WISCH: Efficient data signing via correlated signatures for Bitcoin

Ariel Futoransky[1], Fadi Barbàra[1,2], Ramses Fernandez[1], Emilio Garcia[1], Gabriel Larotonda[3,4], and Sergio Demian Lerner[1,5]

[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. To solve this, we introduce **WISCH** (Winternitz-Schnorr), a hybrid cryptographic protocol that securely 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 suitable for high-value financial infrastructure. 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 by approximately 33× compared to standard approaches, achieving the theoretical minimum size while maintaining the robust security guarantees of a 1-of-N trust assumption.

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 [15] to Discret Log Contracts (DLCs) [5] and more recently, verifiable computation frameworks like

BitVM [10]. 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 [18,17] cannot easily provide. For instance, mechanisms involving bit-commitments, connectors in BitVM [10], or quantum-resistant fallback paths often necessitate the use of Hash-Based Signatures, such as Lamport [9] or Winternitz One-Time Signatures (WOTS) [12]. 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 [12]. In the constrained economic environment of the Bitcoin blockchain [13], 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 [7,3]—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 [8,1]. 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 [15] and Discreet Log Contracts (DLCs) [5] 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 [10,11,16] 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**

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 [17,18]. 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 [14] 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:

- $\mathsf{KeyAgg}(X_1, \ldots, X_n) \to X_{agg}$: 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 $X_{agg} = \sum_{i=1}^{n} a_i X_i$.
- $\mathsf{Sign}(x_1, \ldots, x_n, m)$: The parties interact in two rounds.
  1. *Nonce Exchange:* Each party i generates random nonces and broadcasts public nonces R_i. The aggregate nonce is $R = \sum_{i=1}^{n} R_i$.
  2. *Signature Aggregation:* The challenge is $c = H(R \parallel X_{agg} \parallel m)$. 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 $s = \sum_{i=1}^{n} s_i \pmod p$.

Verification is identical to $\mathsf{SCH.Ver}(X_{agg}, m, \sigma)$. This linearity is basic in WISCH, as it allows the *Optimistic Path* to appear as a standard single-key spend, indistinguishable from other traffic.

3.3 Winternitz One-Time Signatures (WOTS)

Winternitz One-Time Signatures (WOTS) [12] 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

[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 $(1011_2 = 11, 0001_2 = 1, 1110_2 = 14, 0100_2 = 4)$

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 [19] 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.

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. 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 [6].

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 the *Single-Show, Fixed-Key* model [4]. This implies that for a given setup, each index k corresponds to exactly one valid Schnorr signature

[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.

and one valid WOTS commitment. 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.

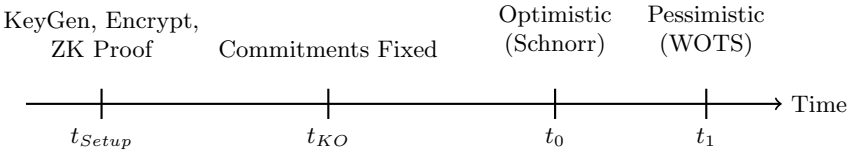

Fig. 1. Protocol Timeline showing the Optimistic and Pessimistic phases.

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. For each $m \in M$, P and V execute $\mathsf{MuSig2.Sign}$ on the fixed message μ using a fresh nonce R_m, obtaining $\sigma_m^{Sch} = (R_m, s_m)$ under pk^{Sch}. 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. 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 $K_{enc} \leftarrow \mathsf{HKDF}(s_m \parallel m \parallel 1)$. 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}: $\sigma_m^W \leftarrow \mathsf{WOTS.Sign}(sk^W, m)$; $C_m \leftarrow \mathsf{Enc}(K_{enc}, \sigma_m^W)$, 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:

$$\mathcal{R}_{enc} = \left\{ \begin{array}{l} (C, X, m) \\ (R_{m,k}, s_{m,k}, k, K_{enc}, sk_k^W) \end{array} \middle| \begin{array}{l} K_{\mathsf{enc}}^{(m)} = \mathsf{HKDF}(s_m \parallel m \parallel 1) \\ C_m = \mathsf{Enc}(K_{\mathsf{enc}}^{(m)}, \mathsf{WOTS.Sign}(sk^W, m)) \\ s_m \cdot G = R_m + H(R_m \| pk^{\mathsf{Sch}} \| m) \cdot pk^{\mathsf{Sch}} \\ pk_i^W = H_W^{w-1-b_i(m)}(\sigma_{m,i}^W) \quad \forall i \in [\ell] \end{array} \right\} \tag{1}$$

This proof prevents $\mathcal{P}$ from committing to "junk" data that cannot be opened.

