leanSPHINCS

Abstract

This specification defines leanSPHINCS, a stateless hash-based signature scheme whose few-time signature is FORS and whose one-time keys form a hypertree of three layers, with hash inputs laid out in 16-byte words (Section 6).

- **NIST security level 1** [Nat16], 127 bits of classical security in the Random Oracle model, and a target of ≈ 64 bits in the Quantum Random Oracle model (Section 6).
- **Public key: 32 bytes.**
- **Signature: 7740 bytes.**
- **Verification: 492 permutations** of Keccak-f.
- **Lifetime:** 2^{30} **signatures per key pair**, proved up to 1.18B (Section 6).

A message digest picks a bottom leaf, and so one FORS instance [Nat24] without its root hash, and one leaf in each of its trees. The openings sign the message, and a hypertree of WOTS$^+$C [HKRY22] keys signs the FORS key up to the public key. The definitions below specify the scheme completely.

1 Hashing and parameters

Write $\{0,1\}^r$ for r-bit strings, 0^r for r zero bits, $\|$ for concatenation, $\mathsf{LE}_r(z)$ for the unsigned r-bit little-endian encoding of z, and LE^{-1} for its inverse. Write $x \xleftarrow{\$} A$ for a uniform sample from A. Indices and bit positions start at zero; $\perp$ denotes failure.

Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{256}$ be Keccak-256 [BDPV11], Ethereum's hash, which is the 256-bit hash of FIPS 202 [Nat15] but for its padding's first byte, `0x01` where the standard has `0x06`. Most operations use its first $n = 128$ output bits:

$$\mathsf{Th}(P, \mathsf{A}, M) = \mathsf{Truncate}_n(\mathsf{H}(P \,\|\, \mathsf{A} \,\|\, M)) \,.$$

Here P is a 128-bit public parameter and A is a 128-bit *address* identifying the operation and its position in the construction. Appendix A gives every address's exact bytes. $\mathsf{Truncate}_r$ always keeps the first r bits, with bits read least significant first within each byte. The parameter and the address take 32 of a block's 136 bytes, so a chain step and a Merkle node each cost one permutation, and every value a hash input holds starts on a 16-byte boundary.

The message m, the seed S_0 and the secret S derived from it are each 256 bits. The signature contains a 128-bit randomizer ρ and one 32-bit counter per layer. Chain values and Merkle nodes are 128 bits; write $\mathcal{H} = \{0,1\}^n$.

Symbol	Value	Meaning
w	2	bit-size of WOTS$^+$C chain positions
v	64	chains per WOTS$^+$C key
T	128	sum of the signed chain positions
d	3	layers of the hypertree
h_0, h_1, h_2	$8, 8, 10$	height of a tree of each layer, from the bottom
h	26	$h_0 + h_1 + h_2$, the bits of an index
k	24	trees of a FORS instance
a	10	height of a FORS tree
$C_{\max}$	2^{32}	maximum encoding attempts per layer

The public parameter and all signing secrets are derived from $S = \mathsf{H}(\texttt{leanSPHINCS} \parallel S_0)$, the hash of the scheme's 11-byte ASCII name then the seed, so that no other scheme derives from the same bytes. In particular,

$$P = \mathsf{Th}\Big(0^{128}, \mathsf{A}_{\text{parameter}}, S\Big)\,.$$

A derivation of secrets hashes S and yields two of them, the two halves of the untruncated output: write H_0 and H_1 for the first and the last 128 bits of H. Functions that read secrets use S implicitly.

2 WOTS$^+$C: signing with hash chains

A one-time key $K = (l, \tau, e)$ is identified by its layer $l < d$, counted from the bottom, its tree τ on that layer and its leaf $e < 2^{h_l}$ in that tree. It contains $v = 64$ chains, each with four positions, numbered 0 through 3. For $0 \le i < v$, derive chain i's start and hash forward:

$$\begin{aligned} C_{i,0} &= \mathsf{H}_{i \bmod 2}(P \parallel \mathsf{A}_{\text{prf}}(K, \lfloor i/2 \rfloor) \parallel S)\,, \\ C_{i,\mu+1} &= \mathsf{Th}(P, \mathsf{A}_{\text{chain}}(K, i, \mu), C_{i,\mu})\,, \qquad 0 \le \mu < 2^w - 1. \end{aligned}$$

