Mosaic: Practical Malicious Security for Garbled Circuits on Bitcoin

Nakul Khambhati*†[1,2], Mukesh Tiwari*‡[1], Azz[1], Sapin Bajracharya[1], Manish Bista[1], Liam Eagen§[3], Christian Lewe[1], and Aaron Feickert[1]

[1]Alpen Labs
[2]UCLA
[3]Ideal Group

Abstract

Bitcoin's scripting language cannot verify arbitrary computation natively, yet applications such as trust-minimized bridges depend on this capability. Recent techniques employ garbled circuits: the prover commits off chain to a garbled circuit encoding a verifier, designed so that evaluating it on an invalid witness reveals a secret. Posting that secret on chain serves as a fraud proof, allowing the verifier to claim the prover's stake without any on-chain computation. To evaluate the garbled circuit and recover the secret, the verifier needs the prover's input labels, which the prover must post on chain. Since Bitcoin charges permanently for block space, minimizing this on-chain footprint is a primary design concern. Achieving malicious security via cut-and-choose compounds this: the prover must produce multiple independently garbled copies of the circuit, requiring one set of labels per copy.

We present Mosaic, a protocol that achieves malicious security via cut-and-choose but reduces the on-chain footprint so that it is independent of the number of garbled copies. The key technique, first introduced by Eagen (Glock, 2025) in this setting, is *polynomial label correlation*: labels across all N garbled copies are arranged as evaluations of a degree-t polynomial, so the t shares revealed during cut-and-choose fall one short of the reconstruction threshold. We use adaptor signatures to arrange that the prover's on-chain witness commitment reveals the missing share as a byproduct; the evaluator then reconstructs labels for all unchallenged copies by interpolation. We sketch why Mosaic is secure against a malicious prover and verifier and instantiate it for trust-minimized Bitcoin bridging with a Groth16 verifier circuit, a full protocol specification, and a Rust implementation.

1 Introduction

Bitcoin's limited scripting language allows locking funds under simple conditions such as equality checks, hash preimages and signatures. This expressiveness is enough to build payment channels and time-locked contracts, but falls well short of verifying arbitrary computation. Bitcoin cannot check whether a given input satisfies some predetermined NP relation.

This limitation becomes an obstacle in the following setting. Two parties, a prover P and a verifier V, agree on an NP relation R and a public instance x. P claims to know a witness w such

*These authors contributed equally to this work.
†Corresponding author: nakul@alpenlabs.io
‡Corresponding author: mukesh@alpenlabs.io
§This author contributed to this work while affiliated with Alpen Labs.

that $R(x, w) = 1$, and V is willing to pay a prize in exchange for it. To deter P from submitting an invalid witness, both parties agree that P puts up a stake, which V should be able to claim if the witness is invalid. Since Bitcoin cannot evaluate R directly, the check must happen off chain, but its outcome must be tied to on-chain consequences in a way that neither party can subvert.

Garbled circuits and on-chain input authentication. Garbled circuits [18, 19] offer a natural way to move the verification of R off chain. The key property we exploit is that we can arrange for P to garble a circuit encoding the relation R and send it to V such that evaluation on an invalid witness reveals a designated secret s^*, while evaluation on a valid witness reveals nothing. Once both parties have agreed on the funds to be locked, they pre-sign a *Witness Fault* transaction whose spending condition requires knowledge of s^*, so that V recovering s^* translates directly into an on-chain claim against P's stake.

What makes garbled circuits particularly well suited to this setting is that their input encoding structure aligns naturally with Bitcoin's transaction validation. The data P must publish on chain to authorize her transaction can be arranged to simultaneously serve as the garbled circuit input labels. There is therefore an exact correspondence between the witness P commits to on chain and the input on which V evaluates the garbled circuit.

Malicious security and associated costs. The above assumes P garbles honestly. A malicious P could produce an incorrect garbled circuit that always evaluates to `true`, preventing V from recovering s^* even for an invalid witness. The standard mitigation is cut-and-choose: P produces N independently garbled copies of the circuit and commits to all of them before seeing which will be checked. V randomly selects t of these to open and verify, leaving $\ell = N - t$ for evaluation, with a statistical guarantee that at least one of the remaining copies is correctly garbled.

This introduces an on-chain cost problem. Authenticating P's input to each of the ℓ unchallenged circuits requires posting the corresponding input labels on chain: one set of labels per circuit. Bitcoin charges permanent block space for every byte in a transaction witness, which can be prohibitive when the witness size is non-trivial.

Authenticated garbling (introduced in [20]) might appear to offer a cheaper alternative, but it is incompatible with our setting's timing constraints. The verification mechanism in authenticated garbling operates during (or after) evaluation: the evaluator checks MACs on wire labels as the circuit is computed, detecting a dishonest garbling only at that point. In our setting, evaluation occurs only after both parties have locked their funds at Kickoff, so by the time authenticated garbling could flag incorrect garbling, it is too late for V to decline participation. Cut-and-choose is appropriate precisely because its challenge-and-open phase runs before any funds are committed: V inspects the opened circuits and proceeds to lock funds only if they are correctly garbled, obtaining a statistical guarantee over the remaining copies.

Our contribution. We present Mosaic, a protocol that achieves malicious security while reducing the on-chain cost to a *single set of labels per input wire*, regardless of ℓ.

The key idea is to correlate the labels across all N garbled circuits using a threshold secret-sharing scheme: the labels for a given wire and input value across all N copies are arranged as evaluations of a polynomial, with reconstruction threshold $t + 1$. The t labels revealed during cut-and-choose are t shares of this polynomial, one short of the reconstruction threshold. Posting a single additional label on chain per wire, the one that completes the threshold, allows V to interpolate the polynomial and derive the correct labels for all ℓ unchallenged circuits algebraically. The on-chain footprint is therefore independent of ℓ.

We informally argue why Mosaic is secure against a malicious prover and a malicious verifier, and give a concrete end-to-end Bitcoin instantiation with a simple transaction graph.

Relation to prior work. The idea of using garbled circuits for optimistic verification on Bitcoin originates with Rubin [7] and Eagen [6] who independently observed that if a garbled circuit encodes a SNARK verifier, evaluation on an invalid proof reveals a secret that can serve as a fraud proof, replacing the bisection-based fraud proofs of BitVM [8, 9] with simple secret revelation.

Eagen [6] presented this observation as the Garbled Lock (Glock) primitive and proposed Glock25, the first concretely practical instantiation that achieved malicious security. Glock25 introduces several ideas that Mosaic builds upon: (i) using Feldman verifiable secret sharing to correlate labels across cut-and-choose copies, so that opening a challenged subset during setup leaves the shared secret information-theoretically hidden; (ii) using Schnorr adaptor signatures, rather than Lamport signatures, as the on-chain authentication mechanism, so that completing a signature simultaneously authorizes the Bitcoin transaction and reveals a polynomial share; and (iii) combining these techniques so that the on-chain footprint per input wire is a single Schnorr signature, independent of the number of unchallenged circuits. The use of secret sharing within cut-and-choose itself has roots in the secure two-party computation literature: Lindell and Pinkas [10] applied it to output-wire determination, and Huang, Katz, and Evans [11] used Feldman VSS on output labels in their symmetric cut-and-choose protocol.

Mosaic provides a complete, self-contained treatment of this approach. Our contributions relative to prior work are as follows. We give a full formal construction (Section 6) with explicit protocol steps, a concrete Bitcoin transaction graph (Section 2), and detailed cost analysis (Section 9) backed by an open-source Rust implementation [26], whereas Glock25 describes the mechanism at a higher level of abstraction. We formally state the security properties required in our setting and sketch how they follow from the underlying cryptographic primitives and the Bitcoin consensus protocol. We also introduce a three-way partition of circuit inputs that reflects the distinct timing at which different parts of the statement become known in a Bitcoin protocol. Each phase maps to a different adaptor signature mechanism with different costs.

Within the broader BitVM landscape, Chen [12] provides a systematization of knowledge organizing the design space. Recent works [14, 13, 15] significantly reduce the size of garbled tables for verifying Groth16 proofs and we explore in Section 9 how Mosaic immediately benefits from some of these improvements.

2 Fair Exchange on Bitcoin

Before presenting the Mosaic cryptographic protocol, we describe the fair exchange protocol it is designed to support and formalize the properties it must achieve.

Earlier works like [27] study fair exchange on Bitcoin, but rely on the predicate check being expressible in Bitcoin script and compatible with Bitcoin's transaction-size, script-validation, and standardness limits.

Recall we are in the setting where a prover P and a verifier V agree on an NP relation R, a public instance x, a prize p₿, and a stake s₿. P claims to know a witness w with $R(x, w) = 1$. We wish to design a fair exchange protocol where P receives the prize if her witness is valid, and forfeits the stake to V if it is not, with no trusted third party.

This protocol flow is depicted in Figure 1.

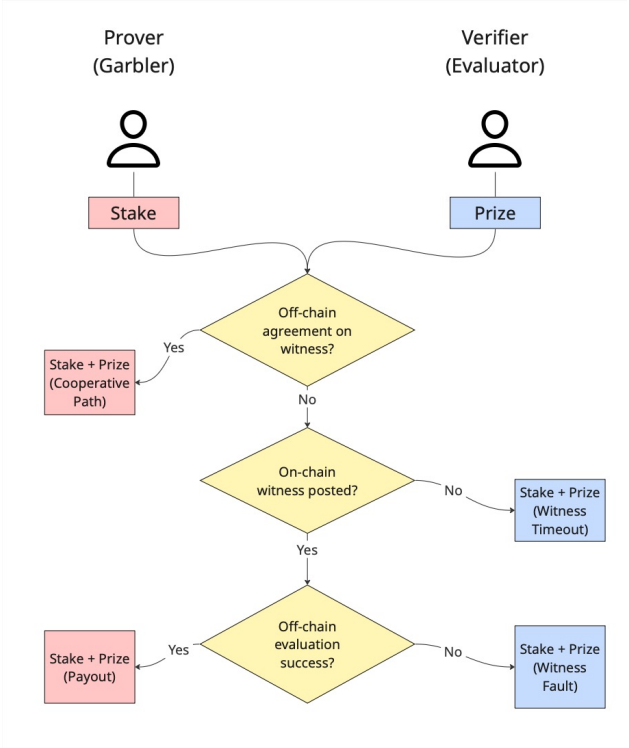


Figure 1: Fair exchange protocol flow. Diamond nodes represent decision points; rectangular nodes indicate fund distribution outcomes

Transaction graph for fair exchange. Bitcoin transactions operate in the UTXO (unspent transaction output) model. A *transaction* tx consumes one or more existing UTXOs as inputs and produces one or more new UTXOs as outputs. Each output carries a *value* and a *spending condition*: a script that specifies who may spend the output and under what circumstances. A transaction is valid only if, for every input it consumes, the accompanying witness data satisfies the corresponding spending condition. A UTXO can be spent at most once; if two transactions attempt to spend the same UTXO, consensus ensures at most one is confirmed.