6. **Collateral Lock:** P funds a Taproot address that commits to the script tree. 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 signatures under the aggregated public key pk^{Sch}. The script uses the Tapscript opcode `OP_CHECKSIGADD`, which verifies each signature and accumulates a count of successes[8].

(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 handles invalid WOTS signatures. 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[9].

(b) The second fraud proof path handles predicate violations, where the committed messages satisfy WOTS verification but violate the application-specific validity condition [10]

[8] We will see in the next section why this is important.

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

[10] 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.

(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 $\sigma_{m^*}^{\mathrm{Sch}} = (R_{m^*}, s_{m^*})$ to redeem the first output, first path.
2. The Bitcoin network verifies $\mathsf{SCH.Ver}(pk^{Sch}, \mu, \sigma_{m^*}^{Sch}) = 1$ with `OP_CHECKSIG`. V then observes the published scalar s_{m^*} and uses $\mathsf{HKDF}(s_{m^*} \| m^\star \| 1)$ 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.

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 $K_{\mathsf{enc}}^{(m^*)}$.
2. $\mathcal{V}$ decrypts C_{m^*} to obtain $\sigma_{m^*}^{W}$.
3. $\mathcal{V}$ checks $\mathsf{WOTS.Ver}(pk^W, m^*, \sigma_{m^*}^{W}) = 1$.

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 consisting of K computational steps (e.g., NAND gates). An execution trace of T is a sequence of values $\{m(0), m(1), \ldots, m(K-1)\}$, where each $m(k) \in \{0,1\}^N$ represents the state of the wires at step k. WISCH allows the Operator to commit to this entire trace efficiently and only reveal specific steps in case of a dispute.

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 cryptographic commitments for the circuit T of size K.

1. **Winternitz Keys:** For each step $k \in \{0, \ldots, K-1\}$, the Operator generates a WOTS key pair (sk_k^W, pk_k^W).
2. **Schnorr Pre-computation:** For every possible wire value m at every step k, the Operator and Watchtowers (or a trusted dealer) compute joint Schnorr signatures $\sigma_{m,k}^{\text{Sch}} = (R_{m,k}, s_{m,k})$ via MuSig2.
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 $S_{N+1} = T(S_N, \boldsymbol{op})$, they determine the correct execution trace $\{m(k)\}_{k=0}^{K-1}$. To finalize this state on Bitcoin:

1. The Operator aggregates the Schnorr signatures corresponding to the true trace: $\sigma_{\text{update}}^{\text{Sch}} = \text{Agg}(\{\sigma_{m(k),k}^{\text{Sch}}\}_k)$. (Note: Ideally, a single aggregate signature for the root or a subset is published, effectively authorizing the update).

2. **Submission:** The Operator publishes this single Schnorr signature on the *Key Path*.
3. **Implicit Reveal:** By publishing the signature (and thus the underlying scalar s), the Operator effectively releases the decryption keys for the specific WOTS signatures corresponding to the execution trace.

Cost: ≈ 64 bytes. To the network, it looks like a standard payment.

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

1. **Decryption:** The Watchtower uses the published Schnorr scalar to decrypt the WOTS signatures for the trace.
2. **Fault Isolation:** They re-execute T locally to find the first step k^* where the Operator's values are inconsistent with gate's logic (e.g., $NAND(1,1) \neq 0$).
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), second-preimage resistance of SHA256 (WOTS), 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 $\mathsf{Game}_{WISCH}(\mathcal{A}, \lambda)$ between a Prover adversary $\mathcal{A}$ and a Challenger $\mathcal{C}$ (acting as the honest Verifier/Contract).

Definition 1 (WISCH Challenge Game).

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:

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

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). We argue soundness via reduction to the security of the underlying AEAD scheme and the HKDF. Suppose $\mathcal{A}$ wins the game with non-negligible probability. This implies $\mathcal{A}$ produces a valid σ^S (deriving a valid key K_{enc}) but the ciphertext C either fails to decrypt or decrypts to an invalid value.

