WISCH: Efficient data signing via correlated signatures for Bitcoin

Ariel Futoransky[1][0009-0000-3780-190X], Fadi Barbàra[1,2][0000-0001-8021-5500], Ramses Fernandez[1][0000-0002-8959-636X], Emilio Garcia[1][0009-0003-3716-1756], Gabriel Larotonda[3,4][0000-0002-6600-8238], and Sergio Demian Lerner[1,5][0009-0003-6506-6254]

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

Abstract. The scalability and expressivity of Bitcoin smart contracts are limited by the network's conservative resource constraints and non-Turing complete scripting language. While new paradigms like BitVM enable complex off-chain computations verified via hash-based signatures (e.g., Winternitz One-Time Signatures), they rely on large data commitments that incur prohibitive on-chain costs. We introduce **WISCH** (Winternitz-Schnorr), a hybrid cryptographic protocol that binds the compact efficiency of Schnorr signatures to the functional expressivity of hash-based commitments. WISCH operates on an optimistic execution model: in the cooperative case, state transitions are authenticated by a single, standard 64-byte Schnorr signature. In the event of a dispute, the protocol leverages the binding between the Schnorr and Winternitz keys to force the on-chain revelation of specific logic gates, enabling granular fraud proofs. We analyze the security of WISCH under a rational adversary model, proving that it achieves economic soundness. Finally, we demonstrate the practical utility of WISCH in the construction of trust-minimized bridges. Our analysis shows that WISCH reduces the optimistic on-chain footprint to a single 64-byte signature, the minimum for any signature-authenticated spend, while maintaining the robust security guarantees of a 1-of-N trust assumption. Against a Winternitz commitment to a value of the same width this is a reduction of 3–5× in witness bytes, obtained at a setup cost exponential in that width; we make the trade-off precise and show how to reduce the resulting commitment tables by a further order of magnitude.

Keywords: Bitcoin · Schnorr Signatures · Winternitz Signatures · Smart Contracts · Layer-2 · BitVM

1 Introduction

Bitcoin has evolved significantly from its origins as a simple distributed ledger. Today, it serves as a robust platform for a variety of complex off-chain protocols, ranging from payment channels in the Lightning Network [20] to Discret Log Contracts (DLCs) [7] and more recently, verifiable computation frameworks like BitVM [14]. A common thread across these innovations is the desire to extend Bitcoin's "smart contract" capabilities without requiring consensus changes or soft forks. Developers seek to build expressive applications—such as covenants, complex state machines, and trust-minimized bridges—using only the existing script opcodes and cryptographic primitives available on the network today.

However, a specific bottleneck arises when these advanced constructions require properties that standard Elliptic Curve Digital Signature Algorithm (ECDSA) or Schnorr signatures [24,22] cannot easily provide. For instance, mechanisms involving bit-commitments, connectors in BitVM [14], or quantum-resistant fallback paths often necessitate the use of Hash-Based Signatures, such as Lamport [13] or Winternitz One-Time Signatures (WOTS) [17]. These schemes offer unique advantages, such as algebraic simplicity and the inability to equivocate without revealing a preimage. Yet, they come with a severe penalty: on-chain size. While a standard Schnorr signature occupies a mere 64 bytes, a single WOTS signature can consume several kilobytes of witness data [17]. In the constrained economic environment of the Bitcoin blockchain [18], relying on these heavy primitives for standard operations renders many potential applications economically unviable.

This paper is motivated by a fundamental observation regarding blockchain protocol design: the vast majority of executions are "optimistic"— or follow what we term the "Optimistic Path." In this scenario, participants act cooperatively to minimize costs, and no fraud occurs. The "Pessimistic Path" is the fallback mechanism invoked only during disputes. It is therefore inefficient to pay the high on-chain cost of "worst-case" security mechanisms for every "Optimistic Path" interaction. The challenge lies in retaining the capability of heavy primitives for the Pessimistic Path while enjoying the efficiency of standard signatures during the Optimistic Path.

To address this, we introduce **WISCH** (Winternitz-Schnorr), a novel protocol designed to bridge the gap between efficiency and expressivity of Bitcoin-based smart contracts. WISCH correlates a compact Schnorr signature with a heavier Winternitz signature, creating a binding mechanism that allows protocols to operate using cheap Schnorr signatures in the optimistic case. The burdensome Winternitz commitment is only revealed and enforced on-chain in the event of a dispute or challenge. This hybrid approach effectively enables protocols to utilize the functional properties of hash-based signatures without paying their cost in the common case.

1.1 Contributions

We present the WISCH protocol and analyze its impact on the Bitcoin development landscape. Our detailed contributions are as follows:

1. **The WISCH Protocol:** We define the formal construction of WISCH, describing how to securely bind a Schnorr public key to a Winternitz commitment such that the former acts as a valid proxy for the latter under normal conditions. Among other things, this enables smart contract like functionality for Bitcoin.
2. **Efficiency Analysis:** We provide a comparative analysis demonstrating that WISCH achieves an order-of-magnitude reduction in on-chain footprint for optimistic protocol executions compared to naive implementations of hash-based schemes.
3. **Economic Security:** We analyze the security of the scheme through a game-based and economic lens. We argue that under the rational adversary model—standard in blockchain cryptoeconomics [10,5]—the threat of the expensive fallback path is sufficient to secure the system, making honest behavior the dominant strategy.
4. **Application:** We demonstrate the utility of WISCH by showing how it can be used to build a Trust-Minimized Bridge compatible with the BitVM paradigm.

2 Related Work

Our work sits at the intersection of hash-based signatures, Bitcoin's native Schnorr signature integration, and the paradigm of optimistic protocol design. In this section, we review these foundational technologies and position WISCH within the broader landscape of Bitcoin layer-two protocols.

2.1 Hash-Based Signatures

Hash-based signatures, such as Lamport Signatures and Winternitz One-Time Signatures (WOTS) (see Section 3.3), rely solely on the security of cryptographic hash functions. Recently, they have been studied primarily for their post-quantum resistance [12,1]. However, within the Bitcoin ecosystem, they have found renewed relevance due to their algebraic simplicity, which allows for verification using basic opcodes like `OP_SHA256` and `OP_EQUAL`.

Recent constructions like BitVM utilize Lamport signatures to enable bit-commitments within Bitcoin scripts, allowing for the verification of complex computational statements on-chain. While powerful, these schemes suffer from significant data inefficiency. A standard WOTS signature can require revealing dozens of hash preimages, consuming kilobytes of block space per signature. This high cost restricts their usage to high-value or low-frequency settlement transactions, rendering them unsuitable for high-throughput Bitcoin-based applications.

2.2 Adaptor Signatures

The activation of the Taproot upgrade introduced native Schnorr signatures to Bitcoin, bringing linearity and key aggregation capabilities. These properties have enabled a wave of "Scriptless Script" protocols, where secure conditions are enforced off-chain through the exchange of valid signatures rather than explicit on-chain scripts, improving smart contract *de facto* expressibility while retaining its security.

Adaptor Signatures, for instance, allow unrelated parties to bind the revelation of a secret to the publication of a signature. While highly efficient, standard adaptor signatures typically reveal a single scalar value. They do not natively support the structured data exposure required for more complex "witnessing" applications where a third party must verify a specific message content against a public commitment without an oracle.

2.3 Optimistic Protocols in Bitcoin

The "optimistic" execution model is a cornerstone of modern Bitcoin scalability solutions. Protocols like the Lightning Network [20] and Discreet Log Contracts (DLCs) [7] operate on the premise that parties will usually cooperate. In these systems, complex execution logic is kept off-chain, and the blockchain is used only as a dispute resolution layer.

The "BitVM" paradigm [14,15,21] extends this optimistic model to general purpose computation. In the first iteration of BitVM, for example, complex computations are represented as logic circuits (NAND gates) committed on-chain using hash-based bit commitments (like Lamport or Winternitz signatures). The operator asserts a result, and if they lie, a verifier can challenge a specific step of the computation on-chain, proving the fraud. While powerful, a major bottleneck is the sheer volume of on-chain data required: verifying a circuit with millions of gates using standard WOTS signatures requires gigabytes of commitment data.

Table 1 summarizes the trade-offs between these schemes, highlighting how WISCH combines the efficiency of Schnorr signatures with the expressive power of hash-based commitments.

Table 1. Comparison of Cryptographic Schemes for Bitcoin Data Verification

Scheme	**Optimistic Size**	**Complex Data Binding**	**Bitcoin Compatible**
Lamport Signatures	High (∼KB)	Yes	Yes
Winternitz (WOTS)	High (∼KB)	Yes	Yes
Adaptor Signatures	Low (64 B)	No (Scalar only)	Yes
WISCH (Ours)	**Low (64 B)**	**Yes**	**Yes**

2.4 Garbled Locks

Garbled Locks (Glock) [8] also produces garbled-circuit labels on Bitcoin from signature material, and is the closest relative of the construction of Section 7. Glock observes that *projective* signature schemes — those whose signatures decompose into per-bit components — directly supply input labels for garbled circuit evaluation, so that a garbler authenticating an input w with $C(w) = 0$ lets an evaluator derive the output label $o^{(0)}$ as a fraud proof. The practical instantiation Glock25 realises projectivity for Bitcoin-native Schnorr through adaptor signatures combined with verifiable secret sharing, and reduces garbled-circuit complexity using a designated verifier SNARK, which makes it permissioned.

WISCH reaches the same interface by encapsulation rather than projection: the binding is external to the signature, carried by a table of ciphertexts keyed on pre-signed scalars. Three consequences separate the two. First, WISCH imposes no structure on the signature scheme — the optimistic spend is an unmodified BIP-340 key-path spend — whereas projectivity constrains the scheme and, in Glock25, requires an additional secret-sharing layer. Second, our composition theorem is generic over any garbling scheme exposing committed input labels, where Glock25 is tied to its designated-verifier instantiation. Third, and against WISCH, the setup cost is $\Theta(2^\ell)$ in the chunk width ℓ, against Glock's cost linear in circuit size; WISCH is therefore restricted to narrow chunks and buys its on-chain saving with an exponential setup, while Glock does not. The two occupy opposite corners of one trade-off: WISCH pays at setup for signature-scheme genericity and a 64-byte permissionless optimistic path, Glock pays in permissioning for a setup that scales with the circuit.

3 Preliminaries

We denote the security parameter by λ. We work in the context of the Bitcoin blockchain, which utilizes the `secp256k1` elliptic curve group we refer to as $\mathbb{G}$, of prime order p. Let G be the generator of $\mathbb{G}$. We assume a cryptographically secure hash function $H : \{0,1\}^* \to \mathbb{Z}_p$.

3.1 Schnorr Signatures

The Schnorr signature scheme is a digital signature scheme known for its simplicity and provable security [22,24]. We define the Schnorr signature scheme SCH as a tuple of polynomial-time algorithms ($\mathsf{SCH.KeyGen}, \mathsf{SCH.Sign}, \mathsf{SCH.Ver}$):

- $\mathsf{SCH.KeyGen}(1^\lambda) \to (x, X)$: Samples a private key $x \leftarrow \mathbb{Z}_p$ uniformly at random and computes the public key $X = x \cdot G$.
- $\mathsf{SCH.Sign}(x, m) \to \sigma$: To sign a message m, sample a random nonce $r \leftarrow \mathbb{Z}_p$, compute the commitment $R = r \cdot G$, the challenge $e = H(R \parallel X \parallel m)$, and the response $s = r + e \cdot x \pmod p$. Output $\sigma = (R, s)$.

– $\mathsf{SCH.Ver}(X, m, \sigma) \to \{0,1\}$: Given a public key X, message m, and signature $\sigma = (R, s)$, parse σ. Compute $e = H(R \parallel X \parallel m)$ and check if $s \cdot G = R + e \cdot X$. Output 1 if equality holds, else 0.

The scheme is correct because $s \cdot G = (r + e \cdot x) \cdot G = r \cdot G + e \cdot (x \cdot G) = R + e \cdot X$.

