,m_t} = \mathsf{Extract}(\tilde{\sigma}_{t,m_t}, \sigma_t)$ for every $t \in R_j$. Once all K transactions confirm, the prover holds exactly the ℓ secrets $\{y_{t,m_t}\}_{t=1}^{\ell}$ selected by m, and the verifier has never used the complementary secrets $\{y_{t,1-m_t}\}_{t=1}^{\ell}$.

7.3 Security

We now state the guarantees the gadget provides when a party deviates from the protocol. An *execution* consists of the Setup and Online phases of Scheme 5, and a *spend* of the gadget output is any confirmed transaction whose witness (σ_V, σ, μ) satisfies Φ. A setup is *accepting* if the verifier's VerAdaptor checks pass. The verifier *tail-authorizes* a spend by producing the tail signature for its (transaction, branch) pair (in the multi-bit construction, the transaction and its last executed branch); the term refers to this pair alone, not to the remaining witness components.

Model and Assumptions The lemmas below hold in an idealized script model and under the following conventions and assumptions:

(M1) *Script semantics.* The gadget output is spendable only by a witness satisfying Φ, and each signature check in Φ is $\mathsf{Sig}_{\mathrm{BTC}}$.Verify over the BIP-342 message SigMsg (Section 3.8).

(M2) *Message injectivity.* SigMsg is modeled as injective in the skeleton, the spent outpoint, the executed tapleaf, and the codeseparator position. In particular $\overline{\mathsf{tx}}^0 \neq \overline{\mathsf{tx}}^1$, and the sighashes are pairwise distinct across bits, ranges, and parallel executions of the scheme (distinct OP_CODESEPARATOR positions and spent outpoints by construction).

(M3) *Reveal-key discipline.* The *honest* prover uses sk_R *exclusively* in SASig.GenAdaptor, with a fresh nonce r' per pre-signature, and *never* to produce an ordinary signature (Section 8.4). Otherwise, two pre-signatures sharing a nonce on distinct messages reveal sk_R by the standard Schnorr nonce-reuse computation. The constraint is on the honest algorithms only (hence on the reduction of Lemma 11, which plays the honest prover); a malicious prover, which holds sk_R, is unconstrained (Section 7.1).

(M4) *Assumptions.* We assume the hardness of discrete logarithms in the random oracle model, under which SASig (Scheme 3, whose parity normalization preserves both notions, Section 3.4) satisfies pre-signature adaptability (perfectly) and *full extractability* in the presence of pre-signing and signing oracles: except with negligible probability, any valid signature not produced by the signing oracle discloses the witness of a pre-signature issued on its message. Specifically, a valid signature on a never-pre-signed message is a forgery. The notion is due to [DOY22], which subsumes the unforgeability and single-instance witness extractability of [AEE+21], and is satisfied by Schnorr adaptor signatures in the ROM [GSST24]. We additionally assume EUF-CMA security of $\mathsf{Sig}_{\mathrm{BTC}}$ and preimage resistance of $\mathsf{Hash}_{\mathrm{BTC}}$.

We adopt the standard security game notation. For a game G against a primitive Π, we write $\mathbf{Adv}^{\mathsf{G}}_{\Pi,\mathcal{B}}(\lambda) := \Pr[\mathcal{B} \text{ wins } \mathsf{G}]$ for the advantage of an algorithm $\mathcal{B}$ (a reduction when it runs another adversary $\mathcal{A}$ internally). We abbreviate the games of (M4) as fExt (full extractability, granting a pre-signing oracle pSign and a signing oracle Sign under the game's key; the adversary wins by outputting a valid signature, not produced by Sign, that discloses the witness of no pre-signature issued on its message), EUF (EUF-CMA, granting Sign; the adversary wins with a valid signature on a never-queried message), and OW (preimage inversion: given $z = \mathsf{Hash}_{\mathrm{BTC}}(x)$ for uniform $x \xleftarrow{\$} \{0,1\}^\lambda$, output a preimage of z). All per-execution quantities (ℓ, K, the number of pre-signatures) are $\mathsf{poly}(\lambda)$. By (M4), each advantage is negligible for PPT $\mathcal{B}$.

The case of both parties being honest is covered by the correctness arguments of Sections 7.1 and 7.2. Here, we consider one-sided corruption. When the prover is honest and the verifier may be malicious, we prove extraction (Lemma 11) and polarity binding (Lemma 12(ii)): every confirmed spend discloses a witness of the adaptor point of each executed branch, and every executed branch corresponds to a Lamport preimage the prover has revealed. When the verifier is honest and the prover may be malicious, we prove spendability (Lemma 10), exclusivity (Lemma 12(i)), and complementary secrecy (Lemma 13): the verifier can always assemble a script-accepted spend,

every confirmed spend is tail-authorized by the verifier, and the verifier's messages depend on the inactive secret only through its adaptor point. In this latter setting, we do not claim that the verifier generated every component of a confirmed witness (the tail commits to the transaction and its executed branch, not to the remaining witness elements), and extraction is claimed only for an honest prover. No security claim is made when both parties are corrupted.

Lemma 10 (Spendability)**.** *Suppose the verifier accepted the setup. Then, for every (possibly malicious) prover strategy: upon observing any preimage μ with $\mathsf{Hash}_{\mathrm{BTC}}(\mu) = \mathsf{lpk}_P^b$ in its ledger view, the honest verifier computes a witness (σ_V, σ, μ) satisfying Φ for $\overline{\mathsf{tx}}$ via branch b. The claim is unconditional.*

Proof. By the (perfect) pre-signature adaptability property of SASig, we have

$$\mathsf{VerAdaptor}(\mathsf{pk}_R, T_b, \overline{\mathsf{tx}}^b, \tilde{\sigma}_b) = 1 \wedge [y_b] = T_b \implies \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Verify}\big(\mathsf{pk}_R, \overline{\mathsf{tx}}^b, \mathsf{Adapt}(\tilde{\sigma}_b, y_b)\big) = 1.$$

The left-hand side holds after an accepting setup, and the verifier holds y_b. With $\sigma := \mathsf{Adapt}(\tilde{\sigma}_b, y_b)$, $\sigma_V \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_V, \overline{\mathsf{tx}}^b)$, and μ opening the polarity gate of branch b, the witness (σ_V, σ, μ) satisfies Φ. □

Lemma 11 (Extraction)**.** *Suppose the prover is honest and let $\mathcal{A}$ be a PPT adversary controlling the verifier. If a spend (σ_V, σ, μ) of the gadget output via branch β is confirmed, then $y := \mathsf{SASig}.\mathsf{Extract}(\tilde{\sigma}_\beta, \sigma)$ satisfies $[y] = T_\beta$, except with probability $\mathsf{negl}(\lambda)$.*

Proof. First, we define the failure event

$$\mathsf{Fail}: \quad \text{a spend } (\sigma_V, \sigma, \mu) \text{ via branch } \beta \text{ is confirmed} \wedge [\mathsf{SASig}.\mathsf{Extract}(\tilde{\sigma}_\beta, \sigma)] \neq T_\beta.$$

In the multi-bit setting, Fail instead requires some executed pair (t, β_t) with $[\mathsf{SASig}.\mathsf{Extract}(\tilde{\sigma}_{t,\beta_t}, \sigma_t)] \neq T_{t,\beta_t}$. Next, we construct the reduction $\mathcal{B}^{\mathsf{pSign},\mathsf{Sign}}(\mathsf{pk})$ against fExt:

1. Set $\mathsf{pk}_R := \mathsf{pk}$ and run $\mathcal{A}$, playing the honest prover.
2. For each polarity $\gamma \in \{0,1\}$ (each pair $\gamma = (t, \beta) \in [\ell] \times \{0,1\}$ in the multi-bit setting), obtain $\tilde{\sigma}_\gamma \leftarrow \mathsf{pSign}(\overline{\mathsf{tx}}^\gamma, T_\gamma)$, where T_γ is the prover's setup input and the messages $\overline{\mathsf{tx}}^\gamma$, fixed by the skeletons, are pairwise distinct by (M2).
3. Upon a confirmed spend, let m' be the sighash of its executed branch (of a failing pair, in the multi-bit setting) and σ the signature its witness supplies for that branch; output (m', σ).

The simulation is perfect: pSign is GenAdaptor with fresh nonces (M3), no other prover action depends on sk_R, and Sign is never queried.

By (M1), σ is a valid signature on m'. If the spending transaction is the pre-agreed tx, then $m' = \overline{\mathsf{tx}}^\beta$, which determines the unique pre-signature $(\tilde{\sigma}_\beta, T_\beta)$ issued on it by (M2), and failing extraction is precisely a win. Otherwise, m' differs from every pre-signed message by (M2), and $\mathcal{B}$ wins outright. Hence Fail implies that $\mathcal{B}$ wins:

$$\Pr[\mathsf{Fail}] \leq \mathbf{Adv}^{\mathsf{fExt}}_{\mathsf{SASig},\mathcal{B}}(\lambda) \leq \mathsf{negl}(\lambda),$$

with no loss over the pre-signatures issued under one reveal key (Section 8.4). As discrete logarithms are unique, when the verifier is honest the extracted y equals its secret y_β. □

Lemma 12 (Exclusivity and polarity binding)**.** *For every PPT adversary $\mathcal{A}$, except with probability $\mathsf{negl}(\lambda)$:*

(i) ($\mathcal{A}$ controls the prover.) No gadget-output spend not tail-authorized by the honest verifier is confirmed.

(ii) ($\mathcal{A}$ controls the verifier.) No confirmed spend executes a branch whose preimage the honest prover has not revealed; in particular, once the prover has revealed the single preimage $\mu = \mathsf{lsk}_P^b$, every confirmed spend executes the branch of polarity b.

Proof. (i) We construct the reduction $\mathcal{B}^{\mathsf{Sign}}(\mathsf{pk})$ against EUF as follows:

1. Set $\mathsf{pk}_V := \mathsf{pk}$; simulate the honest verifier, answering each signature it issues under this key (the tail on $\overline{\mathsf{tx}}^b$ and any signatures on other outputs) with Sign, recording the query set Q.
2. Upon a confirmed spend not tail-authorized by the verifier, whose witness contains (by (M1)) a valid σ'_V on the sighash m' of its executed branch, output (m', σ'_V).

By (M2), $m' \notin Q$: queries for outputs other than the gadget's differ in the spent outpoint, and every gadget-output query is the tail sighash of a tail-authorized pair, valid for no other (transaction, branch) pair. Hence, (m', σ'_V) is a forgery, and

$$\Pr\left[\text{confirmed spend not tail-authorized}\right] \leq \mathbf{Adv}^{\mathsf{EUF}}_{\mathsf{Sig}_{\mathrm{BTC}},\mathcal{B}}(\lambda) \leq \mathsf{negl}(\lambda).$$

(ii) We construct the reduction $\mathcal{B}(z)$ against OW, on challenge $z = \mathsf{Hash}_{\mathrm{BTC}}(x)$ with $x \xleftarrow{\$} \{0,1\}^\lambda$:

1. Sample $\gamma^* \xleftarrow{\$} \{0,1\}$ (respectively $\gamma^* \xleftarrow{\$} [2\ell]$ in the multi-bit setting); set $\mathsf{lpk}_P^{\gamma^*} := z$, sample the remaining Lamport secrets, and simulate the honest prover.
2. Abort if the committed message selects γ^*, as simulating the reveal requires the unknown preimage of z.
3. Upon a confirmed spend executing branch γ^*, whose witness contains by (M1) a value μ' with $\mathsf{Hash}_{\mathrm{BTC}}(\mu') = z$, output μ'.

The entry $\mathsf{lpk}_P^{\gamma^*} = z$ is distributed identically to an honestly generated one, so the guess γ^* is independent of $\mathcal{A}$'s view, and therefore

$$\Pr\left[\text{spend via an unrevealed polarity confirmed}\right] \leq 2 \cdot \mathbf{Adv}^{\mathsf{OW}}_{\mathsf{Hash}_{\mathrm{BTC}},\mathcal{B}}(\lambda) \leq \mathsf{negl}(\lambda)$$

(replacing 2 with 2ℓ in the multi-bit setting). □

Combining Lemma 10 and Lemma 12(i) with ledger liveness (Section 3.8), the honest verifier's spend not only exists but confirms, except with probability $\mathsf{negl}(\lambda)$.

Lemma 13 (Complementary secrecy)**.** *Suppose (T_0, T_1) and b are fixed. All messages exchanged during the scheme execution (the pre-signatures $(\tilde{\sigma}_0, \tilde{\sigma}_1)$ and the spend witness (σ_V, σ, μ)) are a (randomized) function of the Setup-fixed skeletons (hence the sighashes $\overline{\mathsf{tx}}^\beta$), the parties' keys and nonces, y_b, and (T_0, T_1) alone. In particular, the prover's view depends on y_{1-b} only through T_{1-b}.*

Proof. By inspection. Every element of the execution is a function of the arguments displayed:

$$\tilde{\sigma}_\beta = \left(\varepsilon_\beta r'_\beta + e_\beta \cdot \mathsf{sk}_R,\ R'_\beta,\ \varepsilon_\beta\right), \quad e_\beta = H\left(R'_\beta + T_\beta,\ \mathsf{pk}_R,\ \overline{\mathsf{tx}}^\beta\right), \quad \beta \in \{0,1\},$$
$$\sigma = \left(s'_b + \varepsilon_b y_b,\ \varepsilon_b(R'_b + T_b)\right), \quad \sigma_V \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_V, \overline{\mathsf{tx}}^b), \quad \mu = \mathsf{lsk}_P^b.$$

Here $\varepsilon_\beta = \varepsilon(R'_\beta + T_\beta)$ is a function of R'_β and T_β (Scheme 3). All terms are independent of y_{1-b}, as it only enters the protocol view via T_{1-b}. □

Lemma 13 says the gadget *itself* discloses exactly y_b. In our protocol, the hiding of the inactive zero-share $y_{1-b} = f_{1-b}(0)$, given the published Feldman commitments and the d_{CC} shares opened during cut-and-choose, therefore reduces to the (computational, discrete-logarithm) secrecy of VSS (Section 3.3).

Corollary 2 (Multi-bit, partitioned variant)**.** *Lemmas 10–13 extend to the ℓ-bit construction of Section 7.2 and its K-range partition as follows. The spendability property holds per range, upon observing the preimages of every $t \in R_j$. The extraction and polarity binding properties hold per bit t (extraction incurs no security loss, as the full-extractability game is multi-query; the guessing step in Lemma 12(ii) ranges over the 2ℓ Lamport key entries, costing a factor 2ℓ).*

A single *tail signature per range transaction suffices for exclusivity, its sighash committing to the entire transaction though not to the interior branch positions. Their substitution is available only to the prover and changes neither the transaction nor the verifier's tail authorization; extraction (Lemma 11) is a guarantee for the honest prover and is unaffected (Section 7.2).*

Lastly, secrecy holds jointly for all inactive polarities $\{y_{t,1-m_t}\}_t$.

8 The TRITONE Protocol

We are now equipped with the necessary tools to present our main bridge construction, which can be described as a maliciously secure N-party generalization of BABE. As standard, the protocol is partitioned into the setup phase (or "peg-in"), where a user's funds are locked under the operator committee, and the proving phase (or "peg-out"), where an operator claims them.

We distinguish the roles of a *user* U and an *operator*. A user is a party who wishes to bridge their Bitcoin to another chain. An operator is a party responsible for facilitating the bridge protocol. Recall the operator set $\mathcal{O} := \{O_1, \ldots, O_N\}$ of all operators participating in the bridge protocol (Section 4). During peg-out, operators front the user's bitcoin out-of-pocket and are subsequently reimbursed by the protocol. Additionally, our protocol allows operators to simultaneously function as users. In this case, no funds are fronted.

The relation $\mathcal{R}$ embeds the underlying peg-out behavior. The initialization algorithm $\mathsf{Gen}(\mathcal{R}) \to \mathsf{crs}$ directly invokes $\mathsf{Groth16.Gen}(\mathcal{R})$ and returns the relation-specific common reference string crs (Section 4).

The public parameters for our system pp is the tuple

$$\mathsf{pp} = (N, B, k_C, N_{\mathsf{CC}}, M_{\mathsf{CC}}, \Delta_{\mathsf{refund}}, \Delta_{\mathsf{withdrawOpt}}, \Delta_{\mathsf{withdraw}}, \Delta_{\mathsf{assert}}, \Delta_{\mathsf{noWithdraw}}, d_{\mathsf{dust}}).$$

These parameters are implicit to all protocol algorithms. Table 1 summarizes the individual parameters, together with the derived quantities n, d_{CC}, and κ, and the two bounds τ_{eval} and τ_{setup} used in the analysis (Section 9.2). These values are fixed at bridge construction and are static throughout the protocol execution.

8.1 Setup Phase

The setup phase is presented in Protocol 1 and is performed at the *committee* level by the entire operator set $\mathcal{O}$ and at the *pairwise* level between operators O_{i} and O_{j}, for $\mathsf{i}, \mathsf{j} \in [N], \mathsf{i} \neq \mathsf{j}$. It proceeds in three steps: committee-level key generation and covenant emulation (Section 8.1.1), the pairwise cut-and-choose setup over the WE-DRE candidates (Section 8.1.2), and the Lamport-adaptor setup with pre-signature exchange (Section 8.1.3).

8.1.1 Committee-level Setup

Operators first generate all the needed key pairs and broadcast their public keys to each other. Each operator O_{i} also generates the public statement $\boldsymbol{x}_{\mathsf{i}}$ for the instances in which it acts as prover-candidate and broadcasts it alongside its keys. The user then locks their funds under the aggregated identity key by posting $\mathsf{DepositRequest}$. Upon its confirmation, each operator broadcasts a funding

Table 1: Protocol parameters.

N	number of operators, $N = \lvert\mathcal{O}\rvert$
B	pegged deposit value (in BTC)
d_{dust}	dust denomination of the connector outputs: the minimum acceptable value of an output, set as a parameter in a validator node
n	bit length of a BN254 coordinate ($n = 254$, Section 6.1)
k_C	number of ChallengeAssert chunks/bit ranges $R_0, \ldots, R_{k_C-1}$ (we use k_C=2)
N_{CC}	cut-and-choose candidates per dispute instance
M_{CC}	finalized (unopened) candidates, $\lvert\mathcal{I}\rvert = M_{\mathsf{CC}}$
d_{CC}	degree of the Feldman share polynomials: $d_{\mathsf{CC}} = N_{\mathsf{CC}} - M_{\mathsf{CC}}$
κ	statistical security parameter: $\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}^{-1} \leq 2^{-\kappa}$ (we use $\kappa = 40$)
Δ_{refund}	timelock after which the user reclaims an unforwarded deposit (DepositRefund)
$\Delta_{\mathsf{withdrawOpt}}$	optimistic-withdrawal veto window (WithdrawOptimistic)
$\Delta_{\mathsf{withdraw}}$	delay before Withdraw after Claim (challenge window)
Δ_{assert}	prover's window to post Assert after ChallengeClaim; afterward the verifier may post NoAssert
$\Delta_{\mathsf{noWithdraw}}$	prover's window to post WronglyChallenged after the ChallengeAssert chunks; afterward the verifier may post NoWithdraw
τ_{eval}	bound, in blocks, on the honest prover's off-chain work (interpolation, evaluation, decryption) between the last ChallengeAssert chunk and WronglyChallenged (Section 9.2)
τ_{setup}	bound, in blocks, on the duration of an all-honest Protocol 1 from DepositRequest confirming to the broadcast of Deposit (Section 9.2)

outpoint u_{O_i} that funds the dust denominations of its subgraph. The connector outputs of a subgraph carry only multiples of the dust denomination d_{dust} (Table 1), totaling $d_{\mathsf{dust}} \cdot (k_C(N-1)+2)$. Operators abort if any broadcast outpoint fails to cover this amount.

Parties then engage pairwise in the cut-and-choose setup (Section 8.1.2). Once these setups conclude, each operator creates the transaction skeletons of its view of the graph and engages pairwise in the Lamport-adaptor setup and pre-signature exchange (Section 8.1.3). Recall, we emulate covenants on Bitcoin via multi-signatures (Section 3.8). Operators subsequently engage in the interactive MuSig2 signing protocol, first to obtain the covenant signatures on WithdrawOptimistic and Withdraw and, after verifying them and deleting their covenant key shares (so that no covenant signature beyond those exchanged during setup can ever be created), to obtain the identity signature on Deposit. The identity key is used only to lock DepositRequest, which Deposit spends through its key path. Finally, any operator can post the Deposit transaction, forwarding the user's funds under the covenant. The order matters: an operator releases its partial identity signature only once its own copy of the pre-signed graph is complete, and from that point on the outcome of the ceremony is decided by the ledger, which either confirms Deposit (the setup completes) or lets the user reclaim the deposit (Section 9.4).

If any party aborts or times out before the identity signing, the procedure terminates, and the user can submit a DepositRefund transaction to retrieve their funds after Δ_{refund} elapses. A

party that has already released its partial identity signature instead awaits the ledger's verdict, as described above.