A step's address names the position it leaves. Hashing the endpoints in chain order gives the key's Merkle leaf:

$$X_{0,e}^{l,\tau} = \mathsf{Th}(P, \mathsf{A}_{\text{leaf}}(K), C_{0,3} \parallel \cdots \parallel C_{v-1,3})\,.$$

To sign a message M, a FORS key of 48 nodes (Section 4) or a root of the layer below, the signer encodes it as positions $x_0, \ldots, x_{v-1}$ and reveals C_{i,x_i} from each chain. The verifier hashes these values forward to recover the endpoints. All valid encodings have the same sum:

$$\mathcal{C} = \left\{ x \in \{0, \ldots, 2^w - 1\}^v : \sum_{i=0}^{v-1} x_i = T \right\}.$$

Hashing forward can only increase a position. Thus one valid encoding cannot be changed into a distinct valid encoding merely by advancing along the chains.

Encoding a message. $\mathsf{Enc}(P, K, M, c)$ first computes

$$D = \mathsf{Th}(P, \mathsf{A}_{\text{enc}}(K), M \parallel \mathsf{LE}_{32}(c))\,,$$

the counter falling in the last 136-byte block, so that a signer trying counters pays exactly one permutation for each. Interpret D as a little-endian integer; it supplies 64 consecutive two-bit positions:

$$x_i = \left\lfloor D/2^{2i} \right\rfloor \bmod 4, \qquad 0 \le i < v.$$

Return x if $\sum_i x_i = T$; otherwise return $\perp$. An accepted encoding specifies all 128 bits of D.

Signing. $\mathsf{Ots.sign}(P, K, M)$ tries $c = 0, \ldots, C_{\max} - 1$ in order. At the first successful encoding x, it returns

$$(c, \sigma), \qquad \sigma_i = C_{i,x_i}, \quad 0 \le i < v.$$

It returns $\perp$ if every counter fails.

Recovering the leaf. $\mathsf{Ots.leaf}(P, K, M, c, \sigma)$ recomputes $x = \mathsf{Enc}(P, K, M, c)$ and returns $\perp$ if encoding fails. Otherwise, for each i, start with $C'_{i,x_i} = \sigma_i$ and apply the chain recurrence up to position 3. Return

$$\mathsf{Th}\Big(P, \mathsf{A}_{\text{leaf}}(K), C'_{0,3} \parallel \cdots \parallel C'_{v-1,3}\Big).$$

Choosing the least successful counter makes repeated signing of one message under one key reveal the same positions. Revealing different positions for the same message would let an adversary combine the smaller positions from both signatures.

3 The hypertree: authenticating one-time keys

Layer l has $2^{h-(h_0+\cdots+h_l)}$ trees of height h_l: the bottom layer 2^{18}, the middle one 2^{10}, the top one a single tree. The leaves of tree τ of layer l are the $X^{l,\tau}_{0,e}$, and its parent nodes are

$$X^{l,\tau}_{\lambda,j} = \mathsf{Th}\Big(P, \mathsf{A}_{\text{node}}(l, \tau, \lambda, j), X^{l,\tau}_{\lambda-1,2j} \parallel X^{l,\tau}_{\lambda-1,2j+1}\Big),$$

for $1 \le \lambda \le h_l$ and $0 \le j < 2^{h_l-\lambda}$. Children are always hashed left first. The tree's root is $X^{l,\tau}_{h_l,0}$, and the root of the top tree is the public key's.

An index $idx < 2^h$ names one key on every layer: with $\beta_l = h_0 + \cdots + h_{l-1}$ the index bits below layer l, its key there is

$$K_l(idx) = \Big(l,\ \Big\lfloor idx/2^{\beta_l+h_l} \Big\rfloor,\ \Big\lfloor idx/2^{\beta_l} \Big\rfloor \bmod 2^{h_l}\Big).$$

So the bottom key is in tree $\lfloor idx/2^8 \rfloor$ at leaf $idx \bmod 2^8$, the middle key in tree $\lfloor idx/2^{16} \rfloor$ at leaf $\lfloor idx/2^8 \rfloor \bmod 2^8$, and the top key at leaf $\lfloor idx/2^{16} \rfloor$. The key of layer $l+1$ signs the root of the tree holding the key of layer l.

An authentication path supplies the sibling of the chosen node at each height: for the key at leaf e, it is $A = (A_0, \ldots, A_{h_l-1})$ with $A_\lambda = X^{l,\tau}_{\lambda,\lfloor e/2^\lambda \rfloor \oplus 1}$, where $\oplus 1$ flips the low bit of the node index. Paths are ordered from leaf to root.