3.2 MuSig2 and Key Aggregation

MuSig2 [19] is a multi-signature scheme compatible with Schnorr signatures. It allows n parties with public keys $X_1, \ldots, X_n$ to aggregate their keys into a single key X_{Agg}. We describe the core algorithms:

– $\mathsf{KeyAgg}(X_1, \ldots, X_n) \to X_{\text{Agg}}$: let $L = H(X_1 \parallel \cdots \parallel X_n)$. For each $i \in [n]$, compute $a_i = H(L \parallel X_i)$. The aggregate key is $X_{\text{Agg}} = \sum_{i=1}^{n} a_i X_i$.
– $\mathsf{Sign}(x_1, \ldots, x_n, m)$: The parties interact in three rounds; the nonce exchange is preceded by a commitment round.
  1. *Nonce Commitment:* Each party i broadcasts $t_i = H_{\mathsf{com}}(R_i)$. When the 2^ℓ sessions of a WISCH setup are run together, each party broadcasts instead a single Merkle root over its 2^ℓ commitments.
  2. *Nonce Opening:* Each party broadcasts R_i; every party checks $t_i = H_{\mathsf{com}}(R_i)$ and aborts otherwise. The aggregate nonce is $R = \sum_{i=1}^{n} R_i$.
  3. *Signature Aggregation:* The challenge is $c = H(R \parallel X_{\text{Agg}} \parallel m)$. Each party computes partial signature $s_i = r_i + c \cdot a_i \cdot x_i \pmod p$. The final signature is $\sigma = (R, s)$ where $s = \sum_{i=1}^{n} s_i \pmod p$.

The commitment round costs one extra round and, batched over a WISCH setup, one 32-byte root per party. It buys three things. First, a proof in the random oracle model with one nonce per party, where the two-round scheme with two nonces requires the algebraic group model — a distinction that matters at $q = 2^\ell$ concurrent sessions on a single message under a single aggregate key. Second, it removes the adaptivity in nonce selection that the ROS attack exploits [6,4]; this is the three-round scheme of [16], whose proof under the discrete logarithm assumption does not rely on ROS. Third, it makes the aggregate nonces $\{R_m\}$ uniform and independent whatever a corrupt party does, so that a setup-time distinctness check is meaningful. The cost is departure from BIP-327 tooling; if BIP-327 compatibility is required instead, use MuSig2 with $\nu = 4$ nonces and $R = \sum_j b^{j-1} R_j$ for a random-oracle-only proof, and enforce nonce distinctness by an explicit check. Verification is identical to $\mathsf{SCH.Ver}(X_{\text{Agg}}, m, \sigma)$. This linearity is basic in WISCH, as it allows the *Optimistic Path* to appear as a standard single-key spend, indistinguishable in form and cost from any other single-signature payment.

3.3 Winternitz One-Time Signatures (WOTS)

Winternitz One-Time Signatures (WOTS) [17] rely on the security of a hash function $H_W : \{0,1\}^\lambda \to \{0,1\}^\lambda$. It uses a time-memory trade-off parameter

$w \in \mathbb{N}$ (often $w = 4$ or $w = 16$). Increasing w reduces the signature size by shortening the chain count ℓ, but significantly increases the computational cost of signing and verification (which scales with w). A message m of length λ bits is processed in base-w [6]. Let $\ell_1 = \lceil \lambda / \log_2 w \rceil$ be the number of digits for the message, and $\ell_2 = \lfloor (\log_2(\ell_1(w-1))) / \log_2 w \rfloor + 1$ be the digits for the checksum. The total chain count is $\ell = \ell_1 + \ell_2$.

We define the hash chain function $c^k(x)$ recursively: $c^0(x) = x$ and $c^k(x) = H_W(c^{k-1}(x))$ for $k > 0$. We define WOTS as a tuple

$$\mathsf{WOTS} = (\mathsf{WOTS.KeyGen}, \mathsf{WOTS.Sign}, \mathsf{WOTS.Ver}, \mathsf{WOTS.Csum})$$

- $\mathsf{WOTS.KeyGen}(1^\lambda) \to (sk, pk)$: Sample $sk = (sk_1, \ldots, sk_\ell) \leftarrow (\{0,1\}^\lambda)^\ell$. Compute $pk = (pk_1, \ldots, pk_\ell)$ where $pk_i = c^{w-1}(sk_i)$.
- $\mathsf{WOTS.Csum}(m) \to C$: Parse m as base-w integers $(m_1, \ldots, m_{\ell_1})$. Compute the checksum $C = \sum_{i=1}^{\ell_1}(w - 1 - m_i)$. Represent C as base-w integers $(C_1, \ldots, C_{\ell_2})$. Concatenate message and checksum digits to form $M = (M_1, \ldots, M_\ell) = (m_1, \ldots, m_{\ell_1}, C_1, \ldots, C_{\ell_2})$.
- $\mathsf{WOTS.Sign}(sk, m) \to \sigma$: Compute the extended message M using $\mathsf{WOTS.Csum}$. Output signature $\sigma = (\sigma_1, \ldots, \sigma_\ell)$ where $\sigma_i = c^{M_i}(sk_i)$.
- $\mathsf{WOTS.Ver}(pk, m, \sigma) \to \{0,1\}$: Compute M similarly. For each $i \in [\ell]$, compute candidate public key values $pk'_i = c^{w-1-M_i}(\sigma_i)$. Output 1 if $pk'_i = pk_i$ for all i, else 0.

Correctness: Verification succeeds because for a valid signature, $c^{w-1-M_i}(\sigma_i) = c^{w-1-M_i}(c^{M_i}(sk_i)) = c^{w-1}(sk_i) = pk_i$.

3.4 Bitcoin Script and Taproot

Bitcoin's execution environment is defined by a non-Turing complete, stack-based scripting language. Validation of a transaction input involves executing the locking script of the referenced UTXO (Unspent Transaction Output) alongside the unlocking script provided in the spending transaction.

Taproot Architecture (BIP 341). The Taproot upgrade [25] introduced a dual spending condition for outputs, formalized as a Merkle tree structure. A Taproot output key Q is defined as $Q = P + H(P \parallel m) \cdot G$, where P is an internal public key and m is the root of a Merkle tree (MAST) of value-bearing scripts. This structure enables two distinct verification paths:

1. **Key Path:** The output can be spent by a single Schnorr signature for the key Q. This path is highly efficient and private, distinguishing neither the number of signers nor the complexity of the underlying policy on-chain.
2. **Script Path:** If a collaborative signature cannot be produced, a spender may reveal a branch of the Merkle tree m and satisfy the specific conditions of a leaf script.

[6] Specifically, interpret the bitstring m as a sequence of $\log_2 w$-bit integers. For example if $w = 16$ and $m = 1011000111100100$, then m is processed in "blocks" of $\log_2 w = 4$ bits, as $(1011_2 = 11, 0001_2 = 1, 1110_2 = 14, 0100_2 = 4)$

In the context of WISCH, we explicitly map these architectural features to our protocol phases. The *Key Path* corresponds to the "Optimistic Path," where parties cooperate to sign with the aggregate key, incurring minimal cost and revealing no protocol semantics to an observer who does not hold the commitment table. The *Script Path* corresponds to the "Pessimistic Path," used only when cooperation fails, forcing the on-chain revelation of the expensive Winternitz commitments to resolve the dispute.

Schnorr Integration and Scriptless Scripts. The linearity of Schnorr signatures enables key aggregation (e.g., MuSig2, see Section 3.2), allowing complex multi-party policies to appear as a standard single-signature transaction on the Key Path. This property is the foundation of "Scriptless Scripts," which largely rely on *Adaptor Signatures* to enforce conditions off-chain. Adaptor Signatures allow parties to bind the validity of a signature to the revelation of a secret value. However, standard Adaptor Signatures are limited to binding a single *scalar* value (the discrete log of a point). They cannot natively support *complex data binding*, as seen in Section 3.3.

3.5 Symmetric Encryption and AEAD

We utilize an Authenticated Encryption with Associated Data (AEAD) scheme, defined as a tuple $(\mathsf{Enc}, \mathsf{Dec})$:

- $\mathsf{Enc}(\kappa, m) \to (c, \tau)$: Encrypts message m using key κ. Outputs ciphertext c and authentication tag τ.
- $\mathsf{Dec}(\kappa, c, \tau) \to m \cup \{\perp\}$: Decrypts c using κ, and τ. Outputs m if the tag τ is valid for (c), otherwise outputs $\perp$.

We require the property of *Integrity of Ciphertexts* (INT-CTXT) [2], meaning it is computationally infeasible to produce a valid tuple (c, τ) without knowledge of the key k. For concrete instantiation and implementation details of efficient committing AEAD schemes, we refer the reader to [9].

Remark (Signature Message vs. Committed Data). Throughout this paper, we distinguish between the signed message μ, which in Bitcoin is always the spending transaction's sighash as defined by BIP 341, and the committed data m that the Prover wishes to reveal. On-chain verification uses only standard `OP_CHECKSIG` against μ. The binding between a Schnorr signature and a specific data value m is achieved entirely through the off-chain encryption layer, not through the Schnorr message field. Concretely, for each possible data value m, the parties produce a signature on the same transaction μ using a distinct nonce R_m. The resulting scalar s_m then serves as the encryption key seed, binding m to the signature without requiring verification of arbitrary messages on-chain.

4 The WISCH Protocol

The WISCH protocol (Winternitz-Schnorr) allows a Prover P to commit to a large dataset and selectively reveal parts of it to a Verifier V efficiently[7]. It leverages an optimistic *happy path* where only compact Schnorr signatures are published, falling back to the more expensive Winternitz verification only in the event of a dispute.

We work in a *single-show, fixed-key* setting, in which for a given setup each index k corresponds to exactly one valid Schnorr signature and one valid WOTS commitment. The property is enforced by the ledger rather than by the signature scheme: the optimistic output is a single UTXO, so exactly one spending transaction, and hence exactly one scalar, ever becomes public. The Prover is bound to reveal at most one value per index; revealing two conflicting values constitutes a protocol violation (equivocation) which is cryptographically detectable.

4.1 Protocol Phases

We describe the abstract protocol phases here. A concrete end-to-end application (Trust-Minimized Bridge) utilizing this workflow is detailed in Section 5.

Phase 1: Setup (Offline/Pre-computation) In this phase, the Prover prepares the cryptographic material. The goal is to bind a set of Schnorr signatures (which will be used for efficient revealing) to a set of Winternitz signatures (which serve as the binding commitment). The binding is achieved by symmetrically encrypting the Winternitz signatures using keys derived from the Schnorr signatures via an AEAD scheme.

Since the Schnorr signature s is the only secret required to decrypt and reveal the WOTS signature, publishing s is cryptographically equivalent to revealing the WOTS signature data, provided the encryption key derivation is binding.

Let ℓ be the bit-length of the messages, as defined in Section 3.

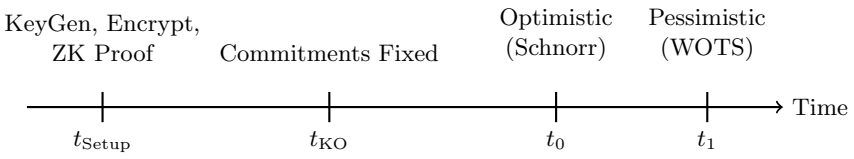

Fig. 1. Protocol Timeline showing the Optimistic and Pessimistic phases.

[7] The verifier can be a smart contract if it's done on-chain, or more generally, it can be a "watchtower" if done off-chain, similar as how it's done in the Lightning Network.