The parties encode the fair exchange contract as a directed graph of transactions, each with spending conditions that capture the different outcomes. To enforce the logic of dispute resolution and timeout paths so it cannot be unilaterally altered after setup, certain transactions are made non-malleable. In practice, this is done by having such transactions collaboratively pre-signed by both users.

The key cryptographic object is a *fault secret* s^*, committed to by P during setup. The spending condition of the Witness Fault transaction requires knowledge of s^*. Mosaic guarantees that V learns s^* if and only if P's on-chain committed witness fails R, so this transaction can be posted by V exactly when P has cheated, awarding all funds to V.

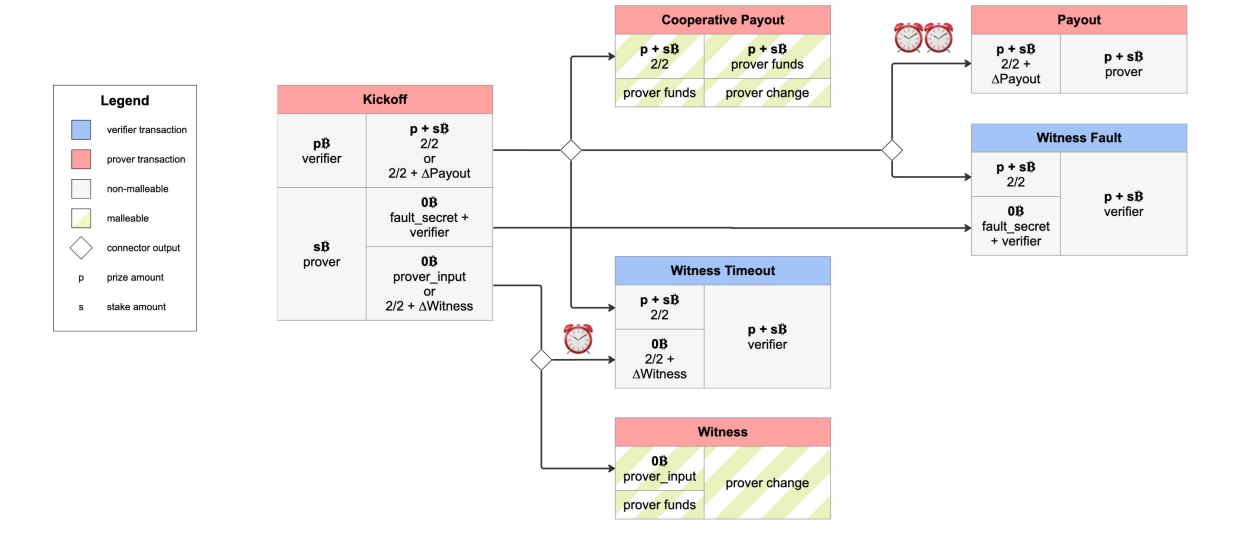


Figure 2: Bitcoin transaction graph implementing the protocol flow described in Figure 1. The left side of each transaction indicates input UTXOs and the right side output UTXOs.

The graph in Figure 2 has the following transactions.

- **Kickoff.** Parties lock their funds here: P contributes the stake s and V contributes the prize p. Before the Kickoff transaction is confirmed on chain, either party controls their UTXO and can back out unilaterally. The exact mechanism for co-signing is described in Section 9.
- **Witness.** Broadcast by P to commit to her witness on chain. The spending condition of the Kickoff output requires P to post one signature per byte of w under V's key. Because P does not hold V's signing key, the only way she can produce these signatures is by completing pre-signatures that V provides before the Kickoff transaction is broadcast, one per possible byte value per position in w. Each pre-signature is constructed so that completing it simultaneously produces a valid Bitcoin signature and reveals the corresponding garbled circuit input label to V. This is possible because Mosaic is designed such that on-chain witness commitment maps precisely to garbled circuit input labels.
- **Witness Fault.** Spendable by V using s^*. If the Witness transaction encodes a failing input, Mosaic guarantees V recovers s^* by evaluating the garbled circuit on the revealed input labels. Posting this awards both funds to V and protects an honest V against a malicious P.
- **Witness Timeout.** Spendable by V after a fixed time window Δ_{Witness} if P has not broadcast the Witness transaction. This forces P to commit to a witness within the allotted period.
- **Payout.** Spendable by P after a delay Δ_{Payout}, provided neither Witness Fault nor Witness Timeout has been posted. If P posts a valid witness that V cannot fault, P collects both funds once the delay expires. This protects an honest P against a malicious V.
- **Cooperative Payout.** Spendable immediately by P with a co-signature from V. During the Δ_{Witness} window, P can send V the witness off chain; if V accepts, both parties settle immediately without posting the Witness transaction at all.

Input timing. Not all input to the garbled circuit is known at the same point in the protocol. Some part of the public statement x is fixed before any funds are locked. However, there may also be public data that depends on the Kickoff transaction's identifier and is only determined once the UTXOs that fund this transaction have been agreed upon by both parties. The witness itself is committed last, via the Witness transaction. Mosaic partitions the circuit inputs into three types accordingly: Phase 1 inputs $\mathbf{x}_1 \in \{0,1\}^{n_1}$, known from the start; Phase 2 inputs $\mathbf{x}_2 \in \{0,1\}^{n_2}$, fixed once the Kickoff transaction is confirmed; and Phase 3 inputs $\mathbf{x}_3 \in \{0,1\}^{n_3}$, corresponding to the witness and committed on chain last.

2.1 Properties of Fair Exchange

We formalize the properties that the fair exchange protocol of Figure 1 must satisfy. Throughout, we model time in discrete rounds (block heights) and write τ_0 for the round at which the Kickoff transaction confirms. The protocol is parameterized by two relative timelocks: $\Delta_{\mathsf{Witness}}$ (the window for P to post the Witness transaction) and Δ_{Payout} (the delay before P may claim the Payout).

We assume the Bitcoin consensus protocol operates correctly, confirming only valid transactions whose witness data satisfies the corresponding spending conditions. In particular, we rely on three properties:

- *Persistence.* A transaction confirmed at some depth k (included in a block with $k-1$ successor blocks built on top of it) remains confirmed in all future honest views.
- *Liveness.* A valid transaction broadcast by an honest party is confirmed within Δ_{conf} blocks.
- *Double-spending protection.* If two transactions spend the same UTXO, at most one is confirmed. This ensures the branches of the transaction graph are mutually exclusive.

Protocol properties. The fair exchange protocol satisfies the following three properties.

- **Completeness.** If both P and V are honest and P holds a valid witness w with $R(x,w) = 1$, then the protocol terminates with P receiving the stake and prize and V learning w.
- **Fairness with Compensation.** The protocol ensures that an honest party is never disadvantaged by a malicious counterparty:
  - (a) **Prover security.** For every PPT adversary $\mathcal{A}$ controlling V: if P is honest and holds a valid witness w with $R(x,w) = 1$, then after the protocol terminates, P receives at least the stake and prize, except with negligible probability.
  - (b) **Verifier security.** For every PPT adversary $\mathcal{A}$ controlling P: after the protocol terminates, exactly one of the following holds except with probability at most $1/\binom{N}{\ell}$:
    - (i) V learns a valid witness w with $R(x,w) = 1$, or
    - (ii) V receives the full stake and prize.
- **Timeliness.** There exists a public round bound T, determined before execution begins, such that for every PPT adversary $\mathcal{A}$ corrupting at most one of $\{P, V\}$: every honest party can unilaterally reach a terminal state by round $\tau_0 + T$, regardless of the adversary's strategy. Concretely, $T = \Delta_{\mathsf{Witness}} + \Delta_{\mathsf{Payout}} + \Delta_{\text{conf}}$.

What Mosaic provides. The safety and accountability properties proved in Section 7 are the cryptographic core of the above guarantees. Accountability ensures that V recovers s^* whenever P posts a failing witness, enabling Fairness with Compensation for the verifier. Safety ensures that an honest P's stake is safe, enabling Fairness with Compensation for the prover. Timeliness follows from the timelock structure of the transaction graph and does not require any cryptographic argument.

Proof of completeness follows from both execution paths (cooperative or on-chain) terminating with P receiving funds when $R(x, w) = 1$, while timeliness follows directly from the timelock structure: each party can unilaterally reach a terminal state by round $\tau_0 + T$. We focus on Fairness with Compensation below, as it ties directly to Mosaic's security guarantees; full proofs of Completeness and Timeliness appear in Appendix A

Proof of Fairness with Compensation. We prove the two directions separately.

(a) Prover security. Let $\mathcal{A}$ be a PPT adversary controlling V. Honest P holds a valid witness w with $R(x, w) = 1$ and broadcasts the Witness transaction before $\tau_0 + \Delta_{\mathsf{Witness}}$.

Witness Timeout is excluded.
Since P posts the Witness transaction before the deadline, it spends the relevant Kickoff output. By double-spending protection, $\mathcal{A}$ cannot also confirm Witness Timeout, which requires spending the same output.

Witness Fault is excluded.
The spending condition of Witness Fault requires knowledge of s^*. By safety of Mosaic (Section 7), since P garbled honestly and $R(x, w) = 1$, no PPT adversary can recover s^* with non-negligible probability.

Payout succeeds.
Since neither Witness Fault nor Witness Timeout can be posted, the Payout output remains unspent. After Δ_{Payout} expires, P broadcasts the Payout transaction and receives $(s + p)$₿.

(b) Verifier security. Let $\mathcal{A}$ be a PPT adversary controlling P. We proceed by case analysis on $\mathcal{A}$'s behavior after the Kickoff transaction confirms.

