ShareLock: Tamper-Localising Threshold Secret Sharing over $\mathrm{GF}(2^8)$

Tingqiu Liu · October 2026 · Mathematics for Systems Security hackathon

Threshold secret sharing distributes a secret so that any t of n shares recover it and any $t-1$ reveal nothing. Classical schemes guarantee confidentiality but say nothing about integrity: a shareholder who receives a corrupted share learns only that reconstruction failed, not who corrupted it. Verifiable variants (Feldman, Pedersen) detect inconsistency at the cost of a discrete-logarithm group and still cannot identify the culprit. ShareLock keeps the scheme in the byte-oriented field $\mathrm{GF}(2^8)$, commits to every share under a Merkle root bound to a session identifier, and thereby (i) verifies any single share in $O(\log n)$ hashes, (ii) names the exact indices of corrupted shares, and (iii) refuses shares replayed from another session.

1. Problem

A secret $s \in \mathrm{GF}(2^8)^k$ is split among n holders with threshold t. Three properties are wanted:

- **Confidentiality:** any set of fewer than t shares yields no information about s.
- **Reconstructability:** any t shares determine s.
- **Accountability:** if some shares are corrupted, the receiver learns *which*.

Shamir's construction gives the first two. The third is what this work adds without leaving the byte field.

2. Preliminaries

2.1 The field

$\mathrm{GF}(2^8)$ is realised with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ (the AES modulus). Addition is bitwise XOR; multiplication uses logarithm tables generated by the primitive element 3. Every non-zero element is invertible, so Lagrange interpolation is available at any set of distinct points.

2.2 Shamir sharing

For each byte s_j of the secret the dealer samples $a_{j,1}, \dots, a_{j,t-1} \leftarrow \mathrm{GF}(2^8)$ uniformly and sets $f_j(x) = s_j + \sum_{i=1}^{t-1} a_{j,i} x^i$. Share i is $\big(x_i, (f_1(x_i), \dots, f_k(x_i))\big)$ for distinct non-zero x_i. Reconstruction uses $\lambda_i = \prod_{j \neq i} \frac{x_j}{x_j - x_i}$ and computes $s = \sum_i \lambda_i y_i$.

3. Construction

Let H be SHA-256 and σ a 128-bit session identifier.

Definition 1 (Share commitment). For share i with coordinate x_i and payload y_i,

$$L_i = H\big(\texttt{"sharelock/v1/share"} \,\|\, |\sigma| \,\|\, \sigma \,\|\, i \,\|\, x_i \,\|\, |y_i| \,\|\, y_i\big).$$

The Merkle root R over $(L_1, \dots, L_n)$ with node function $N(u, v) = H(\texttt{"sharelock/v1/node"} \| u \| v)$ is published together with σ.

Definition 2 (Verification). A share i is *valid* iff recomputing L_i and folding it with its Merkle path reproduces R under the published σ.

4. Security analysis

Theorem 1 (Confidentiality). Let $S \subset \{1, \dots, n\}$ with $|S| \le t - 1$, and let $(x_i, y_i)_{i \in S}$ be observed. For every candidate secret $s' \in \mathrm{GF}(2^8)^k$ there exists a polynomial vector of degree $\le t - 1$ consistent with the observations and with $f(0) = s'$. Consequently the posterior distribution of s given the observations equals its prior.

Proof. Fix a byte position. Let g be the unique polynomial of degree $\le |S| - 1 \le t - 2$ interpolating the observed points, and $h(x) = \prod_{i \in S}(x - x_i)$. Set $f(x) = g(x) + c\,h(x)$ with $c = (s' - g(0)) \cdot h(0)^{-1}$. Since $h(0) \neq 0$ (all x_i are non-zero and distinct), c is well defined. For $i \in S$, $h(x_i) = 0$ hence $f(x_i) = g(x_i) = y_i$; at zero, $f(0) = g(0) + c\,h(0) = s'$. The degree of f is at most $t - 1$. Repeating independently per byte gives a consistent sharing of s'. As the dealer's coefficients are uniform and independent, every s' arises with equal probability, so the observations carry zero information about s. ■

Theorem 2 (Integrity and localisation). Assume H is collision resistant. If share i is modified after commitment, verification of share i fails, and verification of every unmodified share $j \neq i$ succeeds.