8.1.2 Pairwise Cut-and-Choose Setup

The second step of setup is the *pairwise* setup, performed between every (ordered) pair of operators. Each operator functions as both a prover and verifier to all counterparties. Recall the set of ordered dispute instances $\mathcal{D}$ from Section 4, with O_{p} the prover-candidate and O_{v} the verifier of instance (p, v). For example, if we have four operators, we would have

$$\mathcal{D} = \left\{ \begin{array}{cccccc} (1,2) & (1,3) & (1,4) & (2,1) & (2,3) & (2,4) \\ (3,1) & (3,2) & (3,4) & (4,1) & (4,2) & (4,3) \end{array} \right\}.$$

To achieve malicious security, we utilize cut-and-choose (Section 3.7). To optimize the on-chain footprint, we incorporate VSS (first suggested by [Eag25] and later formalized in [KTA+26]) into our protocol. For each input wire $t \in [2n]$ and bit value $\beta \in \{0, 1\}$, the garbler (verifier) samples a uniformly random polynomial $f_{t,\beta}$ of degree at most d_{CC}. The evaluations are the candidates' *input keys*: candidate i is garbled with keys $s_{t-1,\beta} := f_{t,\beta}(i)$, so GSM.Garble encrypts its seed-derived input labels under these shares, one table row per wire and polarity (Scheme 4), and a prover that later interpolates $f_{t,\pi_{1,t}}(i)$ opens exactly row $\pi_{1,t}$ of candidate i's table for wire t.[5] After cut-and-choose, the prover possesses d_{CC} verified shares of every polynomial, which is one share short of the reconstruction threshold, so the prover cannot open a single row of a finalized candidate's tables, and hence cannot evaluate it, until the corresponding zero-shares are disclosed on-chain. Since the prover samples the finalized set $\mathcal{I}$ uniformly at random only after receiving the verifier's commitments, a cheating verifier survives the opening only if its corrupted candidates all lie in $\mathcal{I}$, and corrupts *every* finalized candidate only if that set is exactly $\mathcal{I}$, which occurs with probability at most $\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}^{-1} \leq 2^{-\kappa}$, where $\kappa = 40$ for our parameter choices. The zero-share disclosure and the adaptor pre-signature exchange over these shares are described in Section 8.1.3.

The bound covers deviations that are *local* to a candidate: a malformed msg_i, $\mathsf{ct}_{\mathsf{setup},i}$, or $\mathsf{ct}_{\mathsf{GC},i}$ is exposed as soon as candidate i is opened. The Feldman polynomials are not local, as every candidate draws its input keys from the same $4n$ polynomials, and a verifier can deviate in them without corrupting any single candidate. The input tables of Scheme 4 are analyzed under the invariant that every key opens exactly one row, i.e., that the evaluations $f_{t,\beta}(i)$ at the candidate points are pairwise distinct across all (t, β); the reveal relies on the same property at point 0, where distinct zero-shares keep the two polynomials of a wire apart. The cut cannot enforce this. Two polynomials of degree at most d_{CC} may agree on up to d_{CC} of the N_{CC} evaluation points: for any $S \subseteq [N_{\mathsf{CC}}]$ with $|S| \leq d_{\mathsf{CC}}$ and any $c \in \mathbb{F}_{q_s}^*$, the pair

$$f_{t,1} := f_{t,0} - c \prod_{i \in S} (X - i)$$

[5] The shares cannot serve as the labels themselves. Labels are λ-bit strings while shares lie in $\mathbb{F}_{q_s}$, and the garbling we instantiate uses free-XOR, which fixes $L_{t,1} = L_{t,0} \oplus \Omega$ for a circuit-global secret offset Ω: an XOR relation that polynomial evaluations cannot carry across candidates and that, were it embedded, the shares retained from cut-and-choose would make interpolatable. Encrypting seed-derived labels under the shares keeps the garbling free-XOR compatible and keeps Ω out of the prover's reach. A prover holding both labels of a single wire would recover Ω, hence every label of the candidate, and could evaluate it on arbitrary inputs. That is a decryption oracle for the witness encryption: querying f_r on $\pi = [\alpha]_1$ and $\pi = X$ yields $rY = e(r[\alpha]_1, [\beta]_2) + e(rX, [\gamma]_2)$ and thus $\mathsf{msg} = \mathsf{ct}_{\mathsf{msg}} - H(rY)$ without a valid (π_1, π_2, π_3). This is why the one-label-per-wire guarantee of the tables is necessary (see the distinctness check below).

respects the degree bound and has distinct zero-shares $f_{t,0}(0) \neq f_{t,1}(0)$, yet $f_{t,0}(i) = f_{t,1}(i)$ for every $i \in S$, so both rows of wire t in every candidate $i \in S$ are encrypted under one key. Both VSS instances are individually well-formed, every opened share passes the Feldman check, and re-garbling an opened candidate $i \in S$ from its seed and keys reproduces the same table, so the opening detects nothing, while S may hold all but M_{CC} candidates. Evaluation is unaffected, since GSM.Eval addresses the row by the revealed polarity, so completeness does not rest on the invariant. The one-label-per-wire guarantee does: one interpolated key now opens both rows of wire t, and under the one-time-pad instantiation the two rows XOR to $L_{t,0} \oplus L_{t,1} = \Omega_i$, the free-XOR offset of candidate i, readable from the delivered table without any key and yielding, together with the active labels, every label pair of the candidate. A verifier inflicts this leak only on itself, and an honest verifier's polynomials are collision-free with overwhelming probability, so neither completeness nor knowledge soundness rests on catching it. The check nevertheless turns the invariant from an assumption about the verifier into a property of every commitment the prover accepts, at the cost of one set-membership pass over public data. Concretely, the prover verifies, before sampling $\mathcal{I}$, that the $4n(N_{\mathsf{CC}} + 1)$ share commitments $C_{t,\beta}(i)$ over all $t \in [2n]$, $\beta \in \{0, 1\}$ and $i \in \{0\} \cup [N_{\mathsf{CC}}]$ are pairwise distinct (Protocol 2). The check reads only the public commitments, so it covers finalized and opened candidates alike, and it subsumes the distinctness of the adaptor points $T_{t,\beta} = C_{t,\beta}(0)$, without which the reveal of one zero-share would complete both polynomials of a wire. It penalizes an honest verifier only with negligible probability: for distinct indices $(t, \beta, i) \neq (t', \beta', i')$, the difference $f_{t,\beta}(i) - f_{t',\beta'}(i')$ is a nonzero linear function of uniformly random coefficients, hence uniform in $\mathbb{F}_{q_s}$, and a union bound over all pairs gives a failure probability of at most $\binom{4n(N_{\mathsf{CC}}+1)}{2}/q_s < 2^{-220}$.

We can now describe our construction in detail. Each cut-and-choose *candidate* $i \in [N_{\mathsf{CC}}]$ consists of the following quantities:

- Witness encryption: a randomly sampled msg_i, random secret scalar r_i, and partial ciphertext $\mathsf{ct}_{\mathsf{setup},i}$.
- VSS polynomials $f_{t,\beta}$, evaluated at i, and their commitments $C_{t,\beta}$.
- DRE for garbled scalar multiplication $\mathsf{ct}_{\mathsf{GC},i}$, whose input tables are encrypted under the shares at i.

The verifier O_{v} commits to all of these quantities and sends these commitments to the prover O_{p}. The prover then samples a random subset $\mathcal{I} \subseteq [N_{\mathsf{CC}}]$ of size M_{CC} to remain unopened (the finalized candidates) and sends the finalized set $\mathcal{I}$ to the verifier. The verifier opens the commitments in $[N_{\mathsf{CC}}] \setminus \mathcal{I}$ by sending the messages, secret random scalars, garbling seeds, and secret shares to the prover. The verifier additionally sends the ciphertexts of the unopened candidates to the prover, i.e., $\mathsf{ct}_{\mathsf{setup},i}, \mathsf{ct}_{\mathsf{GC},i}$ for $i \in \mathcal{I}$.

Upon receiving the opened candidates' secrets and the unopened ciphertexts, the prover verifies these values against the commitments received in the prior round. The prover stores the ciphertexts of the unopened candidates, the shares of the opened candidates, the adaptor points $T_{t,\beta}$, and the witness encryption commitments $h_{\mathsf{msg},i}$ for $i \in \mathcal{I}$. We denote $\Upsilon_{(\mathsf{p},\mathsf{v})} := \left(\left\{ T_{(\mathsf{p},\mathsf{v}),t,\beta} \right\}_{t \in [2n], \beta \in \{0,1\}}, \left\{ h_{\mathsf{msg},(\mathsf{p},\mathsf{v}),i} \right\}_{i \in \mathcal{I}_{(\mathsf{p},\mathsf{v})}} \right)$ as the *commitment record* for the pair (p, v).

The commitment record is never broadcast beyond the pair, as no third party consumes it. Each operator constructs its own view of the transaction set (Protocol 6) from the broadcast keys and funding outpoints together with the records of the instances it participates in. The committee and optimistic regions of the graph depend only on the keys and funding outpoints, so they are identical across all operators. Their agreement is enforced by the MuSig2 ceremony itself: divergent views produce different covenant sighashes, causing the signing to fail and the setup to abort. The dispute region of instance (p, v) is needed only by O_{p} and O_{v}, who both hold $\Upsilon_{(\mathsf{p},\mathsf{v})}$ from their

pairwise channel. An operator's security against any other pair rests entirely on its own instances.

8.1.3 Lamport-Adaptor Setup and Pre-signature Exchange

Using the adaptor points $T_{t,\beta}$ from the first pairwise setup, each prover–verifier pair engages in the multi-bit Lamport-adaptor setup of Section 7. We instantiate the gadget once per dispute instance (p,v), with prover O_p and verifier O_v, and omit the instance subscript where unambiguous.

The committed message is the proof element $\pi_1 \in \mathbb{G}_1$, whose two affine coordinates are n bits each, such that $\ell = 2n$. The prover Lamport-signs π_1 in Assert under its key $\mathsf{lpk}_{O_\mathsf{p}}$, making the preimages $\mu_t = (\mathsf{lsk}_{O_\mathsf{p}})_t^{\pi_{1,t}}$ public. The bits are partitioned into $K = k_C$ chunk ranges $R_0, \ldots, R_{k_C-1}$ ($\bigcup_j R_j = [2n]$), where k_C is the protocol's chunking parameter, and the jth gadget output is the match connector $\mathsf{Assert}[j]$, carrying leaf Φ_j. The pre-agreed spending transaction of range j is the chunk transaction $\mathsf{ChallengeAssert}^j$, which consumes this connector as its single input (index 0). We write $j(t)$ for the chunk containing bit t.

The verifier's secrets are the zero-shares $y_{t,\beta} = f_{t,\beta}(0)$ of the VSS polynomials $f_{t,\beta}$ dealt during the cut-and-choose setup (Section 8.1.2), whose share $f_{t,\beta}(i)$ keys the row of candidate i's input table for bit t and polarity β (Scheme 4). The adaptor points coincide with the coefficient-0 Feldman commitments published during cut-and-choose:

$$T_{t,\beta} = \big[f_{t,\beta}(0)\big] = C_{t,\beta}(0),$$

where $C_{t,\beta}(0)$ is the homomorphic evaluation of the commitment vector $C_{t,\beta}$ at 0, i.e., the coefficient-0 commitment (Scheme 2). The binding therefore requires no additional setup data, as the adaptor point is the published commitment. Unlike in BABE, the spend need not additionally open hash commitments to the garbled circuit input labels. The binding proceeds in two steps: $T_{t,\beta}$ binds the zero-share and, through the Feldman commitment $C_{t,\beta}$, every key $f_{t,\beta}(i)$; the table commitment $H_i = \mathsf{CommitTable}(\mathsf{ct}_{\mathsf{GC},i})$ binds the row that this key opens. For an opened candidate the prover re-garbles from the seed and keys and checks the result against H_i, and the cut extends this consistency to the finalized candidates (Section 8.1.2), so the revealed zero-share determines the finalized labels without further on-chain data. Concretely, for $t \in R_j$ and $\beta \in \{0,1\}$, the sighashes and pre-signatures generated during Setup are

$$\begin{aligned}\overline{\mathsf{tx}}^{t,\beta}_{\mathsf{ChA}^{j(t)}} &:= \mathsf{SigMsg}\big(\overline{\mathsf{tx}}_{\mathsf{ChA}^{j(t)}},\, 0,\, c_{t,\beta}\big),\\ \tilde{\sigma}_{t,\beta} &\leftarrow \mathsf{GenAdaptor}\big(\mathsf{sk}_{R,\mathsf{p}},\, T_{t,\beta},\, \overline{\mathsf{tx}}^{t,\beta}_{\mathsf{ChA}^{j(t)}}\big),\end{aligned}$$

where $\overline{\mathsf{tx}}_{\mathsf{ChA}^j}$ is the skeleton of $\mathsf{ChallengeAssert}^j$ (whose executed tapleaf is Φ_j), and the jth match-connector leaf is

$$\Phi_j = \mathsf{CheckLampAdaptorMatch}\big(\mathsf{lpk}_{O_\mathsf{p}}^{(R_j)},\, \{T_{t,\beta}\}_{t\in R_j,\,\beta\in\{0,1\}},\, \mathsf{pk}_{R,\mathsf{p}}\big) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{v}}).$$

By the general guarantee of Section 7.2, each confirmed $\mathsf{ChallengeAssert}^j$ is verifier-authorized and reveals the active zero-shares $\{f_{t,\pi_{1,t}}(0)\}_{t\in R_j}$, each provably the Feldman-committed value. Across all k_C chunks the prover obtains the full set $\{f_{t,\pi_{1,t}}(0)\}_{t=1}^{2n}$, which, together with the d_{CC} shares per polynomial opened during cut-and-choose, determines each $f_{t,\pi_{1,t}}$. The prover interpolates the polynomials, evaluates them at the finalized indices to recover the active keys of all M_{CC} finalized candidates, and decrypts the active labels from their input tables off-chain (Scheme 4), enabling them to evaluate the GC/DRE. Conversely, an honest verifier can always post $\mathsf{ChallengeAssert}^j$, regardless of the prover's behavior.

Additionally, the prover and verifier exchange a set of pre-signed transactions on the transaction skeletons: the prover pre-signs ChallengeOptimistic, ChallengeClaim, NoAssert, and NoWithdraw, and the verifier pre-signs Assert.[6] The pre-signatures and fixed skeletons satisfy the following properties [GKS+26]:

- *Unilateral posting.* The prover can, in particular, post WithdrawOptimistic, Withdraw, and Assert without relying upon other operators signing the transaction. Similarly, the verifier can post ChallengeOptimistic, ChallengeClaim, NoAssert, and NoWithdraw without relying upon the prover. The remaining transactions require only their broadcaster's own signatures and setup material already exchanged (ChallengeAssert^j: adaptor completions; WronglyChallenged: a hashlock preimage).
- *Output binding.* The deposited value B resides only in the covenant-locked Deposit[0], so it can move only via the covenant-signed WithdrawOptimistic and Withdraw fixed at setup. All remaining outputs carry only dust denominations. The prover therefore cannot divert the deposit outside the pre-agreed upon graph, as they possess neither the committee's nor the individual verifiers' signatures on any other value-carrying spend.
- *Input binding.* Every skeleton in $\mathcal{T}$ commits to its parents' transaction ids through its inputs, so a transaction can confirm only after its designated predecessors. In particular, Withdraw binds to the specific Claim and Deposit, and WronglyChallenged binds to the specific ChallengeAssert^j chunks. Unlike in BABE, Withdraw does not bind to Assert: the prover is instead prevented from withdrawing without asserting by the Claim[0] slash leaf (NoAssert, Section 8.2.2).

The set $\mathcal{T}$ contains the prescribed transaction skeletons created during setup (Protocol 6). Each operator's view of $\mathcal{T}$ covers every transaction it signs or posts.

8.2 Proving Phase

When a user initiates a withdrawal, an operator fronts the requested amount to the user. Later, the operator attempts to reclaim these funds from the protocol by demonstrating possession of a valid proof attesting that the funds were transferred to the user and that the corresponding wrapped BTC tokens were burned on the destination chain. The destination-chain flow (the issuance and burning of wrapped tokens, and how a withdrawal request reaches an operator) is out of scope for the bridge protocol. The proving phase proceeds along one of two paths: the *optimistic path* or the *challenge path*.

8.2.1 Optimistic Path

Utilizing the approach suggested by [WAA+26b], we explicitly incorporate an optimistic withdrawal case (the "happy path") into the construction. This enables efficient withdrawals in the common case. The proving operator first uploads the full proof $\pi = (\pi_1, \pi_2, \pi_3)$ to a centralized "proof repository",[7] then posts ClaimOptimistic, opening the $\Delta_{\mathsf{withdrawOpt}}$ veto window. If the operator goes unchallenged within this window, they can post the WithdrawOptimistic transaction and receive the claimed Bitcoin.

8.2.2 Challenge Path

Conversely, *any* operator can veto the withdrawal. Every operator retrieves π from the proof repository and runs $\mathsf{Groth16.Verify}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \pi)$. If verification fails, or no proof is available, the

[6] Recall, WithdrawOptimistic and Withdraw are covered by the covenant signatures obtained in the committee-level MuSig2 ceremony (Section 8.1.1).

[7] We assume the existence of such a repository and leave its instantiation out of scope.

operator posts the pre-signed $\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$ transaction, which spends $\mathsf{ClaimOptimistic}[0]$ via its verifier-specific leaf, halting the optimistic withdrawal path. This transitions the protocol into the challenge path (the "unhappy path").

Denote $O_\mathsf{v} \in \mathcal{O} \setminus \{O_\mathsf{p}\}$ as one challenging verifier (though all remaining $N-1$ operators are eligible to simultaneously challenge O_p). The challenge path utilizes the following transactions:

1. Claim: posted by the prover upon observing $\mathsf{ChallengeOptimistic}$, and now the only path to $\mathsf{Withdraw}$. The transaction carries the slash chokepoint at output 0, which starts the Δ_withdraw clock, along with $N-1$ challenge outputs, one per operator eligible to challenge.
2. $\mathsf{ChallengeClaim}$: posted by a challenging verifier, consuming exactly one of these outputs so that each verifier can challenge the prover exactly once, and compelling the prover to respond within Δ_assert.
3. Assert: posted by the prover within Δ_assert of $\mathsf{ChallengeClaim}$, revealing the proof element π_1 via Lamport signature.
4. $\mathsf{NoAssert}$: posted by the challenging verifier if the prover fails to respond within Δ_assert. It spends the $\mathsf{Claim}[0]$ slash leaf, permanently preventing $\mathsf{Withdraw}$.
5. $\mathsf{ChallengeAssert}$: posted by the challenging verifier as k_C chunk transactions $\mathsf{ChallengeAssert}^j$, containing the completed adaptor signatures selected by the individual bits of π_1 (Section 8.1.3). Confirming all k_C chunks reveals the active zero-shares to the prover and starts the $\Delta_\mathsf{noWithdraw}$ response window.

If the prover is honest (i.e., they possess a valid proof):

6. $\mathsf{WronglyChallenged}$: posted by the prover within $\Delta_\mathsf{noWithdraw}$ after interpolating the share polynomials, evaluating all unopened garbled circuit candidates to obtain $r_i\pi_1, i \in \mathcal{I}$, and decrypting the corresponding witness encryption ciphertexts. Decryption yields a message msg_i, the preimage of the committed hash $h_{\mathsf{msg},i}$ that unlocks this transaction's hashlock.[8] A valid preimage from any one unopened candidate is sufficient to post this transaction.
7. $\mathsf{Withdraw}$: posted by the prover once Δ_withdraw blocks have elapsed since Claim and the $\mathsf{Claim}[0]$ chokepoint remains unspent, withdrawing the bitcoin. The soundness requirement stipulates $\Delta_\mathsf{withdraw} > \Delta_\mathsf{assert} + \Delta_\mathsf{noWithdraw}$, plus "slack" for transaction confirmation, such that a full challenge round ($\mathsf{ChallengeClaim}$, the prover's Δ_assert response window, the $\mathsf{ChallengeAssert}$ chunks, and the $\Delta_\mathsf{noWithdraw}$ window) always fits inside the withdrawal window opened by Claim.

If the prover is malicious (i.e., they do not possess a valid proof):

8. $\mathsf{NoWithdraw}$: posted by the verifier after $\Delta_\mathsf{noWithdraw}$ blocks have elapsed since the $\mathsf{ChallengeAssert}$ chunks without a $\mathsf{WronglyChallenged}$ response, as the prover is unable to produce a valid preimage of any $h_{\mathsf{msg},i}$. It spends the $\mathsf{Claim}[0]$ slash leaf together with all k_C $\mathsf{ChallengeAssert}^j[0]$ outputs, permanently preventing $\mathsf{Withdraw}$.

All $N-1$ dispute instances interact with $\mathsf{Withdraw}$ through the single chokepoint output $\mathsf{Claim}[0]$: any single slash-leaf spend ($\mathsf{NoAssert}$ or $\mathsf{NoWithdraw}$) permanently blocks it. Conversely, if no slash leaf is ever spent, the prover withdraws via the Δ_withdraw path. In particular, a verifier that posts only a subset of the k_C $\mathsf{ChallengeAssert}^j$ chunks slashes nothing: $\mathsf{NoWithdraw}$ consumes all k_C $\mathsf{ChallengeAssert}^j[0]$ outputs, so the chokepoint is never spent and the prover withdraws without needing to respond to the partial challenge. The dust of Claim's unconsumed challenge outputs remains reclaimable by the prover via their timelocked leaves (Appendix A).

[8] The implementation sets $h_{\mathsf{msg},i} = \mathsf{Hash}_{\mathrm{BTC}}(\mathsf{SHA256}(\mathsf{msg}_i))$ and reveals $\mathsf{SHA256}(\mathsf{msg}_i)$, which keeps each input witness at 32 bytes in place of the 384-byte msg_i.

8.3 Transaction Graph

Figure 1 depicts the transaction graph for our construction, generated by $\mathsf{CreateTxSet}_{\mathsf{mp}}$ (Protocol 6). We divide it into three regions corresponding to peg-in (one per deposit, Figure 1a), optimistic peg-out (one per operator O_{p}, Figure 1b), and dispute (one per instance $(\mathsf{p}, \mathsf{v}) \in \mathcal{D}$, Figure 1c). Boxes represent transactions, their left (respectively, right) stubs are inputs (respectively, outputs), and edges are connectors labeled by their spending conditions. Stubs are unlabeled except for the deposit value B and the P2A anchor. A connector is identified by its position in the source transaction, as $\mathsf{X}[i]$ (Section 3.8). Dashed edges mark connectors that continue in another region. A detailed breakdown of the individual transactions is given in Appendix A. The sweep outputs of the terminal transactions are omitted from the figure; Appendix A lists them. The symbol ⟐ indicates that a transaction output can be used as an input by any one out of multiple transactions. Arrows entering a transaction are colored based on who can post the transaction (blue for the proving operator, red for a verifying operator, and green for a user). We additionally color $\mathsf{CheckSig}(\mathsf{pk}_O)$ to indicate that the transaction taking that input was pre-signed during setup (same scheme as the arrows, with purple corresponding to the covenant).