Case 1: $\mathcal{A}$ does not post a Witness transaction before $\tau_0 + \Delta_{\mathsf{Witness}}$.
After the deadline, V broadcasts the Witness Timeout transaction, which awards $(s + p)$₿ to V. By liveness it confirms within Δ_{conf} rounds. Outcome (ii) holds.

Case 2: $\mathcal{A}$ posts a Witness transaction encoding an input with $R(x, w) = 0$.
V extracts the garbled-circuit input labels from the on-chain signatures and evaluates the garbled circuit. By accountability of Mosaic (Section 7), V recovers s^* except with probability at most $1/\binom{N}{\ell}$. V then posts the Witness Fault transaction, receiving $(s+p)$₿. By double-spending protection, $\mathcal{A}$ cannot simultaneously claim Payout from the same output. Outcome (ii) holds.

Case 3: $\mathcal{A}$ posts a Witness transaction encoding an input with $R(x, w) = 1$.
The Witness transaction commits $\mathcal{A}$ to an input that satisfies R. V extracts the input labels from the on-chain signatures, and these labels encode a valid witness w. Outcome (i) holds.

□

3 Notation

Let $\mathbb{G}$ be a prime-order group of order p with generator G and scalar field $\mathbb{F}_p$. We write $[s] := s \cdot G$ for point-scalar multiplication, which serves as a commitment to the scalar s.

Parties. We write G for the *Garbler* and E for the *Evaluator*. In the fair exchange setting of Section 2, the prover P takes the role of G, and the verifier V takes the role of E. We use P and V when discussing the fair exchange protocol and its properties, and G and E when discussing the cryptographic protocol and its construction.

Parameters.

- κ: computational security parameter.
- $n = n_1 + n_2 + n_3$: total number of input wires, partitioned into *Phase 1*, *Phase 2*, and *Phase 3* wires. Each wire carries a byte value $v \in \{0, \ldots, 255\}$. [1].
- N: number of garbled circuits.
- t: number of challenged circuits; equivalently, the degree of the underlying polynomials.
- $\ell := N - t$: number of unchallenged circuits.
- C: a public Boolean circuit encoding the NP relation R, with n byte-valued input wires and a single-bit output.

Notation conventions. We index wires by $j \in [n]$ and byte values by $v \in \{0, \ldots, 255\}$. Circuit indices run from 1 to N; index 0 is reserved for the secret in the secret-sharing scheme. Wires $1, \ldots, n_1$ are Phase 1 wires; $n_1 + 1, \ldots, n_1 + n_2$ are Phase 2 wires; and $n_1 + n_2 + 1, \ldots, n$ are Phase 3 wires. We write $\mathcal{C} \subset [N]$ for the challenge set of size t and $\mathcal{U} := [N] \setminus \mathcal{C}$ for the unchallenged set of size ℓ.

4 Preliminaries

4.1 Verifiable Secret Sharing (VSS)

Mosaic uses Feldman's verifiable secret sharing [1] over $\mathbb{F}_p$. The key property is that polynomial commitments can be *homomorphically evaluated*: given a commitment to a polynomial, one can compute the commitment to any evaluation without knowing the polynomial itself.

Definition 1 (Polynomial)**.** *A* polynomial of degree t *is a vector of* $t + 1$ *scalar coefficients* $f = (a_0, a_1, \ldots, a_t) \in \mathbb{F}_p^{t+1}$*, representing*

$$f(X) = \sum_{k=0}^{t} a_k X^k.$$

Definition 2 (Feldman Commitment)**.** *The* Feldman commitment *to a polynomial* $f = (a_0, \ldots, a_t)$ *is the vector of curve points:*

$$\mathsf{Com}(f) := \big([a_0],\ [a_1],\ \ldots,\ [a_t]\big) \in \mathbb{G}^{t+1}.$$

We write $C_f := \mathsf{Com}(f)$.

[1] The construction extends to any fixed wire width. Smaller widths increase the number of wires needed to encode a fixed input, which in turn increases the number of adaptors to store and verify and the amount of on-chain data. Larger widths reduce the number of wires, but require more auxiliary material to translate the representation into bit-wide wires to be used in Boolean circuits. We fix the wire width to 8 throughout the rest of the paper.

Homomorphic evaluation. Given $C_f = ([a_0], \ldots, [a_t])$ and an index α, one can compute $[f(\alpha)]$ without knowing f:

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

Share and share verification. A *share* of f at index α is the pair $(\alpha, f(\alpha))$. A share (α, s) is *valid* with respect to C_f if and only if $[s] = C_f(\alpha)$, which can be checked without knowing f.

Lagrange interpolation. Given $t+1$ shares $\{(\alpha_i, f(\alpha_i))\}_{i=1}^{t+1}$ of a degree-t polynomial, the polynomial is uniquely determined. For a set of known points $\{(\alpha_i, y_i)\}_{i=1}^{t+1}$ and a target index β:

$$f(\beta) = \sum_{i=1}^{t+1} y_i \cdot \prod_{\substack{j=1 \\ j \neq i}}^{t+1} \frac{\beta - \alpha_j}{\alpha_i - \alpha_j}.$$

Security properties. We rely on two standard properties of Feldman VSS:

1. **Secrecy.** For an honest dealer, the commitment C_f together with any set of at most t shares is information-theoretically independent of $f(0)$. In particular, t shares of a degree-t polynomial are consistent with every possible value of $f(0)$.
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 Feldman commitment therefore binds the dealer to a unique polynomial and a unique secret $f(0)$.

4.2 Schnorr Adaptor Signatures

An adaptor signature scheme [3] binds the completion of a signature to the disclosure of a witness. We use Schnorr adaptor signatures where the completed signature is a valid Schnorr signature and it reveals a discrete logarithm witness. In Mosaic, E holds a signing keypair $(\mathsf{sk}, \mathsf{pk})$ with $\mathsf{pk} = [\mathsf{sk}]$ and generates adaptors; G holds pk and completes them.

Definition 3 (Adaptor Signature Scheme)**.** *An adaptor signature scheme over $(\mathbb{G}, \mathbb{F}_p)$ consists of four algorithms.*

$\mathsf{GenAdaptor}(\mathsf{sk}, S, m) \to \widetilde{\sigma}$**.** *Given secret key $\mathsf{sk} \in \mathbb{F}_p$, an adaptor point $S = [y] \in \mathbb{G}$ (with y unknown to the signer), and a message m:*

(i) Sample $r' \xleftarrow{\$} \mathbb{F}_p^*$ and compute $R' := [r']$.
(ii) Compute $R := R' + S$ and challenge $e := H(R, \mathsf{pk}, m)$.
(iii) Compute $s' := r' + e \cdot \mathsf{sk}$.
(iv) Output $\widetilde{\sigma} := (s', R', S)$.

$\mathsf{VerAdaptor}(\mathsf{pk}, m, \widetilde{\sigma}) \to \{0,1\}$**.** *Parse $\widetilde{\sigma} = (s', R', S)$, compute $R := R' + S$ and $e := H(R, \mathsf{pk}, m)$, and accept if and only if $[s'] = R' + e \cdot \mathsf{pk}$.*

$\mathsf{Complete}(\widetilde{\sigma}, y) \to \sigma$**.** *Given $\widetilde{\sigma} = (s', R', S)$ and $y \in \mathbb{F}_p$ with $S = [y]$, output $\sigma := (s' + y,\ R' + S)$. This is a valid Schnorr signature on m under* pk*.*

$\mathsf{Extract}(\widetilde{\sigma}, \sigma) \to y$. *Given* $\widetilde{\sigma} = (s', R', S)$ *and* $\sigma = (s, R)$, *return* $y := s - s'$. *Correctness holds since* $s - s' = (r' + y + e \cdot \mathsf{sk}) - (r' + e \cdot \mathsf{sk}) = y$.

Security properties. We require three properties that we informally state and refer the reader to [3] for a formal treatment.

1. **Adaptability.** If $\mathsf{VerAdaptor}(\mathsf{pk}, m, \widetilde{\sigma}) = 1$ and $S = [y]$, then $\mathsf{Complete}(\widetilde{\sigma}, y)$ outputs a valid signature under pk on m.
2. **Witness extractability.** If $\widetilde{\sigma}$ is a valid adaptor with point $S = [y]$ and σ is any valid signature on m under pk with nonce $R = R' + S$, then $\mathsf{Extract}(\widetilde{\sigma}, \sigma) = y$. More formally, it is infeasible to produce a valid σ for which extraction fails to yield a valid witness; we rely on the multi-instance variant of this property from [3].
3. **Unforgeability (aEUF-CMA).** An adversary with access to signing and adaptor generation oracles cannot produce a valid signature on a fresh message.

4.3 Privacy-Free Garbled Circuits

Mosaic uses *privacy-free* garbled circuits [16], where there is no privacy guarantee: the evaluator sees all wire values in the clear during evaluation.

Definition 4 (Garbling Scheme)**.** *A garbling scheme consists of three algorithms:*

- $\mathsf{GenTable}(1^\kappa, \mathsf{seed}, \{s_{j,v}\}_{j\in[n],v\in[256]}, s^{\mathsf{out}}) \to T$*: given a seed* $\mathsf{seed} \in \{0,1\}^\kappa$*, input labels* $s_{j,v} \in \{0,1\}^\kappa$ *for each wire* j *and value* v*, and an output secret* $s^{\mathsf{out}} \in \mathbb{F}_p$*, produces a garbling table* T *deterministically. The label* $s_{j,v}$ *encodes value* v *on wire* j*; the output secret* s^{out} *is embedded as the label produced when the circuit evaluates to* false*.*
- $\mathsf{EvalTable}(T, (s_1, \ldots, s_n)) \to s$*: given a garbling table and one input label per wire, returns a candidate output secret.*
- $\mathsf{CommitTable}(T) \to H$*: returns a binding commitment to the garbling table.*

Seeded generation. Because T is generated deterministically from $(\mathsf{seed}, \{s_{j,v}\}, s^{\mathsf{out}})$, a challenger can verify any garbling table by re-deriving it from the opened seed and labels. This is what allows challenged circuits to be verified in cut-and-choose without transmitting the tables themselves. In practice, this can provide a significant improvement to communication efficiency.

