

Cryptographic Stateless Stitching: Sparse Merkle Trees & Bilateral Relationships

Achieving Unbreakable Network Integrity Through Mathematical Tripwires

Brandon “Cryptskii” Ramsay

June 19, 2025

Abstract

Traditional blockchain systems rely on global consensus mechanisms that create bottlenecks, energy inefficiencies, and scalability limitations. I present a novel paradigm called *cryptographic stateless stitching* that achieves network-wide integrity without requiring global coordination. Through device-level meta-commitments combining Sparse Merkle Trees (SMTs) with bilateral relationship chains, I create an atomic interlock system where any attempt at fraud or double-spending mathematically “trips” permanent detection mechanisms. This transforms trust from a coordination problem into a cryptographic impossibility problem, enabling truly scalable decentralized systems that maintain perfect integrity even under adversarial conditions.

1 Introduction: The Fundamental Challenge of Decentralized Trust

The creation of trustworthy decentralized systems faces a fundamental tension: how can independent participants maintain consistent state without a central authority, while remaining resistant to fraud, double-spending, and Byzantine failures? Traditional approaches have relied on global consensus mechanisms—from Bitcoin’s proof-of-work to Ethereum’s proof-of-stake—that require all participants to agree on a single, universal state.

However, this global consensus approach introduces severe limitations. Every transaction must be validated by the entire network, creating computational bottlenecks that limit throughput to thousands of transactions per second at best. More critically, these systems create temporal windows where malicious actors can exploit the delay between transaction submission and final consensus confirmation.

Consider Alice attempting to double-spend: she could submit two conflicting transactions to different parts of the network simultaneously, gambling that network partitions or consensus delays might allow both transactions to appear valid until the network eventually converges on one version. Even sophisticated consensus algorithms cannot eliminate this fundamental race condition entirely.

This approach, *cryptographic stateless stitching*, solves this problem by transforming trust from a coordination challenge into a problem of mathematical impossibility. Rather than requiring global agreement to prevent fraud, I construct cryptographic structures that make it **mathematically impossible for fraud to remain undetected by any honest participant**. This creates what I term *cryptographic tripwires*, which permanently “brick” any device attempting inconsistent behavior by making further valid interaction mathematically impossible.

2 Background: Understanding the Building Blocks

To understand cryptographic stitching, we must first establish the foundational concepts that make this approach possible.

2.1 Sparse Merkle Trees: Efficient Cryptographic Commitments

A Sparse Merkle Tree (SMT) is a cryptographic data structure that allows efficient commitment to large sets of key–value pairs. Unlike traditional Merkle trees that only work with densely packed data, SMTs handle sparse datasets efficiently by using hash-based default values for empty positions.

Definition 1 (Sparse Merkle Tree). *A Sparse Merkle Tree T over a key-space $\{0, 1\}^n$ is a binary tree where:*

- *Each leaf corresponds to a position in the key-space.*
- *Non-empty leaves contain actual data values.*
- *Empty leaves contain a predetermined default hash value.*
- *Internal nodes contain the hash of their children.*
- *The root provides a succinct commitment to the entire key–value mapping.*

The crucial property of SMTs is that they enable *membership proofs*: given a root hash, one can prove that a specific key–value pair is included in the committed set using only $O(\log n)$ data, regardless of how much other data the tree contains.

2.2 Hash Chains: Temporal Ordering Without Coordination

A hash chain creates tamper-evident temporal ordering by linking each new state to the cryptographic hash of its predecessor.

Definition 2 (Hash Chain). *A hash chain $\mathcal{C} = \{s_0, s_1, \dots, s_k\}$ is a sequence of states where each state s_{i+1} contains $H(s_i)$ as a component, with H a collision-resistant hash.*

2.3 Bilateral Relationships: Localized Trust Domains

Rather than maintaining a single global state, our system organises interactions into bilateral relationships. Each pair of participants maintains their own shared hash chain that records their mutual interactions.

Definition 3 (Bilateral Relationship Chain). *For participants A and B , their chain $\mathcal{C}_{A,B}$ is a hash chain where each state records a mutually agreed interaction between A and B .*

Key advantages:

- **Privacy:** Participants only learn about interactions that involve them.
- **Scalability:** Unlimited participants without global coordination.
- **Fault Isolation:** Problems in one relationship do not affect others.
- **Offline Operation:** Two participants can interact without network connectivity.

