

CUTS: Chained Uniform-sum Tree Signatures

Jamie Judd, Louis Tajan, Conor Deegan, and Furkan Akal

Project Eleven

{jamie.judd,lt,cd,fa}@projecteleven.com

Abstract

Stateful hash-based signatures fail badly when a one-time key signs twice, as happens when the signer’s state is lost or rolled back. We study what a second signature reveals in a broad class of hash-based schemes whose key is a tree of hash chains. In this class, a message selects a set of tree nodes that covers every leaf exactly once, together with a position on the chain of each selected node, and the signature reveals the chain values at those positions. For the whole class, we prove that the values derivable from several signatures are exactly the union of those derivable from each signature alone, and we bound forgery after b signatures in the classical random-oracle model by a combinatorial coverage exponent λ_b , also when the signer grinds a counter. We then present CUTS (Chained Uniform-sum Tree Signatures), a member of the class built on one binary tree, in which both the selected nodes and their chain positions depend on the message, with the positions summing to a fixed value S . Signature length and verification cost are fixed by the parameters and do not depend on the tree height. For CUTS, we compute the two-use coverage exactly as one coefficient of a polynomial and derive deterministic lower bounds on λ_2 ; one parameter set with 1424-byte signatures has $\lambda_2 \geq 128.6$. A counter variant, CUTS+C, trades a short counter and extra signing work for a proven grinding gain of g bits in the coverage term.

1 Introduction

Hash-based signatures rest only on properties of a hash function, which makes them the most conservative choice for post-quantum authentication [1–3]. Their building blocks are one-time and few-time signatures. A one-time signature such as WOTS+ [4] is secure for a single message. Stateful schemes such as XMSS [2] authenticate many one-time keys under one root and rely on the signer never to use a key twice. Stateless schemes such as SLH-DSA [3, 5] avoid state by selecting a FORS (Forest of Random Subsets) few-time key pseudorandomly, at the price of larger signatures.

The weak point of the stateful design is key reuse. If the state is lost or rolled back, the signer may sign twice with one Winternitz key, and two signatures under one such key can reveal enough chain values for an observer to forge; the SHRINCS proposal for Bitcoin records this risk for its WOTS+C keys [6]. Few-time schemes in the line of HORS (Hash to Obtain a Random Subset) and FORS degrade gradually as signatures accumulate [5, 7], but each signature carries one authentication path per opened tree. Constant-sum Winternitz encodings fix the verification cost of every message [8–10], yet they remain one-time. How much a second signature gives away is therefore a central question for these components, and it is usually answered scheme by scheme, by simulation.

These components share one structure. Each key is a rooted tree: the leaves are derived from a secret seed, every internal node hashes the values of its children, and the root is the public key. A run of single-child nodes is a hash chain. A signature selects a set of nodes that meets every root-to-leaf path exactly once, which we call a frontier, and reveals their values; the verifier hashes them up to the root. A Winternitz key is a set of chains whose ends are hashed

into the root; its frontier takes one position on every chain, and only the positions depend on the message. A FORS key is a set of Merkle trees whose roots are hashed together; its signature reveals one leaf secret per tree together with the siblings on that leaf's authentication path, which form a frontier. Graph-based one-time signatures [11–13] also reveal sets of nodes from which the public key can be computed. What a signature discloses, and what several signatures disclose together, therefore depends only on the tree and on the family of frontiers that messages select. This lets us analyze reuse once, for the whole class, and then design one member of it.

CUTS (Chained Uniform-sum Tree Signatures) is that member. Its key is a single binary tree in which every non-root node carries a chain of w positions. A message selects a frontier of L nodes and a chain position at each, with the positions summing to a fixed S , so both the frontier and the positions vary with the message. The signature reveals the L addressed chain values and no authentication path. Fixing L fixes the signature length, and fixing L , w and S fixes the verification cost, which does not depend on the tree height h . A taller tree only increases the number of encodings, and with it the cost of key generation and uncached signing. We are not aware of another hash-based component that fixes both signature length and verification work with a single tree, a message-chosen frontier and no authentication path, and that comes with a proved bound on what a second signature under the same key gives away.

Contributions.

1. *A generic tree-chain scheme.* We define a class of hash-based signatures whose keys are trees of hash functions and whose signatures reveal the values on a message-selected frontier, together with a +C variant in which the signer grinds a short counter (Section 4). Winternitz keys, FORS keys and CUTS are instances of this class (Sections 5.1 and 5.5.3).
2. *Union preservation and the reuse exponent.* For every scheme in the class, the nodes derivable from several frontier disclosures are exactly the union of those derivable from each disclosure alone (Lemma 4.3). The identity fails for arbitrary disclosures and underlies every reuse bound in this paper. It reduces reuse to a coverage exponent λ_b : the probability that b independent signatures reveal enough to complete another encoding is $2^{-\lambda_b}$. Positive evaluation and roots-of-unity filtering of a counting polynomial bound λ_b from below without sampling (Section 4.2).
3. *Few-time unforgeability.* In the classical random-oracle model, with an ideal uniform selector, an adversary making Q random-oracle queries and at most b signing queries forges with probability at most $(Q + 1)2^{-\lambda_b} + ((d + 1)(Q + V) + L)2^{-8n}$, where d is the largest fan-in, V the verification cost, L the largest frontier and n the hash length in bytes (Theorem 4.5). The coverage term is the mean $2^{-\lambda_b}$; the proof needs no concentration bound over the signed messages. In the +C variant, the coverage term gains the factor 2^{-g} , where g is the grinding gap (Theorem 4.10).
4. *The CUTS construction.* We specify CUTS and its counter variant CUTS+C as instances of the generic scheme with $d = 2$, prove correctness, and count hash calls exactly: a signature has $(L + 1)n$ bytes, verification makes $V = (L - 1)(2w - 1) - S$ calls, and the uncached signer makes $G - V$ calls, where G is the cost of key generation (Theorem 5.2). We count the CUTS encoding family exactly and give canonical unranking algorithms that biject an integer rank onto it (Theorem 5.6). The generic theorems then bound forgery against CUTS with an ideal selector (Corollary 5.3) and against CUTS+C directly (Corollary 5.26).
5. *Exact two-use coverage for CUTS.* We express the number of covered triples as one coefficient of a polynomial defined by a recurrence of depth h (Theorems 5.17 and 5.18),

and evaluate the generic bounds on it (Theorems 5.19 and 5.20). A single signature covers no encoding other than its own (Proposition 5.16), and a dynamic program counts the covered encodings for any fixed set of disclosures (Proposition 5.22).

6. *Parameters.* We give ten parameter sets for CUTS and CUTS+C and compare them with WOTS+C and FORS (Section 5.5). Each set has a proved lower bound on λ_2 ; for example, a CUTS set with 1 424-byte signatures and 607 verification calls has $\lambda_2 \geq 128.6$ (Table 2).
7. *Artifacts.* A reference implementation of CUTS and CUTS+C, together with the scripts that generate every table, the λ_2 certificates and the test vectors in this paper, will be released at github.com/p-11/cuts.

Scope. All security statements are classical. Union preservation, the coefficient bounds and the two forgery theorems hold for every scheme in the class; the counting, the exact coverage recurrence and the numerical bounds are specific to CUTS, and the recurrence is stated for two signatures. The base theorem treats the selector as one ideal oracle, so applying it to the rejection sampler of CUTS leaves a query-accounting step, which we list as open. The +C theorem covers CUTS+C directly. It does not cover WOTS+C, which accepts a digest by its digit sum, so the grinding gap of WOTS+C rests on the heuristic argument of SPHINCS+C [10].

Organization. Section 2 reviews related work and Section 3 fixes notation and the security notions. Section 4 defines the generic tree-chain scheme, proves union preservation and the forgery bound, and treats the +C variant. Section 5 specializes to CUTS: the scheme and its costs, the encoding family and its unranking, the exact two-use coverage, CUTS+C, and example parameter sets compared with WOTS+C and FORS. Section 6 discusses quantum adversaries, Section 7 outlines applications and open problems, and Section 8 concludes. Appendices A and B prove the two security theorems of Section 4, and Appendices C, D and E give the serialization and domain separation of the reference implementation, a worked encoding, and test vectors.

2 Related work

Winternitz chains and constant-sum encodings. Merkle’s tree authentication and the Winternitz one-time signature are the starting point of the line [1]. A Winternitz signature reveals one value per hash chain, at a position given by a digit of the message digest. A checksum over the digits stops a forger from advancing message positions, because the checksum would then have to decrease, and WOTS+ builds the chains from a keyed function with per-position masks [4]. A constant-sum encoding removes the checksum instead. Every codeword has the same digit sum, so no codeword dominates another, and verification costs the same for every message. Bos and Chaum introduced such encodings for binary digits [14]. Cruz, Yatani, and Kaji used one in a Winternitz scheme with a probabilistic encoding [15], and Perin et al. gave an efficient deterministic encoding [9]. Zhang, Cui, and Yu proved that constant-sum WOTS+ is size-optimal among tree-based one-time signatures, and applied it to SPHINCS+ and XMSS [8]. Khovratovich, Kudinov, and Wagner map messages into the top layers of the hypercube of digit vectors, each layer a fixed-sum set, and obtain better size-time trade-offs [16]. WOTS+C reaches a constant sum without an encoding map: the signer searches for a counter until the message digest itself has the target digit sum, so the checksum is no longer signed [10].

The decoration of a CUTS encoding is a constant-sum vector in this sense, and Algorithm 5 uses the same first-digit recursion as the deterministic encodings above [8,9]. What CUTS adds is the frontier. A Winternitz key fixes its chains in advance, whereas a CUTS message also

chooses which L nodes of a single tree are opened. Both modes rejection-sample a rank and unrank it exactly (Section 5.2); CUTS+C, like WOTS+C, also carries the accepted counter in the signature (Section 5.4).

Few-time signatures. HORS signs by revealing a message-chosen subset of secret values, and its security degrades gradually as more signatures are published [7]. FORS, the few-time component of SPHINCS+ and SLH-DSA, splits this subset across k Merkle trees and authenticates one leaf of each tree with its authentication path [3, 5]. FORS+C grinds the message until the last tree opens its first leaf, so that tree’s authentication path is not sent [10]. PORS+FP, an alternative to FORS built on a single Merkle tree (PORS [17]), grinds until the pruned authentication set of the opened leaves has at most a fixed number of nodes [18]. CUTS is also a few-time scheme with a message-chosen disclosure, but the disclosure is a frontier of one tree. The frontier values determine the root by themselves, so a signature carries no authentication path, and fixing L and S fixes both signature length and verification work. Section 5.3 computes what two such disclosures reveal together.

Graph-based one-time signatures. Bleichenbacher and Maurer described one-time signatures by directed acyclic graphs: the secret values sit at the sources, the public key is computed along the edges, and a signature reveals a set of nodes from which the public key can be recomputed [11]. They then characterized the optimal tree-based schemes in this model [12]. Hevia and Micciancio gave security proofs for graph-based one-time signatures from properties of the underlying functions [13]. The generic scheme of Section 4 is a tree-shaped instance of this view, studied for few-time rather than one-time use: its question is what several disclosures reveal together, and its answer is the union identity of Lemma 4.3.