Correctness. For any $(\mathsf{seed}, \{s_{j,v}\}, s^{\mathsf{out}})$ and any input $\mathbf{x} \in \{0, \ldots, 255\}^n$ with $C(\mathbf{x}) = \mathsf{false}$:

$$\mathsf{EvalTable}\big(T,\ (s_{1,x_1}, \ldots, s_{n,x_n})\big) = s^{\mathsf{out}}.$$

Authenticity. We require that an evaluator holding labels for a *satisfying* input cannot recover the false-output secret. Formally, consider the following game:

(i) The challenger samples $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\kappa$, $s^{\mathsf{out}} \xleftarrow{\$} \mathbb{F}_p$, and input labels $s_{j,v} \xleftarrow{\$} \{0,1\}^\kappa$ for each $j \in [n]$, $v \in [256]$, and computes the garbling table $T \leftarrow \mathsf{GenTable}(1^\kappa, \mathsf{seed}, \{s_{j,v}\}, s^{\mathsf{out}})$.

(ii) The adversary $\mathcal{A}$ receives T, outputs an input $\mathbf{x} \in \{0, \ldots, 255\}^n$, and receives the corresponding labels $(s_{1,x_1}, \ldots, s_{n,x_n})$.

(iii) $\mathcal{A}$ outputs a candidate $\tilde{s}$.

We say $\mathcal{A}$ wins if $C(\mathbf{x}) = \mathsf{true}$ and $\tilde{s} = s^{\mathsf{out}}$. The scheme satisfies *authenticity* if $\Pr[\mathcal{A} \text{ wins}] \leq \mathsf{negl}(\kappa)$ for all PPT $\mathcal{A}$.

Relation to standard definitions. The standard garbling scheme formalism [17] introduces the following security properties: privacy, obliviousness, and authenticity, of which we require only the last. We tailor the definitions to the fraud-proof setting in several ways. In our setting, we collapse the formalism to three algorithms: input labels are external parameters to $\mathsf{GenTable}$ (absorbing an encoding procedure) and the output is a single field element compared against a commitment (eliminating the need for a decoding procedure). $\mathsf{CommitTable}$ provides binding commitments for cut-and-choose, and generation is deterministic from a seed so that challengers can re-derive tables without receiving them. We also emphasize that, in our setting, we require correctness holds for *failing* inputs ($C(\mathbf{x}) = \mathsf{false}$ yields s^{out}) and authenticity protects s^{out} from satisfying inputs, reflecting that the secret is evidence of fraud rather than a proof of knowledge. Finally, our wires carry byte values (256 labels per wire) rather than single bits.

5 Construction Overview

Before formally presenting Mosaic, we provide an overview of how the building blocks of Section 4 compose to achieve the properties required in Section 2.

Polynomial label correlation. A central design choice in Mosaic is that the N garbled circuits are not generated independently. For each input wire $j \in [n]$ and byte value $v \in \{0, \ldots, 255\}$, the Garbler samples a uniformly random degree-t polynomial $f_{j,v} \in \mathbb{F}_p[X]_{\leq t}$ and uses $f_{j,v}(i)$ as the input label for wire j, value v, in circuit i. A single output polynomial f^{out} of degree t is sampled similarly, and $f^{\mathsf{out}}(i)$ is embedded in circuit i as the false output label. The fault secret is the constant term:

$$s^* \;:=\; f^{\mathsf{out}}(0).$$

G publishes Feldman commitments $\mathcal{C}_{j,v} = \mathsf{Com}(f_{j,v})$ and $\mathcal{C}^{\mathsf{out}} = \mathsf{Com}(f^{\mathsf{out}})$ for all j, v, along with commitments to each of the N garbling tables. These commitments bind G to a unique polynomial per wire-value pair and to a unique s^*, before any challenge is issued.

Cut and choose. E challenges a uniformly random subset $\mathcal{C} \subset [N]$ of size t. For each challenged circuit $i \in \mathcal{C}$, G opens the garbling seed, all input labels $\{f_{j,v}(i)\}_{j,v}$, and the output share $f^{\mathsf{out}}(i)$. E verifies each opening against the Feldman commitments and re-derives the garbling table from the seed to check table consistency; it aborts if any check fails.

After this interaction, E holds:

- For each wire j and value v: t verified evaluations $\{f_{j,v}(i)\}_{i \in \mathcal{C}}$, which are t shares of a degree-t polynomial with reconstruction threshold $t + 1$.
- t output shares $\{f^{\mathsf{out}}(i)\}_{i \in \mathcal{C}}$, similarly one short of the reconstruction threshold.
- The ℓ unchallenged garbling tables $\{T_i\}_{i \in \mathcal{U}}$, whose commitments were verified against the pre-committed hashes.

Adaptor signatures as the missing share. There is one missing share needed to reach the reconstruction threshold for input polynomials, and we arrange that G reveals this on chain while authorizing the Witness transaction. Recall from Section 2 that the Witness transaction requires G to post one signature per input wire under E's signing key. Since G does not hold that key, the only way it can produce these signatures is by completing adaptors that E provides.

During the Adaptor Setup phase, E uses the Feldman commitments to input polynomials to derive $[f_{j,v}(0)] = \mathcal{C}_{j,v}(0)$ without knowing the shares themselves, and for each Phase 3 wire $j \in [n_3]$

and byte value $v \in \{0, \ldots, 255\}$ generates an adaptor with point $[f_{j,v}(0)]$, yielding 256 per wire each binding a potential on-chain signature to the disclosure of $f_{j,v}(0)$. G verifies each adaptor against its retained reserved shares and aborts if any mismatch is detected.

When G commits to its witness input v_j^* on wire j, it completes the adaptor for value v_j^* using the witness $f_{j,v_j^*}(0)$, producing a valid Schnorr signature that it posts on chain as the Witness transaction. By witness extractability (Section 4.2), E reads the completed signature, tries extraction against all 256 adaptors, and recovers the unique $f_{j,v_j^*}(0)$ whose commitment matches the missing $t+1$-th share. E now holds $t+1$ shares of f_{j,v_j^*} and applies Lagrange interpolation to recover the full polynomial, from which it evaluates $f_{j,v_j^*}(i)$ for every unchallenged circuit $i \in \mathcal{U}$. The on-chain cost is one field element per input wire, regardless of ℓ.

The three input types. The mechanism above applies in full to Phase 3 wires, where the value v_j^* is unknown until G commits to the witness. Phase 1 wires carry values (corresponding to the agreed-upon public statement) known to both parties from the start; G opens the reserved share $f_{j,x_{1,j}}(0)$ directly during Garbling Setup, with no adaptor required. Phase 2 wires carry values that become known once the Kickoff transaction is confirmed on chain (Section 2); E generates a single (rather than 256) adaptor per wire corresponding to the known public value.

Output reconstruction and fault secret recovery. After obtaining input labels corresponding to the public statement $\mathbf{x}$ and garbler's witness $\mathbf{w}$ for all ℓ unchallenged garbled circuits, E evaluates them in sequence. For each unchallenged circuit $i \in \mathcal{U}$, it runs $\mathsf{EvalTable}(T_i, \{f_{j,v_j^*}(i)\}_{j\in[n]})$, obtaining a candidate output value s_i^{out}. If $C(\mathbf{x}) = \mathsf{false}$, garbling correctness guarantees $s_i^{\mathsf{out}} = f^{\mathsf{out}}(i)$ for any honestly constructed circuit i. Together with the t output shares from the challenge set, E holds $t+1$ points on the degree-t polynomial f^{out}, which suffices to interpolate and recover $s^* = f^{\mathsf{out}}(0)$. E confirms correctness by checking $[s^*] = \mathcal{C}^{\mathsf{out}}(0)$ against the public commitment. If $C(\mathbf{x}) = \mathsf{true}$, the authenticity property of the garbling scheme ensures that s_i^{out} is computationally independent of $f^{\mathsf{out}}(i)$, so E cannot reconstruct s^*.

Soundness probability. For G to cheat without being detected, all ℓ unchallenged circuits must be corrupted: if even one is honestly garbled, evaluation on an invalid witness yields $f^{\mathsf{out}}(i)$ and E recovers s^*. At the same time, all corrupted circuits must fall outside the challenge set, or E detects the inconsistency and aborts during Garbling Setup. Since the challenge set is chosen uniformly at random after G commits, the probability that a fixed set of ℓ corrupted circuits all avoid the t-element challenge is

$$\frac{\binom{N-\ell}{t}}{\binom{N}{t}} = \frac{1}{\binom{N}{\ell}}.$$

We set (N, ℓ) such that the above is less than $2^{-\rho}$ for some statistical security parameter ρ, typically set to 40 in the literature.

6 Mosaic Construction

The Mosaic protocol involves two parties: the **Garbler** G and the **Evaluator** E. Execution proceeds in three phases: **Garbling Setup**, **Adaptor Setup**, and **Witness Resolution**. Each step is attributed to the acting party by a colored sidebar: Garbler (left) Evaluator (right)

Garbling Setup Phase

Both parties hold common public input $\mathbf{x}_1 = (x_{1,1}, \ldots, x_{1,n_1}) \in \{0, \ldots, 255\}^{n_1}$ and have access to independent sources of randomness. This phase takes place before any funds are locked.

Garbler: Commit

S1. Sample polynomials. For each input wire $j \in [n]$ and each byte-value $v \in \{0, \ldots, 255\}$, sample a random degree-t polynomial:

$$f_{j,v} \xleftarrow{\$} \mathbb{F}_p[X]_{\leq t}.$$

Additionally, sample one output polynomial:

$$f^{\mathsf{out}} \xleftarrow{\$} \mathbb{F}_p[X]_{\leq t}.$$

The *fault secret* is defined as $s^* := f^{\mathsf{out}}(0)$.

S2. Commit to polynomials. Compute polynomial commitments:

$$C_{j,v} \leftarrow \mathsf{Com}(f_{j,v}) \quad \forall\, j \in [n],\ v \in [256], \qquad C^{\mathsf{out}} \leftarrow \mathsf{Com}(f^{\mathsf{out}}).$$

S3. Generate garbled circuits. For each circuit index $i \in [N]$, sample a garbling seed $\mathsf{seed}_i \xleftarrow{\$} \{0,1\}^\kappa$ and compute:

$$T_i \leftarrow \mathsf{GenTable}\Big(1^\kappa,\ \mathsf{seed}_i,\ \{f_{j,v}(i)\}_{j\in[n],\, v\in[256]},\ f^{\mathsf{out}}(i)\Big).$$

Let $H_i \leftarrow \mathsf{CommitTable}(T_i)$ denote the garbling table commitment.