3 Model and Construction

Participants. Let \mathcal{U} be the set of devices. Each device $A \in \mathcal{U}$ holds a long-term signing keypair (sk_A, pk_A) for a post-quantum signature scheme Σ (e.g., SPHINCS+), and a collision-resistant hash H (e.g., BLAKE3 with domain separation).

Relationship keys. For each counterparty identifier ID_i , define a fixed relationship key

$$k_i \leftarrow H(\text{"DSM/rel"} \parallel ID_i) \in \{0, 1\}^\lambda.$$

Let $M_A : \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ map $k_i \mapsto h_i$, where h_i is the current head digest of the bilateral chain $\mathcal{C}_{A,i}$.

Sparse Merkle commitment. Device A maintains a Sparse Merkle Tree (SMT) commitment to M_A :

$$r_A \leftarrow \text{Com}(M_A).$$

We write $\text{InclProof}(r_A, k_i, h_i)$ for a standard $O(\log \lambda)$ Merkle inclusion proof and $\text{VerifyIncl}(r_A, k_i, h_i, \pi) = 1$ for a correct verification.

Stitched transitions (atomic receipts). Let A and B update their bilateral chain at index $t \rightarrow t+1$, producing a new head h'_i for key k_i (where $i \equiv ID_B$ in A 's map). Define

$$r'_A \leftarrow \text{Update}(r_A, k_i, h'_i).$$

A constructs a *stitched receipt* that binds the **old and new global roots** together with the extbfrelationship-local change:

$$au_{A \rightarrow B} := \text{enc}(\text{extsf}^{\text{DSM} - \text{StitchedReceipt} - v1}, \text{ID}_A, \text{ID}_B, t, k_i, h_i, h'_i, r_A, r'_A, \pi_{\text{old}}, \pi_{\text{new}}),$$

where π_{old} is an $O(\log \lambda)$ inclusion proof for $(k_i \mapsto h_i)$ under r_A and π_{new} is an inclusion proof for $(k_i \mapsto h'_i)$ under r'_A . The function $\text{enc}(\cdot)$ is a deterministic, canonical byte encoding (e.g., CBOR-DET with length-prefixing).

Sign (by A). Let

$$m_A := H(\text{"DSM/receipt/A"} \parallel \tau_{A \rightarrow B}), \quad \sigma_A \leftarrow \Sigma.\text{Sign}(\text{sk}_A, m_A).$$

Verify (by B). B checks, *in order*:

$$\Sigma.\text{Vrfy}(\text{pk}_A, m_A, \sigma_A) = 1, \quad \text{VerifyIncl}(r_A, k_i, h_i, \pi_{\text{old}}) = 1, \quad \text{VerifyIncl}(r'_A, k_i, h'_i, \pi_{\text{new}}) = 1,$$

then recomputes $r''_A \leftarrow \text{Update}(r_A, k_i, h'_i)$ and rejects unless $r''_A = r'_A$. B also rejects if $t \neq t_{\text{local}} + 1$ for this relationship, or if it has already accepted any other receipt $(\cdot, \cdot, \cdot, r_A, \cdot)$ at the same prior root r_A and key k_i .

Countersign (by B). Bind the full transcript (including σ_A) with domain separation:

$$m_B := H(\text{"DSM/receipt/B"} \parallel \text{ID}_A \parallel \text{ID}_B \parallel t \parallel \tau_{A \rightarrow B} \parallel \sigma_A), \quad \sigma_B \leftarrow \Sigma.\text{Sign}(\text{sk}_B, m_B).$$

Persist (both sides). Each device appends the tuple

$$\rho_{A,B} := (\tau_{A \rightarrow B}, \sigma_A, \sigma_B)$$

to the bilateral log $C_{A,B}$ and advances its local view of A 's state from r_A to r'_A . Any future verifier that observes $\rho_{A,B}$ will accept r'_A *iff* the signature verifications hold and the inclusion proofs validate under (r_A, r'_A) as above; otherwise the receipt is rejected and the counterparty may irreversibly mark the fork attempt at (r_A, k_i) .