$\mathsf{Tree.fold}(P, K, X, A)$ recovers the root from a leaf value X and its path. Start with $V_0 = X$. For $0 \le \lambda < h_l$, let $j = \lfloor e/2^{\lambda+1} \rfloor$ and compute

$$V_{\lambda+1} = \begin{cases} \mathsf{Th}(P, \mathsf{A}_{\text{node}}(l, \tau, \lambda+1, j), V_\lambda \parallel A_\lambda) & \text{if bit } \lambda \text{ of } e \text{ is zero,} \\ \mathsf{Th}(P, \mathsf{A}_{\text{node}}(l, \tau, \lambda+1, j), A_\lambda \parallel V_\lambda) & \text{otherwise.} \end{cases}$$

Return V_{h_l}.

4 FORS: signing digest fields

A FORS instance is identified by a bottom leaf $idx \in \{0, \ldots, 2^h - 1\}$. It contains $k = 24$ trees of height $a = 10$, whose leaves hash secrets. For tree $\kappa < k$, leaf $s < 2^a$ and $1 \le \lambda \le a - 1$:

$$\begin{aligned} sk_{\kappa,s} &= \mathsf{H}_{s \bmod 2}(P \parallel \mathsf{A}_{\mathrm{fprf}}(idx, \kappa, \lfloor s/2 \rfloor) \parallel S)\,, \\ Y^{\kappa}_{0,s} &= \mathsf{Th}(P, \mathsf{A}_{\mathrm{fleaf}}(idx, \kappa, s), sk_{\kappa,s})\,, \\ Y^{\kappa}_{\lambda,j} &= \mathsf{Th}\Big(P, \mathsf{A}_{\mathrm{fnode}}(idx, \kappa, \lambda, j), Y^{\kappa}_{\lambda-1,2j} \parallel Y^{\kappa}_{\lambda-1,2j+1}\Big)\,. \end{aligned}$$

Trees without a root. No tree has a root hash: what stands for a root is the pair of nodes just below it, and the instance's key is those 48 nodes themselves,

$$\mathsf{Fts.key}(P, idx) = \Big(Y^{0}_{a-1,0}, Y^{0}_{a-1,1}, \ldots, Y^{k-1}_{a-1,0}, Y^{k-1}_{a-1,1}\Big)\,,$$

768 bytes. No hash folds them: the bottom one-time key's encoding absorbs all of them with its counter (Section 2).

Opening. The digest selects a leaf s_κ in each tree. $\mathsf{Fts.sign}(P, idx, s)$ reveals, per tree, the secret sk_{κ,s_κ} and the path $(Y^{\kappa}_{\lambda,\lfloor s_\kappa/2^\lambda \rfloor \oplus 1})_{0 \le \lambda < a}$, from the leaves up: its last element is the top node the fold does not reach, 11 values per tree.

Recovering the key. Fts.recover hashes each revealed secret into its leaf and folds it $a-1$ heights as Tree.fold does, with the bits of s_κ and the tree's node addresses. The node reached and the path's last element are the tree's two top nodes, ordered by bit $a-1$ of s_κ. Return the 48.

An instance may sign several digests. Reuse leaks rather than breaks: a forgery at an instance needs, in every tree, the selected leaf's secret already revealed.

The digest. For message m and randomizer ρ, define

$$\mathsf{Digest}(P, m, \rho) = D_0 \parallel D_1, \qquad D_c = \mathsf{H}(P \parallel \mathsf{A}_{\mathrm{msg}}(c) \parallel m \parallel \rho)\,.$$

The root is not hashed: P binds the key. Each input is 80 bytes, one block. Interpret the 512-bit digest as a little-endian integer N, of which the low $h + ka = 266$ bits are used: the index, then the trees' leaves,

$$idx = N \bmod 2^h, \qquad s_\kappa = \Big\lfloor N/2^{h+a\kappa} \Big\rfloor \bmod 2^a.$$

Every digest is accepted: every index is a leaf of the key.