S4. Send commitment. $\mathsf{G} \to \mathsf{E}$:

$$\mathsf{CommitMsg} = \Big(\{C_{j,v}\}_{j,v},\ C^{\mathsf{out}},\ \{H_i\}_{i\in[N]}\Big).$$

Evaluator: Challenge

S5. Sample challenge set. E samples a uniformly random subset $\mathcal{C} \subset [N]$ of size $|\mathcal{C}| = t$. Let $\mathcal{U} := [N] \setminus \mathcal{C}$ denote the set of ℓ unchallenged indices.

S6. Send challenge. $\mathsf{E} \to \mathsf{G}$:

$$\mathsf{ChallengeMsg} = \mathcal{C}.$$

Garbler: Respond

S7. Open challenged circuits. For each $i \in \mathcal{C}$, open:

- Input shares: $\{f_{j,v}(i)\}_{j\in[n],\, v\in[256]}$.
- Output share: $f^{\mathsf{out}}(i)$.
- Garbling seed: seed_i.

S8. Open reserved setup-input shares. For each Phase 1 wire $j \in [n_1]$, open the share at the reserved index corresponding to the known input value:

$$s_j^{(0)} := f_{j,\,x_{1,j}}(0) \quad \text{for } j \in [n_1].$$

S9. Provide unchallenged garbling tables. For each $i \in \mathcal{U}$, send the garbling table T_i.

S10. Retain reserved shares. G retains for later use the reserved shares for Phase 2 and Phase 3 wires:

$$\left\{f_{j,v}(0)\right\}_{j\in[n_2]\cup[n_3],\, v\in[256]}.$$

S11. Send response. $\mathsf{G} \to \mathsf{E}$:

$$\mathsf{ResponseMsg} = \left(\left\{f_{j,v}(i)\right\}_{\substack{i\in\mathcal{C},\, j\in[n],\\ v\in[256]}},\ \left\{s_j^{(0)}\right\}_{j\in[n_1]},\ \left\{f^{\mathsf{out}}(i)\right\}_{i\in\mathcal{C}},\ \left\{\mathsf{seed}_i\right\}_{i\in\mathcal{C}},\ \left\{T_i\right\}_{i\in\mathcal{U}}\right).$$

Evaluator: Verify

S12. Verify response. E checks the following; abort if any check fails.

(a) *Input share consistency.* For every $i \in \mathcal{C}$, $j \in [n]$, $v \in [256]$:

$$\left[f_{j,v}(i)\right] = C_{j,v}(i).$$

(b) *Output share consistency.* For every $i \in \mathcal{C}$:

$$\left[f^{\mathsf{out}}(i)\right] = C^{\mathsf{out}}(i).$$

(c) *Challenged garbling table consistency.* For every $i \in \mathcal{C}$, re-derive the garbling table from the opened seed and shares and verify its commitment:

$$\mathsf{CommitTable}\Big(\mathsf{GenTable}\big(1^\kappa,\ \mathsf{seed}_i,\ \{f_{j,v}(i)\}_{j,v},\ f^{\mathsf{out}}(i)\big)\Big) = H_i.$$

(d) *Unchallenged garbling table consistency.* For every $i \in \mathcal{U}$:

$$\mathsf{CommitTable}(T_i) = H_i.$$

(e) *Phase 1 input share consistency.* For every Phase 1 wire $j \in [n_1]$:

$$\left[s_j^{(0)}\right] = C_{j,\,x_{1,j}}(0).$$

S13. Store derived data. E computes and stores the *input share commitments* at the reserved index for Phase 2 and Phase 3 wires:

$$Y_{j,v} := C_{j,v}(0) = \left[f_{j,v}(0)\right] \quad \text{for } j \in [n_2] \cup [n_3],\ v \in [256]$$

and the output secret commitment:

$$Y^* := C^{\mathsf{out}}(0) = [s^*].$$

E also retains all opened shares $\{f_{j,v}(i)\}_{i\in\mathcal{C}}$, output shares $\{f^{\mathsf{out}}(i)\}_{i\in\mathcal{C}}$, and unchallenged garbling tables $\{T_i\}_{i\in\mathcal{U}}$.

Adaptor Setup Phase

Once the input UTXOs that fund the Kickoff transaction are agreed, both parties know the Phase 2 component of the statement: the Phase 2 input $\mathbf{x}_2 = (x_{2,1}, \ldots, x_{2,n_2}) \in \{0, \ldots, 255\}^{n_2}$ and the messages $\{m_j\}_{j \in [n_2+n_3]}$ tied to the Kickoff txid (one per Phase 2 and Phase 3 wire). The rationale for this partition of input types is given in Section 5. E holds a signing secret key sk with public key $\mathsf{pk} = [\mathsf{sk}]$; G knows pk.

Evaluator: Generate Adaptors

D1. Phase 2 wire adaptors. For each Phase 2 wire $j \in [n_2]$, the input value $x_{2,j}$ is known. Generate a single adaptor:

$$\widetilde{\sigma}_j \leftarrow \mathsf{GenAdaptor}\big(\mathsf{sk},\ Y_{j,\,x_{2,j}},\ m_j\big).$$

The adaptor point $Y_{j,\,x_{2,j}} = [f_{j,\,x_{2,j}}(0)]$ binds this adaptor to the reserved share for the known value.

D2. Phase 3 wire adaptors. For each Phase 3 wire $j \in [n_3]$, the input value is not yet known. Generate one adaptor per possible value:

$$\widetilde{\sigma}_{j,v} \leftarrow \mathsf{GenAdaptor}\big(\mathsf{sk},\ Y_{j,v},\ m_{n_2+j}\big) \quad \text{for } v \in \{0, \ldots, 255\}.$$

D3. Send adaptors. $\mathsf{E} \to \mathsf{G}$:

$$\mathsf{AdaptorsMsg} = \Big(\big\{\widetilde{\sigma}_j\big\}_{j \in [n_2]},\ \big\{\widetilde{\sigma}_{j,v}\big\}_{j \in [n_3],\, v \in [256]}\Big).$$

Garbler: Verify Adaptors

D4. Verify all adaptors. G verifies each adaptor and checks that its adaptor point matches the expected reserved share commitment. Abort if any check fails.

(a) For each $j \in [n_2]$: check that $\widetilde{\sigma}_j$ has adaptor point $[f_{j,\,x_{2,j}}(0)]$ and

$$\mathsf{VerAdaptor}\big(\mathsf{pk},\ m_j,\ \widetilde{\sigma}_j\big) = 1.$$

(b) For each $j \in [n_3]$, $v \in [256]$: check that $\widetilde{\sigma}_{j,v}$ has adaptor point $[f_{j,v}(0)]$ and

$$\mathsf{VerAdaptor}\big(\mathsf{pk},\ m_{n_2+j},\ \widetilde{\sigma}_{j,v}\big) = 1.$$

D5. Store. G stores the verified adaptors $\{\widetilde{\sigma}_j\}_{j \in [n_2]}$, $\{\widetilde{\sigma}_{j,v}\}_{j \in [n_3], v}$ along with its retained reserved shares, partitioned as:

$$\begin{aligned} &\text{Phase 2 input shares:} && s_j^{(2)} := f_{j,\,x_{2,j}}(0) \quad \text{for } j \in [n_2], \\ &\text{Phase 3 input shares:} && \big\{f_{j,v}(0)\big\}_{j \in [n_3],\, v \in [256]}. \end{aligned}$$

Witness Resolution Phase

G receives the witness $\mathbf{x}_3 = (x_{3,1}, \ldots, x_{3,n_3}) \in \{0, \ldots, 255\}^{n_3}$ and broadcasts the Witness transaction (Section 2) by completing and posting the required signatures on chain.

Garbler: Sign *(on chain)*

W1. Complete Phase 2 adaptors. For each Phase 2 wire $j \in [n_2]$:

$$\sigma_j \leftarrow \mathsf{Complete}\big(\widetilde{\sigma}_j,\ s_j^{(2)}\big).$$

W2. Complete Phase 3 adaptors. For each Phase 3 wire $j \in [n_3]$, complete the adaptor corresponding to $x_{3,j}$:

$$\sigma_{n_2+j} \leftarrow \mathsf{Complete}\big(\widetilde{\sigma}_{j,\,x_{3,j}},\ f_{j,\,x_{3,j}}(0)\big).$$

W3. Post on chain. G publishes the $n_2 + n_3$ completed signatures $\{\sigma_k\}_{k\in[n_2+n_3]}$ on chain as the Witness transaction.

Evaluator: Reveal Secret

W4. Extract reserved input shares. E reads the on-chain signatures and extracts the reserved shares used to complete each adaptor.

(a) For each Phase 2 wire $j \in [n_2]$:

$$s_j^{(0)} \leftarrow \mathsf{Extract}\big(\widetilde{\sigma}_j,\ \sigma_j\big) = f_{j,\,x_{2,j}}(0).$$

(b) For each Phase 3 wire $j \in [n_3]$, E does not learn $x_{3,j}$ directly from G; instead it determines the committed input value by trying extraction against all 256 adaptors. For each $v \in \{0, \ldots, 255\}$, compute

$$y_v \leftarrow \mathsf{Extract}\big(\widetilde{\sigma}_{j,v},\ \sigma_{n_2+j}\big)$$

and check $[y_v] \stackrel{?}{=} Y_{j,v}$. Let v_j^* be the unique value for which this holds, and set $s_{n_2+j}^{(0)} := y_{v_j^*} = f_{j,v_j^*}(0)$.