Verification rules. Any verifier checking A 's claimed state r'_A after the update must validate:

1. **Transcript signatures.** $\Sigma.\text{Vrfy}(\text{pk}_A, m_A, \sigma_A) = 1$ and $\Sigma.\text{Vrfy}(\text{pk}_B, m_B, \sigma_B) = 1$, where $m_A := H(\text{"DSM/receipt/A"} \parallel \tau_{A \rightarrow B})$ and $m_B := H(\text{"DSM/receipt/B"} \parallel \text{ID}_A \parallel \text{ID}_B \parallel t \parallel \tau_{A \rightarrow B} \parallel \sigma_A)$.
2. **Inclusion proofs.** $\text{VerifyIncl}(r_A, k_i, h_i, \pi_{\text{old}}) = 1$ and $\text{VerifyIncl}(r'_A, k_i, h'_i, \pi_{\text{new}}) = 1$.
3. **Deterministic replay.** $r''_A \leftarrow \text{Update}(r_A, k_i, h'_i)$ and $r''_A = r'_A$.

Acceptance predicates (local). A device rejects any receipt at (r_A, k_i) if it has (i) already accepted a receipt for the same (r_A, k_i) , (ii) observes $t \neq t_{\text{local}}+1$ for this relationship, or (iii) the post-state r'_A differs from $\text{Update}(r_A, k_i, h'_i)$.

Canonical encoding and identifiers. We use a deterministic, canonical byte encoding $\text{enc}(\cdot)$ (e.g., CBOR-DET with length-prefixing). All device identifiers ID_X are the 32-byte raw public key (little-endian) encoded as bytes (not hex), and all domain tags use ASCII. Concatenation \parallel is byte concatenation. All transcripts use deterministic encoding $\text{enc}(\cdot)$ and domain-separated hashes:

$$\begin{aligned} k_i &= H(\text{"DSM/rel"} \parallel \text{ID}_i), \\ m_A &= H(\text{"DSM/receipt/A"} \parallel \tau_{A \rightarrow B}), \\ m_B &= H(\text{"DSM/receipt/B"} \parallel \text{ID}_A \parallel \text{ID}_B \parallel t \parallel \tau_{A \rightarrow B} \parallel \sigma_A). \end{aligned}$$

Encoding and hash separation (authoritative). $\text{enc}(\cdot)$ is deterministic (e.g., CBOR-DET with length-prefixing). The operator \parallel denotes *byte concatenation* of encoded fields in the exact order shown. SMT hashing is domain-separated:

$$\begin{aligned} \text{Leaf}(k, v) &= H(\text{"DSM/leaf"} \parallel k \parallel v), \\ \text{Node}(L, R) &= H(\text{"DSM/node"} \parallel L \parallel R), \end{aligned}$$

so left/right swaps or leaf/node confusions are infeasible without a collision in H . Device identifiers ID_X are bytes (not hex strings); implementations **MUST** use a canonical byte form (e.g., raw public key).

Relationship-locality. All inclusion proofs are keyed by k_i ; a proof (k_i, h_i, π) is *not* valid for $k_j \neq k_i$ except with a collision on H .

4 Security Definitions

Forkability Game ForgeFork. An adversary \mathcal{A} controls a device A and interacts with two honest counterparties B, C . \mathcal{A} wins if it produces two stitched receipts

$$au' = (k_j, h_j, h'_j, r, r'), \quad \tau'' = (k_j, h_j, h''_j, r, r'')$$

for the *same* relationship key k_j and the *same* prior root r , satisfying: (i) both verify (signatures and inclusion proofs); (ii) $h'_j \neq h''_j$; and (iii) $(\tau', \sigma'_A, \sigma'_B)$ is accepted by B while $(\tau'', \sigma''_A, \sigma''_C)$ is accepted by C . We define the fork advantage $\text{Adv}_{\mathcal{A}}^{\text{ForgeFork}}$ as the probability that \mathcal{A} wins.

