SAKSI: Witness-Validated Secret Sharing with Unconditional Cheater Identification over GF($2^{127} - 1$)

Shermaine Yap Shi Min
Asia Pacific University of Technology and Innovation (APU), Kuala Lumpur, Malaysia

November 2026

Abstract

A master key that exists in one place is one compromise away from total loss, and a key that is split with plain Shamir sharing can be silently corrupted: a single dishonest share-holder can make everyone else reconstruct garbage while learning the real key itself (the Tompa–Woll attack). We present **SAKSI** (Malay: *witness*), a (t, n) secret-sharing scheme for master keys over the Mersenne prime field $\mathbb{F}_p$, $p = 2^{127} - 1$, that *validates* every share before it is used. Each share carries one-time affine *witness tags*, one per other custodian, bound to a public context (session, epoch, index) and checked in a two-round reveal; a generalised Reed–Solomon syndrome independently cross-checks the accepted shares and Gao decoding localises any residual inconsistency, including a dishonest dealer. Secrecy is perfect, a forged share passes a single witness with probability exactly $1/p = 2^{-127}$, and with at most $\lfloor (n-1)/2 \rfloor$ cheaters the correct key is returned and every modified share is named, except with probability at most $2^n/p$. No block cipher, hash function or computational assumption is used anywhere, so the guarantees hold against computationally unbounded (including quantum) adversaries. The core is written in branch-free Rust; a dudect leakage study over up to 1,000,000 measurements finds no timing dependence on secrets ($|t| \leq 1.11$), while deliberately leaky controls are flagged ($|t| \geq 192$). An attack harness of eight experiments, including Z3-assisted collusion and active tampering on the real parameters, is published with the code, and the scheme is deployed in a browser agent that can use an API credential only after validated reconstruction.

1 Introduction

Threshold secret sharing [12, 2] removes the single point of failure of a master key: the key is encoded into n shares such that any t of them recover it and any $t-1$ reveal nothing. Plain Shamir sharing, however, gives no way to tell a correct share from a wrong one. Tompa and Woll [13] showed that the last participant to submit can choose a share so that the others reconstruct a wrong value while the cheater, knowing what it submitted, recovers the true secret. Data corruption, a buggy custodian or a malicious insider all look the same to Lagrange interpolation: it always returns *some* key.

This matters for key custody in software agents. An AI agent that needs a credential should never hold it permanently, and the provider of the language model it talks to should never see it. Splitting the credential across custodians solves the first problem only if reconstruction is *validated*: a corrupted share must be detected and attributed, and the system must fail closed rather than use a wrong key.

What is from the literature. Polynomial sharing over a field is Shamir's [12]. Information-theoretic "check vectors" that let one player authenticate another's share are due to Rabin and Ben-Or [9]; robust sharing with short tags and the role of rushing adversaries are analysed by Cevallos et al. [3]. The relation between Shamir shares and Reed–Solomon codewords and

Gao's decoder are classical [5]. Proactive refresh by adding a sharing of zero is due to Herzberg et al. [6]. Computational verifiable secret sharing (Feldman [4], Pedersen [8]) and CRT-based sharing (Mignotte [7], Asmuth–Bloom [1]) are the main alternatives.

Contributions.

1. **A concrete, fully specified construction** combining (i) Vandermonde/Shamir encoding over the Mersenne field $2^{127}-1$ in 120-bit limbs, (ii) *context-bound inner-product witness tags* $\tau_{j,i} = \beta + \alpha_0 \mathsf{ctx}_i + \sum_\ell \alpha_\ell y_{i,\ell}$ that authenticate the whole multi-limb share *and* its session, epoch and index with a single field element, (iii) a two-round reveal with a $\lfloor (n-1)/2 \rfloor$ vouching rule, and (iv) a parity-check (syndrome) cross-check with Gao decoding, plus a keyless Reed–Solomon fallback.
2. **Proofs** of perfect secrecy including the tags, an exact forgery bound, robustness with cheater identification, and the necessity of the two-round order (Section 4).
3. **A constant-time Rust implementation** of the Mersenne arithmetic with a published dudect-style leakage study and positive controls (Section 7).
4. **An open attack harness** (eight experiments) that attacks SAKSI next to unvalidated controls (Section 6).
5. **Deployment** in a browser agent that custodies an API credential (Section 9).

The individual ingredients are textbook; the contribution is a careful, minimal combination specified down to the wire format, with proofs that match the code line by line and measurements that test the claims.

2 Preliminaries and threat model

Notation. $p = 2^{127}-1$ is prime and $\mathbb{F}_p$ is the field of integers modulo p. Custodians are indexed $i \in [n] = \{1, \dots, n\}$ and custodian i is associated with the public evaluation point $x_i = i$. A secret S of λ bytes is split into $L = \lceil \lambda/15 \rceil$ limbs $s_1, \dots, s_L \in [0, 2^{120}) \subset \mathbb{F}_p$ (big-endian, zero padded). The public header is $H = (n, t, L, \lambda, \mathsf{sid}, e)$ with a uniformly random session identifier $\mathsf{sid} \in \{0, \dots, 2^{53}-1\}$ (53 bits so that it round-trips exactly through JSON and JavaScript numbers in the browser demo) and epoch e. The public context of share i is the field element

$$\mathsf{ctx}_i \;=\; \mathsf{sid} \;+\; 2^{64} e \;+\; 2^{96} i \qquad (\mathsf{sid} < 2^{53} \le 2^{64},\; e < 2^{32},\; i < 2^{16}),$$

an injective encoding of (sid, e, i) into $[0, 2^{112}) \subset \mathbb{F}_p$.