5 The complete signature scheme

5.1 Key generation

Gen samples $S_0 \xleftarrow{\$} \{0,1\}^{256}$, derives S and P as in Section 1, builds the top tree of 2^{h_2} one-time keys, and returns

$$pk = (root, P), \qquad sk = S_0.$$

5.2 Signing

Given $sk = S_0$ and a 256-bit message m, $\mathsf{Sig}(sk, m)$ proceeds as follows:

1. Derive $\rho = \mathsf{Th}(P, \mathsf{A}_{\mathrm{rnd}}, S \parallel m)$, and extract idx and the leaves from $\mathsf{Digest}(P, m, \rho)$.
2. Compute the opening $\mathsf{Fts.sign}(P, idx, s)$ and $M_0 = \mathsf{Fts.key}(P, idx)$.
3. For $l = 0, 1, 2$: compute $(c_l, \sigma_l) = \mathsf{Ots.sign}(P, K_l(idx), M_l)$, returning $\perp$ on failure, and the authentication path A_l of $K_l(idx)$ in its tree, and let M_{l+1} be that tree's root.
4. Return ρ, the opening, and (c_l, σ_l, A_l) for each layer.

Signing is deterministic and changes no secret state. Repeating a message returns the same signature, or the same failure. A key signs at most 2^{30} messages.

5.3 Verification

$\mathsf{Ver}(pk, m, \sigma)$ rejects $\perp$ and malformed inputs. Otherwise, parse $pk = (root, P)$ and σ as above:

1. Compute $\mathsf{Digest}(P, m, \rho)$ and extract idx and the leaves.
2. Set $M_0 = \mathsf{Fts.recover}(P, idx, s, \text{opening})$.
3. For $l = 0, 1, 2$: compute $X_l = \mathsf{Ots.leaf}(P, K_l(idx), M_l, c_l, \sigma_l)$, rejecting if $X_l = \perp$, and $M_{l+1} = \mathsf{Tree.fold}(P, K_l(idx), X_l, A_l)$.
4. Accept exactly when $M_3 = root$.

5.4 Serialization and cost

Serialize the public key as $root \parallel P$, 32 bytes. The secret key is S_0, 32 bytes.

A signature starts with ρ, followed by the FORS opening in increasing tree, each tree's secret then its path in increasing height, then the layers from the bottom up, each $\mathsf{LE}_{32}(c_l)$, the v chain values in increasing chain index and the path in increasing height. The size is

$$\underbrace{16}_{\rho} + \underbrace{k\,(1+a)\,16}_{\text{FORS opening}} + \underbrace{d\,(4+v\,16)}_{\text{WOTS}^{+}\text{C}} + \underbrace{h\,16}_{\text{paths}} = 7740 \text{ bytes.}$$

Count permutations of Keccak-f: an input of r bytes costs $\lfloor r/136 \rfloor + 1$. Every signature takes exactly

$$\underbrace{2}_{\text{digest}} + \underbrace{k\,a}_{\text{FORS}} + \underbrace{6+1+1}_{\text{encodings}} + \underbrace{d\big((2^w-1)v - T + 8\big)}_{\text{WOTS}^{+}\text{C}} + \underbrace{h}_{\text{paths}} = 492$$

permutations to verify: per FORS tree a leaf and 9 folds; 6 for the bottom encoding, which absorbs the FORS key, and 1 for each encoding above it; per layer 64 chain steps and 8 for the one-time leaf.

A one-time key takes 32 derivations and 192 chain steps, so its leaf 232 permutations, and key generation, with the scheme seed and the public parameter, $232 \cdot 2^{10} + (2^{10} - 1) + 2 = 238\,593$. Signing spends about $4.6 \cdot 10^5$: 61 392 for the FORS instance, 119 294 for the two lower trees, 238 591 for the top tree, which a signer may keep from key generation instead, and per layer the expected 14 700 encoding attempts, one permutation each once the message is absorbed.

6 Security

Model H as one shared random oracle. The challenger generates a key pair and gives pk to a classical probabilistic adversary, which may adaptively query the hash and request signatures on messages of its choice, at most N of them. It finally returns (m^*, σ^*) and wins exactly when $\mathsf{Ver}(pk, m^*, \sigma^*) = 1$ and no signing request for m^* returned σ^*: *strong unforgeability*. A query budget q bounds every hash call of the complete experiment, including key generation, signing and final verification.

For $N = 1\,182\,000\,000 \approx 2^{30.14}$, every adversary wins with probability at most $q/2^{127}$. This is proved in Lean 4, with only Lean's three standard axioms, in `formal/lean_sphincs/` (`FORS.h26_layered`), for the scheme of this document: its hash inputs and its signer.

Layout. The address is chosen so that a prover whose machine words are 16 bytes never cuts a value in two: it is 16 bytes, so every value after it starts a cell, and its fields are little-endian.