1. **Winternitz Key Generation:** The prover P generates a WOTS key pair $(sk^W, pk^W) \leftarrow \mathsf{WOTS.KeyGen}(1^\lambda)$, where superscript W denotes Winternitz, and publishes pk^W.
2. **Schnorr Key Aggregation:** The prover P and the verifier V both use $\mathsf{SCH.KeyGen}$ to generate (x_P, X_P) and (x_V, X_V) respectively. Then they perform $pk^{Sch} \leftarrow \mathsf{MuSig2.KeyAgg}(X_P, X_V)$
3. **Schnorr Pre-computation:** For every possible input m in the message space chunk $\mathcal{M} = \{0,1\}^\ell$ (where ℓ is small, e.g., $\ell \approx 16 - 24$ bits), let μ denote the sighash of the spending transaction for the first output. $\mathcal{P}$ and $\mathcal{V}$ each publish a Merkle root over their 2^ℓ nonce-share commitments and then open; $\mathcal{V}$ verifies that the aggregate nonces $\{R_m\}_m$ are pairwise distinct and aborts otherwise. For each $m \in \mathcal{M}$ the parties then execute $\mathsf{MuSig2.Sign}$ on the fixed message μ with nonce R_m. *$\mathcal{V}$ releases its partial signature $s_{\mathcal{V},m}$ and $\mathcal{P}$ releases none*, so that at the end of setup $\mathcal{P}$ holds all 2^ℓ complete signatures $\sigma_m^{Sch} = (R_m, s_m)$ under pk^{Sch} and $\mathcal{V}$ holds none. Each σ_m^{Sch} is a valid Schnorr signature on the same transaction; the distinct nonces ensure distinct scalars s_m, which serve as key material for binding m to the encryption layer. $\mathcal{V}$ verifies the setup proof *before* releasing any partial signature; since setup requires T-of-T participation, one honest watchtower doing so blocks a malformed setup, which is what preserves the 1-of-T security claim. For larger datasets, the data is split into multiple such chunks.
4. **Binding via Encryption:** This is a two-step process. For each $m \in \mathcal{M}$
   (a) P parses the Schnorr signature (R_m, s_m) and derives a symmetric key $K_{\text{Enc}} \leftarrow \mathsf{HKDF}(s_m \parallel m \parallel 1)$. Here, HKDF is a standard Key Derivation Function (e.g., HMAC-SHA256).
   (b) P computes and encrypts the WOTS signature using the derived key K_{Enc}: $\sigma_m^W \leftarrow \mathsf{WOTS.Sign}(sk^W, m)$; $C_m \leftarrow \mathsf{Enc}(K_{\text{Enc}}, \sigma_m^W)$, where Enc is the encryption function of the AEAD scheme defined in Section 3.
5. **Commitment:** P publishes the table of encrypted commitments $\{C_m\}$ and the public keys to the Verifier (or on a public bulletin board). P must prove that the encryption is correct (that the Schnorr signature decrypts a valid WOTS signature for m). This can be done via a Zero-Knowledge Proof (ZKP) for the relation: Write $\mathsf{x} = (\mathsf{ctx}, \mu, pk^{\mathsf{Sch}}, pk^W, \{R_m\}_m, \{C_m\}_m)$ for the statement and $\mathsf{w} = (\{s_m\}_m, \{K_{\mathsf{enc}}^{(m)}\}_m, sk^W)$ for the witness, and set $S_m := R_m + H(R_m \| pk^{\mathsf{Sch}} \| \mu) \cdot pk^{\mathsf{Sch}}$ and $M(m) := \mathsf{WOTS.Csum}(m)$, where $\ell = \ell_1 + \ell_2$ is the chain count. The clauses below are required for all $m \in \mathcal{M}$ and all $i \in [\ell]$.

$$\mathcal{R}_{\text{Enc}} = \left\{ (\mathsf{x}; \mathsf{w}) \;\middle|\; \begin{array}{l} s_m \cdot G = S_m \\ K_{\mathsf{enc}}^{(m)} = \mathsf{HKDF}(s_m \| m \| \mathsf{ctx}) \\ \sigma_m^W = \mathsf{WOTS.Sign}(sk^W, m) \\ C_m = \mathsf{Enc}(K_{\mathsf{enc}}^{(m)}, \sigma_m^W) \\ pk_i^W = H_W^{w-1-M_i(m)}(\sigma_{m,i}^W) \end{array} \right\} \tag{1}$$

Keeping pk^W in the statement is what forces a single Winternitz key across all m, which the fraud-proof script hardcodes; without it $\mathcal{P}$ could use a

different key per message and no on-chain script would verify. This proof prevents $\mathcal{P}$ from committing to "junk" data that cannot be opened.

6. **Collateral Lock:** P constructs, but does not yet broadcast, the transaction funding a Taproot address that commits to the script tree; its txid is therefore known, and the sighash μ of step 3 commits to it. The funding transaction is broadcast only once the commitment table and the setup proof have been verified, and its broadcast defines t_{KO}. This UTXO must contain sufficient collateral S to cover the penalty in case of fraud. The collateral is locked in a script separate from the WOTS verification script, specifically accessible via a "Slash Path" defined in Section 6.1. The transaction has two output. The first output has two spending paths:
   (a) the first allows the prover to claim the output by providing a valid Schnorr signature under the aggregated public key pk^{Sch}. This is the Taproot *key path*: no script is revealed, and the spend is a single `OP_CHECKSIG`-equivalent verification against pk^{Sch}, which is what the cost figures of Section 6.2 assume.
   (b) The second path allows the verifier to claim the output after timeout t_0 if the prover fails to reveal. The script checks that the transaction's locktime is at least t_0 using `OP_CHECKLOCKTIMEVERIFY`, then verifies a signature from the verifier.

   The second output holds the collateral and provides three spending paths.
   (a) The first fraud proof path verifies a WOTS signature supplied as witness. It cannot be used to assert that a commitment fails to open — a script that slashed on a failed verification could be satisfied by anyone submitting arbitrary data — so malformed commitments are excluded at setup by the correctness proof, and the residual case is handled by the challenge of Section 7. Since Bitcoin Script lacks loops, the hash iterations must be unrolled using conditional branches for each possible digit value. While verbose, this fits within Tapscript limits for reasonable Winternitz parameters[8].
   (b) The second fraud proof path handles predicate violations, where the committed messages satisfy WOTS verification but violate the application-specific validity condition [9]
   (c) The third path is the prover's timeout reclaim. If no valid fraud proof is submitted by block height t_1, the prover spends the output with a simple signature, recovering their collateral.

Phase 2: Optimistic Commitment After the setup is frozen (time t_{KO}), $\mathcal{P}$ reveals the chosen message m^*:

1. $\mathcal{P}$ publishes the Schnorr signature $\sigma^{\mathrm{Sch}}_{m^*} = (R_{m^*}, s_{m^*})$ to redeem the first output, first path.

[8] The interested reader can see the full script `anonymizedlink`

[9] For example, for a bridge using BitVM-ish computations, the predicate might encode circuit gate correctness, and the fraud proof would demonstrate a specific gate whose output is inconsistent with its inputs.

2. The Bitcoin network verifies $\mathsf{SCH.Ver}(pk^{Sch}, \mu, \sigma^{Sch}_{m^\star}) = 1$ with `OP_CHECKSIG`. V then observes the published scalar $s_{m^\star}$ and uses $\mathsf{HKDF}(s_{m^\star} \| m^\star \| 1)$ to derive the decryption key, thereby learning the committed data $m^\star$.

If valid, the data m^* is accepted. This costs only 64 bytes. If $\mathcal{P}$ fails to provide a valid signature by timeout t_0, $\mathcal{V}$ wins.

The reveal must be final before any part of the setup is reused. The single-show property rests on the ledger admitting one spend per output, so under a chain reorganisation two distinct scalars could become public, yielding two WOTS signatures under one pk^W, which plain WOTS does not survive. Implementations should require a confirmation depth accordingly.

Phase 3: Pessimistic Challenge (Optional) If $\mathcal{V}$ suspects that $\mathcal{P}$ has equivocated or if the Schnorr signature is invalid, $\mathcal{V}$ can challenge $\mathcal{P}$ to decrypt the underlying WOTS signature:

1. $\mathcal{V}$ uses the revealed scalar s_{m^*} to derive $K^{(m^*)}_{\mathsf{enc}}$.
2. $\mathcal{V}$ decrypts C_{m^*} to obtain $\sigma^W_{m^*}$.
3. $\mathcal{V}$ checks $\mathsf{WOTS.Ver}(pk^W, m^*, \sigma^W_{m^*}) = 1$.

If the WOTS signature is valid and matches m^*, $\mathcal{P}$ is honest. If $\mathcal{P}$ has not produced a valid Schnorr signature that decrypts to a valid WOTS signature for the committed data, $\mathcal{P}$ is slashed. $\mathcal{V}$ enforces this by constructing a Fraud Proof Transaction that spends the collateral UTXO via the script path, using the decrypted WOTS signature as witness data.

5 Trust-Minimized Bridge

We now generalize WISCH to handle multi-step executions, demonstrating its application in a Trust-Minimized (Cross-Chain) Bridge. This construction is critical for scaling Bitcoin, allowing assets to be moved to a Layer-2 (L2) sidechain while inheriting Bitcoin's security for dispute resolution.

5.1 Architecture Overview

The bridge operates on an *Optimistic* model. For simplicity, a centralized Operator is responsible for advancing the state of the L2 according to a deterministic transition function. A decentralized network of Watchtowers monitors these updates. The system is comprised of three core components:

1. **The Operator (Prover):** The entity tasked with executing state transitions.
2. **The Watchtowers (Verifiers):** Independent actors who verify the Operator's assertions off-chain. Their security model is 1-of-T: as long as one honest Watchtower is online, the bridge is secure.

3. **The L1 Contract:** A Bitcoin Taproot tree encoding the arbitration logic, with an *Optimistic Key Path* for efficient updates and a *Pessimistic Script Path* for fraud proofs.

We model the L2 execution as a deterministic state transition function T. Given a current state S_N and a vector of operations $\boldsymbol{op} = (op_1, op_2, \ldots)$, the function computes the next state: $S_{N+1} = T(S_N, \boldsymbol{op})$. The function T is represented as a boolean circuit of K computational steps, whose execution trace $\{m(0), \ldots, m(K-1)\}$ the Operator commits to off-chain by a Merkle root, exactly as in BitVM2 [15].

WISCH is applied to the *assert*, not to the trace. The reason is structural. Because every pre-signature of Section 4 signs the same sighash μ, and μ commits to the spending transaction's outputs under BIP-341 [25], the update transaction is fixed at setup and cannot carry a value that is unknown at that time. What is not fixed is *which* of the 2^ℓ pre-signed alternatives is used to spend a given input. WISCH is therefore best understood as a channel: one input conveys ℓ bits at the cost of a single 64-byte signature, decodable by any party holding the commitment table. A claim of L bits is conveyed by $c = \lceil L/\ell \rceil$ independent instances spent as c inputs of one transaction, and no aggregation of signatures is required or possible.

5.2 Bridge Lifecycle

The lifecycle of a state update seamlessly integrates the WISCH protocol phases defined in Section 4 with the bridge operations.

1. Setup and Circuit Commitment Before the bridge launches (or during an epoch setup), the Operator prepares the commitments for one assert, that is for a claim of L bits split into $c = \lceil L/\ell \rceil$ chunks.

1. **Winternitz Keys:** For each chunk $j \in \{1, \ldots, c\}$, the Operator generates a WOTS key pair (sk_j^W, pk_j^W).
2. **Schnorr Pre-computation:** For every chunk j and every possible value $m \in \{0,1\}^\ell$ of that chunk, the Operator and Watchtowers compute joint Schnorr signatures $\sigma_{m,j}^{\mathrm{Sch}} = (R_{m,j}, s_{m,j})$ under an aggregate key pk_j^{Sch} specific to chunk j, following the setup of Section 4. Each chunk has its own output, so the c instances are independent.
3. **Binding via Encryption:** The Operator encrypts the WOTS signatures using keys derived from the corresponding Schnorr scalars:

$$K_{\mathsf{enc}}^{(m,k)} \leftarrow \mathsf{HKDF}(s_{m,k} \| m \| k), \quad C_{m,k} \leftarrow \mathsf{Enc}(K_{\mathsf{enc}}^{(m,k)}, \sigma_{m,k}^W)$$

4. **Commitment:** The Operator publishes the encrypted table $\{C_{m,k}\}$ and proves via Zero-Knowledge that the encryption is correct. This binds the *cheap* Schnorr signatures to the *expensive* execution trace.