Security proofs. SPHINCS+ is analyzed through tweakable hash functions [5]. Kudinov, Kiktenko, and Fedorov found a flaw in the tight proof of SPHINCS+, traced to its WOTS+ component [19], and separately updated the original W-OTS+ security proof [20]. Hülsing and Kudinov recovered a tight bound for SPHINCS+, losing a factor of w , by proving WOTS-TW secure against non-adaptive chosen-message attacks [21]. Barbosa et al. verified the Hülsing–Kudinov proof in EasyCrypt [22]. The addressed interfaces of Section 5.1.3 have the same shape as these tweakable hashes, but our security theorems (Sections 4.3 and 4.4) treat the public interfaces as classical random oracles. They bound few-time forgery for every scheme in the tree-chain class of Section 4, and hence for CUTS and CUTS+C, rather than for one scheme at a time.

Stateful use and SHRINCS. XMSS authenticates many one-time keys under one Merkle root, and the signer must keep state so that no key signs twice [2]. SHRINCS, a draft signature scheme for Bitcoin, combines such a stateful tree of WOTS+C keys with an SLH-DSA instance as a stateless fallback for when state is lost; its specification notes that reusing a state value can let any observer forge [6]. Kudinov and Nick survey SPHINCS+ variants for Bitcoin and sketch the combination of an unbalanced XMSS tree with a stateless scheme that SHRINCS builds on [23]. A CUTS+C key could take the place of the WOTS+C key in such a design, since Corollary 5.26 bounds forgery after an accidental second signature. Section 7.1 outlines this use, and Section 5.5.3 compares sizes, verification cost and two-use exponents with WOTS+C, FORS, and the SHRINCS one-time key.

3 Preliminaries

For an integer $k \geq 1$, write $[k] = \{0, \dots, k-1\}$. All logarithms are binary unless specified otherwise. For a polynomial P , the notation $[x^a z^b]P$ denotes the coefficient of the monomial $x^a z^b$ in P , and likewise for more variables. The hash length n is measured in bytes, and $\Sigma = \{0, 1\}^{8n}$ is the set of n -byte words. For integers $a \geq 0$ and $d > 0$, the operation $\text{divmod}(a, d)$ returns $(\lfloor a/d \rfloor, a \bmod d)$. All counting and rank arithmetic uses mathematical integers, not machine-word arithmetic.

The schemes in this paper are built from two kinds of function: keyed functions of the secret seed $\text{SK} \in \Sigma$, which derive leaf values and the message randomizer, and public hash functions, which compute the rest of the tree and select an encoding from the message. The generic scheme of Section 4 states their interfaces abstractly, and Section 5.1.3 gives them for CUTS. This section defines the notions used to model them.

3.1 Pseudorandom functions

The keyed functions are domain-separated calls to one pseudorandom function. Let $\text{PRF}: \Sigma \times \{0, 1\}^* \rightarrow \Sigma$ be keyed by a uniform $\text{SK} \in \Sigma$. A distinguisher $\mathcal{A}$ is given oracle access either to $\text{PRF}(\text{SK}, \cdot)$ or to a uniform random function $\mathcal{R}$ of the same domain and codomain, and it outputs a bit. Its advantage is

$$\text{Adv}^{\text{prf}}(\mathcal{A}) = |\Pr[\mathcal{A}^{\text{PRF}(\text{SK}, \cdot)} = 1] - \Pr[\mathcal{A}^{\mathcal{R}} = 1]|.$$

The first probability is over SK and the coins of $\mathcal{A}$. The second is over $\mathcal{R}$ and the coins of $\mathcal{A}$. A leaf call and a randomizer call are distinct inputs; Appendix C records one encoding of CUTS that makes the distinction injective.

The security theorems of Sections 4.3 and 4.4 model the keyed functions as random functions. A reduction may replace them by a pseudorandom function and charge the difference to Adv^{prf} ; this paper does not prove that step.

3.2 Random oracles

In the classical random-oracle model used by Sections 4.3 and 4.4, each public function is an independent uniform random function on its domain-separated inputs. The adversary queries these functions on inputs of its choice and receives the outputs. A repeated input returns the same output. An instantiation realizes the public functions as domain-separated calls to one hash, as with the addressed hashes of SLH-DSA [3]; in the random-oracle model each domain is an independent random function. Theorems 5.7 and 5.24 use the special case in which distinct counter inputs to the CUTS selector hash are independent and uniform.

Queries in this definition are classical. Quantum queries are a different model. Section 6 states the generic quantum search costs and records that a quantum random-oracle reduction is open.

4 A generic tree-chain signature scheme

This section defines a class of hash-based signatures whose keys are trees of hash functions and whose signatures reveal the values on a message-selected frontier. A run of single-child nodes in the tree is a hash chain. We specify the scheme and its costs, show that the nodes derivable from several signatures are the union of those derivable from each, bound forgery in the classical random-oracle model by the resulting coverage exponent, and extend the bound to a signer that grinds a counter. CUTS and CUTS+C are instances (Section 5).

4.1 Scheme description

4.1.1 Trees, disclosures, and function interfaces

Let T be a finite rooted ordered tree with root o and at least one edge. Every internal node v has a fixed ordered list of children

$$\text{ch}(v) = (v_1, \dots, v_{d_v}), \quad d_v \geq 1.$$

Write $\mathcal{V}$ for the set of nodes, $\mathcal{V}_{\text{leaf}}$ for the leaf nodes and $\mathcal{V}_{\text{int}}$ for the internal nodes. Let $d := \max_{v \in \mathcal{V}_{\text{int}}} d_v$. Let $\text{Fr}(T)$ be the set of *frontiers*, namely

$$\text{Fr}(T) := \{E \subseteq \mathcal{V} : |E \cap P| = 1 \text{ for every root-to-leaf path } P\},$$

where a path is identified with its set of nodes. Let $\mathcal{E} \subseteq \text{Fr}(T)$ be a nonempty family of frontiers, and let $M = |\mathcal{E}|$. For $E \in \mathcal{E}$, let $\text{ord}(E) = (e_1, \dots, e_\ell)$ be its nodes listed in depth-first preorder using the prescribed child order. Also let

$$\text{Anc}^+(A) := \{v \in \mathcal{V} : v \text{ lies on the root-to-}a \text{ path for some } a \in A\},$$

including the endpoints.

Words are elements of $\Sigma = \{0, 1\}^{8n}$ (Section 3), each represented by exactly n bytes, and $\mathcal{M}$ is the set of finite byte strings. A scheme in this framework is specified by $\Theta = (n, T, \mathcal{E}, \mathcal{H})$, where

$$\mathcal{H} = ((F_v)_{v \in \mathcal{V}_{\text{leaf}}}, (H_v)_{v \in \mathcal{V}_{\text{int}}}, R, G)$$

consists of deterministic functions with interfaces

$$\begin{aligned} F_v &: \Sigma \longrightarrow \Sigma & (v \in \mathcal{V}_{\text{leaf}}), \\ H_v &: \Sigma^{d_v} \longrightarrow \Sigma & (v \in \mathcal{V}_{\text{int}}), \\ R &: \Sigma \times \Sigma \times \mathcal{M} \longrightarrow \Sigma, \\ G &: \Sigma \times \Sigma \times \mathcal{M} \longrightarrow \mathcal{E}. \end{aligned}$$

The functions are deterministic in the construction. In the security model, their separately indexed interfaces are replaced by mutually independent random functions. A concrete realization using a common hash specifies domain separation and injective input encodings.

4.1.2 Key generation, signing, and verification

All algorithms have the fixed public parameters Θ as implicit input. Key generation returns (SK, pk) , where SK is the secret key. The signer retains SK and pk .

For seed $\text{SK} \in \Sigma$, define canonical labels bottom-up:

$$x_v = F_v(\text{SK}) \quad (v \text{ leaf}), \quad x_v = H_v(x_{v_1}, \dots, x_{v_{d_v}}) \quad (v \text{ internal}). \quad (1)$$

The public key is $\text{pk} = x_o$.

```

KeyGen():
  SK = uniform n-byte string
  compute all canonical labels x[v] bottom-up
  pk = x[o]
  return (SK,pk)

Sign(SK,pk,m):
  rho = R(SK,pk,m)
  E = G(pk,rho,m)
  compute or retrieve x[v] for v in E
  return (rho, (x[v] for v in E in canonical order))

Verify(pk,m,sigma):
  reject unless pk has n bytes
  parse (rho,z[1],...,z[ell]); reject malformed n-byte words
  E = G(pk,rho,m)
  if ell != |E|: reject
  assign the supplied words to E in canonical order
  Recover(v):
    if v in E: return its supplied word
    return H_v(Recover(c) for c in ch(v), in order)
  accept exactly when Recover(o) == pk

```

The signature is the concatenation of its n -byte words; parsing rejects missing, incomplete, or extra words. The verifier does not recompute R . Repeated signing of the same message returns the same signature. Every leaf reached by `Recover` belongs to E : otherwise recovery would have stopped at the selected ancestor on its root-to-leaf path. Thus the recursive branch evaluates only internal nodes.

Proposition 4.1 (Correctness). *For arbitrary functions of the stated types, every honest signature verifies.*

Proof. Both parties derive the same frontier. Its supplied labels are canonical; upward induction in `Recover` therefore gives x_o . $\square$

Let

$$w(E) := |\text{Anc}^+(E) \setminus E|, \quad L := \max_{E \in \mathcal{E}} |E|, \quad V := \max_{E \in \mathcal{E}} w(E).$$

A signature has $(|E| + 1)n$ bytes. The public key and secret seed each have n bytes; retaining both requires $2n$ bytes. A *node-function call* evaluates an F_v or H_v ; an *internal-node hash call* evaluates an H_v only. Each evaluation counts as one call, regardless of arity. The tree-evaluation costs are

$$\begin{array}{ll}
\text{Full key generation:} & |\mathcal{V}| \text{ node-function calls,} \\
\text{Uncached signing for } E: & |\mathcal{V}| - w(E) \text{ node-function calls,} \\
\text{Verification for } E: & w(E) \text{ internal-node hash calls.}
\end{array}$$

The signing count evaluates each subtree rooted at a node of E once. These subtrees are disjoint and contain precisely the nodes outside $\text{Anc}^+(E) \setminus E$. With all selected labels cached, signing requires no node-function calls. An already known label, including the public root when $E = \{o\}$, can also be reused. Base signing additionally uses one randomizer call and one selector call; verification uses one selector call. Integer arithmetic and other non-oracle operations are excluded from these counts.

4.2 Reuse security

4.2.1 Closure and mean coverage

Definition 4.2 (Forward closure). For $A \subseteq \mathcal{V}$, let $\text{cl}(A)$ be the least set containing $A \cup \{o\}$ and containing each internal node all of whose children it contains. Closure is a set of nodes; equal labels at distinct nodes do not identify those nodes.