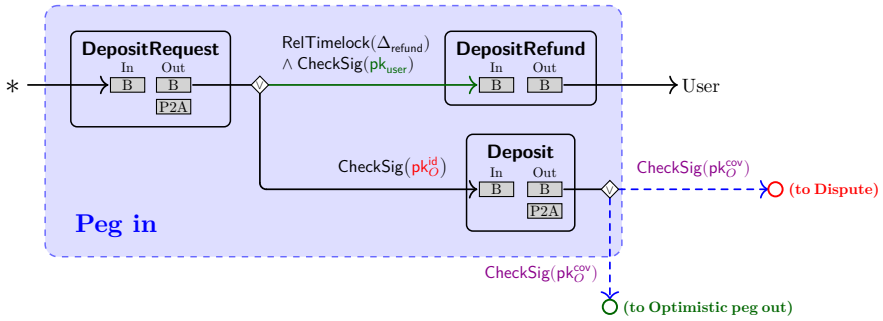
Every transaction additionally carries a pay-to-anchor (P2A) output, used to attach all fees at broadcast time (Section 8.4).

8.4 On-Chain Optimizations

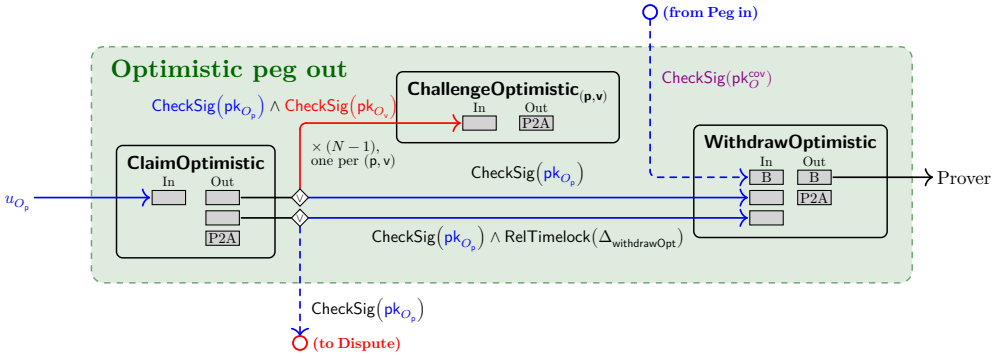
We leverage the following on-chain optimizations.

Transaction Versioning In our construction, we specify that all transactions are version 3 (Topologically Restricted Until Confirmation, or "TRUC") [Zha24]. This choice is dictated by the fee model, rather than for stylistic reasons. The dispute and covenant transactions are pre-signed during setup, and a pre-signature commits to the complete transaction body. Specifically, it commits to the difference between input and output values (i.e., the fee). The fee market prevailing at dispute time (potentially years later) is unknowable at setup, and a fixed fee that proves too low can never be adjusted: an underpaying slash or refutation transaction that fails to confirm is a liveness failure of the dispute process itself. We therefore build every protocol transaction to pay exactly *zero* fees (i.e., its outputs conserve the full input value) and defer the entire fee to broadcast time, when the broadcaster attaches a child spending the zero-value P2A anchor and submits the pair as a one-parent-one-child (1P1C) package (Section 8.3). Version 3 is what makes this construction relayable, which means that under current relay policy [Bit24], a zero-fee parent carrying a zero-value anchor output propagates only as a TRUC 1P1C package. The identical structure at version 2 is nonstandard and will not relay.

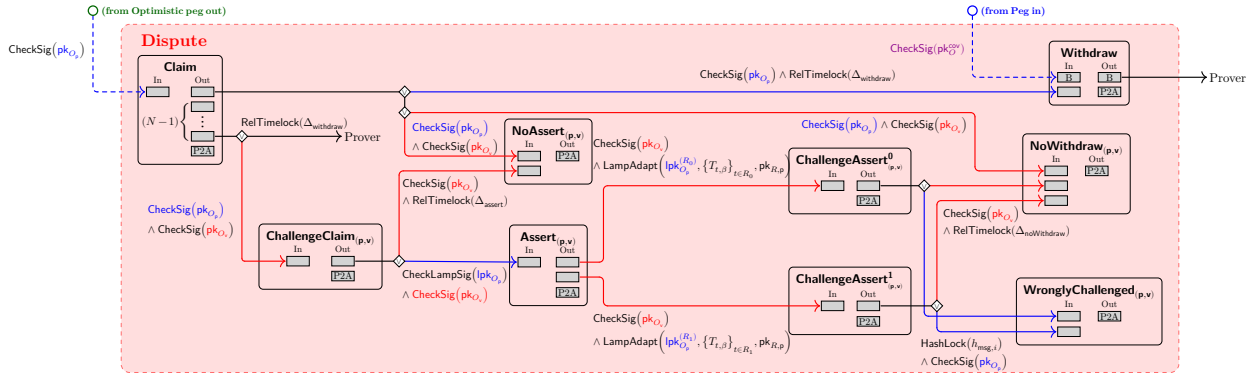
Additionally, TRUC supplies pinning resistance, which we require for our adversarial broadcast setting. The terminal transactions race for shared outputs ($\mathsf{NoWithdraw}$ against $\mathsf{Withdraw}$ for the $\mathsf{Claim}[0]$ chokepoint, and $\mathsf{WronglyChallenged}$ against $\mathsf{NoWithdraw}$ for the $\mathsf{ChallengeAssert}^j[0]$ outputs) such that an adversary able to "pin" an honest party's pre-signed transaction in the mempool at an uncompetitive feerate until a timelock expires would win the corresponding race outright. TRUC exactly forbids the graph topologies this type of attack exploits: an unconfirmed version-3 transaction may have at most one unconfirmed ancestor and one unconfirmed descendant of at most 1,000 vB, is always replaceable, and a competing anchor child at a higher feerate evicts a cheaper sibling. Because the P2A anchor is keyless, *any* party (not only the broadcaster) can attach the fee-paying child. An adversarial pin is thus limited to a single small child that any honest party can displace, in contrast to the far larger low-feerate descendant sets admissible under



(a) Peg-in region: U locks B into DepositRequest[0]. The committee forwards it to the covenant-locked Deposit[0], or U reclaims via DepositRefund after Δ_{refund}.



(b) Optimistic peg-out region for operator O_{p}: ClaimOptimistic^{p} opens the $\Delta_{\text{withdrawOpt}}$ veto window. An unchallenged, WithdrawOptimistic^{p} pays out B. Any of the $N-1$ pre-signed ChallengeOptimistic$_{(\text{p},\text{v})}$ forces the non-optimistic path via Claim^{p}.



(c) Dispute region for instance (p, v). ChallengeClaim forces Assert (else NoAssert slashes). The k_C ChallengeAssert^j chunks force WronglyChallenged (else NoWithdraw slashes via the Claim^{p}[0] chokepoint). The script label LampAdapt abbreviates CheckLampAdaptorMatch.

Figure 1: The TRITONE transaction graph, divided into its three regions.

version-2 policy.

These guarantees introduce two standardness restrictions, both of which determine the shape of the transaction graph. First, the size of a TRUC transaction is capped at 10,000 vB. A single ChallengeAssert completing the adaptor signatures for all $\ell = 508$ bits of π_1 would occupy 17.7 kvB. Our solution is to *split* it into $k_C = 2$ transactions $\mathsf{ChallengeAssert}^j$ of one 254-bit chunk each (8.9 kvB), whose outputs reconverge as inputs of the subsequent NoWithdraw and WronglyChallenged.[9] Second, the one-unconfirmed-ancestor rule forbids chains of unconfirmed protocol transactions, so each hop of a dispute must confirm before the next hop broadcasts. The k_C chunks of a single hop spend distinct connectors and carry their own anchors. The chunks are broadcast in parallel, and the confirmation depth of a dispute is therefore independent of k_C.

However, splitting ChallengeAssert introduces the possibility of a *partial challenge*, in which a (potentially malicious) verifier posts only a proper subset of the k_C chunks. A partial challenge slashes nothing and cannot be refuted: both NoWithdraw and WronglyChallenged consume *every* $\mathsf{ChallengeAssert}^j[0]$ output (via its timeout and hashlock leaves, respectively), such that while any chunk is absent from the ledger, neither transaction can be assembled. An honest prover therefore ignores a partial challenge: it lets $\Delta_{\mathsf{withdraw}}$ elapse and posts Withdraw, remaining answerable to the other operators' challenges in the interim. An honest verifier prevented from completing the challenge due to extenuating circumstances (such as hardware or network failures) loses nothing immediately since the pre-signed chunks remain valid while their parent outputs $\mathsf{Assert}[j]$ are unspent. However, to retain its ability to slash the prover, the verifier must post the remaining chunks early enough that the $\Delta_{\mathsf{noWithdraw}}$ window they open closes before Withdraw matures.

Lamport Commitment Hash Algorithm A followup optimization motivated by minimizing the transaction footprint relates to the hash function used for Lamport signatures. We instantiate the Lamport commitment hash $\mathsf{Hash}_{\mathrm{BTC}}$ as $\mathsf{HASH160} = \mathsf{RIPEMD160} \circ \mathsf{SHA256}$ rather than solely utilizing SHA256 (Section 3.8). A Lamport verification carries every bit's commitment in the leaf script and its revealed preimage in the witness, so the on-chain cost scales with the size of the commitment. Switching from 32- to 20-byte commitments shrinks the ℓ-bit Assert reveal from 12.1 kvB to 8.7 kvB.

Furthermore, this optimization is necessary because all Bitcoin transactions are version 3, as using SHA256 would cause the reveal to exceed the 10,000 vB TRUC cap. We would subsequently need to split Assert across multiple transactions in the same manner as ChallengeAssert. This would ultimately duplicate the $\mathsf{ChallengeClaim}[0]/\mathsf{NoAssert}$ machinery per chunk and further enlarge the dust budget. Under HASH160 the full ℓ-bit commitment to π_1 is revealed in a single transaction. Additionally, the same commitments appear in each match leaf Φ_j, further shrinking the $\mathsf{ChallengeAssert}^j$ chunks as well. Crucially, security of our system is unaffected by this optimization: a one-time Lamport signature needs only preimage resistance of the commitment hash. The smaller collisions (2^{80} versus 2^{128} in SHA256) confer neither forgery nor equivocation because the signer knows both preimages of every bit by construction, and that the polarity of a revealed preimage is determined by which committed hash it matches.

Key Reuse Across Instances Our last optimization relates to the key pairs each operator generates. An operator O_{p} is the prover-candidate of $N - 1$ dispute instances (p, v). Instantiating the gadget of Section 8.1.3 independently per instance would give each instance its own Lamport key and reveal keypair. Instead, we let O_{p} use a *single* Lamport key $\mathsf{lpk}_{O_{\mathsf{p}}}$ (with its 2ℓ preimages) and a single reveal keypair $(\mathsf{sk}_{R,\mathsf{p}}, \mathsf{pk}_{R,\mathsf{p}})$ across all of its instances. This key material (and the setup

[9] Assert, at 8.7 kvB under 20-byte HASH160 Lamport commitments, fits within the cap unsplit.

state derived from it) is per-operator rather than per-instance. Reusing the one-time signature key here is sound because one-time-ness constrains *messages*, rather than the broadcasts: the committed message is the same π_1 in every instance, an honest prover therefore never signs two distinct messages, and posting several Assert transactions merely republishes the same preimages. A prover that equivocates across instances by Lamport-signing different proof elements toward different verifiers publishes both preimages of some bits.

This weakens no security guarantees, as each instance checks the preimages revealed in its own Assert against the committed hashes, and each dispute's security argument is local to its pair. Reusing the reveal key is sound because every pre-signature under it binds to a distinct per-instance, per-bit, per-polarity sighash, such that a completed signature cannot be replayed across instances. However, the key must remain *specific to adaptor reveals*, i.e., O_{p} never issues an ordinary signature under $\mathsf{pk}_{R,\mathsf{p}}$ on any $\mathsf{ChallengeAssert}^j$ sighash. Any signature appearing on-chain under $\mathsf{pk}_{R,\mathsf{p}}$ is the completion of an adaptor pre-signature and necessarily reveals its zero-share (Section 7).

8.5 Protocol Algorithms

Protocol 1: Committee setup and peg-in. Each operator $O_{\mathsf{i}} \in \mathcal{O}$ runs this protocol concurrently.

procedure $\mathsf{Setup}_{\mathcal{O}}(\mathsf{crs})$**:**

$\left((\mathsf{sk}_{O_{\mathsf{i}}}, \mathsf{pk}_{O_{\mathsf{i}}}), (\mathsf{sk}^{\mathsf{id}}_{O_{\mathsf{i}}}, \mathsf{pk}^{\mathsf{id}}_{O_{\mathsf{i}}}), (\mathsf{sk}^{\mathsf{cov}}_{O_{\mathsf{i}}}, \mathsf{pk}^{\mathsf{cov}}_{O_{\mathsf{i}}}), (\mathsf{lsk}_{O_{\mathsf{i}}}, \mathsf{lpk}_{O_{\mathsf{i}}}), (\mathsf{sk}_{R,\mathsf{i}}, \mathsf{pk}_{R,\mathsf{i}})\right) \leftarrow \mathsf{GenAllKeys}(1^{\lambda}, \mathsf{i})$

$\boldsymbol{x}_{\mathsf{i}} \leftarrow \mathsf{GenStmt}(\mathsf{i})$ ▷ *application-specific public statement, O_{i} is the prover-candidate*

broadcast $(\mathsf{pk}_{O_{\mathsf{i}}}, \mathsf{pk}^{\mathsf{id}}_{O_{\mathsf{i}}}, \mathsf{pk}^{\mathsf{cov}}_{O_{\mathsf{i}}}, \mathsf{lpk}_{O_{\mathsf{i}}}, \mathsf{pk}_{R,\mathsf{i}}, \boldsymbol{x}_{\mathsf{i}})$ **to** all $O_{\mathsf{j}} \in \mathcal{O}, \mathsf{j} \neq \mathsf{i}$

upon *receiving all key material* **do**

$\mathsf{pk}^{\mathsf{id}}_{\mathcal{O}} \leftarrow \mathsf{MuSig2.KeyAgg}(\left\{\mathsf{pk}^{\mathsf{id}}_{O_{\mathsf{j}}}\right\}_{\mathsf{j}\in[N]})$

$\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}} \leftarrow \mathsf{MuSig2.KeyAgg}(\left\{\mathsf{pk}^{\mathsf{cov}}_{O_{\mathsf{j}}}\right\}_{\mathsf{j}\in[N]})$

User: authenticate $\mathsf{pk}^{\mathsf{id}}_{\mathcal{O}}$, send pk_U to $\mathcal{O}$, then call $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{DepositRequest})$

upon *observing* DepositRequest *confirmed at reorg-safe depth in* $\mathcal{L}$ **do**

verify DepositRequest[0] has the prescribed script for pk_U

broadcast funding outpoint $u_{O_{\mathsf{i}}}$ **to** all $O_{\mathsf{j}} \in \mathcal{O}, \mathsf{j} \neq \mathsf{i}$

upon *receiving* $u_{O_{\mathsf{j}}}$ *from all* $\mathsf{j} \neq \mathsf{i}$ **do**

verify $u_{O_{\mathsf{j}}}$ is confirmed, unspent, and locked to $\mathsf{pk}_{O_{\mathsf{j}}}$

if $\mathsf{val}(u_{O_{\mathsf{j}}}) \neq d_{\mathsf{dust}} \cdot (k_C(N-1)+2)$ **then abort**

for $\mathsf{j} \in [N] \setminus \{\mathsf{i}\}$ **in parallel do**

$\mathsf{st}^{P}_{(\mathsf{i},\mathsf{j})} \leftarrow P^{\mathsf{cc}}_{\mathsf{Setup}}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{i}})$ **with** O_{j}

$\mathsf{st}^{V}_{(\mathsf{j},\mathsf{i})} \leftarrow V^{\mathsf{cc}}_{\mathsf{Setup}}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{j}})$ **with** O_{j}

$\mathcal{D}_{\mathsf{i}} \leftarrow \{(\mathsf{p},\mathsf{v}) \in \mathcal{D} : \mathsf{i} \in \{\mathsf{p},\mathsf{v}\}\}$

$(\mathcal{T}, \mathcal{S}) \leftarrow \mathsf{CreateTxSet}_{\mathsf{mp}}\left(\left\{\mathsf{pk}_{O_{\mathsf{j}}}, \mathsf{lpk}_{O_{\mathsf{j}}}, \mathsf{pk}_{R,\mathsf{j}}, u_{O_{\mathsf{j}}}\right\}_{\mathsf{j}\in[N]}, \mathsf{pk}^{\mathsf{id}}_{\mathcal{O}}, \mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}}, \mathsf{pk}_U, \left\{\Upsilon_{(\mathsf{p},\mathsf{v})}\right\}_{(\mathsf{p},\mathsf{v})\in\mathcal{D}_{\mathsf{i}}}\right)$

for $\mathsf{j} \in [N] \setminus \{\mathsf{i}\}$ **in parallel do**

$\mathsf{st}^{P}_{(\mathsf{i},\mathsf{j})} \leftarrow P^{\mathsf{ps}}_{\mathsf{Setup}}(\mathcal{T}, \mathsf{st}^{P}_{(\mathsf{i},\mathsf{j})})$ **with** O_{j}

$\mathsf{st}^{V}_{(\mathsf{j},\mathsf{i})} \leftarrow V^{\mathsf{ps}}_{\mathsf{Setup}}(\mathcal{T}, \mathsf{st}^{V}_{(\mathsf{j},\mathsf{i})})$ **with** O_{j}

for $\mathsf{p} \in [N]$ **in parallel do**

$\sigma^{\mathsf{cov}}_{\mathsf{WithdrawOpt},\mathsf{p}} \leftarrow \mathsf{MuSig2.Sign}(\mathsf{sk}^{\mathsf{cov}}_{O_{\mathsf{i}}}, \mathsf{WithdrawOptimistic}^{\mathsf{p}})$

$\sigma^{\mathsf{cov}}_{\mathsf{Withdraw},\mathsf{p}} \leftarrow \mathsf{MuSig2.Sign}(\mathsf{sk}^{\mathsf{cov}}_{O_{\mathsf{i}}}, \mathsf{Withdraw}^{\mathsf{p}})$

verify $\mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Verify}\left(\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}}, \mathsf{WithdrawOptimistic}^{\mathsf{p}}, \sigma^{\mathsf{cov}}_{\mathsf{WithdrawOpt},\mathsf{p}}\right) = 1$

verify $\mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Verify}\left(\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}}, \mathsf{Withdraw}^{\mathsf{p}}, \sigma^{\mathsf{cov}}_{\mathsf{Withdraw},\mathsf{p}}\right) = 1$

delete $\mathsf{sk}^{\mathsf{cov}}_{O_{\mathsf{i}}}$

$\sigma^{\mathsf{id}}_{\mathsf{Deposit}} \leftarrow \mathsf{MuSig2.Sign}\left(\mathsf{sk}^{\mathsf{id}}_{O_{\mathsf{i}}}, \mathsf{Deposit}\right)$

any $O_{\mathsf{j}} \in \mathcal{O}$ **with** a validly signed Deposit: $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{Deposit})$

$\mathsf{st}_{O_{\mathsf{i}}} \leftarrow \left(\mathsf{sk}_{O_{\mathsf{i}}}, \mathsf{sk}^{\mathsf{id}}_{O_{\mathsf{i}}}, \mathsf{lsk}_{O_{\mathsf{i}}}, \left\{\mathsf{st}^{P}_{(\mathsf{i},\mathsf{j})}, \mathsf{st}^{V}_{(\mathsf{j},\mathsf{i})}\right\}_{\mathsf{j}\in[N]\setminus\{\mathsf{i}\}}, \sigma^{\mathsf{cov}}_{\mathsf{WithdrawOpt},\mathsf{i}}, \sigma^{\mathsf{cov}}_{\mathsf{Withdraw},\mathsf{i}}\right)$

upon *observing* Deposit *(resp.* DepositRefund*) confirmed in* $\mathcal{L}$ **do**

return $(\{\boldsymbol{x}_{\mathsf{j}}\}_{\mathsf{j}\in[N]}, \mathcal{T}, \mathcal{S}, \mathsf{st}_{O_{\mathsf{i}}})$ (resp. $\perp$)

Protocol 2: Prover-side setup algorithms, run by O_{p} for one instance (p,v). The (p,v) subscript is omitted.

procedure $P^{\mathsf{cc}}_{\mathsf{Setup}}(\mathsf{crs}, \boldsymbol{x})$: ▷ *cut-and-choose*

  upon *receiving* $\big(\{h_{\mathsf{msg},i}, h_{\mathsf{ct}_{\mathsf{setup}},i}, H_i\}_{i=1}^{N_{\mathsf{CC}}}, \{C_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}\big)$ *from* O_{v} **do**

    verify $|C_{t,\beta}| = d_{\mathsf{CC}} + 1$ for all $t \in [2n]$, $\beta \in \{0,1\}$