2. Optimistic Operation When the Operator computes $S_{N+1} = T(S_N, \boldsymbol{op})$, it obtains the claim to be asserted and parses it as c chunks $m^{(1)}, \ldots, m^{(c)}$. To finalise this state on Bitcoin:

1. **Selection:** For each chunk j the Operator selects the pre-signature $\sigma^{\mathrm{Sch}}_{m^{(j)},j}$ matching the value it is asserting. No aggregation is performed: the scalars $s_{m^{(j)},j}$ must remain individually visible, since each is the key material for its own chunk.
2. **Submission:** The Operator publishes one transaction spending the c outputs, each on its *Key Path* with its own 64-byte signature.
3. **Implicit Reveal:** Each published scalar $s_{m^{(j)},j}$ releases the decryption key for chunk j, so the c inputs jointly convey the L-bit claim.

Cost: ≈ 64 bytes. To the network it looks like a standard payment: an observer learns neither the number of signers, nor the chunk width ℓ, nor that a WISCH instance exists. It is not, however, semantically hidden from a party holding the commitment table, which recovers the revealed index in one lookup from the published nonce. Since the tables are distributed to the watchtower set, unlinkability holds only against observers outside the protocol.

3. Dispute Resolution If a Watchtower detects S_{N+1} is invalid:

1. **Decryption:** The Watchtower uses the c published Schnorr scalars to decrypt the WOTS signatures for the asserted claim, obtaining a commitment the Operator cannot repudiate.
2. **Fault Isolation:** They re-execute T locally and enter the bisection protocol against the Operator's Merkle-committed trace, narrowing the dispute to a single step k^* over $\lceil \log_2 K \rceil$ rounds. Only this step's values are ever opened on-chain, which is why the trace itself need not be committed by WISCH.
3. **Challenge:** The Watchtower initiates an on-chain Challenge via the *Script Path*, revealing the decrypted WOTS signatures $\sigma^{W}_{m(k^*),k^*}$ for step k^*.
4. **Adjudication:** The Bitcoin script verifies the WOTS signatures (proving the Operator committed to these values) and the logic of gate k^* (proving the values are wrong).
5. **Slashing:** The Operator's bond is slashed.

6 Analysis

In this section, we analyze the WISCH protocol from both security and efficiency perspectives. We employ a hybrid security model that combines traditional cryptographic game-based definitions with economic rationality, followed by a detailed efficiency comparison against standard approaches.

6.1 Security Analysis

Adversarial Model We consider a system with a Prover P and a Verifier V.

1. **Rationality Assumption:** All participants are rational, profit-maximizing agents. An adversary will only deviate from the honest protocol if the expected utility of cheating is strictly greater than the expected utility of honest behavior.
2. **Capabilities:** The adversary $\mathcal{A}$ has full control over the network but is computationally bounded (PPT). $\mathcal{A}$ cannot break the underlying cryptographic primitives, such as: the discrete log problem (Schnorr), one-wayness of the SHA256 hash chain together with collision resistance for the checksum encoding (WOTS; see [11] for why second-preimage resistance alone is insufficient), or the AEAD encryption scheme.

Remark (Watchtower Network). In practice, the Verifier V is instantiated as a decentralized network of T watchtowers. The security assumption is 1-of-T: as long as at least one honest watchtower is online to challenge an invalid commitment, the protocol is secure. Use of MuSig2 allows these watchtowers to coordinate, but strictly speaking, any single honest party can enforce the protocol. Since there are T watchtowers, it is a reasonable expectation that at least one of them will be online. For this reason, we are talking about one verifier that is online.

Game-Based Security We model the security of the optimistic commitment as a game $\mathsf{Game}_{\mathrm{WISCH}}(\mathcal{A}, \lambda)$ between a Prover adversary $\mathcal{A}$ and a Challenger $\mathcal{C}$ (acting as the honest Verifier/Contract). The challenge game is given by:

1. **Setup:** $\mathcal{A}$ outputs a set of commitments $\{C_m\}$ and public keys.
2. **Commit:** $\mathcal{A}$ tries to open a value m by producing a Schnorr signature σ^S.
3. **Challenge:** $\mathcal{C}$ checks $\mathsf{SCH.Ver}$. If valid, $\mathcal{C}$ may choose to challenge $\mathcal{A}$ to decrypt the underlying WOTS signature.
4. **Response:** If challenged, $\mathcal{A}$ must provide the key K_{Enc} (derived from σ^S) such that the decrypted σ^W satisfies $\mathsf{WOTS.Ver}$.

Winning Condition. The adversary $\mathcal{A}$ wins the game if $\mathcal{A}$ outputs a valid Schnorr signature σ^S (i.e., $\mathsf{SCH.Ver}(pk^S, m, \sigma^S) = 1$) such that:

$$\Pr[(\mathsf{Dec}_{K_{\mathrm{Enc}}}(C) = \bot) \vee (\mathsf{WOTS.Ver}(pk^W, m, \mathsf{Dec}_{K_{\mathrm{Enc}}}(C)) = 0)] \geq \epsilon \quad (2)$$

for non-negligible ϵ. Here, the first event represents a decryption failure (integrity violation), and the second represents a binding violation (equivocation to a WOTS signature for some $m' \neq m$ or an invalid signature).

Soundness (Proof Sketch). Soundness rests on the setup proof, and not on the AEAD. The distinction matters because the adversary in this game is the Prover: it constructed the table and knows every encryption key, so no property quantified over adversaries *lacking* that key, INT-CTXT among them, constrains it.

Suppose $\mathcal{A}$ wins, producing a valid σ^S for which C either fails to decrypt or decrypts to a value that $\mathsf{WOTS.Ver}$ rejects.

– *Case 1 (Decryption failure).* The relation $\mathcal{R}_{\text{Enc}}$ constrains

$$C_m = \mathsf{Enc}(K_{\mathsf{enc}}^{(m)}, \sigma_m^W), \qquad K_{\mathsf{enc}}^{(m)} = \mathsf{HKDF}(s_m \| m \| \mathsf{ctx}),$$

where s_m is the scalar whose public point S_m is fixed by R_m, pk^{Sch} and μ. An accepted setup proof therefore certifies that C_m decrypts under exactly the key a verifier derives at reveal time, so producing a ciphertext that does not violates the knowledge soundness of that proof, and nothing else.

– *Case 2 (Invalid opening).* Likewise the clause $pk_i^W = H_W^{w-1-M_i(m)}(\sigma_{m,i}^W)$ certifies that the plaintext is a Winternitz signature valid under the published pk^W for the checksum-extended digits of m. Opening to any other value would require a second witness for the same statement, again contradicting knowledge soundness.

Soundness therefore reduces to the setup proof alone. Section 7 states this as Theorem 18 and proves it, with the knowledge error of the proof system and the collision resistance of the commitment structure as the only assumptions.

Confidentiality of the unopened branches. What the AEAD does secure is orthogonal to the above, and is worth stating separately because it is the more surprising claim. During setup $\mathcal{P}$ produces 2^ℓ Winternitz signatures under a single key pair pk^W, which in plain WOTS would be catastrophic. It is not, for two independent reasons. First, the ledger admits one spend per output, so exactly one scalar ever becomes public; this is a property of Bitcoin rather than of the encryption, and it is what bounds the number of openings to one, as Proposition 12 makes precise. It also requires the reveal to be final, since a reorganisation making two scalars public would open two branches. Second, the branches that are never opened stay hidden because their keys derive from scalars that remain computationally unpredictable, which needs both the semantic security of the AEAD and a hardness property of the signature layer; Lemma 11 isolates the latter and Theorem 17 gives the resulting bound.

Economic Security and Optimistic Assumptions The security of WISCH relies on the "Optimistic" paradigm common to Layer-2 scaling solutions. In these systems, security is not enforced by verifying every transition on-chain, but by the credible threat of a fraud proof.

Let C_{Happy} be the cost of the optimistic execution (Schnorr signature) and C_{Sad} be the cost of the challenge response (Winternitz verification). Let S be the slashed collateral penalty for losing a challenge. A rational prover will cheat only if $E[\text{Gain}] > P(\text{Challenge}) \cdot S$. In WISCH, since the Verifier (or a Watchtower) can deterministically check the validity of the commitment off-chain, $P(\text{Challenge}) \approx 1$ for any invalid commitment. As long as $S > C_{\text{Sad}}$ (plus opportunity costs), the adversary has a negative expected utility for cheating. Thus, the system is economically secure, preserving the integrity of the data while enjoying the efficiency of C_{Happy} in the overwhelming majority of cases.

Remark (Parameter Selection). The specific value of the collateral S is an implementation parameter. Crucially, this collateral is locked on-chain during

the Setup phase in the funding UTXO of the contract. It remains locked until the protocol concludes. It must be sufficient to cover the transaction fees of the challenge (which may be high during congestion) and the verifier's operational costs. We leave the precise game-theoretic parameterization of S under different network conditions (e.g., MEV, fee spikes) to future applied research.

6.2 Efficiency Analysis

We provide a complexity analysis and concrete cost comparison, demonstrating the significant efficiency gains of WISCH in the optimistic case.

Complexity Parameters Let λ be the security parameter (e.g., 256 bits). Let L be the total length of the committed data (execution trace) in bits. We split L into K chunks of size ℓ bits, such that $L = K \cdot \ell$. Let w be the Winternitz parameter (typically $w = 16$). Let S_W be the size of a single WOTS signature for an ℓ-bit message.

Computational and Storage Complexity Setup Phase (Off-Chain). The setup requires generating pre-signed Schnorr signatures for every possible value in the chunks.

- **Prover Complexity:** $O(K \cdot 2^\ell)$. For $\ell = 16$, this is $65,536$ operations per step. While exponential in ℓ, for small ℓ this is feasible and parallelizable.
- **Verifier Storage:** $O(K)$ (only commitments) or $O(K \cdot 2^\ell)$ if storing full encrypted tables (though typically hosted on public bulletin boards).

Optimistic Execution (On-Chain).

- **Communication:** $O(1)$. Identifying the valid state requires publishing a single aggregate Schnorr signature.
- **Verification Cost:** 1 `OP_CHECKSIG` (or `OP_CHECKSIGADD` for multisig).

Pessimistic Execution (On-Chain).

- **Communication:** $O(S_W)$. Revealing the WOTS signature requires publishing the hash pre-images.
- **Verification Cost:** $O(\ell / \log w)$ hash operations.

Concrete Cost Comparison Table 2 compares the on-chain costs of committing to a chunk of $\ell = 16$ bits using standard WOTS against WISCH. We assume $w = 16$, so the message occupies $\ell_1 = 4$ digits and the checksum $\ell_2 = 2$, giving 6 digests in total. This is the largest width for which the 2^ℓ pre-signatures of the setup remain comfortably practical; wider chunks improve the ratio but not the order, as discussed below.

For $w = 16$ and 32-byte digests the saving in witness bytes is $(\ell / \log_2 w + \ell_2) \cdot 32/64 = \ell/8 + 1$, so a wider commitment saves proportionally more — $4\times$

Metric	Standard WOTS / BitVM	WISCH (Optimistic)
Witness size	192 bytes (6×32)	**64 bytes**
Hash operations in script	90 (unrolled)	**0**
Marginal input weight	$\approx$ 1,120 WU	**230 WU**
Reduction, witness bytes	–	**3.0**$\times$
Reduction, input weight	–	$\approx$ **4.9**$\times$

Table 2. Concrete on-chain comparison at equal commitment width, $\ell = 16$ bits, $w = 16$. Weights are marginal per input; the Winternitz script size is an estimate and depends on the encoding.

at $\ell = 24$ — but the setup cost is 2^ℓ pre-signatures, so the saving grows only logarithmically in the work required to obtain it. The reduction applies to every step of the execution trace, and for a bridge verifying a large circuit the aggregate saving is the difference between a commitment layer that fits in a block and one that does not.

7 Recursive Translation Tables

The flat commitment table of Section 4 stores one ciphertext per value of m, each carrying the full payload. For garbled-circuit input labels this is 2^ℓ entries of $\ell\lambda$ bits, some 33 MB per chunk at $\ell = 16$ and 13 GB at $\ell = 24$. The table, and with it the witness of the setup proof, is the practical bottleneck.

We reorganise the payload by divide and conquer. A *glue* table retains one entry per value of m but carries only two sub-keys; two *sub-tables*, of $2^{\ell/2}$ entries each, map half-values to labels. The signature layer is unchanged throughout: 2^ℓ MuSig2 pre-signatures on a fixed sighash and one 64-byte optimistic reveal.