Parties and channels. A dealer runs Deal once per epoch and is trusted for that epoch (as in Shamir sharing). Reconstruction is performed by a combiner that receives messages from the custodians over *authenticated* channels (the combiner knows which custodian sent which message) and enforces two ordered rounds: no key from round 2 is released before every round-1 message has been fixed. If the channel is not authenticated and two shares arrive under the same index (a cheater racing an honest custodian), the combiner keeps the candidate with the most witness confirmations; a forgery under a stolen index therefore cannot displace the honest share (tested in both implementations).

Adversary. A static adversary $\mathcal{A}$ corrupts a set $C \subseteq [n]$ of $c = |C|$ custodians before dealing, sees their entire state, and controls everything they send. It may be *rushing* within a round (it sees honest messages of the current round before choosing its own), but cannot change a round-1 message after round 2 starts. It is computationally unbounded. Robustness requires $c \le T := \lfloor (n-1)/2 \rfloor$ and $t \le n - c$; privacy requires $c \le t - 1$.

Definition 1 (Properties)**. Privacy:** the joint view of any $c \le t-1$ custodians before reconstruction is independent of S. **Correctness:** if all custodians are honest, reconstruction from any t of them outputs S. **Robustness:** with at most T corrupted custodians, reconstruction outputs S or fails closed ($\perp$), and outputs S whenever at least t honest custodians respond. **Identification:** every corrupted custodian whose round-1 share differs from the dealt share is reported.

3 The scheme

3.1 Algorithms

$\mathsf{Deal}(S, n, t)$

1: **assert** $2 \le t \le n,\ n \ge 3,\ S \ne \varepsilon$
2: $(s_1, \ldots, s_L) \leftarrow \mathsf{Limbs}_{15}(S)$; $\mathsf{sid} \xleftarrow{\$} \{0, \ldots, 2^{53}-1\}$; $e \leftarrow 0$
3: **for** $\ell = 1..L$ **do**
4: $\quad a_{\ell,0} \leftarrow s_\ell$; $a_{\ell,1}, \ldots, a_{\ell,t-1} \xleftarrow{\$} \mathbb{F}_p$
5: **for** $i = 1..n$ **do**
6: $\quad y_{i,\ell} \leftarrow \sum_{k=0}^{t-1} a_{\ell,k}\, i^k$ (i.e. $\mathbf{y}_\ell = V\mathbf{a}_\ell$)
7: **return** $\mathsf{Witness}(H, \{\mathbf{y}_i\}_{i\in[n]})$

$\mathsf{Witness}(H, \{\mathbf{y}_i\})$

1: **for** ordered $(j, i),\ j \ne i$ **do**
2: $\quad \kappa_{j,i} = (\alpha_0, \ldots, \alpha_L, \beta) \xleftarrow{\$} \mathbb{F}_p^{L+2}$
3: $\quad \tau_{j,i} \leftarrow \beta + \alpha_0 \mathsf{ctx}_i + \sum_{\ell=1}^{L} \alpha_\ell y_{i,\ell}$
4: custodian i gets $\big(\mathbf{y}_i, \{\tau_{j,i}\}_{j\ne i}\big)$
5: custodian j gets $\{\kappa_{j,i}\}_{i\ne j}$
6: **return** all custodian states

Figure 1: Dealing. V is the $n \times t$ Vandermonde matrix $V_{ik} = i^k$. All randomness comes from the operating-system CSPRNG; field elements are sampled by rejection from 127 random bits (rejecting only the value p).

Syndrome. For a set $A = \{x_1, \ldots, x_m\}$ of distinct points and values $\mathbf{y} \in \mathbb{F}_p^m$ define

$$\mathrm{Syn}_A(\mathbf{y})_k \;=\; \sum_{i=1}^{m} v_i\, x_i^k\, y_i \quad (k = 0, \ldots, m-t-1), \qquad v_i = \prod_{j\ne i} (x_i - x_j)^{-1}. \tag{1}$$

This is the parity-check matrix $H_A = (v_i x_i^k)$ of the generalised Reed–Solomon code $\mathcal{C}_A = \{(f(x_1), \ldots, f(x_m)) : \deg f < t\}$; see Lemma 4.

Refresh. $\mathsf{Refresh}$ samples $z_\ell(X) = \sum_{k=1}^{t-1} b_{\ell,k} X^k$ (so $z_\ell(0) = 0$), sets $y'_{i,\ell} = y_{i,\ell} + z_\ell(i)$, increments the epoch $e \leftarrow e+1$ and calls $\mathsf{Witness}$ again with fresh one-time keys.

Encoding sanity check. An honest reconstruction always produces limbs below 2^{120} with zero padding. If a wrong limb appears (possible only beyond the robustness bound), it lies in $[0, 2^{120})$ with probability $2^{120}/p \approx 2^{-7}$, so the check rejects it with probability about $1 - 2^{-7}$ per limb. This is a heuristic extra layer; none of the proofs rely on it.

$\mathsf{Reconstruct}(H)$

Round 1: each responding i sends $(i, \mathbf{y}_i, \{\tau_{j,i}\}_j)$; drop malformed shares and $y \notin [0, p)$
if several shares claim index i, keep the one with most votes (after round 2)
Round 2: each responding j sends $\{\kappa_{j,i}\}_i$
for each received share i **do**
  $\mathsf{votes}_i \leftarrow \#\{j \neq i : \tau_{j,i} \overset{\text{ct}}{=} \beta_{j,i} + \alpha_{j,i,0}\mathsf{ctx}_i + \sum_\ell \alpha_{j,i,\ell}\, y_{i,\ell}\}$
$A \leftarrow \{i : \mathsf{votes}_i \geq T\}$; report $[n] \cap \{\text{received}\} \setminus A$ as rejected
if $|A| < t$ **then return** $\perp$
if $\mathrm{Syn}_A(\mathbf{y}_\ell) = \mathbf{0}$ for all ℓ **then** $s_\ell \leftarrow \sum_{i \in A_t} \lambda_i y_{i,\ell}$ (A_t = first t of A)
else $(f_\ell, E_\ell) \leftarrow \mathsf{Gao}(A, \mathbf{y}_\ell, t)$ for all ℓ; **if** any fails **then return** $\perp$
  $s_\ell \leftarrow f_\ell(0)$; report $\cup_\ell E_\ell$ as located; **if** $|A \setminus \cup E_\ell| < t$ **then return** $\perp$
if any $s_\ell \geq 2^{120}$ or padding $\neq 0$ **then return** $\perp$ (encoding sanity check)
return $S = \mathsf{Bytes}_\lambda(s_1, \ldots, s_L)$