    $T_{t,\beta} := C_{t,\beta}(0)$ for all $t \in [2n]$, $\beta \in \{0,1\}$ ▷ *adaptor points*

    verify the $4n(N_{\mathsf{CC}}{+}1)$ points $\{C_{t,\beta}(i)\}_{t\in[2n],\,\beta\in\{0,1\},\,i\in\{0\}\cup[N_{\mathsf{CC}}]}$ are pairwise distinct

    sample $\mathcal{I} \subseteq [N_{\mathsf{CC}}]$, $|\mathcal{I}| = M_{\mathsf{CC}}$, uniformly at random

    send $\mathcal{I}$ **to** O_{v} ▷ *finalized set*

  upon *receiving* $\big(\{\mathsf{msg}_i, r_i, \mathsf{seed}_i, s_i\}_{i\in[N_{\mathsf{CC}}]\setminus\mathcal{I}},\ \{\mathsf{ct}_{\mathsf{setup},i}, \mathsf{ct}_{\mathsf{GC},i}\}_{i\in\mathcal{I}}\big)$ *from* O_{v}, *where* $s_i = \{f_{t,\beta}(i)\}_{t\in[2n],\,\beta\in\{0,1\}}$ **do**

    for $i \in [N_{\mathsf{CC}}] \setminus \mathcal{I}$ **do** ▷ *if any check fails, abort*

      verify $f_{t,\beta}(i)\cdot G = C_{t,\beta}(i)$ for all $t \in [2n]$, $\beta \in \{0,1\}$ ▷ *Feldman share check*

      verify $H\big(\mathsf{WE}.\mathsf{Enc}_{\mathsf{setup}}(\mathsf{crs}, \boldsymbol{x}, \mathsf{msg}_i; r_i)\big) = h_{\mathsf{ct}_{\mathsf{setup}},i}$

      verify $\mathsf{Hash}_{\mathrm{BTC}}(\mathsf{msg}_i) = h_{\mathsf{msg},i}$

      verify $\mathsf{CommitTable}\big(\mathsf{GSM}.\mathsf{Garble}(r_i, \mathsf{seed}_i, s_i)\big) = H_i$ ▷ *re-garble*

    for $i \in \mathcal{I}$ **do** ▷ *consistency with round 1*

      verify $\mathsf{CommitTable}(\mathsf{ct}_{\mathsf{GC},i}) = H_i$

      verify $H(\mathsf{ct}_{\mathsf{setup},i}) = h_{\mathsf{ct}_{\mathsf{setup}},i}$

  $\Upsilon := \big(\{T_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}},\ \{h_{\mathsf{msg},i}\}_{i\in\mathcal{I}}\big)$ ▷ *commitment record*

  $\mathsf{st}^P := \big(\{\mathsf{ct}_{\mathsf{setup},i}, \mathsf{ct}_{\mathsf{GC},i}\}_{i\in\mathcal{I}},\ \{s_i\}_{i\in[N_{\mathsf{CC}}]\setminus\mathcal{I}},\ \Upsilon\big)$

  return st^P

procedure $P^{\mathsf{ps}}_{\mathsf{Setup}}(\mathcal{T}, \mathsf{st}^P)$: ▷ *pre-signing*

  for $t \in [2n]$, $\beta \in \{0,1\}$ **do**

    $\tilde{\sigma}_{t,\beta} \leftarrow \mathsf{SASig}.\mathsf{GenAdaptor}(\mathsf{sk}_{R,\mathsf{p}}, T_{t,\beta}, \overline{\mathsf{tx}}^{t,\beta}_{\mathsf{ChA}^{j(t)}})$ ▷ *adaptor signatures*

  $\sigma^P_{\mathsf{X}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{p}}}, \mathsf{X})$ for $\mathsf{X} \in \{\mathsf{ChallengeOptimistic}, \mathsf{ChallengeClaim}, \mathsf{NoAssert}, \mathsf{NoWithdraw}\}$

  $\mathsf{presigs}_P \leftarrow \Big\{\sigma^P_{\mathsf{ChallengeOptimistic}}, \sigma^P_{\mathsf{ChallengeClaim}}, \sigma^P_{\mathsf{NoAssert}}, \sigma^P_{\mathsf{NoWithdraw}}\Big\}$

  send $(\{\tilde{\sigma}_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \mathsf{presigs}_P)$ **to** O_{v}

  upon *receiving* $\mathsf{presigs}_V = \{\sigma^V_{\mathsf{Assert}}\}$ *from* O_{v} **do**

    abort if $\mathsf{VerifySigs}_P(\mathsf{pk}_{O_{\mathsf{v}}}, \mathcal{T}, \mathsf{presigs}_V) = 0$

  $\mathsf{st}^P \leftarrow \mathsf{st}^P \cup \big(\{\tilde{\sigma}_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \mathsf{presigs}_V\big)$

  return st^P

Protocol 3: Verifier-side setup algorithms, run by O_{v} for one instance (p,v). The (p,v) subscript is omitted.

procedure $V^{\mathsf{cc}}_{\mathsf{Setup}}(\mathsf{crs}, \boldsymbol{x})$: ▷ *cut-and-choose*

  for $t \in [2n]$, $\beta \in \{0,1\}$ **do**

    $f_{t,\beta} \xleftarrow{\$} \mathbb{F}_{q_s}[X]_{\leq d_{\mathsf{CC}}}$

    $C_{t,\beta} \leftarrow \mathsf{Com}(f_{t,\beta})$ ▷ *Feldman polynomials*

  for $i = 1, \ldots, N_{\mathsf{CC}}$ **do**

    $\mathsf{msg}_i \xleftarrow{\$} \mathcal{M}$; $r_i \xleftarrow{\$} \mathbb{F}_q^*$; $\mathsf{seed}_i \xleftarrow{\$} \{0,1\}^\lambda$

    $\mathsf{ct}_{\mathsf{setup},i} \leftarrow \mathsf{WE}.\mathsf{Enc}_{\mathsf{setup}}(\mathsf{crs}, \boldsymbol{x}, \mathsf{msg}_i; r_i)$

    $\mathsf{ct}_{\mathsf{GC},i} \leftarrow \mathsf{GSM}.\mathsf{Garble}\big(r_i, \mathsf{seed}_i, \{f_{t,\beta}(i)\}_{t\in[2n],\,\beta\in\{0,1\}}\big)$

    $H_i \leftarrow \mathsf{CommitTable}(\mathsf{ct}_{\mathsf{GC},i})$

    $h_{\mathsf{msg},i} \leftarrow \mathsf{Hash}_{\mathrm{BTC}}(\mathsf{msg}_i)$

    $h_{\mathsf{ct}_{\mathsf{setup}},i} \leftarrow H(\mathsf{ct}_{\mathsf{setup},i})$

  send $\big(\{h_{\mathsf{msg},i}, h_{\mathsf{ct}_{\mathsf{setup}},i}, H_i\}_{i=1}^{N_{\mathsf{CC}}}, \{C_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}\big)$ **to** O_{p} ▷ *commitments only*

  upon *receiving* $\mathcal{I}$ *from* O_{p} **do**

    send $\Big(\Big\{\mathsf{msg}_i, r_i, \mathsf{seed}_i, \{f_{t,\beta}(i)\}_{t\in[2n],\,\beta\in\{0,1\}}\Big\}_{i\in[N_{\mathsf{CC}}]\setminus\mathcal{I}}, \{\mathsf{ct}_{\mathsf{setup},i}, \mathsf{ct}_{\mathsf{GC},i}\}_{i\in\mathcal{I}}\Big)$ **to** O_{p}

  $\Upsilon \leftarrow \big(\{T_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \{h_{\mathsf{msg},i}\}_{i\in\mathcal{I}}\big)$

  $\mathsf{st}^V \leftarrow \big(\{f_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \Upsilon\big)$

  return st^V

procedure $V^{\mathsf{ps}}_{\mathsf{Setup}}(\mathcal{T}, \mathsf{st}^V)$: ▷ *pre-signing*

  upon *receiving* $(\{\tilde{\sigma}_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \mathsf{presigs}_P)$ *from* O_{p} **do**

    for $t \in [2n]$, $\beta \in \{0,1\}$ **do**

      verify $\mathsf{SASig}.\mathsf{VerAdaptor}(\mathsf{pk}_{R,\mathsf{p}}, T_{t,\beta}, \overline{\mathsf{tx}}^{t,\beta}_{\mathsf{ChA}^{j(t)}}, \tilde{\sigma}_{t,\beta}) = 1$

    if $\mathsf{VerifySigs}_V(\mathsf{pk}_{O_{\mathsf{p}}}, \mathcal{T}, \mathsf{presigs}_P) = 0$ **then abort**

  $\sigma^V_{\mathsf{Assert}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{v}}}, \mathsf{Assert})$

  $\sigma^V_{\mathsf{ChallengeOpt}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{v}}}, \mathsf{ChallengeOptimistic})$

  $\mathsf{presigs}_V \leftarrow \{\sigma^V_{\mathsf{Assert}}\}$

  send $\mathsf{presigs}_V$ **to** O_{p}

  $\mathsf{st}^V \leftarrow \mathsf{st}^V \cup \big(\{\tilde{\sigma}_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \mathsf{presigs}_P, \sigma^V_{\mathsf{ChallengeOpt}}\big)$

  return st^V

Protocol 4: Prover-side peg-out algorithms.

procedure $P_{\mathsf{Prove}}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \mathcal{T}, \mathcal{S}, \mathsf{st}_{O_{\mathsf{p}}}, \boldsymbol{w})$: ▷ *run by prover-candidate* O_{p}

  $\big(\mathsf{sk}_{O_{\mathsf{p}}}, \cdot, \mathsf{lsk}_{O_{\mathsf{p}}}, \big\{\mathsf{st}^{P}_{(\mathsf{p},\mathsf{j})}, \cdot\big\}_{\mathsf{j}\in[N]\setminus\{\mathsf{p}\}}, \sigma^{\mathsf{cov}}_{\mathsf{WithdrawOpt},\mathsf{p}}, \sigma^{\mathsf{cov}}_{\mathsf{Withdraw},\mathsf{p}}\big) \leftarrow \mathsf{st}_{O_{\mathsf{p}}}$

  foreach $\mathsf{v} \in [N] \setminus \{\mathsf{p}\}$ **do** ▷ *instance values*

    $\big(\{\mathsf{ct}_{\mathsf{setup},i}, \mathsf{ct}_{\mathsf{GC},i}\}_{i\in\mathcal{I}}, \{s_i\}_{i\in[N_{\mathsf{CC}}]\setminus\mathcal{I}}, \Upsilon, \{\tilde{\sigma}_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \sigma^{V}_{\mathsf{Assert}}\big) \leftarrow \mathsf{st}^{P}_{(\mathsf{p},\mathsf{v})}$

    $\big(\cdot, \{h_{\mathsf{msg},i}\}_{i\in\mathcal{I}}\big) \leftarrow \Upsilon$

  $(\pi_1, \pi_2, \pi_3) \leftarrow \mathsf{Groth16.Prove}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \boldsymbol{w})$

  upload $\pi = (\pi_1, \pi_2, \pi_3)$ to the proof repository ▷ *verifiers retrieve* π *and run* Groth16.Verify

  $\mu \leftarrow \mathsf{Lamp.Sign}(\pi_1, \mathsf{lsk}_{O_{\mathsf{p}}})$

  $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{ClaimOptimistic}^{\mathsf{p}})$ ▷ *starts the* $\Delta_{\mathsf{withdrawOpt}}$ *veto window*

  upon *observing* $\Delta_{\mathsf{withdrawOpt}}$ *new blocks after* $\mathsf{ClaimOptimistic}^{\mathsf{p}}$ *in* $\mathcal{L}_P$ *with no* $\mathsf{ChallengeOptimistic}$ **do**

    $\mathsf{WithdrawOptimistic}^{\mathsf{p}}.\mathsf{witness} \leftarrow (\sigma^{\mathsf{cov}}_{\mathsf{WithdrawOpt},\mathsf{p}}, \sigma_{P,\mathsf{WithdrawOpt}}, \sigma_{P,\mathsf{WithdrawOpt}})$ where $\sigma_{P,\mathsf{WithdrawOpt}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{p}}}, \mathsf{WithdrawOptimistic}^{\mathsf{p}})$, one per input

    $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{WithdrawOptimistic}^{\mathsf{p}})$

  upon *observing some* $\mathsf{ChallengeOptimistic}$ *in* $\mathcal{L}_P$ **do**

    $\mathsf{Claim}^{\mathsf{p}}.\mathsf{witness} \leftarrow (\sigma_{P,\mathsf{Claim}})$ where $\sigma_{P,\mathsf{Claim}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{p}}}, \mathsf{Claim}^{\mathsf{p}})$

    $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{Claim}^{\mathsf{p}})$ ▷ *optimistic exit vetoed, non-optimistic phase begins (starts* $\Delta_{\mathsf{withdraw}}$*)*

  upon *observing* $\mathsf{ChallengeClaim}_{(\mathsf{p},\mathsf{v})}$ *in* $\mathcal{L}_P$ *(for each such* v*)* **do**

    $\mathsf{Assert}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness} \leftarrow \big(\mu, \sigma^{V}_{\mathsf{Assert}}\big)$

    $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{Assert}_{(\mathsf{p},\mathsf{v})})$ ▷ *within* Δ_{assert}

  upon *observing all* k_C $\mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})}$ *in* $\mathcal{L}_P$ *(for each such* v*)* **do**

    for $t \in [2n]$ **do**

      extract σ_t from $\mathsf{ChallengeAssert}^{j(t)}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness}$

      $y_t \leftarrow \mathsf{SASig.Extract}(\tilde{\sigma}_{t,\pi_{1,t}}, \sigma_t)$ ▷ $y_t = f_{t,\pi_{1,t}}(0)$

      $f_{t,\pi_{1,t}} \leftarrow \mathsf{Interpolate}\big(\{(0, y_t)\} \cup \big\{(i, f_{t,\pi_{1,t}}(i))\big\}_{i\in[N_{\mathsf{CC}}]\setminus\mathcal{I}}\big)$

    for $i \in \mathcal{I}$ **do**

      $K_i \leftarrow \big(\pi_{1,t},\, f_{t,\pi_{1,t}}(i)\big)_{t\in[2n]}$ ▷ *active keys with their bits*

      $\mathsf{ct}_{\mathsf{prove},i} \leftarrow \mathsf{GSM.Eval}(\mathsf{ct}_{\mathsf{GC},i}, K_i)$

      if $\mathsf{ct}_{\mathsf{prove},i} = \perp$ **then** skip candidate i

      $\mathsf{msg}_i \leftarrow \mathsf{WE.Dec}\big((\mathsf{ct}_{\mathsf{setup},i}, \mathsf{ct}_{\mathsf{prove},i}), (\pi_2, \pi_3)\big)$ ▷ $\mathsf{ct}_{\mathsf{prove},i} = r_i\pi_1$

    find $i^* \in \mathcal{I}$ such that $\mathsf{Hash}_{\mathrm{BTC}}(\mathsf{msg}_{i^*}) = h_{\mathsf{msg},i^*}$

    $\mathsf{WronglyChallenged}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness} \leftarrow (\mathsf{msg}_{i^*}, \sigma_{P,\mathsf{WronglyChallenged}})$ on each of its k_C inputs, where $\sigma_{P,\mathsf{WronglyChallenged}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{p}}}, \mathsf{WronglyChallenged}_{(\mathsf{p},\mathsf{v})})$

    $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{WronglyChallenged}_{(\mathsf{p},\mathsf{v})})$ ▷ *within* $\Delta_{\mathsf{noWithdraw}}$

  upon *observing* $\Delta_{\mathsf{withdraw}}$ *new blocks after* $\mathsf{Claim}^{\mathsf{p}}$ *in* $\mathcal{L}_P$ *with* $\mathsf{Claim}^{\mathsf{p}}[0]$ *unspent* **do**

    $\mathsf{Withdraw}^{\mathsf{p}}.\mathsf{witness} \leftarrow (\sigma^{\mathsf{cov}}_{\mathsf{Withdraw},\mathsf{p}}, \sigma_{P,\mathsf{Withdraw}})$ where $\sigma_{P,\mathsf{Withdraw}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{p}}}, \mathsf{Withdraw}^{\mathsf{p}})$

    $\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{Withdraw}^{\mathsf{p}})$

  upon *observing* $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ *or* $\mathsf{Withdraw}^{\mathsf{p}}$ *in* $\mathcal{L}_P$ **do**

    return ▷ O_{p} *receives* B

  upon *observing some* $\mathsf{NoAssert}$ *or* $\mathsf{NoWithdraw}$ *in* $\mathcal{L}_P$ **do**

    return ▷ $\mathsf{Claim}^{\mathsf{p}}[0]$ *slash leaf spent*

Protocol 5: Verifier-side peg-out algorithms, run by O_{v} for each instance (p,v) it disputes.

procedure $V_{\mathsf{Prove}}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \mathcal{T}, \mathcal{S}, \mathsf{st}_{O_{\mathsf{v}}})$**:**

$\left(\mathsf{sk}_{O_{\mathsf{v}}}, \cdot, \cdot, \left\{\cdot, \mathsf{st}^{V}_{(\mathsf{j},\mathsf{v})}\right\}_{\mathsf{j}\in[N]\setminus\{\mathsf{v}\}}, \cdot, \cdot\right) \leftarrow \mathsf{st}_{O_{\mathsf{v}}}$

$\left(\{f_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \cdot, \{\tilde{\sigma}_{t,\beta}\}_{t\in[2n],\,\beta\in\{0,1\}}, \sigma^{P}_{\mathsf{ChallengeOptimistic}}, \sigma^{P}_{\mathsf{ChallengeClaim}}, \sigma^{P}_{\mathsf{NoAssert}}, \sigma^{P}_{\mathsf{NoWithdraw}}, \sigma^{V}_{\mathsf{ChallengeOpt}}\right) \leftarrow \mathsf{st}^{V}_{(\mathsf{p},\mathsf{v})}$

upon *observing* $\mathsf{ClaimOptimistic}^{\mathsf{p}}$ *in* $\mathcal{L}_V$ **do**

retrieve $\pi = (\pi_1, \pi_2, \pi_3)$ from the proof repository

if $\mathsf{Groth16.Verify}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \pi) \neq 1$ *or no proof is available* **then**

$\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness} \leftarrow (\sigma^{P}_{\mathsf{ChallengeOptimistic}}, \sigma^{V}_{\mathsf{ChallengeOpt}})$ ▷ *pre-signed at setup*

$\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})})$; ▷ *eliminates* $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$

upon *observing* $\mathsf{Claim}^{\mathsf{p}}$ *in* $\mathcal{L}_V$ **do**

$\mathsf{ChallengeClaim}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness} \leftarrow (\sigma^{P}_{\mathsf{ChallengeClaim}}, \sigma_{V,\mathsf{ChallengeClaim}})$ where

$\sigma_{V,\mathsf{ChallengeClaim}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{v}}}, \mathsf{ChallengeClaim}_{(\mathsf{p},\mathsf{v})})$

$\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{ChallengeClaim}_{(\mathsf{p},\mathsf{v})})$ ▷ *creates the assert input* $\mathsf{ChallengeClaim}[0]$

upon *observing* Δ_{assert} *new blocks after* $\mathsf{ChallengeClaim}_{(\mathsf{p},\mathsf{v})}$ *in* $\mathcal{L}_V$ *with* $\mathsf{Assert}_{(\mathsf{p},\mathsf{v})} \notin \mathcal{L}_V$ **do**

$\mathsf{NoAssert}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness} \leftarrow (\sigma^{P}_{\mathsf{NoAssert}}, \sigma_{V,\mathsf{NoAssert}};\ \sigma_{V,\mathsf{NoAssert}})$ where

$\sigma_{V,\mathsf{NoAssert}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{v}}}, \mathsf{NoAssert}_{(\mathsf{p},\mathsf{v})})$

$\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{NoAssert}_{(\mathsf{p},\mathsf{v})})$ ▷ *slash*

upon *observing* $\mathsf{Assert}_{(\mathsf{p},\mathsf{v})}$ *in* $\mathcal{L}_V$ **do**

extract π_1 and μ from $\mathsf{Assert}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness}$

for $t \in R_j, j \in \{0,\ldots,k_C{-}1\}$ **do** ▷ *in parallel, once* Assert *confirms*

$\sigma_t \leftarrow \mathsf{SASig}.\mathsf{Adapt}\left(\tilde{\sigma}_{t,\pi_{1,t}}, f_{t,\pi_{1,t}}(0)\right)$

$\mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness} \leftarrow \left(\sigma_{V,\mathsf{ChallengeAssert}^{j}}, (\sigma_t, \mu_t)_{t\in R_j}\right)$ where

$\sigma_{V,\mathsf{ChallengeAssert}^{j}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{v}}}, \mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})})$

$\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})})$

upon *observing* $\Delta_{\mathsf{noWithdraw}}$ *new blocks after all* k_C $\mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})}$ *in* $\mathcal{L}_V$ *with every* $\mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})}[0]$ *unspent* **do**

$\mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})}.\mathsf{witness} \leftarrow \left(\sigma^{P}_{\mathsf{NoWithdraw}}, \sigma_{V,\mathsf{NoWithdraw}};\ (\sigma^{(j)}_{V,\mathsf{NoWithdraw}})_{j\in\{0,\ldots,k_C-1\}}\right)$ where

$\sigma_{V,\mathsf{NoWithdraw}}, \sigma^{(j)}_{V,\mathsf{NoWithdraw}} \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Sign}(\mathsf{sk}_{O_{\mathsf{v}}}, \mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})})$, one per input

$\mathcal{F}_{\mathrm{BTC}}.\mathsf{WRITE}(\mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})})$ ▷ *slash*

upon *observing some* NoAssert *or* NoWithdraw *in* $\mathcal{L}_V$ **do**

return ▷ $\mathsf{Claim}^{\mathsf{p}}[0]$ *slash leaf spent*

upon *observing* $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ *or* $\mathsf{Withdraw}^{\mathsf{p}}$ *in* $\mathcal{L}_V$ **do**

return ▷ O_{p} *receives* B

Protocol 6: Locking scripts and transaction set, run locally by each operator over the records $\mathcal{D}_{\mathsf{i}}$ of its own instances.