This requires the payload to decompose as a product across disjoint bit-blocks of m, which garbled labels do and WOTS signatures do not, their checksum chains depending on all message digits. Hoisting the checksum into the glue entry recovers a valid recursion for WOTS, but yields 1.5–2$\times$ against 7.9–12$\times$ for labels, the saving being proportional to the payload elements per value: ℓ for labels against $\ell/\log_2 w + \ell_2$ for WOTS. We state everything for labels.

7.1 Construction

Labels and keys are λ bits. Set $\ell_L = \lceil \ell/2 \rceil$, $\ell_R = \lfloor \ell/2 \rfloor$, and parse $m = m_L \| m_R$ accordingly; m_i is the i-th bit of m. We use a pseudorandom function PRF, a key-derivation function H_{kdf} modelled as a random oracle, a Merkle compression function H_{M}, and the commitment hash H. We fix $H =$ SHA256, so that an on-chain opening is a 32-byte witness against `OP_SHA256 <com> OP_EQUAL`; `OP_RIPEMD160` would save 12 bytes per commitment but leave 80-bit collision security, on which Theorems 18 and 19 depend.

The WISCH layer supplies keys $K_m = H_{\mathsf{kdf}}(s_m \| m \| \mathsf{ctx})$, where we have that $\sigma_m^{\mathrm{Sch}} = (R_m, s_m)$ is the m-th pre-signature on the sighash μ under the aggregate key $X := pk^{\mathsf{Sch}}$. Writing $c_m = H(R_m \| X \| \mu)$ and $S_m := R_m + c_m X$, the signature is valid iff $s_m G = S_m$; note S_m is computable from public data. The context string is $\mathsf{ctx} = \mathsf{sid} \| k \| c \| X \| \mathsf{rt}_{\mathsf{nonce}}$, with k the execution step, c the chunk index and $\mathsf{rt}_{\mathsf{nonce}}$ the nonce-commitment root of Lemma 10. Including k and c is necessary: without them the same sub-key value could be addressed from two chunks' glue layers, breaking Theorem 17.

Definition 1 (Translation table). $\mathsf{RTT.Gen}(1^\lambda, \ell, \{L_i^b\}, \{K_m\}, \mathsf{ctx})$ *samples* $K_L(u), K_R(v) \xleftarrow{\$} \{0,1\}^\lambda$ *for each* $u \in \{0,1\}^{\ell_L}$, $v \in \{0,1\}^{\ell_R}$ *and outputs*

$$
\begin{aligned}
C_m^{\mathsf{g}} &= \big(K_L(m_L) \,\|\, K_R(m_R)\big) \oplus \mathsf{PRF}\big(K_m, \mathsf{ctx} \| \mathsf{g} \| m\big), \\
C_u^{L} &= \big(L_1^{u_1} \| \cdots \| L_{\ell_L}^{u_{\ell_L}}\big) \oplus \mathsf{PRF}\big(K_L(u), \mathsf{ctx} \| \mathsf{L} \| u\big), \\
C_v^{R} &= \big(L_{\ell_L+1}^{v_1} \| \cdots \| L_{\ell}^{v_{\ell_R}}\big) \oplus \mathsf{PRF}\big(K_R(v), \mathsf{ctx} \| \mathsf{R} \| v\big),
\end{aligned}
$$

together with $\mathsf{rt} = \mathsf{MerkleRoot}(\{C_m^{\mathsf{g}}\}, \{C_u^L\}, \{C_v^R\})$ *over the lexicographically ordered leaves.* $\mathsf{RTT.Open}(\mathsf{Tab}, m, K_m)$ *recovers* $(K_L(m_L), K_R(m_R))$ *from* C_m^{g}*, then the two label blocks, and concatenates them.*

Remark 2 (Proof of correctness). Fix m and parse $m = m_L \| m_R$. $\mathsf{RTT.Open}$ recomputes $\mathsf{PRF}(K_m, \mathsf{ctx} \| \mathsf{g} \| m)$ from its own inputs, and this is exactly the $K_L(m_L) \| K_R(m_R)$. With $K_L(m_L)$ in hand it recomputes the pad of $C_{m_L}^L$ and recovers

$$
L_1^{m_{L,1}} \| \cdots \| L_{\ell_L}^{m_{L,\ell_L}} \;=\; L_1^{m_1} \| \cdots \| L_{\ell_L}^{m_{\ell_L}},
$$

since $m_{L,i} = m_i$ for $i \le \ell_L$; symmetrically the right branch returns $L_{\ell_L+i}^{m_{R,i}} = L_{\ell_L+i}^{m_{\ell_L+i}}$ for $i \le \ell_R$. Concatenating the two blocks gives $(L_1^{m_1}, \ldots, L_\ell^{m_\ell})$. Every step is deterministic, so this holds with probability 1.

PRF is used at two output lengths, and we take it in counter mode, the pad being the concatenation of $\mathsf{PRF}(K, \mathsf{lbl} \| 0)$, then $\mathsf{PRF}(K, \mathsf{lbl} \| 1)$, and so on, so that the hybrid of Theorem 17 runs over blocks and no variable-output-length assumption is needed.

Per-entry authentication is replaced by three mechanisms. Ciphertext integrity comes from rt, committed as an unspendable leaf of the collateral output's Taproot tree, which fixes it by consensus at zero marginal cost and leaves a bulletin-board copy checkable against the tapleaf path. Wrong-key detection comes from the public $\{R_m\}$, which make a published (R, s) self-identifying by $O(1)$ lookup rather than trial decryption. Garbage detection comes from the commitments $\mathsf{com}_i^b = H(L_i^b)$. Semantic security rests on PRF alone; no AEAD assumption survives.

7.2 Size

Writing $T(\ell)$ for the total size in bits,

$$T(\ell) = \begin{cases} 2^\ell \ell\lambda & \ell \le 3, \\ 2^\ell \cdot 2\lambda + T(\lceil \ell/2 \rceil) + T(\lfloor \ell/2 \rfloor) & \ell > 3, \end{cases} \tag{3}$$

the floor at 3 being the least width at which recursion still pays: a payload entry costs $\ell\lambda$ against 2λ for a glue entry, but the saving $2^\ell(\ell-2)\lambda$ must also cover the two sub-tables, and at $\ell = 3$ it does not, recursing gives $2^3 \cdot 2\lambda + T(2) + T(1) = 26\lambda$ against $2^3 \cdot 3\lambda = 24\lambda$ flat. The rounding is needed: the case $\ell = 24$ descends $24 \to 12 \to 6 \to 3$, and a bare $2T(\ell/2)$ is undefined at 3.

Proposition 3 (Closed form). *Let $\mathcal{T}(\ell)$ denote the multiset of node widths of the recursion tree of* (3)*: the root has width ℓ; a node of width $w > \ell_0$ has children of widths $\lceil w/2 \rceil$ and $\lfloor w/2 \rfloor$; $n_1(\ell) = \#\{w \in \mathcal{T}(\ell) : w = 1\}$. Then*

$$T(\ell) \;=\; 2\lambda\Big(\sum_{w \in \mathcal{T}(\ell)} 2^{\,w} \;-\; n_1(\ell) \Big). \tag{4}$$

If moreover $\ell = 2^k$ with $k \ge 1$, then every node at depth j has width $\ell/2^j$ and there are 2^j of them, every leaf has width 2 and $n_1(\ell) = 0$, so (4) *collapses to*

$$T(\ell) \;=\; 2\lambda \sum_{j=0}^{k-1} 2^j \, 2^{\,\ell/2^j}. \tag{5}$$

Proof. We proceed by structural induction over the recursion tree.

Leaves. For a leaf of width 2, (3) gives $T(2) = 2^2 \cdot 2 \cdot \lambda = 8\lambda$, and the right-hand side of (4) is $2\lambda(2^2 - 0) = 8\lambda$. For a leaf of width 1, (3) gives $T(1) = 2^1 \cdot 1 \cdot \lambda = 2\lambda$, while $2\lambda \cdot 2^1 = 4\lambda$; the term n_1 is exactly this discrepancy. A width-2 leaf costs precisely what a glue entry of that width would cost, which is why the leaf layer obeys the glue formula, whereas a width-1 leaf costs half as much and must be corrected for.

Internal nodes. Let $w > \ell_0$ and write $\mathcal{T}_L, \mathcal{T}_R$ for the subtrees rooted at its two children, so that $\mathcal{T}(w) = \{w\} \uplus \mathcal{T}_L \uplus \mathcal{T}_R$ and $n_1(w) = n_1(\mathcal{T}_L) + n_1(\mathcal{T}_R)$, the root contributing nothing to n_1 because $w > \ell_0 \ge 1$. By (3) and the induction hypothesis applied to each child,

$$\begin{aligned} T(w) &= 2\lambda 2^w + 2\lambda\Big(\sum_{w' \in \mathcal{T}_L} 2^{w'} - n_1(\mathcal{T}_L) \Big) + 2\lambda\Big(\sum_{w' \in \mathcal{T}_R} 2^{w'} - n_1(\mathcal{T}_R) \Big) \\ &= 2\lambda\Big(\sum_{w' \in \mathcal{T}(w)} 2^{w'} - n_1(w) \Big). \end{aligned}$$

Power-of-two case. If $\ell = 2^k$ then $\lceil w/2 \rceil = \lfloor w/2 \rfloor = w/2$ at every node, so the tree is complete: the widths at depth j are all equal to $\ell/2^j$ and there are 2^j of them, the leaves sit at depth $k-1$ with width 2, and $n_1(\ell) = 0$. Substituting into (4) gives (5). □

Corollary 4 (Sharp ratio). *For $\ell = 2^k$ with $\ell \geq 8$, writing $\mathsf{Flat}(\ell) = 2^\ell \ell\lambda$,*

$$\frac{T(\ell)}{2^{\ell+1}\lambda} = 1 + 2^{\,1-\ell/2} + \theta,\ 0 \leq \theta \leq 2^{\,3-3\ell/4},\ \frac{\mathsf{Flat}(\ell)}{T(\ell)} = \frac{\ell}{2}\Big(1 - 2^{\,1-\ell/2} + O(2^{\,3-3\ell/4})\Big).$$

Proof. Dividing (4) by $2^{\ell+1}\lambda$ turns the j-th summand into $2^j 2^{\ell/2^j - \ell}$, which is 1 at $j = 0$ and $2^{1-\ell/2}$ at $j = 1$. For $\ell \geq 8$ the remaining terms decrease by a factor at least 2, so their sum is at most $2 \cdot 2^2 2^{\ell/4-\ell} = 2^{3-3\ell/4}$. The second identity follows from $(1+x)^{-1} = 1 - x + O(x^2)$. □

Lemma 5. *For every $\ell \geq 1$,*

$$2^{\ell+1}\lambda \;\leq\; T(\ell) \;\leq\; 2^{\ell+1}\lambda\big(1 + 2^{\,5/2-\ell/2}\big), \tag{6}$$

so $T(\ell) = 2^{\ell+1}\lambda\big(1 + O(2^{-\ell/2})\big)$ and $\mathsf{Flat}(\ell)/T(\ell) \;\geq\; \frac{\ell}{2}\big(1 - 2^{\,5/2-\ell/2}\big)$.

Proof. The lower bound is the glue term of (3).

For the upper bound we first show $T(\ell) \leq 2^{\ell+2}\lambda$ by strong induction. The cases $\ell \leq 4$ are direct from (3): $T(1) = 2\lambda$, $T(2) = 8\lambda$, $T(3) = 24\lambda$ and $T(4) = 2^5\lambda + 2T(2) = 48\lambda$, against the bounds 8λ, 16λ, 32λ and 64λ respectively. For $\ell \geq 5$, using $\ell_R \leq \ell_L$ and the hypothesis at $\ell_L, \ell_R < \ell$,

$$T(\ell) = 2^{\ell+1}\lambda + T(\ell_L) + T(\ell_R) \;\leq\; 2^{\ell+1}\lambda + 2^{\ell_L+2}\lambda + 2^{\ell_R+2}\lambda \;\leq\; 2^{\ell+1}\lambda + 2^{\ell_L+3}\lambda,$$

and $2^{\ell_L+3} \leq 2^{\ell+2} - 2^{\ell+1} = 2^{\ell+1}$ holds iff $\ell_L \leq \ell - 2$, which is true because $\ell_L = \lceil \ell/2 \rceil \leq (\ell+1)/2 \leq \ell - 2$ for $\ell \geq 5$.