At this point, for every input wire $j \in [n]$, E has determined v_j^* and holds:

$$\underbrace{\{f_{j,\,v_j^*}(i)\}_{i\in\mathcal{C}}}_{t \text{ shares from Garbling Setup}} \quad \cup \quad \underbrace{\{f_{j,\,v_j^*}(0)\}}_{1 \text{ share from extraction}} \quad = \ t+1 \text{ shares of } f_{j,\,v_j^*}.$$

W5. Interpolate input polynomials. For each wire $j \in [n]$, interpolate using the $t+1$ shares:

$$f_{j,v_j^*} \leftarrow \mathsf{Interpolate}\Big(\big\{\big(i,\ f_{j,\,v_j^*}(i)\big)\big\}_{i\in\mathcal{C}\cup\{0\}}\Big).$$

Since $\deg(f_{j,v_j^*}) = t$ and $|\mathcal{C} \cup \{0\}| = t+1$, the polynomial is uniquely determined.

W6. Derive input labels for unchallenged circuits. For each unchallenged circuit $i \in \mathcal{U}$ and wire $j \in [n]$:

$$s_{i,j} := f_{j,v_j^*}(i).$$

W7. Evaluate garbled circuits and recover fault secret. E holds t output shares from Garbling Setup: $\{(i, f^{\mathsf{out}}(i))\}_{i \in \mathcal{C}}$. For each unchallenged circuit $i \in \mathcal{U}$ in sequence:

(a) Evaluate the garbled table:

$$s_i^{\mathsf{out}} \leftarrow \mathsf{EvalTable}\big(T_i,\ (s_{i,1}, \ldots, s_{i,n})\big).$$

(b) Append to the output share set:

$$S^{\mathsf{out}} \leftarrow \big\{(i, f^{\mathsf{out}}(i))\big\}_{i \in \mathcal{C}} \cup \big\{(i, s_i^{\mathsf{out}})\big\}.$$

Now $|S^{\mathsf{out}}| = t + 1$.

(c) Interpolate and recover the candidate secret:

$$s^* \leftarrow \mathsf{Interpolate}(S^{\mathsf{out}})(0).$$

(d) Check against the stored commitment:

$$[s^*] \stackrel{?}{=} Y^*.$$

If equal, output s^* and halt.

If no unchallenged circuit yields a matching secret, output $\perp$.

7 Security Analysis

We argue informally that Mosaic satisfies the two properties stated in Section 2: accountability (if G posts a failing witness, E recovers s^* and can post the Witness Fault transaction) and safety (if G's witness is valid, E cannot recover s^* and G eventually reaches Payout).

7.1 Accountability

Claim 1. *If* G *commits on-chain to an input* $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ *such that* $C(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \mathsf{false}$*, then either* E *aborts during Garbling Setup, or* E *recovers* s^* *with probability at least* $1 - 1/\binom{N}{\ell}$.

Proof sketch. If all checks in step **S12.** pass, then by the binding property of Feldman commitments, each $C_{j,v}$ and C^{out} determines a unique polynomial. The checks verify that the opened shares and seeds at challenged indices are indeed evaluations of the committed polynomials, and that the unchallenged garbling tables match their pre-committed hashes. Any corrupted table in the challenge set is detected and causes abort.

The Witness transaction requires one signature per input wire under E's signing key. Since G does not hold that key, aEUF-CMA ensures that any valid signature it posts must be a completion of one of E's pre-signatures; otherwise it would constitute a forgery under E's key. By multi-instance witness extractability [3], each completion necessarily discloses the witness of the adaptor point, which is the reserved share $f_{j,v_j^*}(0)$. For Phase 2 wires, there is one pre-signature per wire, so the share is recovered directly. For Phase 3 wires, E generated 256 pre-signatures per wire each

with a distinct nonce, so the completed signature uniquely identifies which adaptor was used and extraction yields $f_{j,v_j^*}(0)$.

Together with the t input shares opened at challenged indices in step **S7.**, E holds $t+1$ points on each polynomial f_{j,v_j^*} and interpolates to recover the correct input label $f_{j,v_j^*}(i)$ for every unchallenged garbled table $i \in \mathcal{U}$.

Consider the bad event that all unchallenged garbled tables are corrupted, yet G passes **S12.**. For this to happen, G must corrupt exactly ℓ circuits and have all ℓ avoid E's random challenge. Since G commits to the garbling tables before seeing the challenge, this probability is exactly $1/\binom{N}{\ell}$. With the complementary probability, at least one unchallenged circuit is honestly garbled. On that evaluation (recall $\mathbf{x}$ is such that $C(\mathbf{x}) = \mathsf{false}$), garbling correctness gives

$$\mathsf{EvalTable}\big(T_{i^*},\, (f_{1,v_1^*}(i^*), \ldots, f_{n,v_n^*}(i^*))\big) = f^{\mathsf{out}}(i^*).$$

Combined with the t output shares opened at step **S7.**, E holds $t+1$ points on the degree-t polynomial f^{out} and recovers $s^* = f^{\mathsf{out}}(0)$ by interpolation. □

7.2 Safety

Claim 2. *If* G *honestly executes the protocol with an input satisfying* $C(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \mathsf{true}$*, then no PPT adversary playing the role of* E *can recover* s^* *with non-negligible probability.*

Proof sketch. If an adversarial E provides maliciously generated adaptors at step **D3.**, G aborts at step **D4.**. Since the Kickoff transaction has not yet been broadcast, no funds are locked and G suffers no loss. We consider the case where adaptor verification passes and the protocol reaches the Witness Resolution phase.

The adversary's view consists of the Feldman commitments, t opened input and output polynomial shares and garbled tables from the challenge set, unchallenged garbling tables, and completed signatures on the Witness transaction. We argue that none of these reveal s^*.

The t output shares $\{f^{\mathsf{out}}(i)\}_{i \in \mathcal{C}}$ are evaluations of a degree-t polynomial at t non-zero points. By the secrecy of Feldman VSS, these are information-theoretically independent of the constant term $s^* = f^{\mathsf{out}}(0)$.

For each unchallenged circuit $i \in \mathcal{U}$, the garbling table T_i was generated with $f^{\mathsf{out}}(i)$ as the false-output label. Since $C(\mathbf{x}) = \mathsf{true}$, holding appropriate input labels and an honestly generated garbling table T_i does not help an adversary recover $f^{\mathsf{out}}(i)$ by authenticity of the garbling scheme (Definition 4). The adversary also sees $[f^{\mathsf{out}}(i)]$ via the public commitment C^{out}, but this does not reveal $f^{\mathsf{out}}(i)$ by the hiding property of the commitment. Similarly, it is infeasible to recover s^* directly from its public commitment $Y^* = [s^*]$.

The completed Witness transaction signatures reveal the reserved input shares $\{f_{j,v_j^*}(0)\}_j$, which are evaluations of polynomials sampled independently of f^{out}. G's adaptor verification at step **D4.** ensures the adversary cannot engineer adaptors whose completions would reveal information about f^{out}. □

8 Protocol Summary

We summarize the five-message protocol below. G is always on the left, E on the right; the arrow indicates message direction.

Garbling Setup Phase

1. G $\xrightarrow{\text{CommitMsg}}$ E

G *computes*: Sample polynomials $\{f_{j,v}\}, f^{\text{out}}$; commit to each. Generate N garbled circuits from polynomial shares; commit to tables.

2. G $\xleftarrow{\text{ChallengeMsg}}$ E

E *computes*: Sample challenge set $\mathcal{C} \subset [N]$, $|\mathcal{C}| = t$; send $\mathcal{C}$.

3. G $\xrightarrow{\text{ResponseMsg}}$ E

G *computes*: Open challenged shares and seeds; open reserved Phase 1 input shares. Provide unchallenged garbling tables; retain Phase 2/Phase 3 reserved shares.

E *verifies*: Verify all shares against commitments; verify garbling table consistency. Derive adaptor points $\{Y_{j,v}\}$ and output commitment Y^*.

Adaptor Setup Phase

4. G $\xleftarrow{\text{AdaptorsMsg}}$ E

E *computes*: Generate n_2 Phase 2 wire adaptors and $256 \cdot n_3$ Phase 3 wire adaptors.

G *verifies*: Verify all adaptors against retained reserved shares; store.

Witness Resolution Phase

5. G $\xrightarrow{\text{SigMsg}}$ E (on chain)

G *computes*: Complete all Phase 2 and selected Phase 3 adaptors; post signatures on chain.

E *computes*: Extract reserved shares from signatures; interpolate input polynomials. Evaluate unchallenged garbled circuits; interpolate output polynomial. Verify $[s^*] = Y^*$; output fault secret s^* or $\perp$.

9 Instantiation on Bitcoin

In this section, we instantiate Mosaic in the setting of a trust-minimized Bitcoin bridge and explain how the abstract roles and protocol phases of Section 6 map to a concrete deployment. We consider a bridge in which a federation of operators controls a Bitcoin multi-signature wallet backing assets on a sidechain. To participate in the bridge, each operator locks a stake that can be slashed if the operator makes a dishonest withdrawal claim. Users lock Bitcoin into the bridge to mint sidechain tokens and later redeem those tokens by requesting a withdrawal on Bitcoin.

To service a withdrawal, an operator first fronts the corresponding Bitcoin payment to the user and then seeks reimbursement from the bridge by submitting a proof that the withdrawal was

fulfilled correctly. Mosaic is not used to prove this claim directly. Rather, it is used to enable a counterproof: if the claiming operator submits an invalid proof, another operator can demonstrate that the proof is false for the particular withdrawal instance under dispute.

Concretely, each withdrawal claim is tied to a specific *deposit index*, and the identity of the claiming operator is fixed during an initial *operator setup*. Together, these values determine the statement against which the claim is checked. Any non-participating operator can act as a *watchtower*. If the claimant submits an invalid proof, a watchtower can challenge by showing that the submitted proof does not verify for that operator key and deposit index. Mosaic guarantees that a valid counterproof causes the dishonest operator's stake to be slashed, while an honest operator cannot be punished.

9.1 Cryptographic Instantiation

This instantiation is chosen to align all cryptographic objects with mechanisms already native to Bitcoin.

Group. All group operations use secp256k1, the elliptic curve that Bitcoin uses natively. The scalar field $\mathbb{F}_p$ is the secp256k1 scalar field, and G is the standard generator. This fixes the group for all Feldman commitments, for the $[s] := s \cdot G$ commitment notation used throughout, and for the adaptor signatures.