function $\mathsf{CreateTxSet_{mp}}\big(\{\mathsf{pk}_{O_{\mathsf{i}}}, \mathsf{lpk}_{O_{\mathsf{i}}}, \mathsf{pk}_{R,\mathsf{i}}, u_{O_{\mathsf{i}}}\}_{\mathsf{i}\in[N]}, \mathsf{pk}^{\mathsf{id}}_{\mathcal{O}}, \mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}}, \mathsf{pk}_U, \{\Upsilon_{(\mathsf{p},\mathsf{v})}\}_{(\mathsf{p},\mathsf{v})\in\mathcal{D}_{\mathsf{i}}}\big)$:

- $\{T_{t,\beta}\}, \{h_{\mathsf{msg},i}\} \leftarrow \Upsilon_{(\mathsf{p},\mathsf{v})}$
- $\mathsf{AssertScript} := \mathsf{CheckLampSig}\big(\mathsf{lpk}_{O_\mathsf{p}}\big) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{v}})$
- $\mathsf{ChallengeAssertScript}^j := \mathsf{CheckLampAdaptorMatch}\big(\mathsf{lpk}^{(R_j)}_{O_\mathsf{p}}, \{T_{t,\beta}\}_{t\in R_j,\,\beta\in\{0,1\}}, \mathsf{pk}_{R,\mathsf{p}}\big) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{v}})$, $j \in \{0,\ldots,k_C{-}1\}$ ▷ $:= \Phi_j$, *Sec. 7*
- $\mathsf{WronglyChallengedScript} := \big(\bigvee_{i\in\mathcal{I}} \mathsf{HashLock}(h_{\mathsf{msg},i})\big) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}})$ ▷ *any one finalized* msg_i *opens it*
- $\mathsf{DepositRequest}[0] := \big(\text{key path } \mathsf{pk}^{\mathsf{id}}_{\mathcal{O}} \;||\; \mathsf{RelTimelock}(\Delta_{\mathsf{refund}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_U)\big)$
- $\mathsf{Deposit}[0] := \langle \mathsf{CheckSig}(\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}})\rangle$ **for** $\mathsf{p} \in [N]$ **do**
  - $\mathsf{ClaimOptimistic}^{\mathsf{p}}[0] := \langle \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}}) \;||\; \big\{\mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{v}})\big\}_{\mathsf{v}\neq\mathsf{p}}\rangle$
  - $\mathsf{ClaimOptimistic}^{\mathsf{p}}[1] := \langle \mathsf{RelTimelock}(\Delta_{\mathsf{withdrawOpt}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}}) \;||\; \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}})\rangle$
  - $\mathsf{Claim}^{\mathsf{p}}[0] := \langle \mathsf{RelTimelock}(\Delta_{\mathsf{withdraw}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}}) \;||\; \big\{\mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{v}})\big\}_{\mathsf{v}\neq\mathsf{p}}\rangle$
- **for** $(\mathsf{p},\mathsf{v}) \in \mathcal{D}_{\mathsf{i}}$ **do**
  - $\mathsf{Claim}^{\mathsf{p}}[1{+}\mathsf{rk}(\mathsf{p},\mathsf{v})] := \langle \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{v}}) \;||\; \mathsf{RelTimelock}(\Delta_{\mathsf{withdraw}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{p}})\rangle$
  - $\mathsf{ChallengeClaim}[0] := \langle \mathsf{RelTimelock}(\Delta_{\mathsf{assert}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{v}}) \;||\; \mathsf{AssertScript}\rangle$
  - $\mathsf{Assert}[j] := \langle \mathsf{ChallengeAssertScript}^j\rangle$, $j \in \{0,\ldots,k_C{-}1\}$
  - $\mathsf{ChallengeAssert}^j[0] := \langle \mathsf{RelTimelock}(\Delta_{\mathsf{noWithdraw}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_\mathsf{v}}) \;||\; \mathsf{WronglyChallengedScript}\rangle$, $j \in \{0,\ldots,k_C{-}1\}$
- $\mathsf{Deposit}$: $\mathsf{DepositRequest}[0]$ (key path) $\rightarrow \mathsf{Deposit}[0]$ (B)
- $\mathsf{DepositRefund}$: $\mathsf{DepositRequest}[0]$ (refund leaf) $\rightarrow \mathsf{pk}_U$
- **for** $\mathsf{p} \in [N]$ **do**
  - $\mathsf{ClaimOptimistic}^{\mathsf{p}}$: $u_{O_\mathsf{p}} \rightarrow \mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$, $\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$
  - $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$: $\mathsf{Deposit}[0]$, $\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$ (leaf 0), $\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$ (leaf 0) $\rightarrow B$ to O_p
  - $\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$ for $\mathsf{v} \neq \mathsf{p}$: $\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$ (leaf $1{+}\mathsf{rk}(\mathsf{p},\mathsf{v})$) $\rightarrow$ sweep
  - $\mathsf{Claim}^{\mathsf{p}}$: $\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$ (leaf 1) $\rightarrow \mathsf{Claim}^{\mathsf{p}}[0]$, $\{\mathsf{Claim}^{\mathsf{p}}[1{+}\mathsf{rk}(\mathsf{p},\mathsf{v})]\}_{\mathsf{v}\neq\mathsf{p}}$
  - $\mathsf{Withdraw}^{\mathsf{p}}$: $\mathsf{Deposit}[0]$, $\mathsf{Claim}^{\mathsf{p}}[0]$ (leaf 0) $\rightarrow B$ to O_p
- **for** $(\mathsf{p},\mathsf{v}) \in \mathcal{D}_{\mathsf{i}}$ **do**
  - $\mathsf{ChallengeClaim}$: $\mathsf{Claim}^{\mathsf{p}}[1{+}\mathsf{rk}(\mathsf{p},\mathsf{v})]$ (leaf 0) $\rightarrow \mathsf{ChallengeClaim}[0]$
  - Assert: $\mathsf{ChallengeClaim}[0]$ (leaf 1) $\rightarrow \{\mathsf{Assert}[j]\}_{j\in\{0,\ldots,k_C-1\}}$
  - $\mathsf{NoAssert}$: $\mathsf{Claim}^{\mathsf{p}}[0]$ (slash leaf v), $\mathsf{ChallengeClaim}[0]$ (leaf 0) $\rightarrow$ sweep
  - $\mathsf{ChallengeAssert}^j$: $\mathsf{Assert}[j] \rightarrow \mathsf{ChallengeAssert}^j[0]$
  - $\mathsf{NoWithdraw}$: $\mathsf{Claim}^{\mathsf{p}}[0]$ (slash leaf v), $\{\mathsf{ChallengeAssert}^j[0]$ (leaf 0)$\}_{j\in\{0,\ldots,k_C-1\}} \rightarrow$ sweep
  - $\mathsf{WronglyChallenged}$: $\{\mathsf{ChallengeAssert}^j[0]$ (leaf 1)$\}_{j\in\{0,\ldots,k_C-1\}} \rightarrow$ sweep
- $\mathcal{T} :=$ all skeletons above
- $\mathcal{S} := \{\mathsf{WithdrawOptimistic}^{\mathsf{p}}, \mathsf{Withdraw}^{\mathsf{p}}\}_{\mathsf{p}\in[N]}$ **return** $(\mathcal{T},\mathcal{S})$

Protocol 7: Helper functions.

function $\mathsf{GenAllKeys}(1^\lambda, \mathsf{i})$**:** ▷ *$O_\mathsf{i}$'s full key material*

- $(\mathsf{sk}_{O_\mathsf{i}}, \mathsf{pk}_{O_\mathsf{i}}) \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Gen}(1^\lambda)$ ▷ *dispute key*
- $(\mathsf{sk}^{\mathsf{id}}_{O_\mathsf{i}}, \mathsf{pk}^{\mathsf{id}}_{O_\mathsf{i}}) \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Gen}(1^\lambda)$
- $(\mathsf{sk}^{\mathsf{cov}}_{O_\mathsf{i}}, \mathsf{pk}^{\mathsf{cov}}_{O_\mathsf{i}}) \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Gen}(1^\lambda)$ ▷ MuSig2 *identity / covenant shares*
- $(\mathsf{lsk}_{O_\mathsf{i}}, \mathsf{lpk}_{O_\mathsf{i}}) \leftarrow \mathsf{Lamp}.\mathsf{Gen}(2n)$ ▷ *$2n$-bit Lamport key*
- $(\mathsf{sk}_{R,\mathsf{i}}, \mathsf{pk}_{R,\mathsf{i}}) \leftarrow \mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Gen}(1^\lambda)$ ▷ *adaptor-reveal key*
- **return** $\big((\mathsf{sk}_{O_\mathsf{i}}, \mathsf{pk}_{O_\mathsf{i}}), (\mathsf{sk}^{\mathsf{id}}_{O_\mathsf{i}}, \mathsf{pk}^{\mathsf{id}}_{O_\mathsf{i}}), (\mathsf{sk}^{\mathsf{cov}}_{O_\mathsf{i}}, \mathsf{pk}^{\mathsf{cov}}_{O_\mathsf{i}}), (\mathsf{lsk}_{O_\mathsf{i}}, \mathsf{lpk}_{O_\mathsf{i}}), (\mathsf{sk}_{R,\mathsf{i}}, \mathsf{pk}_{R,\mathsf{i}})\big)$

function $\mathsf{VerifySigs}_\rho(\mathsf{pk}, \mathcal{T}, \mathsf{presigs})$**:**

- $\mathcal{X}_P := \{\mathsf{Assert}\}$, $\mathcal{X}_V := \{\mathsf{ChallengeOptimistic}, \mathsf{ChallengeClaim}, \mathsf{NoAssert}, \mathsf{NoWithdraw}\}$
- **if** *the index set of* presigs *is not exactly* $\mathcal{X}_\rho$ **then return** 0
- **for** $\mathsf{X} \in \mathcal{X}_\rho$ **do**
  - **if** $\mathsf{Sig}_{\mathrm{BTC}}.\mathsf{Verify}\big(\mathsf{pk}, \mathsf{X}, \sigma_\mathsf{X}\big) \neq 1$ **then return** 0
- **return** 1

9 Security Proof

We prove that TRITONE (Protocols 1–7) satisfies the properties of Section 4 under static corruption, and derive the system-level guarantee of Remark 1. Throughout, $C \subset \mathcal{O}$ denotes the corrupt set, $\mathcal{A}$ the PPT adversary controlling it, and an operator is honest iff it lies outside C. An honest operator runs the complete protocol, i.e., all $2(N-1)$ instances in which it participates, and the theorems below are stated for this setting so that they compose (Section 9.7). For a fixed instance $(\mathsf{p}, \mathsf{v}) \in \mathcal{D}$, we drop the instance subscript on transactions and state variables, as in Section 8. Transactions, leaves, and witnesses are as specified in Appendix A.

9.1 Assumptions and Conventions

The proofs use the following assumptions, which collect the security notions of the primitives of Sections 3–7. The model items (M1)–(M4) of Section 7.3 are inherited.

(A1) *Ledger.* $\mathcal{F}_{\mathrm{BTC}}$ satisfies safety and liveness with confirmation bound Δ_{conf} (Section 3.8), where Δ_{conf} absorbs the reorg-safe observation depth and the latency of off-chain checks, such as retrieving and verifying a proof. We use liveness in the following form: a valid transaction broadcast at height h is confirmed by height $h + \Delta_{\mathsf{conf}}$ unless a *conflicting* transaction, one sharing an input with it, is confirmed first. Wherever liveness is invoked, we enumerate the conflicting spends that $\mathcal{A}$ can confirm. A transaction is valid at height h if every witness satisfies its leaf, every input is unspent, and every relative timelock has elapsed (Section 3.8). Honest parties react to an event in the round they observe it.

(A2) *Signatures.* $\mathsf{Sig}_{\mathrm{BTC}}$ is EUF-CMA secure and SigMsg is injective in the sense of (M2). The multi-signature scheme $\mathsf{MuSig2}$ [NRS21] is unforgeable in the presence of $N-1$ corrupt cosigners: no PPT adversary produces a valid signature under an aggregate key containing an honest share on a message the honest signer did not sign.

(A3) *Hashing.* $\mathsf{Hash}_{\mathrm{BTC}}$ and H are programmable random oracles (Section 3.1); in particular, they are preimage and collision resistant. CommitTable is H applied to the full output $\mathsf{ct}_{\mathsf{GC}} = (\mathsf{ct}_{\mathsf{C}}, \mathsf{ct}_{\mathsf{in}}, \mathsf{ct}_{\mathsf{DRE}})$ of GSM.Garble (Definition 2).

(A4) *Discrete logarithms.* DLOG is hard in secp256k1. Consequently, Feldman VSS is binding and computationally secret (Section 3.3), and SASig satisfies adaptability and full extractability (M4).

(A5) *Witness encryption.* Construction 2 is adaptively secure (Definition 7) in the generic bilinear group and random oracle models. We use the multi-candidate message-recovery form of this guarantee, Lemma 17.

(A6) *Garbling.* The binary garbling of Scheme 4 satisfies the adaptive privacy of Definition 2, and GSM as a whole, input tables included, is adaptively private for garbler-supplied input keys in the random oracle model [GKS+26, Thm. 5]. That is, there is a PPT simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that no PPT $\mathcal{A}$ distinguishes the following two experiments, in both of which the input keys $\{s_{j,b}\}$ are fixed in advance and hidden from $\mathcal{A}$, which receives only the active keys K_π, and only after choosing π. In the real experiment, $\mathcal{A}$ receives $\mathsf{ct}_{\mathsf{GC}} \leftarrow \mathsf{Garble}(r, \mathsf{seed}, \{s_{j,b}\})$, chooses π, and receives K_π. In the simulated experiment, $\mathcal{A}$ receives $\mathsf{ct}'_{\mathsf{GC}}$, where $(\mathsf{ct}'_{\mathsf{GC}}, \mathsf{st}) \leftarrow \mathsf{Sim}_1(1^\lambda)$, chooses π, and receives K_π after $\mathsf{Sim}_2(\mathsf{st}, \pi, f_r(\pi), K_\pi)$ has programmed the random oracle so that $\mathsf{Eval}(\mathsf{ct}'_{\mathsf{GC}}, K_\pi) = f_r(\pi)$, where $f_r(\pi) = r\pi$ if $\pi \in E(\mathbb{F}_p)$ and $\perp$ otherwise. Keys enter garbling and evaluation only as random-oracle inputs, through H_{key} in the input tables of Scheme 4, so the simulation is consistent as long as $\mathcal{A}$ queries the oracle at no key before receiving K_π, and at no key outside K_π afterwards. The statement covers the optimizations of Section 6.5. The active labels are the rows that the keys in K_π open, and by the share-

distinctness check no other row opens under them.

(A7) *Groth16.* Construction 1 is perfectly complete and knowledge sound in the generic bilinear group model [Gro16], with respect to benign auxiliary inputs [GKS+26].

The counting parameters N, n, and k_C are $\mathsf{poly}(\lambda)$, whereas N_{CC} and M_{CC} are fixed constants that determine the statistical parameter κ (Table 1) and are independent of λ. The message space satisfies $|\mathcal{M}| \geq 2^{\lambda}$. As in [GKS+26], extractors observe the adversary's random-oracle and generic-group queries in addition to its inputs and protocol transcript. We model the proof repository of Section 8.2.1 as a message from the prover-candidate to each verifier, so that the retrieved proof is part of the transcript.

9.2 Timelock Conditions

Each timelock must give an honest party enough time for its response transaction to confirm before the competing timeout leaf becomes spendable. The withdrawal window must also allow a full challenge round. Let τ_{eval} bound, in blocks, the time an honest prover needs for lines 16–27 of Protocol 4 (interpolation, evaluation of the M_{CC} finalized candidates, and decryption), and let τ_{setup} bound the duration of Protocol 1 from the confirmation of DepositRequest to the broadcast of Deposit when all parties are honest.

Definition 11 (Admissible timelocks)**.** *The public parameters are* admissible *if*

(T1) $\Delta_{\mathsf{refund}} > \tau_{\mathsf{setup}} + \Delta_{\mathsf{conf}}$,

(T2) $\Delta_{\mathsf{withdrawOpt}} > \Delta_{\mathsf{conf}}$,

(T3) $\Delta_{\mathsf{assert}} > \Delta_{\mathsf{conf}}$,

(T4) $\Delta_{\mathsf{noWithdraw}} > \tau_{\mathsf{eval}} + \Delta_{\mathsf{conf}}$, *and*

(T5) $\Delta_{\mathsf{withdraw}} > \Delta_{\mathsf{assert}} + \Delta_{\mathsf{noWithdraw}} + 4\Delta_{\mathsf{conf}}$.

Condition (T1) lets an all-honest setup forward the deposit before the refund leaf opens. Conditions (T2)–(T4) protect the party that must respond inside a window: the verifier's veto (ChallengeOptimistic against WithdrawOptimistic), the prover's Assert (against NoAssert), and the prover's WronglyChallenged (against NoWithdraw). Condition (T5) protects an honest verifier: the four confirmation delays account for ChallengeClaim, a late Assert, the ChallengeAssert chunks, and the slash itself (proof of Theorem 4). It makes precise the requirement stated in Section 8.2.2.

9.3 Structural Lemmas

The first lemma formalizes covenant emulation (Section 3.8).

Lemma 14 (Output binding)**.** *Suppose the committee setup completes and at least one operator is honest. Then, except with probability* $\mathsf{negl}(\lambda)$*, every confirmed transaction spending* Deposit[0] *is one of the* $2N$ *skeletons in* $\mathcal{S}$*, and the only confirmed key-path spend of* DepositRequest[0] *is* Deposit*.*

Proof. Deposit[0] is a script-path-only output, with a NUMS internal key and the single leaf $\mathsf{CheckSig}(\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}})$ (Protocol 6), so a spend requires a valid signature under $\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}}$ on the sighash of the spending transaction. Let O_{i} be honest. In Protocol 1, O_{i} contributes its share $\mathsf{sk}^{\mathsf{cov}}_{O_{\mathsf{i}}}$ to exactly $2N$ MuSig2 sessions, on the sighashes of $\mathsf{WithdrawOptimistic}^{\mathsf{j}}$ and $\mathsf{Withdraw}^{\mathsf{j}}$ for $\mathsf{j} \in [N]$, and deletes it in line 25. By (M2), these $2N$ sighashes are pairwise distinct and differ from the sighash of every other transaction. A confirmed spend of Deposit[0] by a transaction outside $\mathcal{S}$ therefore carries a valid signature under $\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}}$ on a message that O_{i} never signed, a forgery against MuSig2 with honest signer O_{i} and corrupt cosigners C, contradicting (A2). The key-path claim is identical for $\mathsf{sk}^{\mathsf{id}}_{O_{\mathsf{i}}}$, which O_{i} uses on the single message Deposit (line 26). □

Under static corruption, the deletion in line 25 is not needed for the argument, as an honest operator never signs with its share again. It makes the pre-signed set exhaustive also against a later compromise of the operator's storage.

Lemma 15 (Chokepoint)**.** *Suppose the committee setup completes and at least one operator is honest. Then, except with probability* $\mathsf{negl}(\lambda)$*, the following hold for every* $\mathsf{p} \in [N]$*.*

(i) At most one transaction of $\mathcal{S}$ *is ever confirmed.*

(ii) If $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ *is confirmed at height* g*, then* $\mathsf{ClaimOptimistic}^{\mathsf{p}}$ *was confirmed at a height* $g_0 \le g - \Delta_{\mathsf{withdrawOpt}}$*, and no other spend of* $\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$ *or* $\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$ *is ever confirmed. In particular, no* $\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$ *and no* $\mathsf{Claim}^{\mathsf{p}}$ *is ever confirmed.*

(iii) If $\mathsf{Withdraw}^{\mathsf{p}}$ *is confirmed at height* h*, then* $\mathsf{Claim}^{\mathsf{p}}$ *was confirmed at a height* $h_0 \le h - \Delta_{\mathsf{withdraw}}$*, and no other spend of* $\mathsf{Claim}^{\mathsf{p}}[0]$ *is ever confirmed. In particular, if* $\mathsf{NoAssert}_{(\mathsf{p},\mathsf{v})}$ *or* $\mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})}$ *is confirmed for some* v*, then no transaction of* $\mathcal{S}_{\mathsf{p}}$ *is ever confirmed.*

(iv) If O_{p} *is honest,* $\mathsf{Claim}^{\mathsf{p}}$ *is confirmed at height* h_0*, and no* $\mathsf{NoAssert}_{(\mathsf{p},\mathsf{v})}$ *or* $\mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})}$ *is ever confirmed, then* $\mathsf{Claim}^{\mathsf{p}}[0]$ *remains unspent until* O_{p} *spends it, and* $\mathsf{Withdraw}^{\mathsf{p}}$ *is valid at every height* $h \ge h_0 + \Delta_{\mathsf{withdraw}}$ *at which* $\mathsf{Deposit}[0]$ *is unspent.*

Proof. (i) Every transaction of $\mathcal{S}$ spends $\mathsf{Deposit}[0]$, a single output, and by Lemma 14 nothing outside $\mathcal{S}$ spends it.

(ii) The skeleton of $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ spends both $\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$ and $\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$ through their leaves 0, and the latter leaf carries $\mathsf{RelTimelock}(\Delta_{\mathsf{withdrawOpt}})$, which gives the height bound. Both outputs are spent at most once; $\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$ spends the former and $\mathsf{Claim}^{\mathsf{p}}$ the latter.