Now set $E(\ell) := T(\ell) - 2^{\ell+1}\lambda = T(\ell_L) + T(\ell_R)$. By the bound just established and $\ell_R \leq \ell_L \leq (\ell+1)/2$,

$$E(\ell) \;\leq\; 2^{\ell_L+2}\lambda + 2^{\ell_R+2}\lambda \;\leq\; 2^{\ell_L+3}\lambda \;\leq\; 2^{(\ell+1)/2+3}\lambda \;=\; 2^{\,\ell/2+7/2}\lambda,$$

whence $E(\ell)/(2^{\ell+1}\lambda) \leq 2^{\,\ell/2+7/2-\ell-1} = 2^{\,5/2-\ell/2}$, which is (6). The final claim follows from $\mathsf{Flat}(\ell)/T(\ell) = \frac{\ell}{2}\big(1 + E(\ell)/2^{\ell+1}\lambda\big)^{-1}$ together with $(1+x)^{-1} \geq 1 - x$ for $x \geq 0$. □

Lemma 6 (Depth one suffices). *Let $T_1(\ell) = 2^{\ell+1}\lambda + 2^{\ell_L}\ell_L\lambda + 2^{\ell_R}\ell_R\lambda$ denote the size of the two-layer construction of Definition 1, in which the two sub-tables are stored flat. Then*

$$1 \;\leq\; \frac{T_1(\ell)}{T(\ell)} \;\leq\; 1 + \ell_L\, 2^{\,\ell_L - \ell} \;=\; 1 + O\big(\ell\, 2^{-\ell/2}\big).$$

Proof. For the lower bound it suffices that $T(w) \leq \mathsf{Flat}(w)$ for every w, since T and T_1 share the glue term and differ only in whether the two sub-tables are expanded recursively or stored flat. This holds with equality for $w \leq \ell_0$ by (3), and for $w > \ell_0$ by induction: $T(w) = 2^{w+1}\lambda + T(w_L) + T(w_R) \leq 2^{w+1}\lambda + \mathsf{Flat}(w_L) + \mathsf{Flat}(w_R)$, and since $w_L, w_R \leq \lceil w/2 \rceil$ the last two terms are at most $2^{\lceil w/2 \rceil + 1}\lceil w/2 \rceil\lambda$, so the total is below $2^w w\lambda = \mathsf{Flat}(w)$ for every $w > \ell_0 = 3$.

For the upper bound, $T_1(\ell) - T(\ell) \leq 2^{\ell_L}\ell_L\lambda + 2^{\ell_R}\ell_R\lambda \leq 2^{\ell_L+1}\ell_L\lambda$ using $\ell_R \leq \ell_L$, while $T(\ell) \geq 2^{\ell+1}\lambda$ by (6). Dividing gives $\ell_L 2^{\,\ell_L - \ell}$, and $\ell_L \leq (\ell+1)/2$ yields the stated order. □

Table 3. Table size at $\lambda = 256$ bits. "Glue" is the glue layer's share of $T(\ell)$; the last column is the excess of the two-layer construction over full recursion.

ℓ	Flat	$T(\ell)$	Ratio	Glue	d=1 excess
12	1.57 MB	0.27 MB	5.8×	95.8%	4.77%
16	33.55 MB	4.23 MB	7.9×	99.1%	2.18%
20	671.1 MB	67.3 MB	10.0×	99.8%	0.76%
24	12.88 GB	1.07 GB	12.0×	99.9%	0.24%

Equation (4) reproduces Table 3 exactly, and Corollary 4 is tight: it predicts a ratio of 7.9375 at $\ell = 16$ against an exact 7.9265, and 15.99951 at $\ell = 32$ against 15.99951. Two consequences deserve emphasis. First, the saving is a $\Theta(\ell)$ *factor* and the order remains $\Theta(2^\ell)$, since the glue layer keeps one entry per value of m; in terms of the number of pre-signatures $N = 2^\ell$ the saving is $\frac{1}{2}\log_2 N$, logarithmic in a cost exponential in ℓ. Second, the glue layer is 95.8% of $T(\ell)$ at $\ell = 12$ and exceeds 98% from $\ell = 14$, so full recursion beats the two-layer construction of Definition 1 by 2.2% at $\ell = 16$ and 0.24% at $\ell = 24$. We therefore analyse security at depth one, giving hybrids of two layers rather than $\log \ell$ at a cost below 3% in size.

7.3 Security definitions

Two adversarial settings occur and must be kept apart. In *Setting A* the prover is honest and the adversary corrupts the verifier, equivalently all T watchtowers; this is the setting for privacy. In *Setting B* at least one watchtower is honest and the adversary corrupts the prover; this is the setting for soundness.

Definition 7 (Withholding setup). *In the setup of Section 4, $\mathcal{P}$ obtains $\mathcal{V}$'s partial signatures $s_{\mathcal{V},m}$ for all m and completes σ_m^{Sch} locally, but releases no partial signature of its own; at the end of setup $\mathcal{P}$ holds all 2^ℓ complete signatures and $\mathcal{V}$ holds none.*

Definition 7 is a requirement, not an optimisation. If $\mathcal{V}$ learned every s_m it would derive every K_m and decrypt the whole table, falsifying Theorem 17; with the WOTS payload it would obtain a valid signature for every message under pk^W.

Definition 8 (Atomicity). RTT *is* atomic *if there is a PPT simulator* Sim *with*

$$\big(\mathsf{Tab}, \mathsf{rt}, K_{m^\star}, \{\mathsf{com}_i^b\}\big) \;\approx_c\; \mathsf{Sim}\big(1^\lambda, \ell, m^\star, \{L_i^{m_i^\star}\}_{i\in[\ell]}, \{\mathsf{com}_i^b\}\big)$$

for every ℓ, every label set and every $m^\star \in \{0,1\}^\ell$.

Definition 9. *Let π_{RTT} have public input $(\mathsf{rt}, \{R_m\}, X, \mu, \{\mathsf{com}_i^b\}, \mathsf{ctx})$.* RTT *is* extractable *if for any PPT $\mathcal{A}$ producing an accepted $(\mathsf{Tab}, \mathsf{rt}, \pi_{\mathsf{RTT}})$, with overwhelming probability, for every m, $\mathsf{RTT.Open}(\mathsf{Tab}, m, K_m) = (\hat{L}_1, \ldots, \hat{L}_\ell)$ with $H(\hat{L}_i) = \mathsf{com}_i^{m_i}$ for all $i \in [\ell]$.*

The bit-indexing in Definition 9 is essential. Label commitments certify that a recovered value is *a* valid label, not that it is the label for the intended bit: an entry C_u^L containing $L_i^{1-u_i}$ would pass a check against $\mathsf{com}_i^{1-u_i}$ and silently deliver the wrong bit. Requiring $\mathsf{com}_i^{m_i}$ rules this out, at the cost of carrying the index into the setup relation.

Finally, RTT satisfies *table binding* if no PPT $\mathcal{A}$ acting as $\mathcal{P}$ can produce a single accepted spend of the optimistic output from which two distinct values are openable.

We use two assumptions on the signature layer. In the *N-OMDL* game the adversary receives $Y_0, \ldots, Y_N \xleftarrow{\$} \mathbb{G}$ and an oracle returning discrete logarithms, callable N times, and wins by outputting all $N+1$ logarithms; all reductions below are algebraic, so they hold under AOMDL as well. In the *q-session one-more unforgeability* game the adversary corrupts one party, engages the honest party in at most q signing sessions on the fixed message μ, receives t completed signatures, and wins by outputting $t+1$ distinct valid signatures on μ under X. The second notion is needed because every σ_m^{Sch} signs the *same* μ, which makes the usual fresh-message notion vacuous here: what WISCH requires is that the number of obtainable signatures on μ not exceed the number of completed sessions.

7.4 The signature layer

Lemma 10. *Suppose each party publishes a binding commitment to its entire nonce-share vector before either opens, and $\mathcal{P}$ samples its shares uniformly. Then the $\{R_m\}$ are uniform and independent whatever the corrupt party does, and $\Pr[\exists m \neq m' : R_m = R_{m'}] \leq 2^{2\ell-1}/p + \mathsf{Adv}^{\mathsf{bind}}$. With the abort-on-collision check, setup never completes with a colliding pair, and the probability of a spurious abort is bounded by the same quantity. Without the commitment round, a party seeing the other's shares first forces a collision with probability* 1.

Proof. Binding fixes both vectors before either is revealed, so the corrupt shares are independent of the honest ones up to $\mathsf{Adv}^{\mathsf{bind}}$; each R_m is then a uniform element shifted by a fixed one, and the family is independent because the $r_{\mathcal{P},m}$ are. Union-bound over $\binom{2^\ell}{2} \leq 2^{2\ell-1}$ pairs. For the last claim, set $R_{\mathcal{A},m'} := R_{\mathcal{A},m} + R_{\mathcal{P},m} - R_{\mathcal{P},m'}$. □

The commitment round also removes the adaptivity that the ROS attack exploits, under which q concurrent sessions on a fixed message under a fixed aggregate key yield $q+1$ signatures [6]; the underlying ROS_q instance is solvable in sub-exponential time by Wagner's generalised birthday algorithm [23], and in polynomial time once $q \geq \log_2 p$ [4]. WISCH runs $q = 2^\ell$, so this applies for every parameter in range; Corollary 16 fails without it.

Lemma 11. *In Setting A under Definition 7, let $\mathcal{A}$ receive its own key material and nonce shares, the public $(X_{\mathcal{P}}, \{R_{\mathcal{P},m}\}, \{R_m\}, X, \mu)$ and the revealed $s_{m^\star}$. If $\mathcal{A}$ outputs $s_{m'}$ with $s_{m'}G = S_{m'}$ for some $m' \neq m^\star$, there is a PPT $\mathcal{B}$ with $\mathsf{Adv}^{\text{1-omdl}}(\mathcal{B}) \geq 2^{-\ell} \Pr[\mathcal{A}\ \textit{wins}]$.*

Proof. $\mathcal{B}$ receives (Y_0, Y_1) and one oracle query. It sets $X_{\mathcal{P}} := Y_0$, samples $\hat{m} \xleftarrow{\$} \{0,1\}^{\ell}$, sets $R_{\mathcal{P},\hat{m}} := Y_1$, and samples $r_{\mathcal{P},m}$ honestly for $m \neq \hat{m}$. It commits, exchanges and opens as in Lemma 10, then queries $\mathsf{DL}(Y_0)$ to obtain $x_{\mathcal{P}}$; this is its only query and is algebraic. By Definition 7 no partial signature is owed during setup, and $\mathcal{B}$ can compute $s_{\mathcal{P},m} = r_{\mathcal{P},m} + c_m a_{\mathcal{P}} x_{\mathcal{P}}$ for every $m \neq \hat{m}$. At reveal $\mathcal{B}$ aborts if $m^\star = \hat{m}$ and otherwise publishes $s_{m^\star}$, distributed exactly as in the real execution. If $\mathcal{A}$ outputs $s_{m'}$ with $m' = \hat{m}$, then $r_{\mathcal{P},\hat{m}} = s_{\hat{m}} - s_{\mathcal{V},\hat{m}} - c_{\hat{m}} a_{\mathcal{P}} x_{\mathcal{P}} = \mathsf{dlog}(Y_1)$, and $\mathcal{B}$ outputs $(x_{\mathcal{P}}, r_{\mathcal{P},\hat{m}})$: two logarithms from one query. The simulation is perfect given $\hat{m} \neq m^\star$, and $\hat{m}$ is independent of $\mathcal{A}$'s view, so $\Pr[\hat{m} = m'] = 2^{-\ell}$, which also forces $\hat{m} \neq m^\star$. □

With two nonces per party, $\mathcal{B}$ instead sets $R_{\mathcal{P},\hat{m},1} := Y_1$ and $R_{\mathcal{P},\hat{m},2} := \beta G$ for known β and recovers $\mathsf{dlog}(Y_1) = r_{\mathcal{P},\hat{m}} - b_{\hat{m}}\beta$.