Figure 2: Two-round reconstruction. $T = \lfloor (n-1)/2 \rfloor$; $\lambda_i = \prod_{j \neq i} x_j/(x_j - x_i)$ are the Lagrange coefficients at 0; Syn is defined in Equation (1). $\mathsf{Reconstruct}_{\text{keyless}}$ skips the votes and runs Gao on all well-formed shares.

3.2 Worked example (toy parameters)

Take $p = 97$, $n = 5$, $t = 3$, a single limb $s = 42$ and coefficients $a_1 = 11$, $a_2 = 7$, i.e. $f(X) = 42 + 11X + 7X^2$. The shares are

$$(y_1, \ldots, y_5) = (60,\ 92,\ 41,\ 4,\ 78).$$

Reconstruction. For $A_t = \{1, 2, 3\}$ the Lagrange coefficients at 0 are $(\lambda_1, \lambda_2, \lambda_3) = (3, 94, 1)$ and $3 \cdot 60 + 94 \cdot 92 + 1 \cdot 41 \equiv 42 \pmod{97}$. *Syndrome.* For all five points $v = (93, 16, 73, 16, 93)$ and $\mathrm{Syn}(\mathbf{y}) = (0, 0)$. If custodian 4 changes y_4 to 9, the syndrome becomes $(80, 29) \neq \mathbf{0}$, and Gao decoding (which corrects $\lfloor (5-3)/2 \rfloor = 1$ error) returns f and locates index 4. *Witness.* With toy key $(\alpha_0, \alpha_1, \beta) = (23, 58, 64)$ and toy context $\mathsf{ctx}_2 = 2$, the tag of share 2 is $\tau = 64 + 23 \cdot 2 + 58 \cdot 92 \equiv 14$. The shifted share $y_2 + 1 = 93$ evaluates to $72 \neq 14$ (the difference is exactly $\alpha_1 = 58$, which the cheater does not know). *Tompa–Woll on plain Shamir.* If custodian 3 adds 1 to its share, plain interpolation over $\{1, 2, 3\}$ returns 43 instead of 42; the cheater knows $\lambda_3 = 1$ and therefore the true key.

3.3 Parameters and cost

For a 32-byte key ($L = 3$) and $n = 5$, each custodian stores its share (L elements), $n - 1$ tags and $n - 1$ keys of $L + 2$ elements: $3 + 4 + 4 \cdot 5 = 27$ field elements, i.e. 432 bytes. Key material is $\Theta(n^2 L)$ in total, which is the price of unconditional validation (Section 10).

4 Security analysis

Theorem 1 (Perfect secrecy, including tags)**.** *For every coalition C with $|C| \leq t - 1$ and every two secrets S, S' of equal length, the joint view of C after* Deal *(shares, tags, keys and header) has the same distribution under S and S'.*

Proof. Limbs are shared with independent polynomials, so it suffices to treat one limb. Fix s and the values $(y_i)_{i \in C}$. The non-constant coefficients $(a_1, \ldots, a_{t-1})$ must satisfy $\sum_{k=1}^{t-1} a_k x_i^k = y_i - s$ for $i \in C$. The coefficient matrix $(x_i^k)_{i \in C,\, 1 \leq k \leq t-1} = \mathrm{diag}(x_i) \cdot (x_i^{k-1})$ has full row rank $|C|$ because the x_i are distinct and non-zero (a Vandermonde matrix times an invertible diagonal). Hence

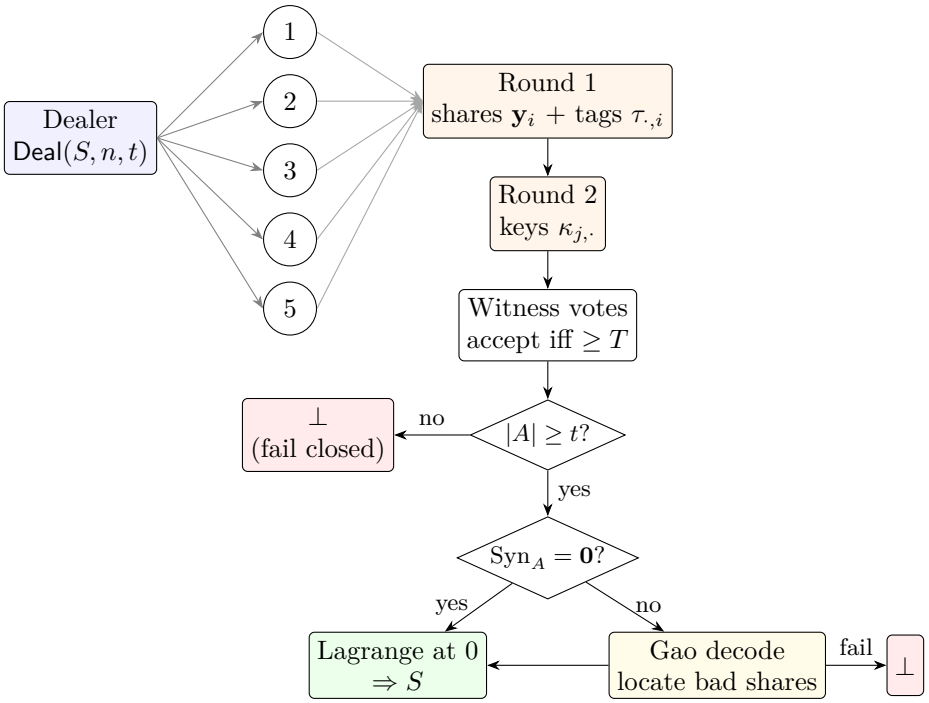


Figure 3: Protocol flow of SAKSI for $n = 5$. Rejected and located custodians are reported by index.