Lemma 4.3 (Frontier closure). *For every frontier E , $\text{cl}(E) = \text{Anc}^+(E)$. Consequently, for every $b \geq 1$ and frontiers $E_1, \dots, E_b$,*

$$\text{cl}(E_1 \cup \dots \cup E_b) = \bigcup_{i=1}^b \text{cl}(E_i). \quad (2)$$

Proof. Every subtree rooted strictly above E contains a frontier, so upward induction derives every ancestor. Conversely, the ancestor set is forward closed and contains E and o , proving equality. A union of ancestor sets is upward closed and hence forward closed; minimality and monotonicity of closure give (2). $\square$

A frontier E is *covered* by $A \subseteq \mathcal{V}$ if $E \subseteq \text{cl}(A)$. Fix an integer $b \geq 1$. For independent uniform $E_0, E_1, \dots, E_b \in \mathcal{E}$, set

$$p_b = \Pr[E_0 \subseteq \text{cl}(E_1 \cup \dots \cup E_b)], \quad (3)$$

$$\lambda_b = -\log_2 p_b. \quad (4)$$

Sampling is with replacement. Since a source covers itself, $p_b \geq 1/M > 0$. The metric averages coverage, not its negative logarithm, and is not a percentile guarantee. Section 4.3 connects this coverage calculation to adaptive unforgeability.

4.2.2 Positive evaluation and congruence filtering

Computing p_b exactly is often computationally infeasible, so we can use the following to get a lower bound for λ_b .

Let $\mathfrak{F}$ be a finite family of frontiers, equipped with a statistic $f : \mathfrak{F} \rightarrow \mathbb{Z}_{\geq 0}^{k_0}$, where $k_0 \geq 1$. Fix $\mathbf{t} \in \mathbb{Z}_{\geq 0}^{k_0}$ such that $\mathcal{E} = \{E \in \mathfrak{F} : f(E) = \mathbf{t}\}$. Introduce $b+1$ blocks of k_0 variables. For matching tuples of variables and nonnegative exponents, write $\mathbf{X}^\nu := \prod_j X_j^{\nu_j}$. Define the polynomial with nonnegative coefficients

$$\mathcal{P}_b(\mathbf{X}_0, \dots, \mathbf{X}_b) = \sum_{\substack{E_0, \dots, E_b \in \mathfrak{F} \\ E_0 \subseteq \text{cl}(E_1 \cup \dots \cup E_b)}} \prod_{i=0}^b \mathbf{X}_i^{f(E_i)}.$$

Let $k = k_0(b+1)$ and let $\mathbf{q}$ concatenate $b+1$ copies of $\mathbf{t}$. Write $[\mathbf{X}^{\mathbf{q}}]\mathcal{P}_b$ for the coefficient of the monomial $\mathbf{X}^{\mathbf{q}}$ in $\mathcal{P}_b$. Then

$$\Gamma_b = [\mathbf{X}^{\mathbf{q}}]\mathcal{P}_b, \quad p_b = \Gamma_b / M^{b+1}. \quad (5)$$

An efficient representation of $\mathcal{P}_b$ is a separate, family-specific question; the coefficient bound does not assume an efficient evaluation algorithm.

Theorem 4.4 (Coefficient bounds). *Write $\mathcal{P}_b = \sum_{\nu} c_{\nu} \mathbf{X}^{\nu}$. For positive radii $\boldsymbol{\rho} \in \mathbb{R}_{>0}^k$ and integer moduli $\mathbf{m} \in \mathbb{Z}_{\geq 1}^k$, define*

$$\mathcal{P}_{b,\mathbf{q}}^{\mathbf{m}}(\boldsymbol{\rho}) = \sum_{\nu_j \equiv q_j \pmod{m_j} \forall j} c_{\nu} \boldsymbol{\rho}^{\nu}.$$

Then

$$\lambda_b \geq (b+1) \log_2 M + \sum_{j=1}^k q_j \log_2 \rho_j - \log_2 \mathcal{P}_{b,q}^m(\rho). \quad (6)$$

With $\omega_j = e^{2\pi i/m_j}$, the filtered value is

$$\frac{1}{\prod_j m_j} \sum_{0 \leq t_j < m_j \forall j} \left(\prod_j \omega_j^{-t_j q_j} \right) \mathcal{P}_b(\rho_1 \omega_1^{t_1}, \dots, \rho_k \omega_k^{t_k}). \quad (7)$$

All $m_j = 1$ gives the positive-evaluation bound.

Proof. The coefficient Γ_b counts precisely the covered ordered tuples whose $b+1$ statistics all equal $\mathbf{t}$. There are M^{b+1} equally likely ordered tuples from $\mathcal{E}$, proving (5). In particular $\Gamma_b > 0$. The desired exponent belongs to every specified residue class, and all coefficients and radii are nonnegative and positive, respectively. Hence

$$\Gamma_b \rho^{\mathbf{q}} \leq \mathcal{P}_{b,q}^m(\rho).$$

Divide by $M^{b+1} \rho^{\mathbf{q}}$ and apply $-\log_2$, which reverses the inequality, to obtain (6).

For an integer a , the finite geometric-sum identity gives

$$\frac{1}{m_j} \sum_{t=0}^{m_j-1} \omega_j^{ta} = \begin{cases} 1, & m_j \mid a, \\ 0, & m_j \nmid a. \end{cases}$$

If $m_j \mid a$, every summand is one; otherwise the ratio $\omega_j^a \neq 1$ has m_j th power one, so the geometric sum vanishes. Expanding the polynomial in (7) and interchanging finite sums therefore yields

$$\begin{aligned} \sum_{\nu} c_{\nu} \rho^{\nu} \prod_{j=1}^k \left(\frac{1}{m_j} \sum_{t=0}^{m_j-1} \omega_j^{t(\nu_j - q_j)} \right) \\ = \sum_{\nu: \nu_j \equiv q_j \pmod{m_j} \forall j} c_{\nu} \rho^{\nu} = \mathcal{P}_{b,q}^m(\rho). \end{aligned}$$

For $m_j = 1$ in every coordinate, no monomial is removed, giving the positive-evaluation bound. $\square$

If $m_j \mid m'_j$ coordinatewise, the $\mathbf{m}'$ filter is at least as strong at fixed radii. Arbitrary increases need not improve it. Moduli exceeding both the coordinate degrees and the corresponding q_j isolate the exact coefficient, but can require an impractical Fourier grid. Every positive choice of radii is valid; optimization only improves the bound.

4.3 Adversarial security in the classical random-oracle model

Model F_v, H_v, R, G as independent random functions of their declared types. Each fresh selector output is uniform in $\mathcal{E}$. The experiment generates one key, gives $\mathbf{pk}$ to the adversary, and permits at most b signing queries and Q public oracle queries, interleaved adaptively. Repetitions count. Oracle access is classical. Honest key generation, signing, and final verification do not consume Q . A final pair (m^*, σ^*) wins exactly when m^* was never submitted for signing and verification accepts. Write $\text{Adv}_{\leq b}^{\text{EUF-CMA}}(Q)$ for the supremum of this probability over adversaries with unrestricted offline computation. One ideal selector call counts as one query. Appendix A.1 states this experiment, *Real*, precisely, with adversaries as decision trees.

Theorem 4.5 (Few-time security). *For every $b \geq 1$ and $Q \geq 0$ in the stated model,*

$$\boxed{\text{Adv}_{\leq b}^{\text{EUF-CMA}}(Q) \leq \min \left\{ 1, (Q+1)p_b + \frac{(d+1)(Q+V)+L}{2^{8n}} \right\}.} \quad (8)$$

Fix any adversary satisfying the stated query budgets. All probabilities in the following three lemmas refer to this adversary. The bound uses the mean p_b without a concentration assumption. Its proof has three parts; Appendix A gives it in full. Appendix A.2 defines a non-aborting simulation Sim with independent labels, uniform source disclosures $E_1, \dots, E_b$, and independent uniform candidate targets $E'_1, \dots, E'_{Q+1}$. Let Cov mean that the final fresh-message target lies in the signing closure. A *selector prequery* is a public query to the selector input subsequently used for signing. The private event Bad records seed guesses, selector prequeries, hidden internal-node input hits, accidental canonical outputs, and hidden payload hits.

Lemma 4.6 (Forgery decomposition). $\Pr_{\text{Real}}[\text{Win}] \leq \Pr_{\text{Sim}}[\text{Cov}] + \Pr_{\text{Sim}}[\text{Bad}]$.

Proof sketch. The games agree until a seed guess, selector prequery, or hidden input hit (Lemma A.1). In the simulation, an accepted uncovered target either supplies a hidden canonical label or produces a canonical output from a noncanonical input. Appendix A.4 proves this and combines the two statements. $\square$

Lemma 4.7 (Adaptive coverage). $\Pr_{\text{Sim}}[\text{Cov}] \leq (Q+1)p_b$.

Proof sketch. A fresh-message target cannot come from a signing-programmed entry, so it is some candidate E'_J , and each candidate is covered by the sources with probability p_b . A union bound over the $Q+1$ candidates, without conditioning on the adaptive index J , gives the claim; Appendix A.4 gives the details. $\square$

Lemma 4.8 (Exceptional events). $\Pr_{\text{Sim}}[\text{Bad}] \leq ((d+1)(Q+V)+L)2^{-8n}$.

Proof sketch. Let N_s, N_m, N_g count public seed-domain (F_v or R), selector (G), and internal-node hash (H_u) calls, respectively. The deferred-sampling argument in Appendix A.5 gives

$$\Pr[\text{Bad}] \leq 2^{-8n}(\mathbb{E}[N_s + N_m + (d+1)N_g] + (d+1)V + L).$$

Use $N_s + N_m + N_g \leq Q$. The d input slots and one output explain $d+1$; each selector prequery is charged to at most one first signing. $\square$

Proof of Theorem 4.5. Combine the three lemmas, take the supremum over adversaries, and cap at one. $\square$

4.4 The +C variant: counter-based selection

4.4.1 Signing, verification, and costs

Keep the tree, disclosure family, and label functions. Fix a bijection $\text{Unrank}_{\mathcal{E}} : [M] \rightarrow \mathcal{E}$, an index width $r \geq 1$ with $M \leq 2^r$, and a counter width $s \geq 1$. Replace the base selector by

$$G^{\text{C}} : \Sigma \times \Sigma \times \mathcal{M} \times [2^s] \rightarrow [2^r].$$

The raw index function always returns an integer. Only the bounded signing search may return the failure symbol $\perp$. Set

$$\alpha = M/2^r, \quad g = -\log_2 \alpha = r - \log_2 M, \quad C_{\max} = 2^s. \quad (9)$$

```

SignC(SK,pk,m):
  rho = R(SK,pk,m)
  for c = 0, ..., 2^s-1:
    u = G^C(pk,rho,m,c)
    if u < M:
      E = Unrank(u)
      compute or retrieve x[v] for v in E
      return (rho,c,(x[v] for v in E in canonical order))
  return failure

VerifyC(pk,m,sigma):
  parse (rho,c,z[1],...,z[ell]); reject malformed fields
  reject unless pk,rho,z[i] have n bytes and 0 <= c < 2^s
  u = G^C(pk,rho,m,c)
  if u >= M: reject
  E = Unrank(u)
  if ell != |E|: reject
  reconstruct from E and the supplied words as in Verify
  accept exactly when the recovered root equals pk

```

Let $\text{enc}_s(c)$ denote the encoding of $c \in [2^s]$ in exactly $\lceil s/8 \rceil$ unsigned big-endian bytes. Encode a signature as $\rho \parallel \text{enc}_s(c) \parallel z_1 \parallel \dots \parallel z_\ell$. Reject missing or extra bytes and counter values outside $[2^s]$, including nonzero unused high bits. Verification tests only the supplied counter, not whether earlier counters were rejected. A successful signature has $(|E| + 1)n + \lceil s/8 \rceil$ bytes. Tree costs and key sizes are unchanged; verification uses one index call. Repeated signing returns the same signature or the same $\perp$. For a successful selection E , signing has the same node-function cost as in the base scheme, plus one randomizer call and the raw index calls made by the counter search. If the search exhausts all counters, no selected labels are computed. The expected number of index calls is given below.