(iii) $\mathsf{Withdraw}^{\mathsf{p}}$ spends $\mathsf{Claim}^{\mathsf{p}}[0]$ through leaf 0, which carries $\mathsf{RelTimelock}(\Delta_{\mathsf{withdraw}})$; the slash transactions of every instance (p,v) spend the same output, through slash leaf v. $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ is excluded by (ii), since $\mathsf{Claim}^{\mathsf{p}}$ is confirmed.

(iv) The leaves of $\mathsf{Claim}^{\mathsf{p}}[0]$ are leaf 0, requiring O_{p}'s signature after the timelock, and the $N-1$ slash leaves, each requiring O_{p}'s signature together with one verifier's. On this output, an honest O_{p} signs only $\mathsf{Withdraw}^{\mathsf{p}}$ at runtime and the skeletons $\mathsf{NoAssert}_{(\mathsf{p},\mathsf{v})}$ and $\mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})}$ at setup ($P^{\mathsf{ps}}_{\mathsf{Setup}}$). By (A2) and (M2), every other spend is a forgery. Hence the output stays unspent until O_{p} spends it, and $\mathsf{Withdraw}^{\mathsf{p}}$, whose witness consists of $\sigma^{\mathsf{cov}}_{\mathsf{Withdraw},\mathsf{p}}$ from setup and O_{p}'s runtime signature, is valid once its timelock has elapsed. □

The last structural lemma is the cut-and-choose guarantee, stated for an honest prover-candidate against a malicious verifier. Fix an instance (p,v) with honest O_{p}, and let M_1 be the round-1 message that O_{p} receives in $P^{\mathsf{cc}}_{\mathsf{Setup}}$, consisting of the hashes $h_{\mathsf{msg},i}$, $h_{\mathsf{ct}_{\mathsf{setup}},i}$, and H_i of every candidate and the Feldman commitments $C_{t,\beta}$. By (A4), each $C_{t,\beta}$ determines a unique polynomial $f_{t,\beta} \in \mathbb{F}_{q_s}[X]_{\le d_{\mathsf{CC}}}$, the one whose coefficients are the discrete logarithms of its entries; we call these the *committed polynomials*. By (A3), except with probability $\mathsf{negl}(\lambda)$, every value that $\mathcal{A}$ ever exhibits as a preimage of a hash in M_1 is the unique such value among $\mathcal{A}$'s oracle queries preceding M_1. We call these the *committed values* $(\mathsf{msg}_i, \mathsf{ct}_{\mathsf{setup},i}, \mathsf{ct}_{\mathsf{GC},i})$ of candidate i (some possibly undefined), and say that candidate i is *well-formed* if there exist $r_i \in \mathbb{F}_q^*$ and seed_i with

$$\mathsf{ct}_{\mathsf{setup},i} = \mathsf{WE}.\mathsf{Enc}_{\mathsf{setup}}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \mathsf{msg}_i; r_i) \quad \text{and} \quad \mathsf{ct}_{\mathsf{GC},i} = \mathsf{GSM}.\mathsf{Garble}\big(r_i, \mathsf{seed}_i, \{f_{t,\beta}(i)\}_{t\in[2n],\,\beta\in\{0,1\}}\big).$$

Well-formedness is a property of a single candidate. The share-distinctness check additionally checks the polynomials shared across candidates (Section 8.1.2): as discrete logarithms are unique, pairwise distinct $C_{t,\beta}(i)$ imply $f_{t,0}(i) \ne f_{t,1}(i)$ for every wire t and every candidate i, finalized or opened, so in every candidate the two rows of each input table are encrypted under distinct keys (the invariant of Section 8.1.2), and $T_{t,0} \ne T_{t,1}$ for every t.

Lemma 16 (Cut-and-choose)**.** *Let O_{p} be honest. Except with probability $\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}^{-1} + \mathsf{negl}(\lambda) \leq 2^{-\kappa} + \mathsf{negl}(\lambda)$, if $P^{\mathsf{cc}}_{\mathsf{Setup}}$ does not abort, then some finalized candidate $i \in \mathcal{I}$ is well-formed, and for every well-formed $i \in \mathcal{I}$ the ciphertexts stored in st^P are the committed values $(\mathsf{ct}_{\mathsf{setup},i}, \mathsf{ct}_{\mathsf{GC},i})$.*

Proof. Let $W \subseteq [N_{\mathsf{CC}}]$ be the set of ill-formed candidates. W is determined by M_1 together with $\mathcal{A}$'s oracle queries preceding it, whereas O_{p} samples $\mathcal{I}$ uniformly after receiving M_1. An opened candidate $i \notin \mathcal{I}$ passes the checks of $P^{\mathsf{cc}}_{\mathsf{Setup}}$ only if its opening $(\mathsf{msg}_i, r_i, \mathsf{seed}_i, s_i)$ consists of committed values (uniqueness of preimages), the shares s_i are the evaluations $f_{t,\beta}(i)$ of the committed polynomials (Feldman check), and the ciphertexts recomputed from them hash to $h_{\mathsf{ct}_{\mathsf{setup}},i}$ and H_i, i.e., only if i is well-formed. Hence no abort implies $W \subseteq \mathcal{I}$, while "no finalized candidate is well-formed" means $\mathcal{I} \subseteq W$. Both hold only if $\mathcal{I} = W$, which for a fixed W has probability at most $\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}^{-1}$ over the choice of $\mathcal{I}$. For the second claim, the round-2 checks $\mathsf{CommitTable}(\mathsf{ct}_{\mathsf{GC},i}) = H_i$ and $H(\mathsf{ct}_{\mathsf{setup},i}) = h_{\mathsf{ct}_{\mathsf{setup}},i}$ for $i \in \mathcal{I}$ force the received ciphertexts to equal the committed values, again by uniqueness of preimages. □

9.4 Setup Safety and Non-Triviality

Recall from Protocol 1 that an honest operator releases its partial signature on Deposit (line 26) only after it has verified all $2N$ covenant signatures (line 23) and completed the pairwise setups of all its instances, and that from then on its output is decided by the ledger: it outputs its state upon observing Deposit confirmed (line 29) and $\perp$ upon observing DepositRefund confirmed.

Theorem 2 (Setup safety and non-triviality)**.** *Under (A1)–(A4) and admissible timelocks, Protocol 1 satisfies Definitions 3 and 6.*

Proof. Non-triviality. Suppose all parties are honest. Every check of Protocols 1–3 and 7 passes by construction: the Feldman checks by the homomorphism of Scheme 2, the hash checks because the openings are the committed values, the re-garbling check because Garble is deterministic in $(r_i, \mathsf{seed}_i, s_i)$, the adaptor checks by the correctness of SASig, the pre-signature checks by the correctness of $\mathsf{Sig}_{\mathrm{BTC}}$, and the signature verifications by the correctness of MuSig2. The share-distinctness check fails with probability below 2^{-220} (Section 8.1.2). Hence Deposit is broadcast by height $h_{\mathsf{DR}}+\tau_{\mathsf{setup}}$, where h_{DR} is the height of DepositRequest. It is valid, and the only other spend of DepositRequest[0], through the refund leaf, is invalid before height $h_{\mathsf{DR}}+\Delta_{\mathsf{refund}} > h_{\mathsf{DR}}+\tau_{\mathsf{setup}}+\Delta_{\mathsf{conf}}$ by (T1). By liveness, Deposit is confirmed, and every operator outputs its state in line 29.

Setup safety. Let O_{i} be honest and suppose the setup aborts. If O_{i} stops before line 26, it never releases a partial signature under $\mathsf{sk}^{\mathsf{id}}_{O_{\mathsf{i}}}$, so by (A2) no valid signature under $\mathsf{pk}^{\mathsf{id}}_{\mathcal{O}}$ on any message exists, and DepositRequest[0] can be spent only through its refund leaf. DepositRefund is valid from height $h_{\mathsf{DR}} + \Delta_{\mathsf{refund}}$ on and has no possible conflicting spend, so by liveness U recovers B by height $h_{\mathsf{DR}} + \Delta_{\mathsf{refund}} + \Delta_{\mathsf{conf}}$. If O_{i} reaches line 26, its transaction set is complete: it holds the $2N$ verified covenant signatures and the states of all its instances, in which the complete counterparty pre-signature set for its role has been verified ($\mathsf{VerifySigs}_\rho$ and VerAdaptor). $\mathcal{A}$ may now hold a valid $\sigma^{\mathsf{id}}_{\mathsf{Deposit}}$ even if the copy O_{i} received is invalid, but O_{i} outputs $\perp$ only upon observing DepositRefund confirmed, in which case U has recovered B, and outputs its complete state upon observing Deposit confirmed, in which case the setup completes. The two transactions conflict on DepositRequest[0], so exactly one of them is confirmed, and DepositRefund is valid from height $h_{\mathsf{DR}} + \Delta_{\mathsf{refund}}$ on.

No honest operator loses funds in either case. The only funds an operator brings to the setup are its funding outpoint $u_{O_{\mathsf{i}}}$, which remains under its control: it releases no signature spending this outpoint during setup and signs $\mathsf{ClaimOptimistic}^{\mathsf{i}}$ only at broadcast time. Every pre-signature O_{i} issues (σ^P_{X} for $\mathsf{X} \in \{\mathsf{ChallengeOptimistic}, \mathsf{ChallengeClaim}, \mathsf{NoAssert}, \mathsf{NoWithdraw}\}$ as

prover-candidate, $\sigma^{V}_{\mathsf{Assert}}$ and $\sigma^{V}_{\mathsf{ChallengeOpt}}$ as verifier) authorizes a spend of a connector funded by the prover-candidate's own dust, and the covenant and identity signatures authorize spends of the user's deposit only along $\mathcal{S}$ and $\mathsf{Deposit}$ (Lemma 14). □

9.5 Completeness

Theorem 3 (Completeness)**.** *Under (A1)–(A4) and (A7), with admissible timelocks,* TRITONE *satisfies Definition 4 with bound* $u = \Delta_{\mathsf{withdrawOpt}} + \Delta_{\mathsf{withdraw}} + 4\Delta_{\mathsf{conf}}$ *blocks, counted from the broadcast of* $\mathsf{ClaimOptimistic}^{\mathsf{p}}$*.*

Proof. Let O_{p} be honest with $(\boldsymbol{x}_{\mathsf{p}}, \boldsymbol{w}) \in \mathcal{R}$, and let $\mathcal{A}$ control all other operators. Since the setup completed, O_{p} holds $\sigma^{\mathsf{cov}}_{\mathsf{WithdrawOpt,p}}$, $\sigma^{\mathsf{cov}}_{\mathsf{Withdraw,p}}$, and, for every v, a state $\mathsf{st}^{P}_{(\mathsf{p},\mathsf{v})}$ all of whose checks passed. As O_{p} Lamport-signs the same π_1 toward every verifier, the only preimages of $\mathsf{lpk}_{O_{\mathsf{p}}}$ that are ever published are those selected by π_1 (Section 8.4). Heights are taken in O_{p}'s view. By Lemma 14, the only possible conflicting spend of $\mathsf{Deposit}[0]$ is a transaction of $\mathcal{S}_{\mathsf{p}'}$ for some $\mathsf{p}' \neq \mathsf{p}$. If such a transaction is confirmed, the exception in the completeness definition applies. For the rest of the proof, we assume that no such transaction is confirmed.

$\mathsf{ClaimOptimistic}^{\mathsf{p}}$ spends O_{p}'s own outpoint and has no conflicting spend, so it is confirmed at some height g_0 at most Δ_{conf} after its broadcast.

Optimistic path. If no spend of $\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$ is confirmed by height $g_0 + \Delta_{\mathsf{withdrawOpt}}$, then O_{p} broadcasts $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ at that height. It is valid: its witness holds the covenant signature and O_{p}'s runtime signatures, and its timelock has elapsed. Its inputs other than $\mathsf{Deposit}[0]$ are $\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$, otherwise spendable only through the verifier leaves ($\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$), and $\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$, otherwise spendable only by O_{p} (leaf 1). Hence either $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$, a transaction of $\mathcal{S}_{\mathsf{p}}$, is confirmed by $g_0 + \Delta_{\mathsf{withdrawOpt}} + \Delta_{\mathsf{conf}}$, or some $\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$ is confirmed by that height, which is the case treated next.

Challenge path. Suppose some $\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$ is confirmed at a height $g_1 \leq g_0 + \Delta_{\mathsf{withdrawOpt}} + \Delta_{\mathsf{conf}}$. O_{p} broadcasts $\mathsf{Claim}^{\mathsf{p}}$, which spends $\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$ through leaf 1; both leaves of this output require O_{p}'s signature, so there is no conflicting spend, and $\mathsf{Claim}^{\mathsf{p}}$ is confirmed at some $h_0 \leq g_1 + \Delta_{\mathsf{conf}}$. We show that no slash leaf of $\mathsf{Claim}^{\mathsf{p}}[0]$ is ever spent, by considering an arbitrary instance (p, v).

(a) $\mathsf{NoAssert}_{(\mathsf{p},\mathsf{v})}$ *is never confirmed.* It spends $\mathsf{ChallengeClaim}[0]$ through leaf 0, so it requires $\mathsf{ChallengeClaim}$ confirmed at some height h_1 and is invalid before $h_1 + \Delta_{\mathsf{assert}}$. Upon observing $\mathsf{ChallengeClaim}$, O_{p} broadcasts Assert with witness $(\mu, \sigma^{V}_{\mathsf{Assert}})$. It is valid, since μ is O_{p}'s Lamport signature on π_1 and $\sigma^{V}_{\mathsf{Assert}}$ was verified at setup. The conflicting spends of $\mathsf{ChallengeClaim}[0]$ are leaf 0, invalid before $h_1 + \Delta_{\mathsf{assert}} > h_1 + \Delta_{\mathsf{conf}}$ by (T3), and leaf 1 with a transaction $\mathsf{X}' \neq \mathsf{Assert}$, which $\mathcal{A}$ can build from the preimages of π_1 in O_{p}'s broadcast Assert transactions, since it holds $\mathsf{sk}_{O_{\mathsf{v}}}$. Thus by $h_1 + \Delta_{\mathsf{conf}}$, either Assert or such an X' is confirmed, and in both cases $\mathsf{ChallengeClaim}[0]$ is spent before $\mathsf{NoAssert}$ becomes valid. In the second case, no $\mathsf{Assert}[j]$ output exists, so neither a $\mathsf{ChallengeAssert}^{j}$ nor $\mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})}$ can ever be confirmed, and the instance is settled in O_{p}'s favor.

(b) $\mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})}$ *is never confirmed.* It spends every $\mathsf{ChallengeAssert}^{j}[0]$ through leaf 0. If some $\mathsf{Assert}[j]$ is never spent, or is spent by a transaction other than $\mathsf{ChallengeAssert}^{j}$, then $\mathsf{NoWithdraw}$ lacks an input and is never valid. So suppose all k_C chunks are confirmed, the latest at height h_3; $\mathsf{NoWithdraw}$ is then invalid before $h_3 + \Delta_{\mathsf{noWithdraw}}$. Each chunk is a spend of the gadget output $\mathsf{Assert}[j]$ of the multi-bit gadget of Section 8.1.3 with honest prover O_{p}. By Lemma 12(ii) (Corollary 2), every executed branch (t, β_t) has $\beta_t = \pi_{1,t}$, the only polarity whose preimage O_{p} has revealed, and by Lemma 11, $y_t := \mathsf{SASig.Extract}(\tilde{\sigma}_{t,\pi_{1,t}}, \sigma_t)$ satisfies

$[y_t] = T_{t,\pi_{1,t}}$, both except with probability $\mathsf{negl}(\lambda)$. Since $T_{t,\pi_{1,t}} = C_{t,\pi_{1,t}}(0) = [f_{t,\pi_{1,t}}(0)]$ for the committed polynomial and discrete logarithms are unique, $y_t = f_{t,\pi_{1,t}}(0)$. Together with the d_{CC} shares $f_{t,\pi_{1,t}}(i)$, $i \notin \mathcal{I}$, verified in $P^{\mathsf{cc}}_{\mathsf{Setup}}$, this determines $f_{t,\pi_{1,t}}$, so the interpolation in line 16 recovers the committed polynomial, and $K_i = ((\pi_{1,t}, f_{t,\pi_{1,t}}(i)))_t$ is candidate i's active key set for every $i \in \mathcal{I}$. By Lemma 16, except with probability $2^{-\kappa}+\mathsf{negl}(\lambda)$ some $i^* \in \mathcal{I}$ is well-formed and O_{p} holds its committed ciphertexts. For it, GSM.$\mathsf{Eval}(\mathsf{ct}_{\mathsf{GC},i^*}, K_{i^*}) = r_{i^*}\pi_1$ by the correctness of GSM ($\pi_1 \in E(\mathbb{F}_p)$, and the active rows of a well-formed candidate were encrypted under exactly the keys in K_{i^*}, so they open), and WE.Dec returns msg_{i^*} by the correctness of WE: Groth16.$\mathsf{Verify}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \pi) = 1$ by (A7), which implies $e(r_{i^*}\pi_1, \pi_2) - e(\pi_3, r_{i^*}[\delta]_2) = r_{i^*}Y$. As $\mathsf{Hash}_{\mathrm{BTC}}(\mathsf{msg}_{i^*}) = h_{\mathsf{msg},i^*}$, O_{p} broadcasts $\mathsf{WronglyChallenged}$ by height $h_3+\tau_{\mathsf{eval}}$ (line 27). It is valid, spending every $\mathsf{ChallengeAssert}^j[0]$ through leaf 1 with a preimage and O_{p}'s signature. The only conflicting spends are through leaf 0 of some $\mathsf{ChallengeAssert}^j[0]$, which requires O_{v}'s signature and $\Delta_{\mathsf{noWithdraw}}$ blocks since that chunk: $\mathsf{NoWithdraw}$ itself is invalid before $h_3 + \Delta_{\mathsf{noWithdraw}} > h_3 + \tau_{\mathsf{eval}} + \Delta_{\mathsf{conf}}$ by (T4), and any other leaf-0 spend of an earlier chunk's output consumes an input of $\mathsf{NoWithdraw}$. Hence either $\mathsf{WronglyChallenged}$ is confirmed by $h_3+\tau_{\mathsf{eval}}+\Delta_{\mathsf{conf}}$, or some input of $\mathsf{NoWithdraw}$ is spent otherwise, and in both cases $\mathsf{NoWithdraw}$ is never valid.

By (a) and (b) for every v, no slash leaf of $\mathsf{Claim}^{\mathsf{p}}[0]$ is ever spent, so by Lemma 15(iv) $\mathsf{Withdraw}^{\mathsf{p}}$ is valid from height $h_0+\Delta_{\mathsf{withdraw}}$ on, with $\mathsf{Deposit}[0]$ its only possible point of conflict. O_{p} broadcasts it at that height, and it is confirmed by $h_0+\Delta_{\mathsf{withdraw}}+\Delta_{\mathsf{conf}} \le g_0+\Delta_{\mathsf{withdrawOpt}}+\Delta_{\mathsf{withdraw}}+3\Delta_{\mathsf{conf}}$, i.e., within u blocks of the broadcast of $\mathsf{ClaimOptimistic}^{\mathsf{p}}$. By a union bound over the $N-1$ instances in which O_{p} is the prover-candidate, applying Lemma 16 to each, the failure probability is at most $(N-1)\cdot 2^{-\kappa} + \mathsf{negl}(\lambda)$. □

This exception distinguishes the multi-party setting from BABE. The deposit output can be spent only once, so a corrupt operator that withdraws first prevents the honest operator from withdrawing. By knowledge soundness, the corrupt operator must know a witness for its own statement, except with the theorem's error probability. Whether both operators' statements can have valid witnesses depends on $\mathcal{R}$ and $\mathsf{GenStmt}$.

9.6 Knowledge Soundness

We first isolate the cryptographic core: a prover that recovers the message of a finalized candidate knows a witness. The first lemma is the guarantee of the witness encryption scheme in the form we need. It differs from Definition 7 in three respects: the adversary must recover a uniformly random message rather than distinguish two chosen ones, it faces N_{CC} independent candidates and may open those outside a set $\mathcal{I}$ of its choice, and extraction is required to succeed whenever recovery does. The last point is essential. Definition 5 conditions on the adversary's success, so a reduction whose extractor succeeds with probability $\varepsilon/(4N_{\mathsf{CC}})$ when the adversary succeeds with probability ε (the black-box route via Definition 7, embedding one challenge in a guessed candidate) would not establish it, even though $4N_{\mathsf{CC}}$ is a constant.