exactly $p^{t-1-|C|}$ coefficient vectors are consistent with the view, independently of s: the shares of C are uniform on $\mathbb{F}_p^{|C|}$ and independent of s. The keys held by C are sampled independently of everything else. A tag $\tau_{j,i}$ with $i \in C$, $j \notin C$ equals $\beta_{j,i} + \langle \alpha_{j,i}, m_i \rangle$ where $\beta_{j,i}$ is uniform and independent of everything in the view except $\tau_{j,i}$ itself; so these tags are independent uniform field elements. Tags $\tau_{j,i}$ with $i, j \in C$ are functions of values already in the view. The header depends only on public parameters, λ and fresh randomness. The joint view is therefore identically distributed for S and S'. □

1 : **Game** $\mathsf{Forge}_{\mathcal{A}}(L)$
2 : $(\alpha_0, \ldots, \alpha_L, \beta) \xleftarrow{\$} \mathbb{F}_p^{L+2}$
3 : $(m, \mathsf{st}) \leftarrow \mathcal{A}_1()$ // $m \in \mathbb{F}_p^{L+1}$, chosen first
4 : $\tau \leftarrow \beta + \langle \alpha, m \rangle$
5 : $(m', \tau') \leftarrow \mathcal{A}_2(\mathsf{st}, \tau)$
6 : **return** $m' \neq m \wedge \tau' = \beta + \langle \alpha, m' \rangle$

Figure 4: Forgery game for one witness key. In SAKSI $m = (\mathsf{ctx}_i, y_{i,1}, \ldots, y_{i,L})$; the cheater i holds $\tau_{j,i}$ but never $\kappa_{j,i}$ before round 1 is fixed.

Theorem 2 (Witness forgery)**.** *For every (unbounded) $\mathcal{A}$,* $\Pr[\mathsf{Forge}_{\mathcal{A}}(L) = 1] \leq 1/p$*, with equality for the best strategy.*

Proof. Write $d = m' - m \neq \mathbf{0}$. The forgery succeeds iff $\tau' - \tau = \langle \alpha, d \rangle$. Given m and τ, the conditional distribution of α is still uniform on $\mathbb{F}_p^{L+1}$, because τ only fixes $\beta = \tau - \langle \alpha, m \rangle$ and β is uniform and independent of α. Since $d \neq \mathbf{0}$, $\alpha \mapsto \langle \alpha, d \rangle$ is a surjective linear map, so $\langle \alpha, d \rangle$ is uniform on $\mathbb{F}_p$ and independent of (m, m', τ, τ'), which the adversary fixes without further information about α. Hence the success probability is exactly $1/p$ whatever τ' is chosen. □

Because ctx_i is part of m, a valid share replayed from another session or epoch, or relabelled with another index, is a forgery attempt with $m' \neq m$; it is also checked under a different, independent key. Either way Theorem 2 applies.

Theorem 3 (Robustness and identification)**.** *Let $c = |C| \leq T = \lfloor (n-1)/2 \rfloor$, let $k = T - c + 1$ and assume the two rounds are ordered. Then:*

(a) every honest custodian's share is accepted;

(b) a share $\mathbf{y}'_i \neq \mathbf{y}_i$ submitted for a corrupted i is accepted with probability at most $\binom{n-c}{k} p^{-k} \leq 2^n/p$;

(c) consequently, if at least t honest custodians respond, Reconstruct *outputs S and names every modified share, except with probability at most $c \cdot 2^n/p$; in every case it never outputs a value other than S unless a modified share was accepted.*

Proof. (a) An honest share i is checked by the $n-1-c$ honest other custodians, who publish the correct keys; each check succeeds deterministically. Since $2T \leq n-1$ and $c \leq T$, $n-1-c \geq n-1-T \geq T$, so $\mathsf{votes}_i \geq T$.

(b) A modified share of $i \in C$ can collect at most $c-1$ votes from the other corrupted custodians (they may publish arbitrary keys). It therefore needs at least $T-(c-1) = k \geq 1$ matching honest keys. For each honest j, matching $\tau'_{j,i}$ is an instance of Figure 4: by the ordering, the adversary fixes $(\mathbf{y}'_i, \tau'_{\cdot,i})$ before any honest key is revealed, and its view contains $\tau_{j,i}$ but no information about $\kappa_{j,i}$ beyond it. The keys $\kappa_{j,i}$ of distinct honest j are independent, so the events are independent given the adversary's choices, each with probability $1/p$. The probability that some k-subset of the $n-c$ honest keys all match is at most $\binom{n-c}{k} p^{-k} \leq 2^n p^{-1}$.

(c) By (a) all honest responding shares are in A; by (b) and a union bound over the c corrupted custodians, all modified shares are outside A except with probability $c \cdot 2^n/p$. Then every share in A is a correct share, so $\mathrm{Syn}_A = \mathbf{0}$ (Lemma 4), $|A| \geq t$, and Lagrange interpolation outputs S. Shares outside A are reported. If a modified share does enter A, the syndrome is non-zero and Gao decoding either removes it (Theorem 5) or the algorithm fails closed. □

For the demonstration parameters $n = 5$, $c \leq 2$, the bound in (c) is at most $2 \cdot 32/p < 2^{-120}$.