Proposition 4.9 (Selection and correctness). *Every successful honest signature verifies. For a transcript (pk, ρ, m) independent of its raw index outputs, if these outputs are independent uniform in $[2^r]$, then*

$$\varepsilon_{\text{ex}} := \Pr[\perp] = (1 - \alpha)^{C_{\text{max}}}, \quad \mathbb{E}[\text{index calls}] = \frac{1 - \varepsilon_{\text{ex}}}{\alpha}. \quad (10)$$

Conditional on success the disclosure is uniform in $\mathcal{E}$, independently of the returned counter. In particular, $\alpha C_{\text{max}} \geq \eta \ln 2$ suffices for $\varepsilon_{\text{ex}} \leq 2^{-\eta}$.

Proof. Correctness follows because both parties repeat the same accepting index call and root reconstruction. For each $e \in \mathcal{E}$ and $0 \leq c < C_{\text{max}}$,

$$\Pr[\text{return } (c, e)] = (1 - \alpha)^c 2^{-r}.$$

Summing gives the distribution claims. Counter c is tried exactly when the preceding c trials were rejected, giving the geometric sum in (10). Finally, $(1 - \alpha)^{C_{\text{max}}} \leq e^{-\alpha C_{\text{max}}}$. $\square$

The cost counts all C_{max} trials on exhaustion. An exhaustion target is an availability constraint; the security theorem itself does not need it. The proposition concerns fresh rows; adaptive prequeries are handled next.

4.4.2 Coverage and adversarial security

The quantities p_b, λ_b still describe the same disclosure family. For fixed sources, a fresh raw index is accepted and covered with probability $|\{E \in \mathcal{E} : E \subseteq \text{cl}(E_1 \cup \dots \cup E_b)\}|/2^r$. Thus

define the raw-trial metrics

$$p_b^C = \alpha p_b, \quad \lambda_b^C = -\log_2 p_b^C = g + \lambda_b. \quad (11)$$

Every lower bound on λ_b therefore increases by g in this metric. This changes the probability per raw trial, not the coverage conditional on an accepted index.

In the counter-mode real experiment, G^C is an independent uniform random function into $[2^r]$. Each public raw index call counts toward Q ; honest search calls do not. The signing budget counts requests, including those returning $\perp$, and a forged message must never have been submitted for signing. All other rules are those of Section 4.3.

Theorem 4.10 (Counter-mode security). *For every $b \geq 1$ and $Q \geq 0$ in this classical random-oracle model,*

$$\text{Adv}_{\leq b}^{\text{EUF-CMA,C}}(Q) \leq \min \left\{ 1, (Q+1)\alpha p_b + \frac{(d+1)(Q+V)+L}{2^{8n}} \right\}. \quad (12)$$

Exhaustion is simulated exactly, so no separate $b\varepsilon_{\text{ex}}$ term is required.

Proof outline. Use the base simulation with independent uniform raw indices in $[2^r]$ as candidate targets. Simulate each signing search with its exact stopping counter and exhaustion distribution. Each fresh candidate is accepted and covered with probability αp_b , so the coverage term is at most $(Q+1)\alpha p_b$. The exceptional-event charges are unchanged. Combining these bounds proves the claim; Appendix B gives the full simulation and coupling argument. $\square$

5 CUTS

CUTS is the generic scheme of Section 4 on one binary tree whose non-root nodes carry hash chains of equal length. A message selects a frontier of the binary tree and a position on the chain of each selected node, with the positions summing to a fixed value. This section specifies the scheme and its exact costs (Section 5.1), counts and unranks its encoding family (Section 5.2), computes its two-use coverage exponent (Section 5.3), defines the counter variant CUTS+C (Section 5.4), and gives concrete parameter sets (Section 5.5).

5.1 Scheme description and costs

CUTS is the generic scheme of Section 4 with a particular tree, particular functions, a particular encoding family and a selector that rejection-samples a message hash. We fix each in turn. The key generation, signing and verification algorithms are then those of Section 4, implemented without building the tree explicitly, and correctness, costs and security follow from the generic statements. The counter variant CUTS+C changes only the selector and the signature layout (Section 5.4).

5.1.1 Parameters

The public parameter tuple is $\theta = (n, h, w, L, S)$, satisfying

$$1 \leq n, w < 2^{64}, \quad 1 \leq h \leq 63, \quad 2 \leq L \leq 2^h, \quad 0 \leq S \leq L(w-1), \quad S < 2^{64}.$$

CUTS+C adds an index width r and a counter width s to θ (Section 5.4). The 64-bit restrictions specify an address-compatible profile; the combinatorial results hold more generally whenever the corresponding finite tree and family are defined. No computation of the family size M (Section 5.2) is restricted to 64 bits.

5.1.2 The tree

Let T_h be the complete rooted binary tree of height h , in heap order, with nodes $V_h = \{1, \dots, 2^{h+1} - 1\}$, root 1, and children $2v, 2v + 1$. The leaves are $2^h, \dots, 2^{h+1} - 1$. Write $u \prec v$ when u is a strict ancestor of v , and let $T(v)$ be the subtree rooted at v .

The tree of the generic scheme is T_h with a chain of w positions on every non-root node. Its nodes are the root 1 and the pairs (v, j) , for non-root $v \in V_h$ and $j \in [w]$. The parent of (v, j) is $(v, j + 1)$ when $j < w - 1$; the parent of $(v, w - 1)$ is $(\lfloor v/2 \rfloor, 0)$, or the root when $\lfloor v/2 \rfloor = 1$. The leaves are the pairs $(v, 0)$ for leaves v of T_h . Thus each pair $(v, 0)$ with v internal, and the root, has two children, which are the chain ends of $2v$ and $2v + 1$; every other internal node $(v, j + 1)$ has the single child (v, j) . The largest fan-in is $d = 2$, and the tree has $(2^{h+1} - 2)w + 1$ nodes. A chain with w positions has $w - 1$ steps; the root has no chain.

5.1.3 The functions

CUTS uses the following deterministic interfaces. All inputs shown are bound to their operation domain; the notation denotes structured inputs, not ambiguous string concatenation.

$$\begin{aligned} F_{\text{leaf}}(\text{SK}; \theta, v) &\in \Sigma, & F_{\text{rand}}(\text{SK}; \theta, \text{pk}, m) &\in \Sigma, \\ H_{\text{merge}}(\theta, v, x_L, x_R) &\in \Sigma, & H_{\text{chain}}(\theta, v, j, x) &\in \Sigma, \\ H_{\text{msg}}(\theta, c, \text{pk}, \rho, m; \beta) &\in \{0, 1\}^{8\beta}. \end{aligned}$$

Here the secret seed is $\text{SK} \in \Sigma$, m is a finite byte string, and $x, x_L, x_R, \text{pk}, \rho \in \Sigma$. The node v is a leaf for F_{leaf} , an internal node (including the root) for H_{merge} , and a non-root node for H_{chain} , whose step index satisfies $0 \leq j < w - 1$. The counter c lies in $[2^{64}]$, and the output length β , in bytes, is public (Section 5.2). Leaf and randomizer derivation are domain-separated uses of one keyed function (Section 3.1). Merge, chain, and selector inputs use distinct operation domains of one hash (Section 3.2), with injective encodings of their parameters and arguments. Appendix C specifies the SHAKE256 instantiation used by the reference implementation.

The functions of the generic scheme are

$$\begin{aligned} F_{(v,0)} &= F_{\text{leaf}}(\cdot; \theta, v), & H_{(v,0)} &= H_{\text{merge}}(\theta, v, \cdot, \cdot), \\ H_{(v,j+1)} &= H_{\text{chain}}(\theta, v, j, \cdot), & H_1 &= H_{\text{merge}}(\theta, 1, \cdot, \cdot), \end{aligned}$$

and $R = F_{\text{rand}}(\cdot; \theta, \cdot, \cdot)$. Writing $X_{v,j}$ for the canonical label (1) of (v, j) , the labels satisfy

$$\begin{aligned} X_{v,0} &= F_{\text{leaf}}(\text{SK}; \theta, v) && \text{if } v \text{ is a leaf,} \\ X_{v,0} &= H_{\text{merge}}(\theta, v, X_{2v,w-1}, X_{2v+1,w-1}) && \text{if } v \neq 1 \text{ is internal,} \\ X_{v,j+1} &= H_{\text{chain}}(\theta, v, j, X_{v,j}) && \text{for } 0 \leq j < w - 1, \end{aligned}$$

and the public key is the root label

$$\text{pk} = H_{\text{merge}}(\theta, 1, X_{2,w-1}, X_{3,w-1}). \quad (13)$$

The secret key is (SK, pk) ; storing pk avoids recomputing the root when signing.

5.1.4 Encodings and selection

A root-to-leaf path of the tree runs through the whole chain of every node on a root-to-leaf path of T_h . A frontier other than $\{1\}$ therefore selects one position $(v, A(v))$ on the chain of each node v of a frontier F of T_h that excludes the root, and every such pair (F, A) gives a frontier. We call (F, A) a *decorated frontier*. The CUTS encoding family $\mathcal{E}_{h,w,L,S}$ consists of the decorated frontiers with $|F| = L$ and $\sum_v A(v) = S$ (Definition 5.4), so every encoding has L

nodes and L is the largest frontier size of the generic scheme. Section 5.2 counts the family and ranks it.

The selector G is **Select** (Algorithm 6): it hashes the transcript with H_{msg} , rejection-samples a rank in $[M]$, and unranks it. When it succeeds, its output is uniform in $\mathcal{E}_{h,w,L,S}$ (Theorem 5.7).

5.1.5 Key generation, signing, and verification

Algorithms 1, 2 and 3 implement the generic algorithms for this instance without storing the tree. For a non-root node v of T_h , $\text{Start}(\text{SK}, v)$ computes the label $X_{v,0}$ and $\text{End}(\text{SK}, v)$ the label $X_{v,w-1}$. Calls for selected nodes have disjoint subtrees.