Hash domains. An input determines its parameter, its address and its payload, and an address determines the call, so distinct calls of one key never share an input. Calls of different keys differ in P. The scheme seed's input, 43 bytes with no parameter and no address, is shorter than every other input, which is at least 48 bytes.

Quantum security. Approximately 64 bits of security in the quantum random-oracle model is a target. No quantum security bound is proved here.

7 Completeness

Signing fails only if a counter search is exhausted; whenever it succeeds, verification accepts, each recovery retracing what the signer computed. An encoding accepts exactly when its 64 two-bit positions sum to 128, so a uniform 128-bit digest is accepted with probability

$$\frac{[z^{128}](1+z+z^2+z^3)^{64}}{2^{128}} = \frac{231952106178391201996588199096936 79}{2^{128}} \geq 2^{-13.85},$$

where $[z^r]$ denotes the coefficient of z^r, and all $C_{\max} = 2^{32}$ counters of a layer fail with probability at most $e^{-2^{18}}$.

A Address encoding

An address is 16 bytes: a 32-bit field lo, a 24-bit field hi, one byte holding a 5-bit type t and a 3-bit chain step μ, one byte holding a layer l, a 32-bit tree τ, and 3 zero bytes,

$$\mathsf{enc}(t, \mu, l, \tau, hi, lo) = \mathsf{LE}_{32}(lo) \,\|\, \mathsf{LE}_{24}(hi) \,\|\, \mathsf{LE}_8(t + 32\mu) \,\|\, \mathsf{LE}_8(l) \,\|\, \mathsf{LE}_{32}(\tau) \,\|\, 0^{24}.$$

A one-time key $K = (l, \tau, e)$ puts its layer and tree in their fields and its leaf in lo. Outside the hypertree the layer and the tree are zero. In a FORS instance, lo is idx and hi packs the tree and a position: $\kappa + 64s$ for a leaf s, $\kappa + 64p$ for the pair of leaves $2p$ and $2p+1$, and $\kappa + 64(\lambda + 16j)$ for node j of height λ.

Address	Encoding
$\mathsf{A}_{\text{prf}}(K, r)$	$\mathsf{enc}(0, 0, l, \tau, r, e)$
$\mathsf{A}_{\text{chain}}(K, i, \mu)$	$\mathsf{enc}(1, \mu, l, \tau, i, e)$
$\mathsf{A}_{\text{leaf}}(K)$	$\mathsf{enc}(2, 0, l, \tau, 0, e)$
$\mathsf{A}_{\text{node}}(l, \tau, \lambda, j)$	$\mathsf{enc}(3, 0, l, \tau, \lambda, j)$
$\mathsf{A}_{\text{enc}}(K)$	$\mathsf{enc}(4, 0, l, \tau, 0, e)$
$\mathsf{A}_{\text{parameter}}$	$\mathsf{enc}(5, 0, 0, 0, 0, 0)$
A_{rnd}	$\mathsf{enc}(7, 0, 0, 0, 0, 0)$
$\mathsf{A}_{\text{msg}}(c)$	$\mathsf{enc}(12, 0, 0, 0, c, 0)$
$\mathsf{A}_{\text{fprf}}(idx, \kappa, p)$	$\mathsf{enc}(14, 0, 0, 0, \kappa + 64p, idx)$
$\mathsf{A}_{\text{fleaf}}(idx, \kappa, s)$	$\mathsf{enc}(15, 0, 0, 0, \kappa + 64s, idx)$
$\mathsf{A}_{\text{fnode}}(idx, \kappa, \lambda, j)$	$\mathsf{enc}(17, 0, 0, 0, \kappa + 64(\lambda + 16j), idx)$

In a chain address, μ is the position the step leaves. Types 0, 5, 7 and 14 are reserved for the signer's derivations from the seed.

References

[BDPV11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, version 3.0, 2011. URL: https://keccak.team/files/Keccak-reference-3.0.pdf.

[HKRY22] Andreas Hülsing, Mikhail Kudinov, Eyal Ronen, and Eylon Yogev. SPHINCS+C: Compressing SPHINCS+ with (almost) no cost. Cryptology ePrint Archive, Paper 2022/778, 2022. URL: https://eprint.iacr.org/2022/778.

[Nat15] National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication 202, U.S. Department of Commerce, 2015. doi:10.6028/NIST.FIPS.202.

[Nat16] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. NIST PQC Call for Proposals, 2016. Section 4.A.5, p. 16.

[Nat24] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Technical Report FIPS 205, National Institute of Standards and Technology, 2024. doi:10.6028/NIST.FIPS.205.