Lemma 4 (Syndrome)**.** *For distinct $x_1, \ldots, x_m$ and $m \geq t$, $\mathrm{Syn}_A(\mathbf{y}) = \mathbf{0}$ if and only if there is f with $\deg f < t$ and $f(x_i) = y_i$ for all i.*

Proof. For any g with $\deg g \leq m-1$, the Lagrange form $g(X) = \sum_i g(x_i) \prod_{j \neq i} (X - x_j)/(x_i - x_j)$ shows that $\sum_i v_i g(x_i)$ is the coefficient of X^{m-1} in g. ($\Rightarrow$ direction, codewords are in the kernel.) If $y_i = f(x_i)$ with $\deg f < t$ then for $k \leq m-t-1$ the polynomial $X^k f$ has degree $\leq m-2$, so $\mathrm{Syn}_A(\mathbf{y})_k = 0$. (Kernel equals code.) $H_A = (v_i x_i^k)$ is an $(m-t) \times m$ Vandermonde matrix times the invertible diagonal $\mathrm{diag}(v_i)$, so it has rank $m-t$ and its kernel has dimension t. The code is the image of the injective map $\mathbf{a} \mapsto V\mathbf{a}$ and has dimension t; it is contained in the kernel, hence equal to it. □

Theorem 5 (Keyless decoding)**.** *Let $m \geq t$ shares be received, e of them wrong. If $e \leq \lfloor (m-t)/2 \rfloor$, Gao decoding returns the dealt polynomial and exactly the e wrong indices. If $1 \leq e \leq m-t$, the syndrome is non-zero, so the corruption is detected.*

Proof. The shares form a codeword of a Reed–Solomon code of length m, dimension t and minimum distance $d = m - t + 1$ (two distinct polynomials of degree $< t$ agree on at most $t-1$ points). Gao's algorithm [5] returns the unique codeword within distance $\lfloor (d-1)/2 \rfloor = \lfloor (m-t)/2 \rfloor$; the wrong indices are the points where it disagrees with the received word. If $1 \leq e \leq d-1$, the received word is not a codeword, so its syndrome is non-zero by Lemma 4. □

Beyond the correction radius the decoder can fail (SAKSI then returns $\perp$) or, if $e \geq d - \lfloor (d-1)/2 \rfloor$, return a different codeword. The keyless mode therefore guarantees correctness only inside the radius. The keyed mode does not depend on decoding for its main guarantee (Theorem 3); it uses decoding only to correct a dishonest dealer.

Proposition 6 (Refresh)**.** Refresh *preserves the secret, and a share from epoch e presented in epoch $e+1$ is accepted with probability at most $\binom{n-c}{k}p^{-k}$.*

Proof. $(f_\ell + z_\ell)(0) = f_\ell(0) + 0 = s_\ell$, and $\deg(f_\ell + z_\ell) < t$. The new keys are fresh and independent of the old ones, and the context changes ($e+1 \neq e$), so an old share and its old tags are an attempted forgery against every new honest key; Theorem 3(b) applies. Old shares also lie on a different polynomial ($z_\ell \neq 0$ with overwhelming probability) and cannot be mixed with new shares by interpolation. □

Proposition 7 (The two-round order is necessary)**.** *If custodian i learns the keys $\kappa_{j,i}$ before its share is fixed, it can make any $\mathbf{y}'_i$ pass every honest check with probability 1.*

Proof. With $\kappa_{j,i}$ known, the cheater sets $\tau'_{j,i} = \beta_{j,i} + \alpha_{j,i,0}\mathsf{ctx}_i + \sum_\ell \alpha_{j,i,\ell} y'_{i,\ell}$ for every j. Experiment E5 (Section 6) confirms this: with keys revealed first, the forged share passed the witness layer in 100.0% of 500 trials; with the SAKSI order, in 0.0%. □

The order is therefore necessary for the *witness* layer. It is not the only line of defence: when more than t shares are accepted, the syndrome layer is independent of the keys, and in the same E5 runs it still localised the forged share and the correct key was output in 100.0% of the one-round trials (defence in depth; with exactly t accepted shares there is no redundancy and this backstop is absent).

Why this is post-quantum. Theorems 1–5 make no computational assumption: they hold against an adversary with unlimited computing power, which includes any quantum computer. Nothing in the trust path depends on factoring, discrete logarithms, lattices or hash functions. What remains is the classic information-theoretic cost: keys are one-time and large, and an honest majority among the custodians is needed.

5 Attack-surface analysis

Vulnerability windows mitigated.

- **Silent corruption** (bit rot, buggy storage): detected by tags and syndrome; the bad share is named; reconstruction proceeds from the rest.
- **Tompa–Woll last-submitter attack**: the shifted share fails its witnesses (E5).
- **Rushing within reconstruction**: keys are released only after shares are fixed (Proposition 7).
- **Replay and cross-session substitution**: tags bind sid, epoch and index (E5).
- **Stolen old shares**: refresh moves all shares to a new polynomial and new keys.
- **Inconsistent dealer**: syndrome detects, Gao localises (E5).
- **Sub-threshold collusion**: perfect secrecy; Z3 finds no narrowing (E3).
- **Biased coefficients** (cf. RUSTSEC-2024-0398 [11]): coefficients are uniform over all of $\mathbb{F}_p$ including 0; E2 shows a biased sampler leaks.
- **Timing**: constant-time arithmetic and comparison (Section 7).
- **Using a wrong key**: every failure path returns $\perp$ rather than a guess.