Proposition 12 (Table binding). *Given Lemma 10 and a ledger admitting at most one spend per output, at most one index is openable per accepted spend.*

Proof. The canonical chain contains one transaction spending the output, carrying one signature $(R^\star, s^\star)$; the reveal index is the unique m with $R_m = R^\star$. If two indices were openable, then by Theorem 18 $s^\star = s_m$ and $s^\star = s_{m'}$, whence $R_m = R^\star = R_{m'}$, contradicting distinctness. □

Proposition 12 identifies what enforces the one-time property: not a cryptographic mechanism but the ledger's double-spend rule. It also requires the reveal to be final, since a reorganisation making two scalars public would open two entries.

Lemma 13 (Fresh nonce). *In any execution in which $\mathcal{A}$ wins $\mathbf{G}^{\mathsf{omuf}}$, at least one of its $t+1$ output signatures carries an aggregate nonce $R^\star$ distinct from the aggregate nonces $R^{(1)}, \ldots, R^{(t)}$ of the completed sessions.*

Proof. Suppose every output nonce lies in $\{R^{(1)}, \ldots, R^{(t)}\}$. Two distinct output signatures cannot share a nonce: if (R, s) and (R, s') are both valid then $sG = R + H(R\|X\|\mu)X = s'G$, and $\mathbb{G}$ has prime order, so $s = s'$. Hence the $t+1$ outputs carry $t+1$ pairwise distinct nonces, all drawn from a set of size t, which is impossible. □

Lemma 14 (Signing simulation). *There is a simulator that plays the honest signer in Π_3 on the fixed message μ without its secret key, whose output is distributed identically to the real one except with probability at most $2q_s(q_h + q_s + 1)/p$, where q_s is the number of sessions and q_h the number of random-oracle queries.*

Proof. Write the honest signer as party 2 and let X_2 be its public key. In round 1 the simulator sends a uniformly random $t_2 \xleftarrow{\$} \{0,1\}^{\lambda}$ in place of $H_{\mathsf{com}}(R_2)$. In round 2, *after* $\mathcal{A}$ has opened its own nonce R_1, the simulator samples $s_2, c \xleftarrow{\$} \mathbb{Z}_p$ and sets

$$R_2 \;:=\; s_2 G - c\, a_2 X_2, \qquad R \;:=\; R_1 + R_2,$$

programs $H_{\mathsf{com}}(R_2) := t_2$ and $H(R\|X\|\mu) := c$, opens R_2, and in round 3 releases s_2. By construction $s_2 G = R_2 + c\, a_2 X_2$, so the partial signature verifies, and R_2 is uniform because s_2 is, so R is uniform and independent of $\mathcal{A}$'s view at the time it committed.

The simulation is perfect unless one of the two programming steps overwrites a previously assigned value. Since R_2 and R are uniform, each collides with one of the at most $q_h + q_s$ points already assigned in the respective oracle with probability at most $(q_h + q_s + 1)/p$; a union bound over the two oracles and the q_s sessions gives the stated term. □

Theorem 15 (One-more unforgeability of Π_3). *Model H_{agg}, H_{com} and H as random oracles. For every PPT $\mathcal{A}$ making at most q_h random-oracle queries and opening at most $q_s \le q$ sessions on the fixed message μ,*

$$\mathsf{Adv}^{\mathsf{omuf}}_{\Pi_3}(q) \;\le\; \left(q_h^3 \cdot \mathsf{Adv}^{\mathsf{dl}}_{\mathbb{G}}\right)^{1/4} \;+\; \frac{2q_s(q_h + q_s + 1)}{p}.$$

Proof. Let $\mathcal{B}$ receive a discrete-logarithm challenge $Y \in \mathbb{G}$ and set $X_2 := Y$. It runs $\mathcal{A}$, which chooses X_1 adaptively, and answers signing queries by Lemma 14; the simulated execution differs from the real one by at most the second term of the bound, and $\mathcal{B}$ never uses x_2.

Suppose $\mathcal{A}$ wins. By Lemma 13 some output signature $(R^\star, s^\star)$ has a nonce outside the set of session nonces. Because the simulator programmed H only at the session nonces, its value at $R^\star\|X\|\mu$ was never assigned by $\mathcal{B}$, so $\mathcal{A}$ must have queried that point; let I be the index of the query.

Inner forking. Apply the generalised forking lemma [3] at query I. Two executions sharing their randomness up to I and differing in the reply yield $(R^\star, s^\star)$ and $(R^\star, \hat{s}^\star)$ with challenges $c \neq \hat{c}$, whence

$$(s^\star - \hat{s}^\star)G = (c - \hat{c})X \quad\Longrightarrow\quad x := \frac{s^\star - \hat{s}^\star}{c - \hat{c}} \;\text{ satisfies }\; X = xG.$$

Thus $\mathcal{B}$ recovers $x = a_1 x_1 + a_2 x_2$, the discrete logarithm of the *aggregate* key, but not yet x_2.

Outer forking. Apply the forking lemma a second time, following the double-forking technique of [16], now at the query $H_{\mathsf{agg}}(L\|X_2)$ that fixes a_2, running the inner extractor in each branch. The two branches share a_1, since $H_{\mathsf{agg}}(L\|X_1)$ is a distinct oracle point unaffected by the fork, and return $x = a_1 x_1 + a_2 x_2$ and $x' = a_1 x_1 + a_2' x_2$ with $a_2 \neq a_2'$. Subtracting, the unknown $a_1 x_1$ cancels and

$$x_2 \;=\; \frac{x - x'}{a_2 - a_2'},$$

which is $\log_G Y$. Two nested applications of the forking lemma [3] turn a success probability ε into ε^4/q_h^3 up to terms of order $1/p$, the same quartic loss as in the analysis of [16], giving the stated bound. □

Corollary 16 (No null opening). *In Setting B, the probability that a PPT $\mathcal{P}$ spends the optimistic output with a signature from which no index is openable*

is at most $\mathsf{Adv}^{\mathsf{omuf}}_{\Pi_3}(2^\ell)$*, which by Theorem 15 is negligible under the discrete logarithm assumption in the random oracle model.*

Proof. If $R^\star = R_m$ with $s^\star \neq s_m$, validity gives $s^\star G = R^\star + c_m X = s_m G$, so $s^\star = s_m$ by primality. Hence $R^\star \notin \{R_m\}$, and $(R^\star, s^\star)$ together with the 2^ℓ signatures obtained from $\mathcal{V}$ is a set of $2^\ell + 1$ distinct valid signatures on μ. □

Corollary 16 closes the case in which a deviation is detectable but not punishable: with nothing decryptable, no watchtower holds a witness for the fraud-proof path.

7.5 Atomicity and extractability

Theorem 17 (Atomicity). *In Setting A under Definition 7, Definition 1 is atomic, with*

$$\mathsf{Adv}^{\mathsf{atom}} \;\le\; 2^\ell \mathsf{Adv}^{\text{1-omdl}} + \big(2^\ell + 2^{\ell_L} + 2^{\ell_R} - 3\big)\mathsf{Adv}^{\mathsf{PRF}} + 2^{2\ell}/2^\lambda.$$

Proof. $\mathbf{G}_0$ is the real game, in which $\mathcal{A}$ sees $(\mathsf{Tab}, \mathsf{rt}, s_{m^\star}, \{R_m\}, X, \{\mathsf{com}_i^b\})$.

$\mathbf{G}_1$ replaces K_m, $m \neq m^\star$, by independent uniform strings. As H_{kdf} is a random oracle the games differ only if $\mathcal{A}$ queries it at $s_m \| m \| \mathsf{ctx}$ for some $m \neq m^\star$. A reduction parsing each query as $s\|m\|\mathsf{ctx}$ and testing $sG = S_m$ is exactly the adversary of Lemma 11, so the gap is at most $2^\ell \mathsf{Adv}^{\text{1-omdl}}$.

$\mathbf{G}_2$ aborts if any two of the $2^\ell + 2^{\ell_L} + 2^{\ell_R}$ keys coincide. All are uniform and independent in $\mathbf{G}_1$, so the gap is at most $\binom{2^\ell + 2^{\ell_L} + 2^{\ell_R}}{2} 2^{-\lambda} \le 2^{2\ell}/2^\lambda$, using $2^{\ell_L} + 2^{\ell_R} \le 2^\ell$. This term has no counterpart under AEAD and arises because the pad is deterministic: two equal keys with equal context strings would give a two-time pad.

$\mathbf{G}_3$ replaces $\mathsf{PRF}(K_m, \mathsf{ctx}\|\mathsf{g}\|m)$, $m \neq m^\star$, by uniform strings, one index at a time. Each step is a PRF game with a fresh uniform key, unused elsewhere by $\mathbf{G}_2$, so the gap is $(2^\ell - 1)\mathsf{Adv}^{\mathsf{PRF}}$. Every unopened glue entry is now uniform and independent of its plaintext.

Consequently, for $u \neq m^\star_L$ the value $K_L(u)$ occurs in $\mathcal{A}$'s view only inside glue entries C^{g}_m with $m_L = u$, all of which satisfy $m \neq m^\star$ and are now uniform; so $K_L(u)$ is information-theoretically hidden, and likewise $K_R(v)$ for $v \neq m^\star_R$. By contrast $K_L(m^\star_L)$ occurs in $C^{\mathsf{g}}_{m^\star}$, which is opened, and in entries that are now uniform, so it is revealed exactly once, as intended.

$\mathbf{G}_4$ replaces the unopened leaf pads by uniform strings, costing $(2^{\ell_L} + 2^{\ell_R} - 2)\mathsf{Adv}^{\mathsf{PRF}}$ by the same argument.

Sim now samples uniform strings for every unopened entry, samples $K_L(m^\star_L)$ and $K_R(m^\star_R)$, sets $K_{m^\star} = H_{\mathsf{kdf}}(s_{m^\star}\|m^\star\|\mathsf{ctx})$, and forms the three opened ciphertexts from its inputs, computing rt. Every quantity it uses is in its input, and the distribution is identical to $\mathbf{G}_4$. Summing the gaps gives the bound. □

The two multiplicative losses compound to $2^{2\ell}$, giving the parameter constraint $\lambda \ge 2\ell + 128$: satisfied by $\lambda = 256$ and $\ell \le 24$, but not by 128-bit labels.

Theorem 18 (Extractability). *In Setting B, if π_{RTT} is knowledge sound with knowledge error $\varepsilon_{\mathsf{ks}}$ and H_{M} is collision resistant, Definition 1 is extractable, and extractability fails with probability at most $\varepsilon_{\mathsf{ks}} + \mathsf{Adv}^{\mathsf{cr}}_{H_{\mathsf{M}}}$.*

Proof. Knowledge soundness yields, except with probability $\varepsilon_{\mathsf{ks}}$, a witness satisfying every clause of (7) below; let $\mathsf{Tab}^{\mathsf{ext}}$ be the table it determines. The clause $s_m G = S_m$ pins each s_m uniquely, $\mathbb{G}$ having prime order and S_m being fixed by public data, so the extracted $K_m = H_{\mathsf{kdf}}(s_m \| m \| \mathsf{ctx})$ is the key any verifier derives at reveal time.

If the served table differs from $\mathsf{Tab}^{\mathsf{ext}}$ at a position whose Merkle path nonetheless verifies against rt, then along the two paths there is a first node where the inputs to H_{M} differ but the outputs agree, giving a collision; this costs $\mathsf{Adv}^{\mathsf{cr}}_{H_{\mathsf{M}}}$. Assume then that they agree.