Algorithm 1 Tree generation and $\text{KeyGen}(\theta)$

```

1  Start(SK,v):                                // requires v != 1
2      if v >= 2^h:
3          return F_leaf(SK;theta,v)
4      left  = End(SK,2*v)
5      right = End(SK,2*v+1)
6      return H_merge(theta,v,left,right)
7
8  End(SK,v):
9      z = Start(SK,v)
10     for j = 0, ..., w-2:
11         z = H_chain(theta,v,j,z)
12     return z
13
14  KeyGen(theta):
15     SK = uniform n-byte string
16     left  = End(SK,2)
17     right = End(SK,3)
18     pk = H_merge(theta,1,left,right)
19     return (secret_key=(SK,pk), public_key=pk)

```

Algorithm 2 $\text{Sign}((\text{SK}, \text{pk}), m)$

```

1  rho = F_rand(SK;theta,pk,m)
2  encoding = Select(pk,rho,m)
3  if encoding is failure:
4      return failure
5  (F,A) = encoding
6  for i = 0, ..., L-1:
7      z = Start(SK,F[i])
8      for j = 0, ..., A[i]-1:
9          z = H_chain(theta,F[i],j,z)
10     Z[i] = z
11  return (rho,Z[0],...,Z[L-1])

```

Loops with upper endpoint below their lower endpoint are empty. In particular, $w = 1$ causes no chain steps. A successful signature is the unambiguous concatenation of $L + 1$ strings of length n bytes. The frontier addresses and positions are not transmitted. Signing is deterministic once the seed, primitives, parameters, and message are fixed.

Why the randomizer is secret. Selection depends on the message only through $H_{\text{msg}}(\theta, c, \text{pk}, \rho, m)$, and the signer derives ρ with the secret key. Without ρ , or with a public

one, anyone could compute the encoding of every message offline. An adversary could then search for two harmless-looking messages whose encodings together cover a large part of $\mathcal{E}$, together with a target message whose encoding lies in that cover, have the two messages signed, and assemble a forgery from the disclosed values. Exponent λ_2 of Section 5.3 is an average over independent uniform encodings, and it says nothing about such chosen pairs. With a secret ρ , the adversary cannot predict which encoding a signing query will disclose, and each disclosed encoding is uniform in $\mathcal{E}$ and independent of the others, as that section assumes. Corollary 5.3 models F_{rand} as a random function; replacing it by a pseudorandom function would cost a PRF advantage (Section 3.1), and that step is not proved here.

Deriving ρ deterministically from the key and the message, as SLH-DSA does with PRF_{msg} [3], has two further consequences: signing needs no fresh randomness, and signing the same message twice returns the same signature, so it does not count as a second use of the key.

For a selected node v_i , verification advances the received Z_i to its endpoint. At unselected ancestors it merges child endpoints and advances the resulting chain, except at the global root. A dictionary maps selected nodes to their signature positions; all other topology comes from unranking. Figure 1 shows one signature and the calls its verification makes.

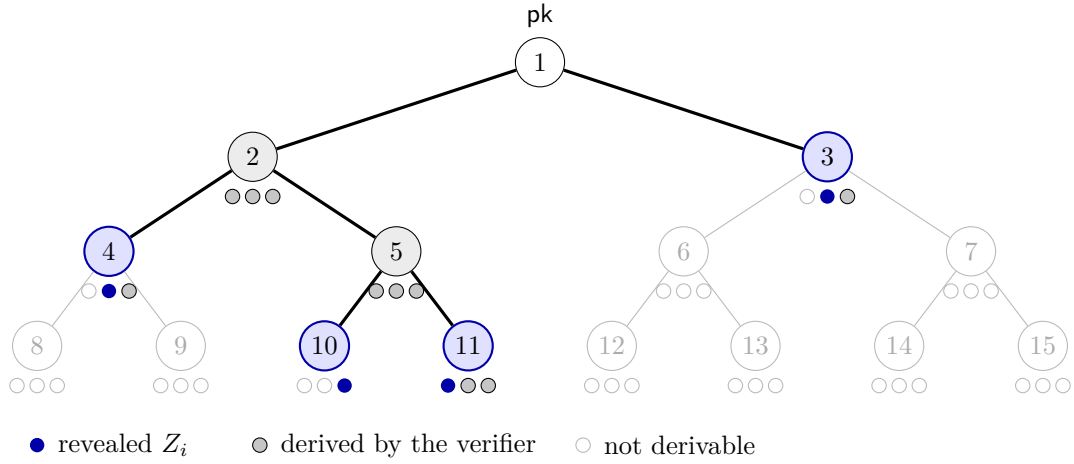


Figure 1: A CUTS signature for $(h, w, L, S) = (3, 3, 4, 4)$: rank 68 of $M = 95$, with frontier $F = (4, 10, 11, 3)$ and positions $A = (1, 2, 0, 1)$. Each non-root node has a chain of $w = 3$ positions, drawn left to right under it. The signature reveals one value per selected node (blue) and no authentication path. The verifier advances each to the end of its chain, merges and advances at the reconstructed nodes 5 and 2 (gray), and merges at the root: $V = (L - 1)(2w - 1) - S = 11$ tree primitive calls along the thick edges. Values below the frontier, and positions before a revealed one, stay hidden.

Completeness of the decoded frontier ensures that recovery stops before it would recurse below a leaf.

The verifier cannot test that ρ was produced by F_{rand} , so the randomizer of a forgery is the adversary's choice. This does not undo the protection above: choosing ρ only lets the adversary re-sample the encoding of its target message, at one H_{msg} call per try, after the signed encodings are fixed. The term $(Q + 1)p_b$ of Corollary 5.3 accounts for this search.

Theorem 5.1 (Correctness). *For any deterministic primitives of the stated types, valid parameters, key pair output by KeyGen, and message m for which signing succeeds,*

$$\text{Verify}(\text{pk}, m, \text{Sign}((\text{SK}, \text{pk}), m)) = 1.$$

Proof. Signing returns $Z_i = X_{v_i, a_i}$, the canonical labels of the selected nodes, and verification

Algorithm 3 Advance, root recovery, and Verify(pk, m, σ)

```
1 Advance(v, a, z):
2   for j = a, ..., w-2:
3     z = H_chain(theta, v, j, z)
4   return z
5
6 Recover(v):                                // uses selected endpoints E
7   if v is selected:
8     return E[v]
9   left = Recover(2*v)
10  right = Recover(2*v+1)
11  z = H_merge(theta, v, left, right)
12  if v == 1:
13    return z
14  return Advance(v, 0, z)
15
16 Verify(pk, m, sigma):
17   reject unless pk has n bytes and sigma has (L+1)*n bytes
18   parse sigma as (rho, Z[0], ..., Z[L-1])
19   encoding = Select(pk, rho, m)
20   reject if encoding is failure
21   (F, A) = encoding
22   for i = 0, ..., L-1:
23     E[F[i]] = Advance(F[i], A[i], Z[i])
24   return (Recover(1) == pk)
```

evaluates the generic recovery of Section 4: Advance walks a chain and Recover merges child chain ends. Apply Proposition 4.1. $\square$

5.1.6 Sizes and exact tree costs

A leaf derivation, one chain step, and one binary merge each count as one *tree primitive call*; these are the node-function calls of Section 4 for this tree. Counts exclude message randomization, selection, integer arithmetic, and memory operations. Calls may have different input lengths and costs; these are operation counts, not compression counts or timings.

Theorem 5.2 (Sizes and operation counts). *The public key has n bytes, the stored secret key has $2n$ bytes, and each successful signature has $(L + 1)n$ bytes. Full key generation uses*

$$G = 2^h + (2^{h+1} - 2)(w - 1) + (2^h - 1) = (2^{h+1} - 2)w + 1 \quad (14)$$

tree primitive calls. For every signature of the correct length whose selection succeeds, verification uses exactly

$$V = L(w - 1) - S + (L - 1) + (L - 2)(w - 1) = (L - 1)(2w - 1) - S \quad (15)$$

tree primitive calls, whether or not it accepts. The uncached signer of Algorithm 2 uses exactly

$$G_{\text{sign}} = 2^h + (2^h - L) + (2^{h+1} - 2L)(w - 1) + S = G - V \quad (16)$$

tree primitive calls. These counts exclude message randomization, selection, and unranking.

Proof. By Section 4, key generation evaluates every node of the tree once, verification evaluates the nodes of $\text{Anc}^+(E) \setminus E$, and the uncached signer evaluates the rest, each subtree rooted at a selected node once. The tree has 2^h leaves, $(2^{h+1} - 2)(w - 1)$ chain steps and $2^h - 1$ merges,

proving (14). For an encoding (F, A) , the set $\text{Anc}^+(E) \setminus E$ consists of the positions above $A(v)$ on each selected chain, $L(w-1) - S$ in all, and the merges and chains of the strict ancestors of F . The pruned tree above F is a full binary tree with $L-1$ internal nodes, including the root, so there are $L-1$ merges and $L-2$ ancestor chains of $w-1$ steps, giving (15). The encoding fixes these calls, whatever values are received. Subtracting from G gives (16), whose first form counts the same calls directly: every leaf, the $2^h - L$ merges not above F , the $2^{h+1} - 2L$ chains neither selected nor above F , and S steps on the selected chains. $\square$

Successful signing also makes one call to F_{rand} and the number of H_{msg} calls specified by the selector. Verification makes the same selector calls. The recursion needs $O(h)$ temporary tree values, plus the L output values and public counting tables. Caching can reduce signing work; the identity $G_{\text{sign}} = G - V$ applies to the uncached routine. The tests of the reference implementation count the tree primitive calls of key generation, signing, and verification, and check them against (14), (15), and (16).

5.1.7 Security

Corollary 5.3 (Few-time security of CUTS). *In the model of Section 4.3, with **Select** replaced by an ideal selector that returns a uniform element of $\mathcal{E}_{h,w,L,S}$, for every $b \geq 1$ and $Q \geq 0$,*

$$\text{Adv}_{\leq b}^{\text{EUF-CMA}}(Q) \leq \min \left\{ 1, (Q+1)2^{-\lambda_b} + \frac{3(Q+V)+L}{2^{8n}} \right\}, \quad V = (L-1)(2w-1) - S.$$

Proof. Apply Theorem 4.5 to this instance, whose largest fan-in is $d = 2$, largest frontier is L and verification cost is V . $\square$

Theorem 5.7 shows that **Select** returns a uniform encoding when it succeeds, and that it fails with probability below $2^{-256 \cdot 2^{64}}$. The corollary counts one ideal selector call as one query (Appendix A.5), so it does not account for the several H_{msg} calls that one **Select** call can make; Section 7.2 lists that step. Forward closure in this tree is the symbolic closure of Definition 5.8, so Section 5.3 gives λ_1 exactly for CUTS and, for $b = 2$, deterministic lower bounds on λ_2 .

5.2 Encodings and unranking

Definition 5.4 (Frontier and decoration). A frontier is a set $F \subseteq V_h$ of nodes of T_h (Section 5.1.2) that intersects every root-to-leaf path exactly once, as in Section 4. Its canonical order is left to right, recursively placing all selected nodes in the left subtree before those in the right subtree. A decoration assigns $A(v) \in [w]$ to each $v \in F$. The encoding family is

$$\mathcal{E}_{h,w,L,S} = \{(F, A) : F \text{ a frontier, } |F| = L, \sum_{v \in F} A(v) = S\}.$$

An encoding can equivalently be written as ordered tuples $F = (v_0, \dots, v_{L-1})$ and $A = (a_0, \dots, a_{L-1})$.