Component	Assumption	Attacker needs	Mitigation	Residual risk
Shares	$\leq t-1$ leak before reconstruction	t shares of one epoch	perfect secrecy (Thm 1); refresh per epoch	t custodians compromised in one epoch
Witness tags	keys one-time, fresh per epoch	guess $\geq k$ honest keys	forgery $1/p$ (Thm 2)	$\leq 2^n/p$
Reveal order	round 1 fixed before round 2	see keys first	combiner enforces order (Prop. 7)	combiner bug
Custodian majority	$c \leq \lfloor (n-1)/2 \rfloor$	$\geq n/2$ custodians	vouching threshold; E5 shows bound is tight	beyond bound: wrong key possible
Dealer	honest for its epoch	malicious dealer	syndrome + Gao localise inconsistent shares	dealer that leaks S
Randomness	OS CSPRNG	predictable coefficients	`getrandom`/`secrets`; no `random` (tested)	OS RNG failure
Parsing	untrusted JSON	malformed input	canonical-element and size checks; dedup; fail closed (E7)	–
Timing	CPU executes as compiled	secret-dependent timing	branch-free Rust, ct compare (Section 7)	compiler/JIT, power/EM
Memory	no swap/core dumps	read process memory	`zeroize` on drop; key live only during use	key in RAM at use

Table 1: Attack surface of SAKSI and how each element is reduced.

6 Cryptanalysis experiments

All experiments are in `attacks/harness.py` and `core/src/bin/saksi.rs`; the numbers below are read from the generated result files.

E1 Exhaustive perfect secrecy. For toy fields $(p,t) = (31,3)$, $(p,t) = (101,2)$, $(p,t) = (13,4)$ we enumerated every polynomial for every secret and tabulated the view of a $(t-1)$-coalition, and for $p = 31$ also the coalition's tags under honest keys. Were the view distributions identical for every secret, in every case? Yes. The mutual information is therefore 0 bits, as Theorem 1 predicts.

E2 Share uniformity. With the secret fixed to the all-zero key, 100,000 sharings at full size; the top 8 bits of three shares were tested with χ^2 over 256 bins: p-values 0.297, 0.941, 0.885 (Figure 5). As a control, a sharer that draws its coefficient from $\{1,\ldots,p-1\}$ (the RUSTSEC-2024-0398 pattern) leaks exactly 0.0056 bits per share in GF(257); SAKSI's uniform sampler leaks 0.0 bits.

E3 Collusion with an SMT solver. Given two of three needed shares, Z3 enumerated every secret consistent with the view. Toy SAKSI ($p = 31$, $t = 3$): 31 of 31 candidates remain (0.00 bits leaked). Mignotte control (moduli $11, 13, 17, 19, 23$, threshold 3): 14 of 1,993 candidates remain, 7.15 bits leaked, which is why CRT-based sharing was not used.

E4 Forgery rate. For nine toy primes from 5 to 257, 200,000 forgery attempts each; the measured success rate tracks $1/p$ (largest absolute deviation 0.0010; Figure 6), confirming Theorem 2 is tight.

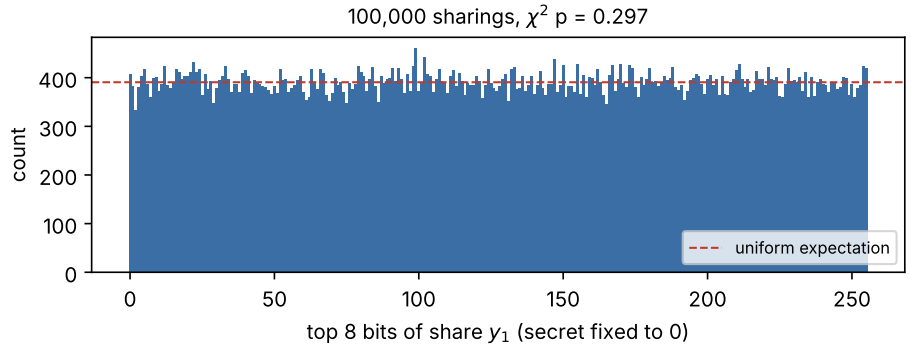


Figure 5: Distribution of the top byte of share y_1 over all sharings of a fixed secret (E2).

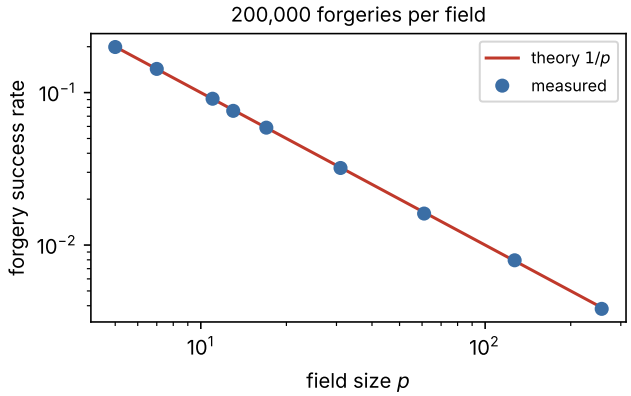


Figure 6: Measured witness-forgery success vs. the bound $1/p$ (E4).

E5 Active attacks at full size. $p = 2^{127} - 1$, $n = 5$, $t = 3$, 32-byte key. Table 2 compares plain Shamir (interpolate the first t shares) with SAKSI.

Every attack in Table 2 was detected and attributed in all runs. Plain Shamir returned a wrong key in all runs except for bit flips (81.9%): with a 32-byte key the last limb holds only 2 bytes, and when a flipped bit in that limb produces an error that stays inside its 13 bytes of zero padding, the unvalidated control truncates the error away. SAKSI rejects those shares as well, through the witness tags (and, as an extra, the padding check).

