Chapter 3 – Tamper-Proof Ledger at Scale

"2 000 gifts $\neq$ 2 000 MB audit."

ndhy

July 2, 2025

Integrity must stay compact.

Outline

1. Hash refresher: collisions and avalanche effect
2. Merkle-tree theory and construction
3. Brief note: Poseidon and other zk-friendly hashes
4. Full chain: leaf hash $\rightarrow$ Merkle root $\rightarrow$ signature
5. Hands-on lab: verify an inclusion proof

Hash Function Essentials

- **Collision resistance**: hard to find $m \neq m'$ with $\mathrm{H}(m) = \mathrm{H}(m')$.
- **Second-preimage**: given m, hard to find m' with the same hash.
- **Avalanche**: flipping one input bit changes roughly half the output bits.

SHA-256 Flip-Bit Demo

```
# Original
echo -n "gift123" | openssl dgst -sha256
#-> 30f8d5...b1e1

# Flip one bit: 'Gift123'
echo -n "Gift123" | openssl dgst -sha256
#-> 4c2e97...a47c (completely different)
```

A 7-byte string produces a 32-byte fingerprint that detects any tweak.

Why a Tree Beats a Flat List

Goal : prove inclusion of one row out of 2 000 without sending 2 000 hashes.

A Merkle tree compresses n leaf hashes into a single *root*. An inclusion proof needs only $\log_2 n$ sibling hashes (2 000 leaves $\rightarrow$ 11 siblings).

Path Construction and Verification

- Leaf index in binary tells which sibling hash to include at each level.
- Verifier re-hashes upward: $\mathrm{H}(\mathrm{left} \,\|\, \mathrm{right})$ until the root is reached.
- Proof length grows logarithmically; bandwidth and verification stay small.

Security of Merkle Proofs

- If the underlying hash is collision-resistant, altering one leaf without changing the root would require a collision.
- Binding therefore reduces to hash security; no additional assumptions.
- Widely deployed in Bitcoin (block headers), Git, certificate transparency.

Brief Note: Poseidon and Friends

- Bit-oriented hashes (SHA-256, Blake2) are costly in finite-field proofs.
- Poseidon, Rescue, and MiMC are designed for $\mathbb{F}_p$ arithmetic, reducing constraint count.
- In this course we keep Poseidon as a drop-in replacement where zk proofs matter; details are left to optional reading.

Bringing the Pieces Together

1. Hash each gift row $\Rightarrow$ 256-bit leaf.
2. Build a Merkle root over all 2 000 leaves.
3. Sign the root with ECDSA (Chapter 2).
4. Distribute $\langle \text{root}, \sigma \rangle$ once per day.
5. Anyone verifies the signature, then verifies inclusion of their row with a logarithmic-size proof.

Lab — Verify an Inclusion Proof

Files:

1. `merkle_root.txt` – 32-byte hex
2. `proof.json` – sibling hashes (hex)
3. `my_gift.txt` – your plaintext row

Run:

```
python verify.py my_gift.txt proof.json merkle_root.txt
# expected: inclusion proof valid
```

Hint: re-hash upward and compare with the published root.

Cheat-Sheet

- Hashes give fixed-size fingerprints with collision resistance.
- Merkle trees compress n hashes into one root; proofs cost $O(\log n)$.
- Root signed with ECDSA makes the ledger tamper-evident.
- Poseidon (optional) is zk-friendly if you need proof systems later.

Questions?