Lemma 1 (Determinism of UPDATE). *Let r commit to map M and let M' be M updated only at key k to value v' . Then $\text{Update}(r, k, v')$ equals $\text{Com}(M')$ uniquely; any $r^* \neq \text{Com}(M')$ that verifies as an update of (r, k, v') implies a collision in H .*

Proof. SMT recomputation along the k -path is canonical; differing roots require a collision in the Merkle compressor (and thus in H). \square

5 Tripwire Theorem (Formal)

Theorem 1 (Atomic Interlock Tripwire). *Assume Σ is existentially unforgeable under chosen-message attack (EUF-CMA) and H is collision-resistant. Then for any PPT adversary \mathcal{A} ,*

$$\text{Adv}_{\mathcal{A}}^{\text{ForgeFork}} \leq \text{negl}(\lambda).$$

Proof. Suppose \mathcal{A} wins. Then w.l.o.g. we have valid receipts $\tau' = (k_j, h_j, h'_j, r, r')$ and $\tau'' = (k_j, h_j, h''_j, r, r'')$ with $h'_j \neq h''_j$ and both verify under $(\text{pk}_A, \text{pk}_B)$ and $(\text{pk}_A, \text{pk}_C)$ respectively.

By Lemma 1, r' is the unique SMT root obtained by updating the map committed by r at key k_j to value h'_j . Likewise, r'' is the unique root obtained by updating the same prior map at the same key to h''_j .

If $r' = r''$ then two different leaf values ($h'_j \neq h''_j$) yield the same SMT root under the same prior map and position, which constitutes a collision on the Merkle compression function and thus on H .

If $r' \neq r''$, both receipts must carry valid signatures by A (and respective counterparties), or else \mathcal{A} has forged an EUF-CMA signature. Therefore, producing *both* acceptable receipts reduces to either (i) a hash collision or (ii) a signature forgery, each with negligible probability.

Hence $\text{Adv}_{\mathcal{A}}^{\text{ForgeFork}}$ is negligible. \square

6 Causal Consistency and Stateless Network Integrity

Causal influence. Each countersigned receipt embeds (r, r') , inducing a DAG of roots across devices. Let \rightsquigarrow denote the smallest relation such that $r \rightsquigarrow r'$ whenever a valid receipt carries (r, r') .

Definition 4 (Causal Dependencies). *For a claimed state r_D of device D , the set $\text{Deps}(r_D)$ is the set of roots required by receipts in any path ending at r_D in the global receipt DAG (as observed locally).*

Theorem 2 (Causal Consistency Requirement). *A verifier accepts r_D iff for every (k_i, h_i) referenced by a receipt on a path into r_D , there exists π with $\text{VerifyIncl}(r_D, k_i, h_i, \pi) = 1$. Otherwise r_D is rejected locally.*

Proof sketch. Soundness follows from Merkle binding and receipt countersignatures; completeness follows from the existence of inclusion proofs along the update path. Any omission contradicts the update semantics or breaks H or Σ . \square

7 Relationship-Locality and Non-Transferability

Lemma 2 (Locality). *Given domain-separated keys $k_i = H(\text{"DSM/rel"} \parallel \text{ID}_i)$, a proof for (k_i, h) under r cannot validate for (k_j, h) with $j \neq i$ except with a collision on H .*

8 First-Contact Security

Theorem 3 (First-Contact Binding). *If an isolated device I presents its first countersigned receipt (r_I, r'_I) to any honest party, then for any future state r_I^* to be accepted, there must exist a path $r_I \rightsquigarrow r_I^*$ composed of stitched receipts. Any attempt to switch to a forked history lacks such a path and is rejected unless H or Σ is broken.*

9 Complexity

Each update requires one SMT update and two inclusion proofs: $O(\log \lambda)$ time and proof size. For $\lambda = 256$, typical proof sizes are 3–8 KiB with a sparse frontier; updates and verifications parallelize across relationships. No global broadcast is required.

10 The Cryptographic Stitching Paradigm

10.1 Device-Level Meta-Commitments: The Atomic Interlock

Each device maintains k independent bilateral chains $\mathcal{C}_1, \dots, \mathcal{C}_k$. At any moment it computes a single SMT root

$$r = \text{SMT}(h_1, \dots, h_k), \quad h_i = H(\text{head of } \mathcal{C}_i),$$

thereby “stitching” all relationships into one atomic commitment.