The root lies on every root-to-leaf path, so a frontier containing it has size one. Since $L \geq 2$, no frontier in the family contains the root, which has no chain (Section 5.1.2); every $v \in F$ therefore has a chain, and A is well defined.

5.2.1 Counting

Let $N[t, \ell]$ count ℓ -node frontiers of a complete height- t subtree. Then $N[t, 1] = 1$ for $t \geq 0$, since the subtree root is the only node on every root-to-leaf path. A frontier is nonempty, and each of its nodes has a leaf below it that no other node shares, so $N[t, \ell] = 0$ for $\ell < 1$ and for $\ell > 2^t$. Let $D_w[k, s]$ count vectors in $[w]^k$ with sum s . Then $D_w[0, 0] = 1$, $D_w[0, s] = 0$ for $s \neq 0$, and $D_w[k, s] = 0$ for $s < 0$.

Lemma 5.5 (Counting recurrences). *For $t \geq 1$ and $\ell \geq 2$,*

$$N[t, \ell] = \sum_{a=\max(1, \ell-2^{t-1})}^{\min(2^{t-1}, \ell-1)} N[t-1, a]N[t-1, \ell-a].$$

For $k \geq 1$ and $s \geq 0$,

$$D_w[k, s] = \sum_{a=0}^{\min(w-1, s)} D_w[k-1, s-a].$$

Consequently,

$$M = |\mathcal{E}_{h,w,L,S}| = N[h, L]D_w[L, S]. \quad (17)$$

Proof. Let F be a frontier of a height- t subtree with root v and $|F| = \ell \geq 2$. Then $v \notin F$: any other $u \in F$ lies on a root-to-leaf path through v , which would meet F twice. Hence F is the disjoint union of $F \cap T(2v)$ and $F \cap T(2v+1)$. Every root-to-leaf path of $T(2v)$ extends to one of $T(v)$ through v , so $F \cap T(2v)$ is a frontier of $T(2v)$, and likewise for $T(2v+1)$. Conversely, the union of a frontier of $T(2v)$ and one of $T(2v+1)$ is a frontier of $T(v)$ not containing v . This gives a bijection between such F and pairs of child frontiers. If the left restriction has size a , the right has size $\ell - a$, and both sizes lie between 1 and 2^{t-1} . Summing over a gives the first recurrence. A decoration is partitioned uniquely by its first digit, giving the second. Each size- L frontier has exactly $D_w[L, S]$ decorations, proving the product formula. $\square$

The admissible parameter conditions imply $M \geq 1$. Indeed, every integer frontier size between 1 and 2^t occurs by the first recurrence and induction, and every sum between 0 and $k(w-1)$ occurs for a bounded digit vector. The selector entropy under uniform selection is $\log_2 M$.

5.2.2 Canonical unranking

The algorithms below assume that their rank lies in the specified counting range. The frontier loop runs over the index range of Lemma 5.5. The decoration loop also omits the digits $a < s - (k-1)(w-1)$, whose terms are zero because the tail sum $s - a$ exceeds $(k-1)(w-1)$. Every remaining term is positive, because every frontier size from 1 to 2^{t-1} and every sum from 0 to $(k-1)(w-1)$ occurs. So every block in either loop is nonempty, and the blocks sum to $N[t, \ell]$ and $D_w[k, s]$. When the frontier algorithm reaches $\ell = 1$, the only valid rank is zero; when the decoration algorithm reaches $k = 0$, its valid input has $s = rk = 0$.

Algorithm 4 $\text{UnrankFrontier}(v, t, \ell, rk)$, for $0 \leq rk < N[t, \ell]$

```

1  if ell == 1:
2      return (v)
3  amin = max(1, ell - 2^(t-1))
4  amax = min(2^(t-1), ell - 1)
5  for a = amin, ..., amax:
6      B = N[t-1, a] * N[t-1, ell-a]
7      if rk < B:
8          (rkL, rkR) = divmod(rk, N[t-1, ell-a])
9          FL = UnrankFrontier(2*v, t-1, a, rkL)
10         FR = UnrankFrontier(2*v+1, t-1, ell-a, rkR)
11         return concatenate(FL, FR)
12     rk = rk - B
13 fail // unreachable for a valid input

```

Algorithm 5 UnrankDecoration(k, s, rk), for $0 \leq rk < D_w[k, s]$

```

1  if k == 0:
2      return ()
3  amin = max(0, s - (k-1)*(w-1))
4  amax = min(w-1, s)
5  for a = amin, ..., amax:
6      B = D_w[k-1, s-a]
7      if rk < B:
8          return concatenate((a), UnrankDecoration(k-1, s-a, rk))
9      rk = rk - B
10 fail // unreachable for a valid input

```

For $rk \in [M]$, define

$$\begin{aligned}
 (rk_F, rk_A) &= \text{divmod}(rk, D_w[L, S]), \\
 \text{Unrank}_{\mathcal{E}}(rk) &= (\text{UnrankFrontier}(1, h, L, rk_F), \text{UnrankDecoration}(L, S, rk_A)). \quad (18)
 \end{aligned}$$

Theorem 5.6 (Encoding bijection). *Algorithms 4 and 5 terminate on their valid inputs and are bijections onto their respective counted sets. Equation (18) defines a bijection $[M] \rightarrow \mathcal{E}_{h,w,L,S}$.*

Proof. Induct on height for frontiers and on length for decorations. The valid base cases are unique. Lemma 5.5 partitions each rank interval into consecutive blocks indexed by the split size or first digit. Within a positive frontier block, quotient and remainder give a unique pair of child ranks; within a decoration block, the residual rank indexes the tail. Induction gives a bijection in every block, and the blocks are disjoint and exhaustive. Recursive height or length strictly decreases, proving termination. The frontier algorithm returns the left restriction before the right one, so by induction its output lists F in canonical order $(v_0, \dots, v_{L-1})$. Pairing it with the decoration $(a_0, \dots, a_{L-1})$ sets $A(v_i) = a_i$, as in Definition 5.4. Finally, the quotient-remainder split in (18) bijectively indexes the Cartesian product of the two counted sets. $\square$

Appendix D works through one example, unranking $rk = 9$ in the family $(h, w, L, S) = (2, 3, 3, 2)$.

5.2.3 Uniform message selection

Let

$$\beta = \left\lceil \frac{\lceil \log_2 M \rceil + 256}{8} \right\rceil, \quad B_\beta = 2^{8\beta}, \quad T_\beta = \left\lfloor \frac{B_\beta}{M} \right\rfloor M, \quad C_{\max} = 2^{64}.$$

The finite cap makes the counter domain explicit; counters range from zero through $2^{64} - 1$ and never wrap. Selector failure is denoted by $\perp$.

Theorem 5.7 (Selection distribution). *Fix a transcript $(\theta, \text{pk}, \rho, m)$ independently of the selector's random-function outputs. Suppose its distinct counter inputs give independent uniform 8β -bit strings. Set $\delta = (B_\beta - T_\beta)/B_\beta$. Then $0 \leq \delta < 2^{-256}$, the selector fails with probability $\delta^{C_{\max}}$, and, conditional on success, its output is uniform in $\mathcal{E}$. The expected number of counter trials is $(1 - \delta^{C_{\max}})/(1 - \delta)$.*

Proof. Each residue in $[M]$ has T_β/M accepted preimages, so its probability of being returned is

$$\frac{1 - \delta}{M} \sum_{c=0}^{C_{\max}-1} \delta^c = \frac{1 - \delta^{C_{\max}}}{M}.$$

Theorem 5.6 gives uniform encodings conditional on success. Since $0 \leq B_\beta - T_\beta < M$ and $B_\beta \geq 2^{256}M$, we have $\delta < 2^{-256}$. Independence gives failure probability $\delta^{C_{\max}}$ and expected trial count $\sum_{c=0}^{C_{\max}-1} \delta^c = (1 - \delta^{C_{\max}})/(1 - \delta)$. $\square$

Algorithm 6 $\text{Select}(\text{pk}, \rho, m)$; public parameters are implicit

```

1  for c = 0, ..., Cmax-1:
2      bytes = H_msg(theta, c, pk, rho, m; beta)
3      u = OS2IP(bytes)
4      if u < T_beta:
5          rk = u mod M
6          (rkF, rkA) = divmod(rk, D_w[L, S])
7          F = UnrankFrontier(1, h, L, rkF)
8          A = UnrankDecoration(L, S, rkA)
9          return (F, A)
10 return failure

```

5.2.4 Implementation cost

The table $N[t, \ell]$, truncated at $t = h, \ell = L$, can be built using $O(hL^2)$ integer operations. The decoration recurrence admits the sliding-window form

$$D_w[k, s] = D_w[k, s - 1] + D_w[k - 1, s] - D_w[k - 1, s - w],$$

with out-of-range entries zero, giving $O(L(S + 1))$ integer operations for a table truncated at $s = S$. A direct unranking implementation performs at most $O(L^2)$ frontier block inspections and $O(Lw)$ decoration block inspections. These are arithmetic-operation bounds; large-integer bit complexity and storage must also be measured in an implementation. The tables are public and can be shared by all keys with the same parameters.

5.3 Reuse security

This subsection studies the coverage exponents of Section 4.2 for CUTS. It characterizes the closure of any number of CUTS disclosures, gives λ_1 exactly, expresses λ_2 as one coefficient of an explicit polynomial and bounds it from below; for $b \geq 3$ it gives an exact count for fixed disclosures (Proposition 5.22). Its symbols (v, a) are the nodes of the tree of Section 5.1.2, and its inference rules are the generic forward rule on that tree, so Theorem 5.12 is Lemma 4.3 for CUTS, and Corollaries 5.3 and 5.26 apply the bounds below.

5.3.1 Symbolic reconstruction

Definition 5.8 (Symbolic values and forward closure). The set of symbolic values is

$$\Omega = \{\text{pk}\} \cup \{(v, a) : v \in V_h \setminus \{1\}, 0 \leq a < w\}.$$

The symbol (v, a) represents the addressed value $X_{v,a}$, and pk represents the public root. A set $C \subseteq \Omega$ is *forward closed* if it is closed under the following inference rules:

$$(v, a) \implies (v, a + 1) \quad (v \neq 1, 0 \leq a < w - 1), \quad (19)$$

$$(2v, w - 1), (2v + 1, w - 1) \implies (v, 0) \quad (v \neq 1 \text{ internal}), \quad (20)$$

$$(2, w - 1), (3, w - 1) \implies \text{pk}. \quad (21)$$

For $D \subseteq \Omega$, define $\text{cl}(D)$ to be the least forward-closed set containing $D \cup \{\text{pk}\}$.

Such a least set exists: it is the intersection of all forward-closed supersets of $D \cup \{\text{pk}\}$. In particular, closure is extensive, monotone, and idempotent. Including the already-public root has no effect on the derivability of non-root values, since the root is not a premise of any rule. No leaf derivation from a secret seed is an available inference rule.