Schnorr adaptor signatures. We instantiate the adaptor signature scheme of Section 4.2 using BIP-340 Schnorr signatures over secp256k1 [21]. The hash function H in GenAdaptor is the BIP-340 tagged hash. Accordingly, the Witness transaction spending condition can be expressed using a standard BIP-340 signature check per wire, without any custom verification logic.

As observed by Linus [5], a single public key suffices for all $n_2 + n_3$ signatures while binding each to a distinct message; the exact script is given in [5].

Verifiable secret sharing. We use Feldman VSS over secp256k1 as defined in Section 4.1.

Garbling scheme. We use a privacy-free garbled circuit scheme, one concrete instantiation of which is the half-gates construction [22] adapted to the privacy-free setting. We also estimate costs based on promising recent work [13, 14, 15] that significantly reduces garbled table size for a Groth16 verifier. Labels are elements of $\{0,1\}^\kappa$ obtained by truncating the binary representation of field elements $f_{j,v}(i) \in \mathbb{F}_p$.

9.2 The NP Relation

A primary motivation for Mosaic is the problem of trust-minimized bridges for Bitcoin. The central challenge is that Bitcoin's scripting language cannot verify arbitrary computation on chain and in particular, it cannot directly verify a SNARK proof. The BitVM line of work [8, 9] addresses this via an optimistic protocol: a prover commits to a garbled SNARK verifier off chain, and if the prover later submits an invalid proof, the verifier uses the garbled circuit to extract a fault secret and post a fraud proof on chain. Indeed, the fair exchange protocol of Section 2 is a simplified instance of this setting that isolates the core properties Mosaic must provide; in the bridge application, the same properties are required to ensure that a watchtower can provide a valid counterproof in exchange for slashing the operator of their stake. The NP relation is a Groth16 verifier: $R(x, w) = 1$ iff w

is a valid Groth16 proof for the statement x. The circuit C encoding R takes three types of input corresponding to the three wire partitions.

- **Phase 1 wires** ($n_1 = 32$): the operator public key, agreed upon before any funds are locked and fixed for the lifetime of the contract. This is used to ensure that the proof being challenged is authenticated by the operator.
- **Phase 2 wires** ($n_2 = 4$): the deposit index, which identifies the specific instance of the relation being proved. This is determined by the Kickoff transaction's identifier and so cannot be fixed until the Adaptor Setup phase. This ensures that proofs are tied to the particular instance and cannot be reused across independent instances of deposits and withdrawals.
- **Phase 3 wires** ($n_3 = 128$): the Groth16 proof bytes, which constitute the witness w that watchtower commits to on chain via the Witness transaction.

The circuit C verifies that the 128-byte Groth16 proof submitted by the operator for a particular operator key and deposit index is invalid.

9.3 Protocol Timeline

The three phases of Mosaic map onto the Bitcoin timeline as follows.

- **Garbling Setup** takes place entirely before the Kickoff transaction is broadcast. The garbling and cut-and-choose interaction depend only on the agreed circuit C and the setup inputs $\mathbf{x}_1$; neither the transaction graph nor the Kickoff txid is needed. Neither party has committed any funds and either can walk away without loss.
- **Adaptor Setup** begins once both parties have agreed on the input UTXOs that will fund the Kickoff transaction, since the Kickoff txid is needed to construct the messages $\{m_j\}$ that the adaptors are bound to. Both parties first pre-sign the Witness Fault, Witness Timeout, Payout, and Cooperative Payout transactions. The Adaptor Setup phase then runs: V generates and sends the adaptors, P verifies them, and both parties co-sign the Kickoff transaction. P broadcasts the Kickoff transaction; once it confirms, the timelocks $\Delta_{\mathsf{Witness}}$ and Δ_{Payout} begin.
- **Witness Resolution** corresponds to P broadcasting the Witness transaction before $\Delta_{\mathsf{Witness}}$ expires. The Witness transaction's input contains the completed adaptor signatures (steps **W1.**–**W3.**), one per Phase 2 and Phase 3 wire, which simultaneously authorize the transaction on chain and deliver the garbled circuit input labels to V. V evaluates the garbled circuit off chain (steps **W4.**–**W7.**): if P's witness fails R, V recovers s^* and broadcasts the Witness Fault transaction, claiming both funds.

Outside the three protocol phases, two additional paths may be taken depending on the behavior of the parties. During the $\Delta_{\mathsf{Witness}}$ window, P may send V her witness off chain; if V accepts, he co-signs the Cooperative Payout transaction and both parties settle without posting the Witness transaction at all. This cooperative path is the expected execution when both parties are honest. If instead P does not post the Witness transaction before $\Delta_{\mathsf{Witness}}$ expires, V broadcasts the Witness Timeout transaction and recovers both funds; if P posts the Witness transaction but V does not post Witness Fault before Δ_{Payout} expires, P broadcasts the Payout transaction and recovers both funds. Before the Kickoff transaction confirms, either party can abort at any time by spending their input UTXO elsewhere.

9.4 Parameters and Costs

In this section, we report on the total communication, storage and computational costs of Mosaic, first expressed in terms of protocol parameters $n, n_1, n_2, n_3, N, t, \ell, |T|$ and cryptographic parameters $\lambda_S, \lambda_P, \lambda_{P_x}, \kappa, \lambda_\sigma, \lambda_H$, where λ_S is the scalar size, λ_P is the compressed point size, λ_{P_x} is the size of x-coordinate of the curve point, κ is the label size (computational security parameter), λ_σ is the signature size, and λ_H is the size of hash output of the garbling table commitment, all expressed in bytes. Using secp256k1 as the underlying curve instantiation: $\lambda_S = 32$, $\lambda_P = 33$, $\lambda_{P_x} = 32$, $\kappa = 16$, $\lambda_\sigma = 64$, $\lambda_H = 32$.

9.4.1 Symbolic Costs

Communication and storage. Table 1 summarizes the total communication and storage costs. CommitMsg carries the Feldman commitments to all $n \cdot 256 + 1$ polynomials (input and output) and the N garbling table hashes. ResponseMsg carries the opened shares $\{f_{j,v}(i)\}$ for all wire-value pairs at challenged indices, the reserved Phase 1 input shares at index 0, the t output shares and garbling seeds for challenged circuits, and the ℓ unchallenged garbling tables. Garbler storage consists of the reserved shares $\{f_{j,v}(0)\}$ for Phase 2 and Phase 3 wires retained at step **S10.**, and the verified adaptor pre-signatures stored at step **D5.**. Evaluator storage consists of the zero-coefficient commitments $C_{j,v}(0)$ retained after step **S13.** in place of the full Feldman commitments, the opened input shares from the challenge set, and the ℓ unchallenged garbling tables.

Artifact	Size
Communication	
CommitMsg (step **S4.**)	$(n \cdot 256 + 1) \cdot (t + 1) \cdot \lambda_P + N \cdot \lambda_H$
ResponseMsg (step **S11.**)	$t \cdot n \cdot 256 \cdot \lambda_S + n_1 \cdot \lambda_S + t \cdot \lambda_S + t \cdot \kappa + \ell \cdot \lvert T \rvert$
AdaptorsMsg (step **D3.**)	$(n_3 \cdot 256 + n_2) \cdot (\lambda_S + \lambda_P)$
Total	CommitMsg + ResponseMsg + AdaptorsMsg
Storage	
Garbler	$(n_2 + n_3) \cdot 256 \cdot \lambda_S + (n_3 \cdot 256 + n_2) \cdot (\lambda_S + \lambda_P)$
Evaluator	$n \cdot 256 \cdot \lambda_P + t \cdot n \cdot 256 \cdot \lambda_S + \ell \cdot \lvert T \rvert$
On-chain	
Locking script	$\lambda_{P_x} + 3 \cdot (n_2 + n_3) - 1$
Script inputs	$(n_2 + n_3) \cdot (\lambda_\sigma + 1)$ [2]

Table 1: Communication, storage, and on-chain costs of Mosaic.

Computation. We report computational costs in terms of garbled circuit operations (Garble, EvalTable, Hash) and group operations (PSM, MSM); scalar field operations are negligible in comparison and are omitted. PSM denotes a point-scalar multiplication in secp256k1, MSM a multi-scalar multiplication, Garble the cost of generating one garbling table, EvalTable the cost of evaluating one garbling table, and Hash the cost of one CommitTable call.

The $t \cdot n \cdot 256$ individual share checks at **S12.**(a) can be batched into a single MSM of size $n \cdot 256 \cdot (t + 1)$ using a random linear combination. Concretely, E samples $\alpha \xleftarrow{\$} \mathbb{F}_p$ and verifies the

[2] The +1 accounts for the varint length prefix per witness item, as specified in BIP-141 [4]

Phase	Step	Party	Operation	Count
Garbling Setup	**S2.**	G	MSM	$(n \cdot 256 + 1)(t + 1)$ points
Setup	**S3.**	G	Garble + Hash	N tables
Garbling Setup	**S9.**	G	Garble + Hash	ℓ tables (from seed)
Garbling Setup	**S12.**(a)	E	MSM	$n \cdot 256 \cdot (t + 1)$ points
Garbling Setup	**S12.**(c)	E	Garble + Hash	t tables
Garbling Setup	**S12.**(d)	E	Hash	ℓ tables
Adaptor Setup	**D2.**	E	PSM	$n_3 \cdot 256 + n_2$
Adaptor Setup	**D4.**	G	PSM	$n_3 \cdot 256 + n_2$
Witness Resolution	**W7.**	E	EvalTable	up to ℓ tables

Table 2: Dominant computational operations per protocol step.

single equation

$$\left[\sum_{k=1}^{M} \alpha^k \cdot s_k\right] = \sum_{k=1}^{M} \alpha^k \cdot C_{j_k,v_k}(i_k),$$

where k indexes all $(j, v, i) \in [n] \times [256] \times \mathcal{C}$ in a canonical ordering and $s_k = f_{j_k,v_k}(i_k)$ is the corresponding opened share. The right-hand side expands into a single MSM over the $n \cdot 256 \cdot (t+1)$ Feldman commitment points. If any individual share is inconsistent, the check fails except with probability M/p over the choice of α, which is negligible. At Witness Resolution, step **W7.** halts as soon as the interpolated output value satisfies $[s^*] = Y^*$, so ℓ evaluations is the (unlikely) worst case.