10.2 The Tripwire Theorem: Mathematical Impossibility of Double-Spend

Theorem (Atomic Interlock Tripwire: Informal). *Given a collision-resistant hash H , a device cannot generate two SMT roots (r, r') at the same logical height that differ in any relationship head without permanent detection by at least one honest counter-party.*

Proof. (Informal sketch — full reduction appears in Section 11.) Forking a single relationship forces the device to publish inconsistent roots $r \neq r'$. Any honest peer verifying inclusion proofs across relationships will see that no single SMT root can satisfy both forks simultaneously without violating collision resistance of H . Hence the attacker irrevocably “bricks” itself. \square

10.3 Understanding the Tripwire Mechanism

11 Formal Foundations and Complete Proof of the Tripwire Theorem

11.1 Preliminaries

Definition 5 (Hash Function). $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ is a collision-resistant hash with security parameter λ .

Definition 6 (Straight Hash Chains). *For relationship i a device stores $\mathcal{C}_i = (s_0, \dots, s_k)$ with $s_{j+1} = H(s_j \parallel \text{payload}_{j+1})$ and head $h_i := H(s_k)$.*

Definition 7 (Device Commitment). Given $\{h_1, \dots, h_k\}$ the device’s global state is

$$r := \text{SMT}(h_1, \dots, h_k).$$

Definition 8 (DBRW Fingerprint [2]). $\text{DBRW}()$ is the Dual-Binding Random Walk hardware fingerprint; distinct hardware yields distinct outputs except with negligible probability.

11.2 Adversary & Network Model

- **Adversary \mathcal{A} :** PPT, controls network scheduling, may corrupt arbitrary devices but *not* their DBRW fingerprints or SPHINCS⁺ keys.
- **Communication:** Eventually-synchronous. Honest devices eventually compare commitments once connectivity returns.

Definition 9 (Double-Spend Game \mathbf{G}_{DS}). *i) Setup: Challenger initialises an honest device D with commitment r_0 .*

ii) Challenge: \mathcal{A} outputs two distinct commitments (r', r'') at the same logical height τ .

iii) Win: Both commitments are accepted by two honest counterparties B and C .

Define $\text{Adv}_{\text{DS}}(\lambda) := \Pr[\mathbf{G}_{\text{DS}} \text{ wins}]$.

11.3 Formal Tripwire Theorem

Theorem 4 (Tripwire—Reduction Form). For any PPT adversary \mathcal{A} ,

$$\text{Adv}_{\text{DS}}(\lambda) \leq \text{Adv}_{\text{CRH}}(\lambda) + 2^{-\lambda},$$

where Adv_{CRH} is the collision advantage for H .

Proof. Assume \mathcal{A} wins Definition 9 with probability ε . We build a reduction \mathcal{R} that breaks collision resistance:

1. \mathcal{R} runs \mathcal{A} , perfectly emulating honest devices, DBRW genesis, SMT logic, etc.
2. On output (r', r'') at height τ , there exists an index j with heads $h'_j \neq h''_j$.
3. \mathcal{R} extracts the two Merkle proof paths (supplied by \mathcal{A} for verification). Paths terminate in the *same* parent hash yet contain $h'_j \neq h''_j$, so

$$H(h'_j) = H(h''_j)$$

is a collision in H .

Hence \mathcal{R} breaks H with probability $\varepsilon - 2^{-\lambda}$, contradicting collision resistance. Therefore ε is negligible. \square

11.4 Liveness and Causal Finality

Every honest device stores the last-seen commitment from each peer. After a partition, a single inclusion-proof exchange suffices to detect forks; a forked device cannot craft a new root acceptable to both branches without the forbidden collision of Theorem 4.

11.5 First-Contact Binding

The first online interaction of an isolated device irrevocably binds it to a single hash-chain branch (DBRW uniqueness) and all subsequent forks violate Theorem 4.

11.6 Polynomial-Time Verifiability

For any transition T ,

$$\text{Time}(\text{Verify}(T)) \leq c|T|^k,$$

for constants c, k , ensuring DoS-resilience even on mobile hardware.

References

- [1] Ramsay, B. “DSM: Decentralized State Machine—The Missing Trust Layer of the Internet,” *Cryptology ePrint Archive*, Paper 2025/592.
- [2] Ramsay, B. “DBRW: Dual-Binding Random Walk for Anti-Cloning,” 2025. Available at <https://decentralizedstatemachine.com>.
- [3] Dahlberg, R.; Pulls, T.; Peeters, R. “Efficient Sparse Merkle Trees,” LNCS, 2016.