Definition 5.9 (Symbolic disclosure). For a decorated frontier $E = (F, A)$ with $1 \notin F$, define

$$D(E) = D(F, A) = \{(v, A(v)) : v \in F\}.$$

Write $\text{cl}(E) = \text{cl}(D(E))$ and

$$\text{Anc}(F) = \{v \in V_h : \exists f \in F \text{ with } v \prec f\}.$$

Unless stated otherwise, sources and targets belong to the fixed family $\mathcal{E} = \mathcal{E}_{h,w,L,S}$ of Section 5.2, of cardinality M . The structural lemmas also apply to arbitrary decorated frontiers that exclude the global root, without common size or sum constraints.

All closures below are closures of symbolic positions. Equal byte strings at different positions are not identified. Thus the model captures the combinatorial forward-reconstruction component separately from collisions, inversions, and other computational attacks.

5.3.2 Closure of frontier disclosures

Lemma 5.10 (Restriction to a subtree). *Let F be a frontier and $v \in V_h$. If $F \cap T(v) \neq \emptyset$, then $F \cap T(v)$ is a frontier of $T(v)$. If, moreover, $v \notin F$, then v is internal and each of $F \cap T(2v)$ and $F \cap T(2v+1)$ is a nonempty frontier of its respective subtree.*

Proof. First, distinct nodes of F are incomparable under ancestry: otherwise a root-to-leaf path through the deeper node would meet F twice. Choose $f \in F \cap T(v)$. No strict ancestor of v can belong to F , since it would be a strict ancestor of f . Every path from v to a leaf, extended upward to the root, therefore meets F at a node in $T(v)$, and does so exactly once. This proves the first assertion. If $v \notin F$, these intersections lie in the child subtrees; both child restrictions are frontiers and nonempty. A leaf v could not satisfy these hypotheses. $\square$

Lemma 5.11 (Exact description of one closure). *Let $E = (F, A)$ be a decorated frontier with $1 \notin F$. Then*

$$\begin{aligned} \text{cl}(E) = \{\text{pk}\} \cup \{(v, a) : v \in \text{Anc}(F) \setminus \{1\}, 0 \leq a < w\} \\ \cup \{(v, a) : v \in F, A(v) \leq a < w\}. \end{aligned} \quad (22)$$

Proof. Denote the right-hand side by Q . Chain progression derives every listed suffix at a selected node. For any $v \in \text{Anc}(F)$, Lemma 5.10 gives frontiers in both child subtrees. Reconstructing bottom-up from the selected nodes derives their child endpoints, the merge at v , and, if $v \neq 1$, all positions on the chain at v . This is a finite induction on subtree height and shows $Q \subseteq \text{cl}(E)$.

Conversely, Q contains the disclosure and the public root and is closed under chain progression. Suppose the premises of a non-root merge at v lie in Q . Membership of $(2v, w-1)$ implies that $2v$ is either selected or a strict ancestor of a selected node. In either case $v \in \text{Anc}(F)$, and hence $(v, 0) \in Q$. The root rule also has its conclusion in Q . Thus Q is forward closed, and minimality of closure gives $\text{cl}(E) \subseteq Q$. $\square$

Theorem 5.12 (Union preservation for frontiers). *Let $b \geq 1$, and let $E_i = (F_i, A_i)$, $1 \leq i \leq b$, be decorated frontiers with $1 \notin F_i$. Their sizes and decoration sums need not agree. Then*

$$\boxed{\text{cl}\left(\bigcup_{i=1}^b D(E_i)\right) = \bigcup_{i=1}^b \text{cl}(E_i).} \quad (23)$$

Proof. Put $U = \bigcup_{i=1}^b \text{cl}(E_i)$. It contains the public root and every disclosure $D(E_i)$. If a chain-rule premise belongs to U , it belongs to some $\text{cl}(E_i)$, which also contains the conclusion. Suppose both premises of a non-root merge at v belong to U . Choose i such that $(2v, w-1) \in \text{cl}(E_i)$.

By Lemma 5.11, the child $2v$ is selected or has a selected descendant. Thus $v \in \text{Anc}(F_i)$ and $(v, 0) \in \text{cl}(E_i) \subseteq U$. The root-rule conclusion is already in U . Therefore U is forward closed. Minimality gives $\text{cl}(\bigcup_i D(E_i)) \subseteq U$. For each i , monotonicity gives $\text{cl}(E_i) \subseteq \text{cl}(\bigcup_j D(E_j))$, proving the reverse inclusion. $\square$

Remark 5.13 (Role of the frontier property). Theorem 5.12 is not a property of arbitrary disclosures. For example, take a height-two tree and let $D_1 = \{(4, w-1)\}$ and $D_2 = \{(5, w-1)\}$. The value $(2, 0)$ lies in $\text{cl}(D_1 \cup D_2)$ but in neither individual closure. These disclosures are not frontiers. The theorem also does not prohibit mixed forgeries: an accepted target may combine values from different individual closures.

Corollary 5.14 (Threshold characterization). *For a non-root node v , define*

$$\tau_E(v) = \begin{cases} 0, & v \in \text{Anc}(F), \\ A(v), & v \in F, \\ w, & \text{otherwise.} \end{cases}$$

For $0 \leq a < w$ and frontier disclosures $E_1, \dots, E_b$,

$$(v, a) \in \text{cl}\left(\bigcup_i D(E_i)\right) \iff a \geq \min_i \tau_{E_i}(v).$$

Proof. Apply Lemma 5.11 to each source and then Theorem 5.12. $\square$

5.3.3 Coverage probability

Definition 5.15 (Covered targets and coverage exponent). For $E_1, \dots, E_b \in \mathcal{E}$, set

$$\text{Forge}(E_1, \dots, E_b) = \{E \in \mathcal{E} : D(E) \subseteq \text{cl}\left(\bigcup_{i=1}^b D(E_i)\right)\}.$$

For independent uniform $E_1, \dots, E_b \in \mathcal{E}$, define

$$p_b = \mathbb{E} \left[\frac{|\text{Forge}(E_1, \dots, E_b)|}{M} \right], \quad \lambda_b = -\log_2 p_b.$$

Previously disclosed encodings remain included in Forge , in accordance with the coverage definition; the target message may nevertheless be fresh.

Proposition 5.16 (One-disclosure antichain property). *For every $E \in \mathcal{E}$, $\text{Forge}(E) = \{E\}$. In particular, $p_1 = M^{-1}$ and $\lambda_1 = \log_2 M$.*

Proof. Write $E = (F, A)$ and let $E' = (F', A') \in \text{Forge}(E)$. Every $v \in F'$ belongs to $F \cup \text{Anc}(F)$ by Lemma 5.11, so $F \cap T(v)$ is a nonempty frontier of $T(v)$. The subtrees rooted at nodes of F' partition the leaves and their corresponding restrictions partition F . Indeed, for any $f \in F$, the unique F' -node on a path through f cannot be strictly below f , by the same closure characterization. Consequently

$$L = |F| = \sum_{v \in F'} |F \cap T(v)| \geq |F'| = L.$$

Every restriction therefore has size one. A singleton frontier of a binary subtree consists of its root, so $F = F'$. Availability now requires $A'(v) \geq A(v)$ for all $v \in F$. Since both sums equal S , every coordinate is equal. Thus $E' = E$, and E is plainly covered by itself. $\square$

For an independent uniform target $E_0 \in \mathcal{E}$, conditioning on the sources gives target coverage probability $|\text{Forge}(E_1, \dots, E_b)|/M$. Taking expectations yields

$$p_b = \Pr[E_0 \in \text{Forge}(E_1, \dots, E_b)]. \quad (24)$$

Consequently, if Γ_2 counts the covered ordered triples in $\mathcal{E}^3$, then $p_2 = \Gamma_2/M^3$.

5.3.4 An exact generating function for two disclosures

For the auxiliary recursion, consider a height- t binary subtree with a w -position chain at *every* node, including its local root. This models a proper subtree of CUTS. At internal nodes, chain starts are derived by merging child endpoints; there is no public-root symbol in this auxiliary model. The frontier lemmas and union theorem have identical proofs in this model. We handle the actual chainless CUTS root separately.

Let $\mathfrak{E}_t$ contain all decorated frontiers of this subtree, with no fixed size or sum. For $E = (F, A)$ write $\ell(E) = |F|$ and $s(E) = \sum_{v \in F} A(v)$. Introduce commuting indeterminates $(x_1, x_2, y, z_1, z_2, u)$ and weights

$$\text{wt}_i(E) = x_i^{\ell(E)} z_i^{s(E)}, \quad \text{wt}_0(E) = y^{\ell(E)} u^{s(E)}.$$

Write $\text{Cov}(E_1, \dots, E_b; E_0)$ when the target disclosure lies in the forward closure of the sources in this auxiliary model. Define polynomials with nonnegative integer coefficients by

$$\begin{aligned} A_{i,t} &= \sum_{E \in \mathfrak{E}_t} \text{wt}_i(E), \\ B_{i,t} &= \sum_{\substack{E_i, E_0 \in \mathfrak{E}_t \\ \text{Cov}(E_i; E_0)}} \text{wt}_i(E_i) \text{wt}_0(E_0), \\ C_t &= \sum_{\substack{E_1, E_2, E_0 \in \mathfrak{E}_t \\ \text{Cov}(E_1, E_2; E_0)}} \text{wt}_1(E_1) \text{wt}_2(E_2) \text{wt}_0(E_0). \end{aligned}$$

Also define

$$\begin{aligned} P(z) &= \sum_{a=0}^{w-1} z^a, & J(z, u) &= \sum_{0 \leq a \leq c < w} z^a u^c, \\ K(z_1, z_2, u) &= \sum_{\substack{0 \leq a, b, c < w \\ \min(a, b) \leq c}} z_1^a z_2^b u^c. \end{aligned}$$

For brevity put

$$a_i = x_i P(z_i), \quad b_i = x_i y J(z_i, u), \quad \gamma = x_1 x_2 y K(z_1, z_2, u), \quad d = y P(u).$$

Theorem 5.17 (Subtree recurrence). *The polynomials above satisfy*

$$A_{i,0} = a_i, \quad B_{i,0} = b_i, \quad C_0 = \gamma.$$

For $t \geq 1$ they satisfy

$$A_{i,t} = a_i + A_{i,t-1}^2, \tag{25}$$

$$B_{i,t} = b_i + d A_{i,t-1}^2 + B_{i,t-1}^2, \tag{26}$$

$$\begin{aligned} C_t &= \gamma + d \left(a_1 A_{2,t-1}^2 + a_2 A_{1,t-1}^2 + A_{1,t-1}^2 A_{2,t-1}^2 \right) \\ &\quad + a_1 B_{2,t-1}^2 + a_2 B_{1,t-1}^2 + C_{t-1}^2. \end{aligned} \tag{27}$$

Proof. At height zero all frontiers stop at the unique node. A single source digit a covers a target digit c precisely when $a \leq c$; two source digits a, b cover it precisely when $\min(a, b) \leq c$. These give the base cases.