Two further rows probe the model's limits. *Rushing control:* if keys are revealed before shares (violating the protocol), the cheater recomputes valid tags and its forged share passes the witness layer in 100.0% of 500 runs, against 0.0% with the SAKSI order (Proposition 7); in those one-round runs the independent syndrome layer still localised the forged share and the correct key was output in 100.0%. *Bound tightness:* three colluding custodians out of five ($c = 3 > T = 2$) forge a consistent, well-formed alternative key, vouch for each other and reject the honest shares. SAKSI then returned the coalition's wrong key in 100.0% of 100 runs. This is the expected boundary, not a new weakness: robust reconstruction is impossible against a dishonest majority, and with $c \geq t$ the coalition holds the key anyway. The condition $c \leq \lfloor (n-1)/2 \rfloor$ in Theorem 3 is tight.

E6 The same attacks on the Rust core. 150 runs through the compiled `saksi` binary (bit flip, Tompa–Woll shift, two cheaters): correct key 100.0%, wrong key 0.0%, cheater named 100.0%.

Attack	Trials	Control fooled	SAKSI wrong key	SAKSI names cheater
bit flip in one share	1000	81.9%	0.0%	100.0%
random share replacement	1000	100.0%	0.0%	100.0%
Tompa–Woll: last submitter shifts share	1000	100.0%	0.0%	100.0%
cross-session substitution	1000	100.0%	0.0%	100.0%
two colluding cheaters, lying keys	500	100.0%	0.0%	100.0%
replay share from previous epoch	500	100.0%	0.0%	100.0%
inconsistent dealer (tags valid)	500	100.0%	0.0%	100.0%

Table 2: E5: rate at which each attack makes the system output a wrong key, and the rate at which SAKSI names the cheater. "Control fooled" counts runs in which plain Shamir returned a wrong key.

E7 Fuzzing. 2,000 randomly mutated bundles (bad indices, non-canonical or malformed field elements, truncated shares, duplicates, missing tags) fed to the Rust binary: 0 crashes and 0 wrong keys returned.

E8 Timing of the Python reference (control). The Python reference multiplies with variable-time big integers. A dudect test (fixed 0 vs. random, 200,000 samples) gives $|t| = 380.2$, i.e. it leaks. This is why the Python code is only the specification and test oracle, and the deployable core is Rust (Section 7).

7 Side-channel engineering

The track brief asks for an algorithm that resists side-channel attacks. The Rust core follows four rules for every operation on secret data (source file `core/src/field.rs`).

1. **Branch-free reduction.** Elements are canonical `u128` values. After an addition, the map $x \mapsto (x \bmod 2^{127}) + \lfloor x/2^{127} \rfloor$ brings x below $p+2$ (since $2^{127} \equiv 1$), and the final conditional subtraction uses the borrow flag turned into an all-ones mask, never a branch.
2. **Fixed-shape multiplication.** The 127×127-bit product is computed from four $64 \times 64 \to 128$-bit products; with $2^{128} \equiv 2$ the 254-bit result $h \cdot 2^{128} + \ell$ becomes $2h + (\ell \bmod 2^{127}) + \lfloor \ell/2^{127} \rfloor < 2^{128}$, then is frozen as above. Measured cost: 9.0 ns.
3. **Fixed ladder inversion.** $a^{-1} = a^{p-2}$ by a 127-step square-and-multiply ladder that *always* multiplies and chooses the result with a masked select; the exponent is public, so the instruction trace is identical for every input (2.43 μs).
4. **Constant-time comparison and hygiene.** Tags are compared with an OR-reduction of $x \oplus y$ (`ct_eq`); `core::hint::black_box` prevents the compiler from turning masks back into branches; shares, keys, coefficients and reconstructed limbs are `zeroize`d on drop.

Methodology. We follow dudect [10]: for each operation two input classes (fixed vs. random) are generated *before* measurement, interleaved in random order, timed with serialised `rdtsc`, the slowest 5% of each class discarded, and compared with Welch's t-test; $|t| > 4.5$ indicates a leak. Our first run generated the random inputs *inside* the timed region, so it measured the operating-system RNG and flagged every operation; we discarded it and redid the study correctly. Two deliberately leaky functions serve as positive controls: an extended-Euclid inverse and an early-exit byte comparison.

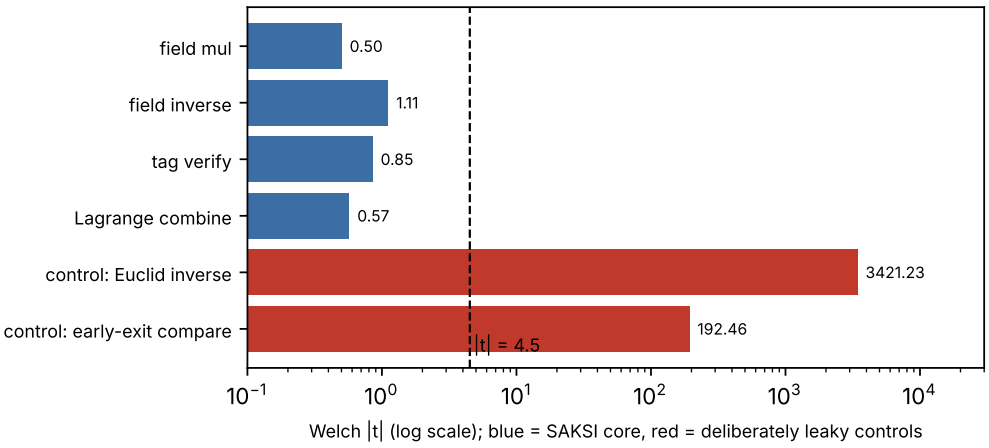


Figure 7: Welch $|t|$ per operation. SAKSI operations stay far below the 4.5 threshold; both leaky controls are detected.