Fix m. Removing the pad from C^{g}_m yields exactly $K_L(m_L) \| K_R(m_R)$ by the glue clause. Removing the pad from $C^{L}_{m_L}$ yields $L_1^{m_1} \| \cdots \| L_{\ell_L}^{m_{\ell_L}}$ by the left clause instantiated at $u = m_L$, and symmetrically on the right, so $\hat{L}_i = L_i^{m_i}$ for every i. The clause $H(L_i^b) = \mathsf{com}_i^b$ at $b = m_i$ gives $H(\hat{L}_i) = \mathsf{com}_i^{m_i}$. □

The bit-indexed superscripts in the leaf clauses carry the whole weight of the third step: written non-indexed, the argument would only give $\hat{L}_i \in \{L_i^0, L_i^1\}$. Note also that collision resistance of H is not needed here; it is consumed in Theorem 19.

7.6 Composition with garbling

Let Gb be a garbling scheme for the predicate circuit P with input encoding $e = \{L_i^b\}$ and output labels $(o^{(0)}, o^{(1)})$ committed as $\mathsf{com}^b_{\mathsf{out}}$. We require correctness, privacy, and *authenticity*: given $(\hat{\mathsf{P}}, \mathsf{Encode}(e, x))$ no PPT adversary outputs $o^{(1-\mathsf{P}(x))}$ except with probability $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{Gb}}$. Authenticity, rather than privacy, is what the fraud-proof direction consumes.

Theorem 19 (Composition). *Let π_{Gb} certify that $(\hat{\mathsf{P}}, \{\mathsf{com}_i^b\}, \{\mathsf{com}^b_{\mathsf{out}}\})$ is a correct garbling of P, let RTT be atomic and extractable, and let H be collision resistant. In an accepted setup, after one reveal of $s_{m^\star}$:* (i) *the evaluator obtains exactly $\mathsf{Encode}(e, m^\star)$;* (ii) *if $\mathsf{P}(m^\star) = 0$ any honest watchtower computes $o^{(0)}$ and satisfies $H(\cdot) = \mathsf{com}^0_{\mathsf{out}}$;* (iii) *if $\mathsf{P}(m^\star) = 1$ no PPT adversary, including a corrupt watchtower holding the whole table, satisfies that check except with probability $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{Gb}} + \mathsf{Adv}^{\mathsf{cr}}_{H} + \mathsf{Adv}^{\mathsf{atom}}$.*

Proof. (i) By Theorem 18 the opening is $(L_i^{m_i^\star})_i$ with $H(L_i^{m_i^\star}) = \mathsf{com}_i^{m_i^\star}$, and by π_{Gb} the commitments are those of a correct garbling, so this tuple is $\mathsf{Encode}(e, m^\star)$. (ii) follows from correctness of Gb. For (iii), let $\mathcal{A}$ output o' with $H(o') = \mathsf{com}^0_{\mathsf{out}}$. If $o' \neq o^{(0)}$ this is an H-collision. If $o' = o^{(0)}$, build $\mathcal{B}$ against authenticity: $\mathcal{B}$ receives $(\hat{\mathsf{P}}, \mathsf{Encode}(e, m^\star))$ and the commitments, runs the simulator of Theorem 17 to fabricate a table, hands $\mathcal{A}$ that table with $\hat{\mathsf{P}}$ and π_{Gb}, and forwards $\mathcal{A}$'s output. The views differ by at most $\mathsf{Adv}^{\mathsf{atom}}$ and $o^{(0)} = o^{(1-\mathsf{P}(m^\star))}$. □

Part (iii) shows that the two layers do not separate as cleanly as the construction suggests. Soundness decomposes into two independent arguments meeting at $\{\mathsf{com}\}$, Theorem 18 and π_{Gb}, but the reduction to authenticity is available only because atomicity lets $\mathcal{B}$ fabricate the table from the opened labels alone. Without it a corrupt watchtower could decrypt input labels for some $m \neq m^\star$, evaluate the circuit on an input of its choosing and obtain $o^{(0)}$ legitimately, slashing an honest operator. Since atomicity rests in turn on Lemma 11, the soundness of the fraud proof depends on a hardness property of the signature layer.

7.7 Realising the setup relation

Write $\mathsf{x} = (\mathsf{rt}, \{R_m\}_m, X, \mu, \{\mathsf{com}_i^b\}, \mathsf{ctx})$, $\mathsf{w} = (\{s_m\}_m, \{K_L(\cdot)\}, \{K_R(\cdot)\}, \{L_i^b\})$ for the statement and witness, and recall $S_m = R_m + H(R_m \| X \| \mu) \cdot X$. The clauses below are required for all $m \in \{0,1\}^\ell$, $u \in \{0,1\}^{\ell_L}$, $v \in \{0,1\}^{\ell_R}$, $i \in [\ell]$ and $b \in \{0,1\}$.

$$\mathcal{R}_{\mathsf{RTT}} = \left\{ (\mathsf{x}; \mathsf{w}) \;\middle|\; \begin{array}{l} s_m G = S_m \\ K_m = H_{\mathsf{kdf}}(s_m \| m \| \mathsf{ctx}) \\ C_m^{\mathsf{g}} = (K_L(m_L) \| K_R(m_R)) \oplus \mathsf{PRF}(K_m, \cdot) \\ C_u^L = (L_i^{u_i})_{i \le \ell_L} \oplus \mathsf{PRF}(K_L(u), \cdot) \\ C_v^R = (L_{\ell_L + i}^{v_i})_{i \le \ell_R} \oplus \mathsf{PRF}(K_R(v), \cdot) \\ H(L_i^b) = \mathsf{com}_i^b \\ \mathsf{rt} = \mathsf{MerkleRoot}(\{C_m^{\mathsf{g}}\}, \{C_\cdot^L\}, \{C_\cdot^R\}) \end{array} \right\} \tag{7}$$

The clauses $H(L_i^b) = \mathsf{com}_i^b$ are global, 2ℓ in all; the rest are per index. Since S_m is computable from public data the only group operation in-circuit is the fixed-base multiplication $s_m G \stackrel{?}{=} S_m$, and it dominates. Taking order-of-magnitude R1CS counts over BN254, SHA-256 compression $\approx 2.7 \times 10^4$, non-native `secp256k1` fixed-base multiplication $\approx 3 \times 10^5$, Poseidon permutation ≈ 240, an instance built from AES-GCM, HMAC-SHA256 and a SHA-256 Merkle path of depth $\ell + 1$ costs 8.5×10^5 constraints, of which the group operation is 35%, the Merkle path 54% and the AEAD 4.7%. Replacing the AEAD alone therefore buys 5%; replacing all symmetric components buys 2.8×, at which point the group operation is 98% of what remains.

It can be removed. Drop $s_m G = S_m$ from (7) and add to the collateral output a path claimable by $\mathcal{V}$ after a delay $t_{\mathsf{chal}} < t_1$ against a challenge bond, which $\mathcal{P}$ answers by publishing the opening for the revealed index on-chain, a successful defence transferring the bond to $\mathcal{P}$. Theorem 18 used that clause only to identify the extracted s_m with the published one; without it a malformed table admits an index whose decryption fails the bit-indexed H check, which is a public predicate on the decrypted value and hence detectable, and the challenge converts detectability into punishability. Extractability weakens to: for every m, either the opening is bit-indexed-consistent, or $\mathcal{P}$ loses the collateral, which leaves Theorem 19 unchanged. The cost is one transaction in an already rare path, and

a griefing surface that is self-limiting since $\mathcal{P}$ pays for its own transaction and forfeits collateral. Where the extra script path is unwelcome, the alternative is to publish a Pedersen commitment $P_m = s_m G + \rho_m H$ and prove the link by a Schnorr proof of knowledge of $\mathsf{dlog}_H(P_m - S_m)$, which is 64 bytes and batch-verifiable, leaving only the cross-field commitment link in-circuit.

With the group operation gone and all symmetric components Poseidon, an instance costs 6.1×10^3 constraints, or 2.5×10^3 if the 2^ℓ instances are folded so that the Merkle tree is built incrementally at ≈ 2 hashes per instance rather than $\ell + 1$. Against the initial 8.5×10^5 this is 344×, or 2.7 minutes against 15.5 hours per chunk per step at 10^6 constraints/second. Folding is preferable to per-instance proofs on two further grounds: at 128 bytes per Groth16 proof, 2^{16} proofs are 8.4 MB, twice the table they accompany; and verification, repeated by every watchtower and every late joiner, falls from 2^ℓ pairings to one. These figures are estimates and should be measured, though their ordering is robust.

8 Conclusion

WISCH introduces an "optimistic binding" primitive that securely correlates Winternitz One-Time Signatures with Schnorr signatures, enabling protocols to operate with a minimal 64-byte on-chain footprint in the common case while preserving hash-based fraud proofs for disputes. Our economic security analysis shows this hybrid approach is robust under the rational adversary model, making it suitable for trust-minimized bridges, complex covenants, and verifiable oracles.

Future work will focus on instantiating WISCH within BitVM bridges and optimizing the setup phase, particularly through recursive proofs (e.g., Halo2, STARKs) to batch encryption correctness proofs and Multi-Party Computation (MPC) to distribute the setup trust.

References

1. Alagic, G., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., Perlner, R., et al.: Status report on the third round of the nist post-quantum cryptography standardization process. Tech. rep., US Department of Commerce, NIST (2022)
2. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Advances in Cryptology—ASIACRYPT 2000. pp. 531–545. Springer (2000)
3. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS). pp. 390–399. ACM (2006)
4. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. In: Advances in Cryptology – EUROCRYPT 2021. Lecture Notes in Computer Science, vol. 12696, pp. 33–53. Springer (2021)
5. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: 2015 IEEE symposium on security and privacy. pp. 104–121. IEEE (2015)

6. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the security of two-round multi-signatures. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 1084–1101. IEEE (2019)
7. Dryja, T.: Discreet log contracts. https://adiabat.github.io/dlc.pdf (2017)
8. Eagen, L.: Glock: Garbled locks for bitcoin. Cryptology ePrint Archive, Paper 2025/1485 (2025), https://eprint.iacr.org/2025/1485
9. Farshim, P., Libert, B., Paterson, K.G., Quaglia, E.A.: Efficient schemes for committing authenticated encryption. In: Advances in Cryptology–EUROCRYPT 2017. pp. 300–329. Springer (2017)
10. Groce, A., Katz, J., Thiruvengadam, A., Zikas, V.: Byzantine agreement with a rational adversary. In: International Colloquium on Automata, Languages, and Programming. pp. 561–572. Springer (2012)
11. Hülsing, A.: W-OTS+ – shorter signatures for hash-based signature schemes. In: Progress in Cryptology – AFRICACRYPT 2013. Lecture Notes in Computer Science, vol. 7918, pp. 173–188. Springer (2013)
12. Kudinov, M., Nick, J.: Hash-based signature schemes for bitcoin. Cryptology ePrint Archive, Paper 2025/2203 (2025), https://eprint.iacr.org/2025/2203
13. Lamport, L.: Constructing digital signatures from a one way function. Tech. Rep. CSL-98, SRI International (1979)
14. Linus, R.: Bitvm: Compute anything on bitcoin. https://bitvm.org/bitvm.pdf (2023)
15. Linus, R., Aumayr, L., Pelosi, A., Avarikioti, Z., Maffei, M.: Bitvm2: Bridging bitcoin to second layers. Cryptology ePrint Archive, Paper 2024/1286 (2024), https://eprint.iacr.org/2024/1286
16. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to Bitcoin. Designs, Codes and Cryptography **87**(9), 2139–2164 (2019)
17. Merkle, R.C.: A certified digital signature. In: Advances in Cryptology—CRYPTO'89 Proceedings. pp. 218–238. Springer (1989)
18. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
19. Nick, J., Ruffing, T., Seurin, Y.: Musig2: Simple two-round multi-signatures. In: Annual International Cryptology Conference. pp. 189–221. Springer (2021)
20. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf (2016)
21. RootstockLabs: Bitvmx: A virtual cpu for universal computation on bitcoin. https://bitvmx.org/knowledge/whitepaper (2024)
22. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
23. Wagner, D.: A generalized birthday problem. In: Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 288–304. Springer (2002)
24. Wuille, P., Nick, J., Ruffing, T.: Bip 340: Schnorr signatures for secp256k1. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki (2020)
25. Wuille, P., Nick, J., Towns, A.: Bip 341: Taproot: Segwit version 1 spending rules. https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki (2020)