Lemma 17 (Multi-candidate message recovery)**.** *Consider the following game between a challenger and a PPT adversary $\mathcal{A}$. The challenger samples* $\mathsf{crs} \leftarrow \mathsf{Gen}(\mathcal{R})$*, and $\mathcal{A}$ outputs a statement $\boldsymbol{x}$. For every $i \in [N_{\mathsf{CC}}]$, the challenger samples* $\mathsf{msg}_i \xleftarrow{\$} \mathcal{M}$ *and* $r_i \xleftarrow{\$} \mathbb{F}_q^*$ *and sends* $\mathsf{ct}_{\mathsf{setup},i} \leftarrow$ WE.$\mathsf{Enc}_{\mathsf{setup}}(\mathsf{crs}, \boldsymbol{x}, \mathsf{msg}_i; r_i)$ *and* $h_i = \mathsf{Hash}_{\mathrm{BTC}}(\mathsf{msg}_i)$*. $\mathcal{A}$ outputs $\mathcal{I} \subseteq [N_{\mathsf{CC}}]$ and receives (msg_i, r_i) for $i \notin \mathcal{I}$. $\mathcal{A}$ outputs π_1 and receives $\mathsf{ct}_{\mathsf{prove},i} = r_i\pi_1$ for $i \in \mathcal{I}$ (or $\perp$ if $\pi_1 \notin \mathbb{G}_1$). Finally, $\mathcal{A}$ outputs (i^*, msg') with $i^* \in \mathcal{I}$ and* wins *if* $\mathsf{Hash}_{\mathrm{BTC}}(\mathsf{msg}') = h_{i^*}$*. In the generic bilinear group and random*

oracle models, for every such $\mathcal{A}$ there is a PPT extractor $\mathcal{E}$, given $\mathcal{A}$'s view and oracle queries, such that

$$\Pr\left[\mathcal{A} \textit{ wins} \wedge (\boldsymbol{x}, \mathcal{E}(\mathsf{view}_{\mathcal{A}})) \notin \mathcal{R}\right] \leq \mathsf{negl}(\lambda).$$

Proof sketch. The argument is that of [GKS+26, Lem. 2], applied to every finalized candidate. Fix $i \in \mathcal{I}$. The component $\mathsf{ct}_{\mathsf{msg},i} = H(r_i Y) + \mathsf{msg}_i$ is a one-time pad of msg_i unless $\mathcal{A}$ queries H at $r_i Y$, and msg_i is uniform in $\mathcal{M}$. If $\mathcal{A}$ never queries $r_i Y$, then msg_i enters its view only through h_i, so exhibiting a preimage of h_i under the random oracle $\mathsf{Hash}_{\mathrm{BTC}}$ succeeds with probability at most $(q+1)(|\mathcal{M}|^{-1} + 2^{-\lambda})$ for q oracle queries, i.e., negligibly. Hence, up to a negligible term, $\mathcal{A}$ wins only if it queries $r_{i^*} Y$ for its output index i^*. In the generic group model, every $\mathbb{G}_T$ element $\mathcal{A}$ forms has a representation as a linear combination of pairings of received handles, and the only handles containing the formal variable r_i for $i \in \mathcal{I}$ are $\mathsf{ct}_{\mathsf{rd},i} = r_i[\delta]_2$ and $\mathsf{ct}_{\mathsf{prove},i} = r_i \pi_1$ (the opened r_i, $i \notin \mathcal{I}$, are known scalars). A query equal to $r_i Y$ therefore has the form $e(\mathsf{ct}_{\mathsf{prove},i}, [a_2]_2) + e([a_3]_1, \mathsf{ct}_{\mathsf{rd},i})$ up to terms free of r_i, with $[a_2]_2$ and $[a_3]_1$ computable from the representation, and $Y = e(\pi_1, [a_2]_2) + e([a_3]_1, [\delta]_2)$ as formal polynomials, except with the negligible probability that two distinct formal polynomials collide (the master theorem of [GKS+26]). Thus $(\pi_1, [a_2]_2, -[a_3]_1)$ satisfies the Groth16 verification equation, and the knowledge extractor of (A7) returns $\boldsymbol{w}$ with $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}$, except with probability $\mathsf{negl}(\lambda)$. The extractor $\mathcal{E}$ runs over all of $\mathcal{A}$'s random-oracle queries: for each query and each $i \in \mathcal{I}$, it reads off $([a_2]_2, [a_3]_1)$ from the query's representation, checks the verification equation, and on success runs the Groth16 extractor. It needs neither r_i nor the challenger's randomness. A union bound over the N_{CC} candidates and the q queries completes the proof. We treat each DRE-decoded $r_i \pi_1$ as a handle in the generic bilinear group model. □

The next lemma transfers this guarantee to the protocol. The *recovery game* $\mathsf{G}_{\mathsf{rec}}$ for an instance (p, v) is played between a challenger, who runs all of O_{v}'s algorithms (in particular $V^{\mathsf{cc}}_{\mathsf{Setup}}$, $V^{\mathsf{ps}}_{\mathsf{Setup}}$, and V_{Prove} for that instance), and a PPT adversary $\mathcal{A}$, who plays every other party, supplies O_{v}'s view of the ledger (in particular the Assert witness from which O_{v} reads the preimages), and receives everything O_{v} sends or broadcasts. $\mathcal{A}$ wins if it outputs msg' with $\mathsf{Hash}_{\mathrm{BTC}}(\mathsf{msg}') = h_{\mathsf{msg},i}$ for some $i \in \mathcal{I}$. The game gives $\mathcal{A}$ at least the view of a malicious prover-candidate in a real execution with honest O_{v}.

Lemma 18 (Finalized-candidate extractability)**.** *Under (A3)–(A7), for every PPT adversary $\mathcal{A}$ in $\mathsf{G}_{\mathsf{rec}}$ there is a PPT extractor $\mathcal{E}$, given $\mathcal{A}$'s view and oracle queries, such that*

$$\Pr\left[\mathcal{A} \textit{ wins} \wedge (\boldsymbol{x}_{\mathsf{p}}, \mathcal{E}(\mathsf{view}_{\mathcal{A}})) \notin \mathcal{R}\right] \leq \mathsf{negl}(\lambda).$$

Proof sketch. We move from $\mathsf{G}_{\mathsf{rec}}$ to the game of Lemma 17 through a sequence of games, each computationally indistinguishable from the previous one for $\mathcal{A}$. Membership in $\mathcal{R}$ is efficiently checkable, so the event "$\mathcal{A}$ wins and $\mathcal{E}$ fails" is efficiently recognizable from $\mathcal{A}$'s view and the oracle transcripts, which the reductions below control, and its probability changes by at most $\mathsf{negl}(\lambda)$ across each hop. The extractor $\mathcal{E}$ is that of Lemma 17 in every game.

Game G_0 is $\mathsf{G}_{\mathsf{rec}}$.

Game G_1 (equivocal commitments). The challenger sends uniformly random strings in place of $h_{\mathsf{ct}_{\mathsf{setup}},i}$ and H_i in round 1 and, when it sends the round-2 values, programs H to map them to these strings (A3). The games differ only if $\mathcal{A}$ has queried H at a round-2 value before round 2; every such value contains the fresh r_i or seed_i, so this happens with probability $\mathsf{negl}(\lambda)$. The challenger may now decide after learning $\mathcal{I}$ how the finalized candidates are formed, and $\mathcal{A}$'s view up to its choice of $\mathcal{I}$ is independent of that decision.

The label-query event. Let Bad be the event that $\mathcal{A}$ queries the random oracle at an input key $f_{t,\beta}(i)$ of a finalized candidate $i \in \mathcal{I}$, either before O_{v} reveals the zero-shares in $\mathsf{ChallengeAssert}$ or afterwards with $\beta \neq \pi_{1,t}$, where π_1 is the string revealed in Assert. We claim

$$\Pr[\mathsf{Bad}] \leq 4n\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}} \cdot \mathbf{Adv}^{\mathsf{DLOG}}_{\mathcal{B}}(\lambda) + \mathsf{negl}(\lambda)$$

for the following $\mathcal{B}$. On a challenge $Z = [z]$, $\mathcal{B}$ guesses a set $\mathcal{I}^*$ of size M_{CC}, a wire t^*, and a polarity β^*, all uniformly. It plays the challenger of G_1, except for the polynomial f_{t^*,β^*}: it samples the shares $f_{t^*,\beta^*}(i)$ for $i \notin \mathcal{I}^*$ uniformly, sets $C_{t^*,\beta^*}(0) := Z$, and derives the coefficient commitments C_{t^*,β^*} by Lagrange interpolation in the group. This vector has the distribution of an honest commitment, so $\mathcal{A}$'s view is independent of the guesses until $\mathcal{B}$ has to open a share it does not know or to reveal the zero-share $f_{t^*,\beta^*}(0)$, at which point it aborts. Thus $\mathcal{B}$ survives whenever $\mathcal{I} = \mathcal{I}^*$ and $\pi_{1,t^*} \neq \beta^*$, with probability exactly $\left(2\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}\right)^{-1}$, independently of $\mathcal{A}$'s behavior. Whenever $\mathcal{A}$ queries the oracle at a value z' with $[z'] = C_{t^*,\beta^*}(i)$ for some $i \in \mathcal{I}$, $\mathcal{B}$ interpolates $f_{t^*,\beta^*}(0) = z$ from z' and its d_{CC} chosen shares and outputs it. If Bad occurs, its first query concerns wire t^* with probability at least $1/(2n)$, which gives the claim. The factor $\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}$ is a constant independent of λ, so $\Pr[\mathsf{Bad}]$ is negligible. This reduction guesses the finalized set before sending the commitments. The guess contributes a factor $\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}$ to the reduction loss, corresponding to $\log_2 \binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}}$ bits of concrete tightness.

Game G_2 (simulated garbling). For every finalized candidate $i \in \mathcal{I}$, the challenger sends in round 2 a garbled circuit $\mathsf{ct}_{\mathsf{GC},i}$ produced by Sim_1. When O_{v} completes the adaptor signatures for the string π_1 revealed in Assert, the challenger runs Sim_2 on π_1, on $f_{r_i}(\pi_1)$, and on the actual active keys $K_i = ((\pi_{1,t}, f_{t,\pi_{1,t}}(i)))_t$, which are fixed by the Feldman polynomials and become computable to $\mathcal{A}$ exactly through the revealed zero-shares. Outside Bad, this is the setting of (A6), so a hybrid over the M_{CC} finalized candidates bounds the distinguishing probability by $\Pr[\mathsf{Bad}] + M_{\mathsf{CC}} \cdot \mathsf{negl}(\lambda)$.

In G_2, the challenger uses the witness-encryption material of the finalized candidates only through $\mathsf{ct}_{\mathsf{setup},i}$, $h_{\mathsf{msg},i}$, and $r_i\pi_1$, and opens (msg_i, r_i) for $i \notin \mathcal{I}$. Everything else it produces (keys, Feldman polynomials and their zero-shares, pre-signatures and adaptor completions, the simulated garbled circuits, the programmed oracle values, and O_{v}'s actions in its other instances) is generated independently of $\{\mathsf{msg}_i, r_i\}_{i \in \mathcal{I}}$. A reduction to the game of Lemma 17 therefore forwards $\boldsymbol{x}_{\mathsf{p}}$, the ciphertexts and hashes, the set $\mathcal{I}$, and the revealed π_1, and simulates the rest itself. $\mathcal{A}$ winning G_2 is winning that game, so Lemma 17 bounds $\Pr[\mathcal{A}\text{ wins} \wedge \mathcal{E}\text{ fails}]$ in G_2 by $\mathsf{negl}(\lambda)$, and hence in G_0. □

Theorem 4 (Knowledge soundness)**.** *Under (A1)–(A7), with admissible timelocks,* Tritone *satisfies Definition 5. The extraction error is in fact* $\mathsf{negl}(\lambda)$*: the* $2^{-\kappa}$ *term of the definition is not needed, as cut-and-choose protects only the honest prover.*

Proof. Let O_{v} be honest, let $\mathcal{A}$ control all other operators, and suppose a transaction of $\mathcal{S}_{\mathsf{p}}$ is confirmed. The extractor $\mathcal{E}$ runs the Groth16 extractor of (A7) on the proof $\mathcal{A}$ delivered to O_{v}, if any, and the extractor of Lemma 18 on $\mathcal{A}$'s view, and outputs whichever result satisfies $\mathcal{R}$. By Lemma 15(i), exactly one transaction of $\mathcal{S}_{\mathsf{p}}$ is confirmed, and we show in each of the two cases that one of the extractors succeeds, except with probability $\mathsf{negl}(\lambda)$. Heights are taken in O_{v}'s view.

Case $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ *confirmed.* By Lemma 15(ii), $\mathsf{ClaimOptimistic}^{\mathsf{p}}$ was confirmed at some g_0, and $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ no earlier than $g_0 + \Delta_{\mathsf{withdrawOpt}}$. Upon observing $\mathsf{ClaimOptimistic}^{\mathsf{p}}$, O_{v} retrieves the proof and, if none verifies, broadcasts $\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$, valid with the two pre-signatures it holds. Its only input is $\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$, whose other spenders all conflict with $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ as well, so by liveness some transaction other than $\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ would spend this output by $g_0 + \Delta_{\mathsf{conf}} < g_0 + \Delta_{\mathsf{withdrawOpt}}$, using (T2), contradicting Lemma 15(ii). Hence

O_{v} retrieved a proof π with $\mathsf{Groth16.Verify}(\mathsf{crs}, \boldsymbol{x}_{\mathsf{p}}, \pi) = 1$, which is part of the transcript, and the Groth16 extractor returns a witness, except with probability $\mathsf{negl}(\lambda)$.

Case $\mathsf{Withdraw}^{\mathsf{p}}$ *confirmed at height* h. By Lemma 15(iii), $\mathsf{Claim}^{\mathsf{p}}$ was confirmed at some $h_0 \leq h - \Delta_{\mathsf{withdraw}}$, and $\mathsf{Claim}^{\mathsf{p}}[0]$ was spent by no other transaction. We follow V_{Prove} and show that by height $h_4 := h_0 + \Delta_{\mathsf{assert}} + \Delta_{\mathsf{noWithdraw}} + 4\Delta_{\mathsf{conf}}$, which is below h by (T5), either a slash leaf of $\mathsf{Claim}^{\mathsf{p}}[0]$ is spent or some $\mathsf{ChallengeAssert}^j[0]$ is spent through leaf 1. At each step we list the spends that can conflict with O_{v}'s transaction. A spend of $\mathsf{Claim}^{\mathsf{p}}[0]$ through another slash leaf establishes the claim directly, and $\mathsf{Withdraw}^{\mathsf{p}}$ is invalid before $h_0 + \Delta_{\mathsf{withdraw}} > h_4$, so neither is listed again.

(a) Upon observing $\mathsf{Claim}^{\mathsf{p}}$, O_{v} broadcasts $\mathsf{ChallengeClaim}$, valid with $\sigma^{P}_{\mathsf{ChallengeClaim}}$ from setup. Its input $\mathsf{Claim}^{\mathsf{p}}[1{+}\mathsf{rk}(\mathsf{p},\mathsf{v})]$ is otherwise spendable only through leaf 1, which is invalid before $h_0 + \Delta_{\mathsf{withdraw}}$. So $\mathsf{ChallengeClaim}$ is confirmed at some $h_1 \leq h_0 + \Delta_{\mathsf{conf}}$.

(b) If Assert is not confirmed by $h_1 + \Delta_{\mathsf{assert}}$, O_{v} broadcasts $\mathsf{NoAssert}$, valid with $\sigma^{P}_{\mathsf{NoAssert}}$ and its own signatures. Its input $\mathsf{ChallengeClaim}[0]$ is otherwise spendable only through leaf 1, which requires a signature under $\mathsf{pk}_{O_{\mathsf{v}}}$ on the spending transaction. O_{v} has signed exactly the skeleton Assert on this output through this leaf, so by (A2) and (M2) the only such spend is Assert. Hence by $h_2 := h_1 + \Delta_{\mathsf{assert}} + \Delta_{\mathsf{conf}}$, either $\mathsf{NoAssert}$ is confirmed (a slash) or Assert is confirmed at some $h_A \leq h_2$.

(c) Upon observing Assert, O_{v} reads the preimages μ_t from its witness, which fix a string $\pi_1 \in \{0,1\}^{2n}$, completes the adaptor signatures σ_t of the polarities $\pi_{1,t}$ with the zero-shares $f_{t,\pi_{1,t}}(0)$, and broadcasts the k_C chunks $\mathsf{ChallengeAssert}^j$. Each is a spend of the gadget output $\mathsf{Assert}[j]$ tail-authorized by the honest verifier O_{v}: by Lemma 10, its witness satisfies Φ_j, and by Lemma 12(i) (Corollary 2), no spend of $\mathsf{Assert}[j]$ not tail-authorized by O_{v} is confirmed. Hence all chunks are confirmed by $h_3 := h_A + \Delta_{\mathsf{conf}} \leq h_0 + \Delta_{\mathsf{assert}} + 3\Delta_{\mathsf{conf}}$.

(d) If every $\mathsf{ChallengeAssert}^j[0]$ is still unspent $\Delta_{\mathsf{noWithdraw}}$ blocks after the latest chunk, O_{v} broadcasts $\mathsf{NoWithdraw}$, valid with $\sigma^{P}_{\mathsf{NoWithdraw}}$ and its own signatures. Each input $\mathsf{ChallengeAssert}^j[0]$ is otherwise spendable through leaf 0 only with O_{v}'s signature, which it has issued for no other transaction, or through leaf 1. Hence by $h_4 \geq h_3 + \Delta_{\mathsf{noWithdraw}} + \Delta_{\mathsf{conf}}$, either $\mathsf{NoWithdraw}$ is confirmed (a slash) or some $\mathsf{ChallengeAssert}^j[0]$ is spent through leaf 1.

Since $\mathsf{Claim}^{\mathsf{p}}[0]$ was spent only by $\mathsf{Withdraw}^{\mathsf{p}}$, no slash was confirmed, so some $\mathsf{ChallengeAssert}^j[0]$ was spent through leaf 1, whose witness contains, by (M1), a value msg' with $\mathsf{Hash}_{\mathrm{BTC}}(\mathsf{msg}') = h_{\mathsf{msg},i^*}$ for some $i^* \in \mathcal{I}$. The execution up to this point is an execution of $\mathsf{G}_{\mathsf{rec}}$ in which $\mathcal{A}$, playing O_{p}, all other operators, and the ledger, wins. By Lemma 18, $\mathcal{E}$ outputs $\boldsymbol{w}$ with $(\boldsymbol{x}_{\mathsf{p}}, \boldsymbol{w}) \in \mathcal{R}$, except with probability $\mathsf{negl}(\lambda)$.

The argument does not depend on $\mathcal{A}$'s behavior in other instances: the preimages O_{v} uses are those of the Assert of its own instance, the Feldman polynomials of (p,v) are independent of every other instance, and a revealed string π_1 that is not a curve point yields $f_{r_i}(\pi_1) = \perp$ in Lemma 18. Nor does it depend on the order in which $\mathcal{A}$ posts its transactions: $\mathsf{Claim}^{\mathsf{p}}$ may be posted without a prior veto, and if $\mathcal{A}$ never posts Assert, step (b) ends with $\mathsf{NoAssert}$. □

9.7 Composition

Corollary 3 (System-level security)**.** *Under (A1)–(A7) and admissible timelocks, for every PPT adversary* $\mathcal{A}$ *statically corrupting a set* $C \subsetneq \mathcal{O}$ *and every honest operator* O_{j}*, the following hold, except with probability* $\mathsf{negl}(\lambda)$ *for (i) and (ii), and* $(N-1)\cdot 2^{-\kappa} + \mathsf{negl}(\lambda)$ *for (iii).*

(i) If the setup aborts, U *recovers* B *within* $\Delta_{\mathsf{refund}} + \Delta_{\mathsf{conf}}$ *blocks of* $\mathsf{DepositRequest}$ *confirming, and no honest operator loses funds. If all parties are honest, the setup completes.*

(ii) If the setup completes, every confirmed spend of $\mathsf{Deposit}[0]$ *is a transaction of* $\mathcal{S}_{\mathsf{p}}$ *for some* p*, and if* $O_{\mathsf{p}} \in C$*, then a PPT extractor outputs* $\boldsymbol{w}$ *with* $(\boldsymbol{x}_{\mathsf{p}}, \boldsymbol{w}) \in \mathcal{R}$ *from* $\mathcal{A}$*'s view.*

(iii) If the setup completes and O_{j} *runs* P_{Prove} *with a witness for* $\boldsymbol{x}_{\mathsf{j}}$*, then a transaction of* $\mathcal{S}_{\mathsf{j}}$ *is confirmed within* u *blocks, unless a transaction of* $\mathcal{S}_{\mathsf{p}}$ *with* $O_{\mathsf{p}} \in C$ *is confirmed first.*

Proof. (i) is Theorem 2. For (ii), the first claim is Lemma 14. For the second, fix $O_{\mathsf{p}} \in C$ and consider the instance (p, j). An adversary $\mathcal{A}'$ for Definition 5 that corrupts everyone but O_{j} is obtained by running $\mathcal{A}$ together with honest executions of the operators outside $C \cup \{O_{\mathsf{j}}\}$, whose keys $\mathcal{A}'$ generates itself; O_{j}'s view is unchanged, so Theorem 4 yields an extractor for (p, j) with error $\mathsf{negl}(\lambda)$. A union bound over the at most $N-1$ corrupt p gives (ii) with error $(N-1)\cdot\mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$. The instances share O_{p}'s Lamport and reveal keys (Remark 1), which the union bound does not require to be independent, and the per-instance theorem already lets $\mathcal{A}$ interact with O_{j} in all of O_{j}'s roles. (iii) is Theorem 3, with the same simulation of the honest operators outside $C \cup \{O_{\mathsf{j}}\}$. □

10 Experimental Evaluation

We implemented the full protocol in Rust. The evaluation combines local benchmarks with transaction measurements from a Bitcoin mainnet test. All off-chain benchmarks ran on an Apple M4 Max with 16 physical cores, using Rust 1.98.0 release builds with platform-specific cryptographic acceleration enabled.

Off-chain setup Single-candidate processing takes 152 ms for the verifier and 113 ms for the prover on one thread:

Table 2: Single-candidate wall times

Garbler / verifier		Evaluator / prover	
Operation	Time (ms)	Operation	Time (ms)
GC garbling	53.39	GC evaluation	56.38
DRE construction and encryption	99.06	DRE decryption and decoding	56.71
Total	**152.41**	**Total**	**112.95**

The GC ciphertext stream and encrypted DRE account for almost all retained storage:

Table 3: Single-candidate storage

Retained data	Size
GC ciphertext stream	12.31 MiB
Encrypted DRE	15.80 MiB
Input tables	≈31.84 KiB
WE setup ciphertext and message hash	480 B
Shares, interpolation weights, adaptor points	287.95 KiB
Adaptor pre-signatures	64.49 KiB
Total	**28.49 MiB**

Full pairwise setup uses cut-and-choose. For each M_{CC}, we use the smallest N_{CC} satisfying $\binom{N_{\mathsf{CC}}}{M_{\mathsf{CC}}} \geq 2^{40}$. Table 4 reports storage and setup timings across six configurations:

Table 4: Pairwise cut-and-choose setup at $\kappa = 40$

N_{CC}	M_{CC}	Setup (s)	Prover storage (MiB)	Communication (MiB)	Peak RSS (GiB)
2268	4	140.01	112.83	258.91	3.77
669	5	42.96	141.00	183.83	2.78
307	6	20.30	169.18	188.58	2.87
181	7	11.50	197.35	208.56	2.79
124	8	8.03	225.53	232.99	2.96
78	10	5.33	281.88	286.25	3.10

Retaining four candidates reduces prover storage to 112.83 MiB per pair, 43% less than the seven-candidate configuration, at the cost of increasing setup time from 11.50 s to 140.01 s.