Proof. A modification to y_i or x_i changes the preimage of L_i; collision resistance implies the recomputed leaf differs from the committed one, and the Merkle fold is a deterministic function of the leaf and its path, so it cannot reproduce R without a collision. For $j \neq i$ the leaf L_j is untouched and its path contains only siblings outside j's subtree, none of which is L_i, so the fold still yields R. Hence the failing index set is exactly $\{i\}$. ■

Theorem 3 (Session binding). A share committed under σ fails verification under any $\sigma' \neq \sigma$.

Proof. σ is part of every leaf preimage and is compared against the commitment before hashing, in constant time. For $\sigma' \neq \sigma$ the recomputed leaf differs from the committed leaf, and the fold cannot reach R without a hash collision. ■

5. Attack surface

- **Dealer trust.** A malicious dealer can commit to a wrong sharing and still publish a valid root. ShareLock detects transport and storage corruption, not a lying dealer; that requires a commitment to the secret itself and is out of scope.
- **Root authenticity.** R must reach verifiers over an authenticated channel, otherwise an adversary substitutes a root for shares of its own choosing.
- **Confidentiality of shares.** A hash commits without hiding. Shares still require a confidential channel; this scheme does not replace one.
- **Timing.** Session comparison uses `hmac.compare_digest`; the Merkle fold is data independent, so verification time does not depend on how close a forgery came.
- **Share reuse.** Shares are bound to (σ, i); replaying a share from another session fails the binding check.

6. Comparison

Scheme	Field	Verifiable	Localises	Group assumption
Shamir [1]	any field	no	no	none
Feldman VSS [2]	$\mathbb{Z}_p$	yes	no	discrete log
Pedersen VSS [3]	$\mathbb{Z}_p$	yes	no	discrete log
ShareLock	$\mathrm{GF}(2^8)$	yes	**yes**	none (hash only)

Feldman and Pedersen check a linear relation over a group; one bad share breaks the relation for the whole set without revealing which share caused it. ShareLock commits per share, so the failing index is recovered in $O(\log n)$ hashes per shareholder and no group is required — which matters on constrained devices where byte-field arithmetic is already available through AES hardware.

7. Implementation and measurements

Reference implementation in Python 3.13 (`sharelock/`), 14 tests, no third-party dependencies. Secret length 32 bytes, 20 repetitions:

(t, n)	share bytes	expansion	split p50	combine p50	locate p50
(2,3)	99	3.09x	0.012 ms	0.008 ms	0.009 ms
(3,5)	165	5.16x	0.037 ms	0.013 ms	0.021 ms
(5,5)	165	5.16x	0.058 ms	0.019 ms	0.026 ms
(4,10)	330	10.3x	0.071 ms	0.017 ms	0.043 ms

Verification of one share costs 0.003–0.004 ms with a proof of $\log_2 n \cdot 32$ bytes (64 bytes at $n = 3$, 96 at $n = 5$). Locating every corrupted share is a single pass over the set.

8. Limitations

- Storage overhead is inherent: shares total $n \cdot k$ bytes plus one coordinate byte each; the threshold cannot be met with less.
- A malicious dealer is not detected; the root commits to the sharing, not to the secret.
- Shares are not hidden by the commitment; a confidential channel is still required.
- Constant-time behaviour is argued for the comparison and the fold, but this Python implementation is not a side-channel-hardened artefact.

References

[1] A. Shamir, *How to share a secret*, Communications of the ACM 22(11), 1979.
[2] P. Feldman, *A practical scheme for non-interactive verifiable secret sharing*, FOCS 1987.
[3] T. Pedersen, *Non-interactive and information-theoretic secure verifiable secret sharing*, CRYPTO 1991.