9.4.2 Concrete Instantiation with PFHG

Using parameters $n_1 = 32$, $n_2 = 4$, $n_3 = 128$ ($n = 164$), $N = 181$, $t = 174$, $\ell = 7$, and the state-of-the-art Groth16 verifier Boolean circuit [23] with 3 billion AND gates, the PFHG garbling table size is $|T| = 3 \times 10^9 \times \kappa = 48$ GB. Our implementation uses `g16` [24], a fork of the BitVM Alliance's garbled SNARK verifier [23] modified for compatibility with SP1 proof structure. Garbling and evaluation are performed using `ckt` [25], our privacy-free garbled circuit implementation optimized for circuits at the billion-gate scale. To the best of our knowledge, before `ckt`, no general-purpose garbling library could handle circuits of this size within practical memory bounds.

The garbling tables dominate communication and storage. Setting ℓ small and t large (many garblings need to be performed, but only few garbled tables need to be transmitted) is appropriate in this regime since garbling with PFHG involves only symmetric-key operations, keeping per-table compute low relative to table size.

The on-chain footprint is independent of ℓ and $|T|$, following directly from the polynomial label correlation. The Witness transaction comprises around 100 bytes of non-witness data (version, input, output, etc) and approximately 9.0 kB of witness data (signatures, locking script, and control block). The total raw transaction size is approximately 9.1 kB; however, under SegWit's weight rules, witness bytes receive a 75% discount, yielding an effective size of approximately 2.4 kvB. Both figures are well within Bitcoin's standardness limits.

We provide an open-source production-grade Rust implementation of Mosaic [26]. Based on single-threaded benchmarks on Apple M1 Pro, Garbling Setup takes approximately 6.3 hours for G and 5.9 hours for E, Adaptor Setup takes approximately 2 seconds for G and 1 second for E, and Witness Resolution is negligible for G with E requiring between 3 and 21 minutes (6 minutes in expectation).

Artifact	Size
Communication	
CommitMsg	231 MB
ResponseMsg (shares)	223 MB
ResponseMsg (tables)	336 GB
AdaptorsMsg	2.1 MB
Total	336.4 GB
Storage	
Garbler	3.1 MB
Evaluator	336.2 GB
On-chain	
Locking script	427 B
Script inputs	8.6 kB

Table 3: Concrete costs with PFHG and current parameters.

9.4.3 Future Instantiation with Compact Tables

Promising future work [13, 14, 15] may reduce the garbled table size for the Groth16 verifier to as little as $|T| \approx 2$ MB, with individual garbling and evaluation each taking under 200 ms. In this regime, balanced parameters $N = 44$, $t = 22$, $\ell = 22$ prove more favorable (still preserving the soundness guarantee as $1/\binom{44}{22} < 2^{-40}$).

Artifact	Size
Communication	
CommitMsg	30.4 MB
ResponseMsg (shares)	28.2 MB
ResponseMsg (tables)	44 MB
AdaptorsMsg	2.1 MB
Total	104.7 MB
Storage	
Garbler	3.1 MB
Evaluator	73.5 MB
On-chain	
Locking script	427 B
Script inputs	8.6 kB

Table 4: Concrete costs with compact tables and balanced parameters.

Now, the VSS machinery (polynomial commitments and opened shares) becomes the dominant cost. Based on reported estimates, in this regime, Garbling Setup takes approximately 33 seconds for G and 12 seconds for E, Adaptor Setup approximately 1.6 seconds for G and 0.7 seconds for E, and Witness Resolution at most 4.4 seconds for E with 400 ms in expectation.

Acknowledgment

The authors thank Rajil Bajracharya for his help in simplification of the transaction graph.

References

[1] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*, pages 427–437. IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.4.

[2] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. Cryptology ePrint Archive, Paper 2014/756, 2014. `https://eprint.iacr.org/2014/756`.

[3] P. Gerhart, D. Schröder, P. Soni, and S. A. K. Thyagarajan. Foundations of adaptor signatures. Cryptology ePrint Archive, Paper 2024/1809, 2024. `https://eprint.iacr.org/2024/1809`.

[4] E. Lombrozo, J. Lau, and P. Wuille. BIP-141: Segregated Witness (Consensus Layer). Bitcoin Improvement Proposal, 2015. `https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki`.

[5] R. Linus. Optimizing On-Chain Costs for Publishing Proofs in BitVM-Style Bridges `https://gist.github.com/RobinLinus/0fc7405ad7485c35465efb7996a7b014`.

[6] L. Eagen. Glock: Garbled Locks for Bitcoin. Cryptology ePrint Archive, Paper 2025/1485, 2025. `https://eprint.iacr.org/2025/1485`.

[7] J. Rubin. Delbrag. 2025. `https://rubin.io/public/pdfs/delbrag.pdf`.

[8] R. Linus. BitVM: Compute anything on Bitcoin. 2023. `https://bitvm.org/bitvm.pdf`.

[9] R. Linus, L. Aumayr, Z. Avarikioti, M. Maffei, A. Pelosi, O. Thyfronitis Litos, C. Stefo, D. Tse, and A. Zamyatin. Bridging Bitcoin to second layers via BitVM2. Cryptology ePrint Archive, Paper 2025/1158, 2025. `https://eprint.iacr.org/2025/1158`.

[10] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. Cryptology ePrint Archive, Paper 2010/284, 2010. doi:10.1007/s00145-011-9107-0. `https://eprint.iacr.org/2010/284`.

[11] Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. Cryptology ePrint Archive, Paper 2013/081, 2013. `https://eprint.iacr.org/2013/081`.

[12] W. Chen. SoK: BitVM with succinct on-chain cost. Cryptology ePrint Archive, Paper 2025/1253, 2025. `https://eprint.iacr.org/2025/1253`.

[13] L. Eagen and Y. T. Lai. Argo MAC: Garbling with elliptic curve MACs. Cryptology ePrint Archive, Paper 2026/049, 2026. `https://eprint.iacr.org/2026/049`.

[14] S. Garg, D. Kolonelos, M. Sergeevitch, S. Sridhar, and D. Tse. BABE: Verifying proofs on Bitcoin made 1000x cheaper. Cryptology ePrint Archive, Paper 2026/065, 2026. `https://eprint.iacr.org/2026/065`.

[15] N. Khambhati, A. Bhattacharya, and D. Heath. Duty-Free Bits: Projectivizing garbling schemes. Cryptology ePrint Archive, Paper 2026/476, 2026. `https://eprint.iacr.org/2026/476`.

[16] T. K. Frederiksen, J. B. Nielsen, and C. Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT 2015*, pages 191–219, 2015. https://eprint.iacr.org/2014/598.

[17] M. Bellare, V. T. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM CCS 2012*, pages 784–796, 2012. https://eprint.iacr.org/2012/265.

[18] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.

[19] Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. https://eprint.iacr.org/2004/175

[20] X. Wang, S. Ranellucci, and J. Katz. Authenticated garbling and efficient maliciously secure two-party computation. Cryptology ePrint Archive, Paper 2017/030, 2017. https://eprint.iacr.org/2017/030.

[21] 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

[22] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology – EUROCRYPT 2015*, pages 220–246. Springer, 2015. https://eprint.iacr.org/2014/756

[23] BitVM Alliance. Garbled Snark Verifier. https://github.com/BitVM/garbled-snark-verifier.

[24] Alpen Labs. g16: Groth16 verifier Boolean circuit for SP1. https://github.com/alpenlabs/g16.

[25] Alpen Labs. ckt: A high-performance garbled circuit framework. https://github.com/alpenlabs/ckt.

[26] Alpen Labs. Mosaic: An end-to-end implementation of the Mosaic Garbler Service. https://github.com/alpenlabs/mosaic.

[27] Iddo Bentov and Ranjit Kumaresan. How to Use Bitcoin to Design Fair Protocols. Cryptology *ePrint* Archive, Paper 2014/129, 2014. https://eprint.iacr.org/2014/129.

A Proofs of Fair Exchange Protocol Properties

Proof of Completeness. Suppose both parties are honest and P holds a valid witness w with $R(x, w) = 1$. Two execution paths lead to a successful exchange:

Cooperative path.
During the $\Delta_{\mathsf{Witness}}$ window, P sends w to V off chain. Since V is honest and $R(x, w) = 1$, V accepts and co-signs the Cooperative Payout transaction. P broadcasts it; by liveness it confirms, and P receives $(s + p)$₿. V has learned w.

On-chain path.
If the cooperative path is not taken, P broadcasts the Witness transaction before $\tau_0 + \Delta_{\mathsf{Witness}}$, committing to w on chain. V extracts the garbled-circuit input labels from the on-chain signatures (step **W4.**), thereby learning the witness w encoded by P. Since $R(x, w) = 1$, the garbled-circuit evaluation does not yield s^*, so V cannot post Witness Fault. After Δ_{Payout} expires, P broadcasts the Payout transaction and receives $(s + p)$₿.

□

Proof of Timeliness. We show that each honest party can unilaterally reach a terminal state by round $\tau_0 + T$ where $T = \Delta_{\mathsf{Witness}} + \Delta_{\mathsf{Payout}} + \Delta_{\mathrm{conf}}$.

Honest P.
P broadcasts the Witness transaction before $\tau_0 + \Delta_{\mathsf{Witness}}$. By liveness it confirms within Δ_{conf} rounds; by persistence the confirmation is not reversed. If V does not post Witness Fault during the Δ_{Payout} window, P broadcasts the Payout transaction, which by liveness confirms by round $\tau_0 + \Delta_{\mathsf{Witness}} + \Delta_{\mathsf{Payout}} + \Delta_{\mathrm{conf}}$. If V does post Witness Fault, P's funds are resolved earlier. In either case, P's funds are unlocked by round $\tau_0 + T$ without requiring cooperation from V.

Honest V.
Two sub-cases arise. If P does not post the Witness transaction by the deadline, V broadcasts Witness Timeout after $\tau_0 + \Delta_{\mathsf{Witness}}$, which confirms within Δ_{conf} rounds by liveness. If P does post the Witness transaction, V evaluates the garbled circuit: if the evaluation yields s^*, V posts Witness Fault, which confirms within Δ_{conf} rounds; if it does not, the input was valid and P will eventually claim Payout, resolving all funds. In all cases, V's funds are resolved by round $\tau_0 + T$ without requiring cooperation from P.

□