Table 5 provides a breakdown of pairwise setup wall time at $(N_{\mathsf{CC}}, M_{\mathsf{CC}}) = (181, 7)$, excluding network transfer time. Generating and checking the candidates dominates setup, with the prover's checks accounting for the largest cost.

Table 5: Pairwise setup timing breakdown

Party	Operation	Time (s)
Verifier	Generate candidates and commitments	3.283
	Prepare openings and ciphertexts	0.286
	Verify adaptor pre-signatures	0.186
	Subtotal	**3.755**
Prover	Check commitments	2.860
	Check openings and retained ciphertexts	4.166
	Generate adaptor pre-signatures	0.165
	Subtotal	**7.191**
Combined total		**10.946**

On-chain cost The largest transactions are Assert and ChallengeAssert, while all other transactions are smaller than 350 vB (Table 6). The transactions from ClaimOptimistic to Withdraw in Table 6 form one complete dispute, in which a single verifier challenges at every stage and the prover-candidate answers with WronglyChallenged. The prover-candidate posts ClaimOptimistic, Claim, Assert, WronglyChallenged, and Withdraw (9,607 vB), and the verifier posts ChallengeOptimistic, ChallengeClaim, and both ChallengeAssert^j chunks (18,160 vB). All transactions used 1 sat/vB, and USD values are those reported by mempool.space for each transaction. Each fee in Table 6 is the fee of a one-parent-one-child (1P1C) package (Section 8.4), in which a 151 vB child spending the P2A anchor pays for the zero-fee parent: the parent size plus 151 vB at 1 sat/vB, plus 2 sat.

Table 6: Mainnet parent transaction sizes

Transaction	Size (vB)	Fee (sat)	Fee (USD)	Transaction ID
Deposit	124	277	0.18	d763108c. . . c5fea30b
ClaimOptimistic	167	320	0.20	f4c386e9. . . d84567cb
ChallengeOptimistic	158	311	0.20	6d6afdf4. . . 355d1b24
Claim	236	389	0.25	928eb4ae. . . 544a73b6
ChallengeClaim	174	327	0.21	a4e600cc. . . 255dfca1
Assert	8,703	8,856	5.64	8136caff. . . f829b54f
ChallengeAssert^0	8,914	9,067	5.79	f723f15f. . . 451e8705
ChallengeAssert^1	8,914	9,067	5.79	9a2e0926. . . e9664ab5
WronglyChallenged	268	421	0.27	59adec6b. . . a026b0b8
Withdraw	233	386	0.25	be732ede. . . 91219ae0
NoWithdraw	341	494	0.31	e49cd2b5. . . a40d4eca

References

[AAL+24] Lukas Aumayr, Zeta Avarikioti, Robin Linus, Matteo Maffei, Andrea Pelosi, Christos Stefo, and Alexei Zamyatin. Bitvm: Quasi-turing complete computation on bitcoin. IACR Cryptology ePrint Archive, Paper 2024/1995, 2024.

[ADH+26] Lukas Aumayr, Xinshu Dong, Runchao Han, Srivatsan Sridhar, David Tse, and Nikhil Vanjani. Trustless Bitcoin vaults. IACR Cryptology ePrint Archive, Paper 2026/2466, 2026.

[AEE+21] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT*, pages 635–664. Springer, 2021.

[AIK14] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. *SIAM Journal on Computing*, 43(2):905–929, 2014.

[Bab26] Babylon Labs. Babylon. https://babylonlabs.io/, 2026.

[Bit24] Bitcoin Core developers. Bitcoin core 28.0. Release notes, 2024. https://bitcoincore.org/en/releases/28.0/.

[Bit26] Bitlayer. Bitvm bridge testnet. https://docs.bitlayer.org/docs/BitVMBridge/Multi-Chain/testnet/, 2026.

[BOB26] BOB. Bob bridge. https://app.gobob.xyz/en/bridge, 2026.

[Cit26] Citrea. Citrea bridge. https://citrea.xyz/bridge, 2026.

[DGGM19] Nico Döttling, Sanjam Garg, Vipul Goyal, and Giulio Malavolta. Laconic conditional disclosure of secrets and applications. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 661–685. IEEE, 2019.

[DOY22] Wei Dai, Tatsuaki Okamoto, and Go Yamamoto. Stronger security and generic constructions for adaptor signatures. In *Progress in Cryptology – INDOCRYPT*, pages 52–77, 2022.

[Eag25] Liam Eagen. Glock: Garbled locks for Bitcoin. IACR Cryptology ePrint Archive, Paper 2025/1485, 2025.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Annual Symposium on Foundations of Computer Science (SFCS)*, pages 427–438. IEEE, 1987.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology – EUROCRYPT*, pages 191–219. Springer, 2015.

[Fou19] Lloyd Fournier. One-time verifiably encrypted signatures A.K.A. adaptor signatures. https://github.com/LLFourn/one-time-VES/, 2019.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *ACM Symposium on Theory of Computing (STOC)*, pages 467–476, 2013.

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *ACM Symposium on Theory of Computing (STOC)*, pages 151–160, 1998.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology – EUROCRYPT*, pages 281–310. Springer, 2015.

[GKS+26] Sanjam Garg, Dimitris Kolonelos, Mikhail Sergeevitch, Srivatsan Sridhar, and David Tse. BABE: Verifying proofs on Bitcoin made 1000x cheaper. IACR Cryptology ePrint Archive, Paper 2026/065, 2026. To appear at ACM CCS 2026.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT*, pages 305–326, 2016.

[GSST24] Paul Gerhart, Dominique Schröder, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. Foundations of adaptor signatures. In *Advances in Cryptology – EUROCRYPT*, pages 161–189, 2024.

[Hio22] Leona Hioki. Trustless Bitcoin bridge creation with witness encryption. Ethereum Research forum, https://ethresear.ch/t/trustless-bitcoin-bridge-creation-with-witness-encryption/11953, 2022.

[Ish13] Yuval Ishai. Randomization techniques for secure computation. In Manoj Prabhakaran and Amit Sahai, editors, *Secure Multi-Party Computation*, volume 10 of *Cryptology and Information Security Series*, pages 222–248. IOS Press, 2013.

[KBH26] Nakul Khambhati, Anwesh Bhattacharya, and David Heath. Duty-free bits: Projectivizing garbling schemes. IACR Cryptology ePrint Archive, Paper 2026/476, 2026.

[KTA+26] Nakul Khambhati, Mukesh Tiwari, Azz, Sapin Bajracharya, Manish Bista, Liam Eagen, Christian Lewe, and Aaron Feickert. Mosaic: Practical malicious security for garbled circuits on bitcoin. IACR Cryptology ePrint Archive, Paper 2026/812, 2026.

[Lam79] Leslie Lamport. Constructing Digital Signatures from a One Way Function. Technical Report CSL-98, October 1979. Edition: SRI International.

[Lin16] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[Lin23] Robin Linus. BitVM: Compute anything on Bitcoin. https://bitvm.org/bitvm.pdf, 2023.

[LVW17] Tianren Liu, Vinod Vaikuntanathan, and Hoeteck Wee. Conditional disclosure of secrets via non-linear reconstruction. In *Advances in Cryptology – CRYPTO*, pages 758–790. Springer, 2017.

[MEGS16] Malte Möser, Ittay Eyal, and Emin Gün Sirer. Bitcoin covenants. In *Financial Cryptography and Data Security (FC)*, pages 126–141. Springer, 2016.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In *Advances in Cryptology – CRYPTO 2021*, pages 189–221. Springer, 2021.

[OP17] Russell O'Connor and Marta Piekarska. Enhancing Bitcoin transactions with covenants. In *Financial Cryptography and Data Security (FC)*, pages 191–198. Springer, 2017.

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In *International Workshop on Fast Software Encryption*, pages 348–358. Springer, 2004.

[Rub25] Jeremy Rubin. Delbrag. https://rubin.io/public/pdfs/delbrag.pdf, 2025.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[WAA+26a] Robin Linus Woll, Ioannis Alexopoulos, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and David Tse. BitVM3: Efficient bitcoin bridges via garbled circuits. IACR Cryptology ePrint Archive, Paper 2026/933, 2026.

[WAA+26b] Robin Linus Woll, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, Andrea Pelosi, Orfeas Stefanos Thyfronitis Litos, Christos Stefo, David Tse, and Alexei Zamyatin. Bridging bitcoin to second layers via BitVM2. In *USENIX Security Symposium*, pages 969–988, 2026.

[WNR20] Pieter Wuille, Jonas Nick, and Tim Ruffing. BIP-340: Schnorr signatures for secp256k1. Bitcoin Improvement Proposal, 2020. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki.

[WNT20a] Pieter Wuille, Jonas Nick, and Anthony Towns. BIP-341: Taproot: SegWit version 1 spending rules. Bitcoin Improvement Proposal, 2020. https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki.

[WNT20b] Pieter Wuille, Jonas Nick, and Anthony Towns. BIP-342: Validation of taproot scripts. Bitcoin Improvement Proposal, 2020. https://github.com/bitcoin/bips/blob/master/bip-0342.mediawiki.

[Yao82] Andrew Yao. Protocols for secure computations. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

[Zha24] Gloria Zhao. BIP-431: Topology restrictions for pinning. Bitcoin Improvement Proposal, 2024. https://github.com/bitcoin/bips/blob/master/bip-0431.mediawiki.

A Transactions

All peg-out transactions use `nVersion` = 3 and carry a 0-value P2A anchor as their final output (omitted from the tables). Throughout, one instance $(\mathsf{p}, \mathsf{v}) \in \mathcal{D}$ is fixed and rk is as in Protocol 6. Input rows read $(\mathsf{Src}[i], \langle\text{spent leaf}\rangle)$ and output rows $(\text{value}, \langle\text{script}\rangle)$. Signature notation follows Section 3.8. $\mathsf{ChallengeAssertScript}^j$ is Φ_j of Section 8.1.3.

A.1 Peg-in

$\mathsf{DepositRequest}$ (broadcaster: U)	
Inputs	(0) user funding outpoint(s), unconstrained
Outputs	(0) $\big(B,\ \langle\text{key path } \mathsf{pk}^{\mathsf{id}}_{\mathcal{O}} \,\|\|\, \mathsf{RelTimelock}(\Delta_{\mathsf{refund}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_U)\rangle\big)$ (pending deposit)
Witness	(0) user's signature(s) on the funding input(s)

$\mathsf{Deposit}$ (broadcaster: any $O_{\mathsf{j}} \in \mathcal{O}$)	
Inputs	(0) ($\mathsf{DepositRequest}[0]$, key path $\mathsf{pk}^{\mathsf{id}}_{\mathcal{O}}$)
Outputs	(0) $\big(B,\ \langle\mathsf{CheckSig}(\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}})\rangle\big)$ (covenant-locked deposit)
Witness	(0) $\sigma^{\mathsf{id}}_{\mathsf{Deposit}}$ (N-of-N MuSig2 under $\mathsf{pk}^{\mathsf{id}}_{\mathcal{O}}$, taproot-tweaked key path)

$\mathsf{DepositRefund}$ (broadcaster: U; only on aborted setup)	
Inputs	(0) ($\mathsf{DepositRequest}[0]$, $\langle\mathsf{RelTimelock}(\Delta_{\mathsf{refund}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_U)\rangle$)
Outputs	(0) $\big(B,\ \mathrm{P2TR}(\mathsf{pk}_U)\big)$
Witness	(0) $\sigma_{U,\mathsf{DepositRefund}}$

A.2 Peg-out (optimistic)

$\mathsf{ClaimOptimistic}^{\mathsf{p}}$ (broadcaster: O_{p})	
Inputs	(0) $u_{O_{\mathsf{p}}}$ (funding outpoint, $\mathsf{val}(u_{O_{\mathsf{p}}}) = d_{\mathsf{dust}}(k_C(N-1)+2)$)
Outputs	(0) $\big(d_{\mathsf{dust}},\ \langle\mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \,\|\|\, \big\{\mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})\big\}_{\mathsf{v}\neq\mathsf{p}}\rangle\big)$ (optimistic veto, one leaf per v)
	(1) $\big(d_{\mathsf{dust}}(k_C(N-1)+1),\ \langle\mathsf{RelTimelock}(\Delta_{\mathsf{withdrawOpt}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \,\|\, \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}})\rangle\big)$ (prover stake, optimistic)
Witness	(0) O_{p}'s signature on its funding outpoint

$\mathsf{ChallengeOptimistic}_{(\mathsf{p},\mathsf{v})}$ (broadcaster: O_{v})	
Inputs	(0) ($\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$, leaf $1{+}\mathsf{rk}(\mathsf{p},\mathsf{v}) = \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})$)
Outputs	(0) $\big(d_{\mathsf{dust}},\ \mathrm{P2TR}(\mathsf{pk}_{O_{\mathsf{v}}})\big)$ (sweep)
Witness	(0) $(\sigma^{P}_{\mathsf{ChallengeOptimistic}}, \sigma^{V}_{\mathsf{ChallengeOpt}})$ (pre-signed at setup)

$\mathsf{WithdrawOptimistic}^{\mathsf{p}}$ (broadcaster: O_{p})	
Inputs	(0) ($\mathsf{Deposit}[0]$, $\langle \mathsf{CheckSig}(\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}}) \rangle$)
	(1) ($\mathsf{ClaimOptimistic}^{\mathsf{p}}[0]$, leaf $0 = \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}})$)
	(2) ($\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$, leaf $0 = \mathsf{RelTimelock}(\Delta_{\mathsf{withdrawOpt}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}})$)
Outputs	(0) $\big(B + d_{\mathsf{dust}}(k_C(N{-}1){+}2),\ \mathrm{P2TR}(\mathsf{pk}_{O_{\mathsf{p}}})\big)$
Witness	(0) $\sigma^{\mathsf{cov}}_{\mathsf{WithdrawOpt},\mathsf{p}}$
	(1) $\sigma_{P,\mathsf{WithdrawOpt}}$
	(2) $\sigma_{P,\mathsf{WithdrawOpt}}$

A.3 Dispute

$\mathsf{Claim}^{\mathsf{p}}$ (broadcaster: O_{p})	
Inputs	(0) ($\mathsf{ClaimOptimistic}^{\mathsf{p}}[1]$, leaf $1 = \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}})$)
Outputs	(0) $\big(d_{\mathsf{dust}},\ \langle \mathsf{RelTimelock}(\Delta_{\mathsf{withdraw}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \ \|\| \ \big\{\mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})\big\}_{\mathsf{v}\neq\mathsf{p}} \rangle\big)$ (slash chokepoint)
	$(1{+}\mathsf{rk}(\mathsf{p},\mathsf{v}))$ $\big(k_C d_{\mathsf{dust}},\ \langle \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}}) \ \|\| \ \mathsf{RelTimelock}(\Delta_{\mathsf{withdraw}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \rangle\big)$ for $\mathsf{v} \neq \mathsf{p}$ (challenge output, one per v)
Witness	(0) $\sigma_{P,\mathsf{Claim}}$

$\mathsf{ChallengeClaim}_{(\mathsf{p},\mathsf{v})}$ (broadcaster: O_{v})	
Inputs	(0) ($\mathsf{Claim}^{\mathsf{p}}[1{+}\mathsf{rk}(\mathsf{p},\mathsf{v})]$, leaf $0 = \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})$)
Outputs	(0) $\big(k_C d_{\mathsf{dust}},\ \langle \mathsf{RelTimelock}(\Delta_{\mathsf{assert}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}}) \ \|\| \ \mathsf{AssertScript} \rangle\big)$ (assert input)
Witness	(0) $(\sigma^{P}_{\mathsf{ChallengeClaim}}, \sigma_{V,\mathsf{ChallengeClaim}})$

$\mathsf{Assert}_{(\mathsf{p},\mathsf{v})}$ (broadcaster: O_{p})	
Inputs	(0) ($\mathsf{ChallengeClaim}_{(\mathsf{p},\mathsf{v})}[0]$, leaf $1 = \mathsf{AssertScript} = \mathsf{CheckLampSig}(\mathsf{lpk}_{O_{\mathsf{p}}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})$)
Outputs	(j) $\big(d_{\mathsf{dust}},\ \langle \mathsf{ChallengeAssertScript}^{j} \rangle\big)$ for $j \in \{0,\ldots,k_C{-}1\}$ (Lamport match, chunk j)
Witness	(0) $(\mu, \sigma^{V}_{\mathsf{Assert}})$ (μ: the $2n$-bit Lamport signature on π_1)

$\mathsf{NoAssert}_{(\mathsf{p},\mathsf{v})}$ (broadcaster: O_{v})	
Inputs	(0) ($\mathsf{Claim}^{\mathsf{p}}[0]$, slash leaf $\mathsf{v} = \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})$)
	(1) ($\mathsf{ChallengeClaim}_{(\mathsf{p},\mathsf{v})}[0]$, leaf $0 = \mathsf{RelTimelock}(\Delta_{\mathsf{assert}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})$)
Outputs	(0) $\big((k_C{+}1)d_{\mathsf{dust}},\ \mathrm{P2TR}(\mathsf{pk}_{O_{\mathsf{v}}})\big)$
Witness	(0) $(\sigma^{P}_{\mathsf{NoAssert}}, \sigma_{V,\mathsf{NoAssert}})$
	(1) $\sigma_{V,\mathsf{NoAssert}}$

$\mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})}$, $j \in \{0,\dots,k_C{-}1\}$ (broadcaster: O_{v})	
Inputs	(0) ($\mathsf{Assert}_{(\mathsf{p},\mathsf{v})}[j]$, $\langle \Phi_j = \mathsf{CheckLampAdaptorMatch}(\mathsf{lpk}^{(R_j)}_{O_{\mathsf{p}}}, \{T_{t,\beta}\}_{t\in R_j,\,\beta\in\{0,1\}}, \mathsf{pk}_{R,\mathsf{p}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})\rangle$)
Outputs	(0) $\big(d_{\mathsf{dust}},\ \langle \mathsf{RelTimelock}(\Delta_{\mathsf{noWithdraw}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})\ \|\|\ \mathsf{WronglyChallengedScript}\rangle\big)$ (response window)
Witness	(0) $\big(\sigma_{V,\mathsf{ChallengeAssert}^{j}}, (\sigma_t, \mu_t)_{t\in R_j}\big)$ (σ_t: adaptor completions)

$\mathsf{Withdraw}^{\mathsf{p}}$ (broadcaster: O_{p})	
Inputs	(0) ($\mathsf{Deposit}[0]$, $\langle \mathsf{CheckSig}(\mathsf{pk}^{\mathsf{cov}}_{\mathcal{O}})\rangle$)
	(1) ($\mathsf{Claim}^{\mathsf{p}}[0]$, leaf $0 = \mathsf{RelTimelock}(\Delta_{\mathsf{withdraw}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}})$)
Outputs	(0) $\big(B + d_{\mathsf{dust}},\ \mathrm{P2TR}(\mathsf{pk}_{O_{\mathsf{p}}})\big)$
Witness	(0) $\sigma^{\mathsf{cov}}_{\mathsf{Withdraw},\mathsf{p}}$
	(1) $\sigma_{P,\mathsf{Withdraw}}$

$\mathsf{NoWithdraw}_{(\mathsf{p},\mathsf{v})}$ (broadcaster: O_{v})	
Inputs	(0) ($\mathsf{Claim}^{\mathsf{p}}[0]$, slash leaf $\mathsf{v} = \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})$)
	(j+1) ($\mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})}[0]$, leaf $0 = \mathsf{RelTimelock}(\Delta_{\mathsf{noWithdraw}}) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{v}}})$) for $j \in \{0,\dots,k_C{-}1\}$
Outputs	(0) $\big((k_C{+}1)d_{\mathsf{dust}},\ \mathrm{P2TR}(\mathsf{pk}_{O_{\mathsf{v}}})\big)$
Witness	(0) $(\sigma^{P}_{\mathsf{NoWithdraw}}, \sigma_{V,\mathsf{NoWithdraw}})$
	(j+1) $\sigma^{(j)}_{V,\mathsf{NoWithdraw}}$ for $j \in \{0,\dots,k_C{-}1\}$

$\mathsf{WronglyChallenged}_{(\mathsf{p},\mathsf{v})}$ (broadcaster: O_{p})	
Inputs	(j) ($\mathsf{ChallengeAssert}^{j}_{(\mathsf{p},\mathsf{v})}[0]$, leaf $1 = \mathsf{WronglyChallengedScript} = \big(\bigvee_{i\in\mathcal{I}} \mathsf{HashLock}(h_{\mathsf{msg},i})\big) \wedge \mathsf{CheckSig}(\mathsf{pk}_{O_{\mathsf{p}}})$) for $j \in \{0,\dots,k_C{-}1\}$
Outputs	(0) $\big(k_C d_{\mathsf{dust}},\ \mathrm{P2TR}(\mathsf{pk}_{O_{\mathsf{p}}})\big)$ (sweep)
Witness	(j) $(\mathsf{msg}_{i^*}, \sigma_{P,\mathsf{WronglyChallenged}})$ for $j \in \{0,\dots,k_C{-}1\}$