Operation (class A vs class B)	Samples	Mean A	Mean B	$\lvert t\rvert$
Fe::mul (fixed 0 vs random)	1,000,000	88.6	88.6	0.50
Fe::inv (fixed 1 vs random)	250,000	4899.9	4899.9	1.11
tag verify (valid vs forged tag)	1,000,000	146.5	146.5	0.85
Lagrange combine 3 shares (zero key vs random key)	1,000,000	130.7	130.7	0.57
control: leaky Euclid inverse (fixed 1 vs random)	250,000	135.1	2667.6	3421.23
control: leaky early-exit compare (equal vs random)	1,000,000	90.1	85.9	192.46

Table 3: dudect results for the Rust core (cycles). Class A is the fixed input, class B random.

Limits. Measurements were taken on a shared cloud virtual machine; they show the absence of a *detectable* leak at this sample size, not a proof. Constant-time behaviour depends on the compiler, and the WebAssembly build in the browser demo runs under a JIT that gives no constant-time guarantee. Gao decoding and polynomial division branch on degrees; they run only when the syndrome is non-zero, and what they reveal (which shares are bad) is reported publicly anyway. Power and electromagnetic channels are out of scope.

8 Implementation and benchmarks

- **Rust core** (`core/`): field, polynomials and Gao decoder, scheme, JSON wire format with input validation, CLI, and a WebAssembly build (`wasm-bindgen`). Tests: 4 unit and property-based tests of the field (256 random cases per property) and 10 integration tests of the scheme.
- **Python reference** (`ref/`): a direct transcription of Figures 1 and 2; 18 tests. A cross-implementation check deals in one language and reconstructs in the other (30 rounds, identical results).
- **Performance** (release build, same VM): deal a 32-byte key 3-of-5 in 40.8 μs; validated reconstruction from 5 shares in 45.2 μs; keyless decoding with one error in 62.6 μs.

9 Applied environment: the Kunci Agent

To show the algorithm in use, the WebAssembly build of the core runs in a browser application (`https://saksi-gamma.vercel.app`). An API credential is dealt 3-of-5 to five custodians (the

browser, a passkey- or password-derived store, the server, and two simulated colleagues). A language-model agent (Gemini through function calling, with Groq as a fallback provider) plans the user's task; when a step needs the credential it calls a tool that runs the two-round SAKSI reconstruction, shows which custodians were accepted, rejected or located, uses the key for a single outbound request and zeroises it. The model only ever receives the tool's result; no share, tag, key or credential is sent to the model provider. A "tamper" switch on each custodian lets a viewer reproduce the attacks of Section 6 live. Source code: `https://github.com/ShermaineYap/saksi`.

10 Limitations

- **Trusted dealer per epoch.** A dishonest dealer is caught if it hands out inconsistent shares, but a dealer that leaks the key is not prevented; dealer-free generation (each custodian dealing a sharing of a random contribution) is future work.
- **Key material grows as $\Theta(n^2 L)$**, 432 bytes per custodian for $n = 5$, $L = 3$; fine for key custody, not for bulk data.
- **Honest majority.** Robustness needs $c \leq \lfloor (n-1)/2 \rfloor$; E5 shows what happens beyond it.
- **Ordered, authenticated channel to the combiner.** Proposition 7 shows the order is essential; the combiner must enforce it. Duplicate submissions under one index are resolved by witness votes, but authenticated channels remain the clean assumption.
- **The key exists in memory at use.** Reconstruction necessarily materialises the key; we minimise the window and zeroise, but malware on the combining host can read it.
- **Refresh is centralised.** Our refresh uses one refresher; full mobile-adversary security needs a distributed refresh [6].
- **Python is variable time** (E8) and is not meant for deployment.
- **The secret's length** λ is public in the header.

11 Compliance with the hackathon brief

- *Encrypts the master key without pre-existing algorithms:* the key is protected only by the SAKSI sharing itself (perfect secrecy); no block cipher, stream cipher, hash or library cryptography is used in the trust path. The only external dependency is the operating system's random number generator.
- *Number theory:* arithmetic in the Mersenne field, $2^{127} \equiv 1$ reduction, Fermat inversion, field-size forgery bounds. *Linear algebra:* Vandermonde encoding, Lagrange coefficients, inner-product tags, the parity-check (syndrome) matrix and its rank argument, Reed–Solomon decoding.
- *Low-level language:* the core is Rust with constant-time arithmetic.
- *Applied environment:* the Kunci Agent (Section 9).
- *Survives cryptanalysis / minimises attack surface:* Sections 4 to 7.

References

[1] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29(2):208–210, 1983.

[2] G. R. Blakley. Safeguarding cryptographic keys. In *Proc. National Computer Conference (AFIPS)*, vol. 48, pages 313–317, 1979.

[3] A. Cevallos, S. Fehr, R. Ostrovsky and Y. Rabani. Unconditionally-secure robust secret sharing with compact shares. In *EUROCRYPT 2012*, LNCS 7237, pages 195–208, 2012.

[4] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS 1987*, pages 427–437, 1987.

[5] S. Gao. A new algorithm for decoding Reed–Solomon codes. In *Communications, Information and Network Security*, Springer, pages 55–68, 2003 (preprint 2002).

[6] A. Herzberg, S. Jarecki, H. Krawczyk and M. Yung. Proactive secret sharing or: how to cope with perpetual leakage. In *CRYPTO '95*, LNCS 963, pages 339–352, 1995.

[7] M. Mignotte. How to share a secret. In *Cryptography, Proc. Workshop Burg Feuerstein 1982*, LNCS 149, pages 371–375, 1983.

[8] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*, LNCS 576, pages 129–140, 1992.

[9] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC 1989*, pages 73–85, 1989.

[10] O. Reparaz, J. Balasch and I. Verbauwhede. Dude, is my code constant time? In *DATE 2017*, pages 1697–1702, 2017.

[11] RustSec Advisory Database. RUSTSEC-2024-0398: Bias of polynomial coefficients in secret sharing (crate `sharks`). 2024.

[12] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[13] M. Tompa and H. Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(2):133–138, 1988.