- *Case 1 (Integrity):* If $\mathsf{Dec}_{K_{enc}}(C) = \perp$, $\mathcal{A}$ has forged a valid authentication tag for the AEAD scheme without the key (since setup integrity is guaranteed by ZK). This breaks the *INT-CTXT* security of the AEAD.
- *Case 2 (Binding/Pseudorandomness):* If decryption succeeds but the inner value is invalid, we rely on the input consistency check ZK proof at setup. For $\mathcal{A}$ to open to a *different* value later, they must control the randomness of HKDF to derive a key collision or break the binding of the encryption. Assuming HKDF acts as a random oracle and AEAD is secure, this probability is negligible.
  Note on Key Reuse: During setup, $\mathcal{A}$ generates multiple WOTS signatures (for different m's) using the *same* WOTS key pair pk^W. In standard WOTS, this would be catastrophic. However, since these signatures are encrypted with *independent* keys (derived from distinct Schnorr signatures), the adversary never reveals more than one valid WOTS signature for a given index. The security of the unrevealed branches is protected by the semantic security of the AEAD scheme.

Thus, no PPT adversary can win the game except with negligible probability.

Economic Security and Optimistic Assumptions The security of WISCH relies on the "Optimistic" paradigm common to Layer-2 scaling solutions (e.g.,

Optimistic Rollups). 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 $S > C_{sad}$ (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 for committing to a 256-bit value ($L = 256$) using standard WOTS vs. WISCH. We assume $w = 16$ (4 bits per digit), requiring 64 digits plus checksum (approx 3 digits) = 67 digests.

Metric	Standard WOTS / BitVM	WISCH (Optimistic)
Witness Size	$\approx 2,144$ bytes (67×32)	**64 bytes**
Script OpCodes	$\approx 1,000$ (Unrolled Loop)	**1** (`OP_CHECKSIG`)
Transaction Weight	$\approx 8,600$ WU	$\approx$ **260 WU**
Cost Reduction	–	$\sim$ **33**$\times$

Table 2. Concrete On-Chain Comparison for a 256-bit commitment.

This 33$\times$ reduction applies to *every* step of the execution trace. For a bridge verifying a large circuit, the aggregate savings make the difference between economic viability and impossibility.

7 Conclusion

The WISCH protocol addresses a critical scalability bottleneck in Bitcoin's evolving application landscape: the efficient verification of complex, structured data. By securely binding the semantic capabilities of Winternitz One-Time Signatures to the efficiency of Schnorr signatures, WISCH introduces a novel "optimistic binding" primitive. This allows protocols to operate with the minimal on-chain footprint of a standard 64-byte signature in the vast majority of cases, only falling back to the data-intensive hash-based signatures when a dispute arises.

Our economic security analysis demonstrates that this hybrid approach is robust under the rational adversary model, making it a viable foundation for high-value verifiable oracles, complex covenants, and trust-minimized bridges like BitVM. By strictly penalizing equivocation or invalidity on the "unhappy path," WISCH incentivizes honest behavior and ensures that the heavy cost of hash-based verification is theoretically possible but practically rare.

Future work will focus on specific instantiations of WISCH within the BitVM bridge framework and optimizing the setup phase requirements. Specifically, we plan to investigate the use of recursive proofs (e.g., Halo2, STARKs) to batch the correctness proofs of the encryption, making the setup feasible for large-scale execution traces. We also aim to explore the potential of using Multi-Party Computation (MPC) to distribute the setup trust, further decentralizing the protocol's reliance on the prover's initial correctness proofs.

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. 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)
4. Buchmann, J., Dahmen, E., Ereth, S., Hülsing, A., Rückert, M.: On the security of the winternitz one-time signature scheme. In: International Conference on Cryptology in Africa. pp. 363–378. Springer (2011)
5. Dryja, T.: Discreet log contracts. `https://adiabat.github.io/dlc.pdf` (2017)
6. 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)
7. 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)
8. Kudinov, M., Nick, J.: Hash-based signature schemes for bitcoin. Cryptology ePrint Archive, Paper 2025/2203 (2025), `https://eprint.iacr.org/2025/2203`
9. Lamport, L.: Constructing digital signatures from a one way function. Tech. Rep. CSL-98, SRI International (1979)
10. Linus, R.: Bitvm: Compute anything on bitcoin. `https://bitvm.org/bitvm.pdf` (2023)
11. 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`
12. Merkle, R.C.: A certified digital signature. In: Advances in Cryptology—CRYPTO'89 Proceedings. pp. 218–238. Springer (1989)
13. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf` (2008)
14. Nick, J., Ruffing, T., Seurin, Y.: Musig2: Simple two-round multi-signatures. In: Annual International Cryptology Conference. pp. 189–221. Springer (2021)
15. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf` (2016)
16. RootstockLabs: Bitvmx: A virtual cpu for universal computation on bitcoin. `https://bitvmx.org/knowledge/whitepaper` (2024)
17. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
18. Wuille, P., Nick, J., Ruffing, T.: Bip 340: Schnorr signatures for secp256k1. `https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki` (2020)
19. 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)