A source frontier either stops at the subtree root or splits into one frontier in each child subtree, proving (25). For (26), if source and target both stop, their contribution is b_i . If the source splits and target stops, the source derives the chain start at the root, so the target digit

is unrestricted, giving $dA_{i,t-1}^2$. If both split, the two child coverage conditions contribute $B_{i,t-1}^2$. A source stopping at the root cannot cover a target that splits, since it supplies no descendant values. These are all cases.

For (27), first suppose the target stops. If both sources stop, the contribution is γ . If exactly source 1 stops, exactly source 2 stops, or neither source stops, the respective contributions are $da_1A_{2,t-1}^2$, $da_2A_{1,t-1}^2$, and $dA_{1,t-1}^2A_{2,t-1}^2$. At least one splitting source derives the root chain start, so every target digit is covered in these three cases.

If the target splits, both sources cannot stop. If only source 1 stops, that source supplies no descendants and source 2 must cover both target restrictions, giving $a_1B_{2,t-1}^2$. The symmetric case gives $a_2B_{1,t-1}^2$. If both sources split, coverage holds independently as a combinatorial condition in each child subtree, giving C_{t-1}^2 . The cases are disjoint and exhaustive. Products combine ordered choices in the two child subtrees, and sizes and sums add, proving the recurrence. $\square$

Theorem 5.18 (Exact triple-count formula). *For the CUTS family $\mathcal{E}$ in Definition 5.4, let $R = C_{h-1}^2$. Then*

$$\Gamma_2 = [x_1^L x_2^L y^L z_1^S z_2^S u^S]R, \quad p_2 = \Gamma_2 / M^3. \quad (28)$$

Proof. All three frontiers have size $L \geq 2$ and thus split at the actual root. Each child subtree has a root chain and is counted by C_{h-1} . The chainless root supplies no descendant values, so target coverage is equivalent to coverage in each child subtree. Hence covered triples are counted by R . The specified coefficient enforces each size and digit sum separately. Apply (24). $\square$

5.3.5 Deterministic bounds by positive evaluation

Set $\mathbf{q} = (L, L, L, S, S, S)$, with variable order $(x_1, x_2, y, z_1, z_2, u)$. For a vector $\mathbf{r} \in \mathbb{R}_{>0}^6$, write $\mathbf{r}^{\mathbf{q}} = \prod_{i=1}^6 r_i^{q_i}$.

Theorem 5.19 (Positive-evaluation bound). *For every $\mathbf{r} \in \mathbb{R}_{>0}^6$,*

$$p_2 \leq \frac{R(\mathbf{r})}{M^3 \mathbf{r}^{\mathbf{q}}}, \quad (29)$$

$$\lambda_2 \geq 3 \log_2 M + \sum_{i=1}^6 q_i \log_2 r_i - \log_2 R(\mathbf{r}). \quad (30)$$

Proof. All coefficients of R are nonnegative. Its target monomial therefore contributes $\Gamma_2 \mathbf{r}^{\mathbf{q}} \leq R(\mathbf{r})$. Divide by $M^3 \mathbf{r}^{\mathbf{q}}$ and take negative binary logarithms. $\square$

Theorem 5.20 (Congruence-filtered bound). *Write $R(\mathbf{X}) = \sum_{\alpha} c_{\alpha} \mathbf{X}^{\alpha}$. For an integer $m \geq 1$, define*

$$R_{\mathbf{q}}^{(m)}(\mathbf{r}) = \sum_{\alpha \equiv \mathbf{q} \pmod{m}} c_{\alpha} \mathbf{r}^{\alpha},$$

where congruence holds coordinatewise. Then

$$\lambda_2 \geq 3 \log_2 M + \log_2 \mathbf{r}^{\mathbf{q}} - \log_2 R_{\mathbf{q}}^{(m)}(\mathbf{r}). \quad (31)$$

For $\omega = \exp(2\pi i/m)$, the filtered value equals

$$\frac{1}{m^6} \sum_{j \in \{0, \dots, m-1\}^6} \omega^{-j \cdot \mathbf{q}} R(r_1 \omega^{j_1}, \dots, r_6 \omega^{j_6}). \quad (32)$$

If m divides m' , then $R_{\mathbf{q}}^{(m')}(\mathbf{r}) \leq R_{\mathbf{q}}^{(m)}(\mathbf{r})$. Thus these bounds improve monotonically along divisibility chains for fixed weights.

Proof. The filtered sum retains the target monomial and contains only nonnegative terms, so

$$\Gamma_2 \mathbf{r}^q \leq R_q^{(m)}(\mathbf{r}) \leq R(\mathbf{r}).$$

This proves (31). To obtain (32), expand each evaluation of R and use

$$\frac{1}{m} \sum_{j=0}^{m-1} \omega^{j(a-q)} = \begin{cases} 1, & a \equiv q \pmod{m}, \\ 0, & \text{otherwise,} \end{cases}$$

in each coordinate. Finally, congruence modulo m' implies congruence modulo m when $m \mid m'$, so the former sum is a sub-sum of the latter. $\square$

Remark 5.21 (Computation and interpretation). Once P, J, K are evaluated, the scalar recurrence has $O(h)$ arithmetic steps. The filter uses m^6 such evaluations. Weight optimization is only an efficiency device: every positive choice is sound. The filter includes all higher-degree aliases and therefore requires no truncation assumption. The nonnegativity argument is exact, even though its evaluation at complex points requires numerical error control in an implementation.

5.3.6 Counting covered targets for fixed disclosures

The following dynamic program counts covered targets for a fixed collection of source disclosures.

Proposition 5.22 (Conditional coverage count). *Fix $E_1, \dots, E_b$ and let $\tau(v) = \min_i \tau_{E_i}(v)$. For each non-root node define*

$$Q_v(x, z) = x \sum_{a=\tau(v)}^{w-1} z^a + \begin{cases} 0, & v \text{ a leaf,} \\ Q_{2v}(x, z) Q_{2v+1}(x, z), & v \text{ internal.} \end{cases}$$

An empty sum is zero. Set $Q_1 = Q_2 Q_3$. Then

$$|\text{Forge}(E_1, \dots, E_b)| = [x^L z^S] Q_1(x, z).$$

All polynomials may be truncated to degrees at most L in x and S in z during computation.

Proof. A covered local frontier either stops at v , at an available position $a \geq \tau(v)$, or splits into covered frontiers in both child subtrees. These cases are disjoint and exhaustive by Corollary 5.14. Sizes and sums add under a split, giving multiplication. The global root cannot be selected, so it has only the split term. All exponents are nonnegative; a term discarded for exceeding a target degree can never contribute to that degree later. $\square$

5.4 CUTS+C

CUTS+C is the +C variant (Section 4.4) of CUTS. It keeps the tree of Section 5.1.2, the public key (13), key generation, and the chain-value checks of Section 5.1, and changes two things: how a message selects an encoding, and which bytes the signature carries. In place of rejection sampling over a wide output, the signer grinds a short counter until a fixed-width message hash lands in the encoding family, and sends that counter with the signature.

The counter changes the cost of selection for everyone. A selector call lands in the family with probability 2^{-g} , where g is defined below, so the signer makes about 2^g calls per signature, and a search for a message whose selection lands in a given set of encodings needs 2^g times as many selector calls as under CUTS. Corollary 5.26 turns this factor into g more bits in the coverage term of the forgery bound. After one signature the only encoding a forger can complete is the signed one (Proposition 5.16), and each selector call reaches it with probability at most 2^{-r} . The price is s_b more signature bytes and the signer's grinding.

5.4.1 Parameters

A CUTS+C instance adds an index width r and a counter width s to (n, h, w, L, S) . Its parameter tuple is $\theta = (n, h, w, L, S, r, s)$, so a CUTS key and a CUTS+C key with the same (n, h, w, L, S) do not share a tree. Write

$$r_b = \lceil r/8 \rceil, \quad s_b = \lceil s/8 \rceil, \quad \alpha = M/2^r, \quad g = r - \log_2 M,$$

so that α is the chance that one counter is accepted and $g = -\log_2 \alpha$. The parameters are valid when (n, h, w, L, S) is a valid CUTS tuple, $1 \leq s \leq 64$, $r \geq 1$, $2^r \geq M$, and

$$(1 - \alpha)^{2^s} \leq 2^{-128}. \quad (33)$$

Condition (33) bounds the chance that every counter is rejected. Since α falls as r grows, it bounds r from above for a given s ; together with $2^r \geq M$ it confines r to an interval. The interval is nonempty whenever $s \geq 7$: at the smallest admissible r , $\alpha \geq 1/2$, so $(1 - \alpha)^{2^s} \leq 2^{-2^s} \leq 2^{-128}$. The following integer test is sufficient.

Lemma 5.23 (Counter rule). *If $89 \cdot 2^r \leq 2^s M$, then (33) holds. Equivalently, (33) holds whenever $g \leq s - \log_2 89 \approx s - 6.48$.*

Proof. Since $1 - \alpha \leq e^{-\alpha}$, $(1 - \alpha)^{2^s} \leq \exp(-\alpha 2^s)$. The hypothesis gives $\alpha 2^s = 2^s M / 2^r \geq 89 > 128 \ln 2$, so $(1 - \alpha)^{2^s} < e^{-128 \ln 2} = 2^{-128}$. $\square$

The rule is close to tight: for $s \geq 16$, the largest g that (33) allows exceeds $s - \log_2 89$ by less than 0.01 bits. Each counter bit therefore buys about one bit of g , at an expected $2^g < 2^s / 89$ selector calls per signature; a 16-bit counter allows g up to about 9.5 bits.

5.4.2 Message selection

For a counter $c \in [2^s]$, the message index is the leading r bits of an r_b -byte selector output,

$$u_c = \left\lfloor \text{OS2IP}(H_{\text{msg}}(\theta, c, \text{pk}, \rho, m; r_b)) / 2^{8r_b - r} \right\rfloor \in [2^r]. \quad (34)$$

The counter is accepted if $u_c < M$, and u_c is then unranked by (18). Algorithm 7 gives the verifier's check of one counter and the signer's search for the first accepted one, and Figure 2 draws the search. The signature carries the counter as $\text{enc}_s(c)$, its unsigned big-endian encoding in exactly s_b bytes. When s is not a multiple of eight, a received encoding can exceed $2^s - 1$; the check rejects it.

Theorem 5.24 (CUTS+C selection distribution). *Fix a transcript $(\theta, \text{pk}, \rho, m)$ independently of the selector's random-function outputs, and suppose its distinct counter inputs give independent uniform $8r_b$ -bit strings. Then each u_c is uniform on $[2^r]$, and GrindC fails with probability $(1 - \alpha)^{2^s}$, which is at most 2^{-128} for valid parameters. Conditional on success, the returned encoding is uniform in $\mathcal{E}$, and the expected number of counters tried is*

$$\frac{1 - (1 - \alpha)^{2^s}}{\alpha} \leq \frac{1}{\alpha} = 2^g,$$

with equality only when $\alpha = 1$.

Proof. Keeping the leading r bits of a uniform $8r_b$ -bit string gives a uniform element of $[2^r]$, so each counter is accepted independently with probability α . All 2^s counters are rejected with probability $(1 - \alpha)^{2^s}$, which (33) bounds. An accepted u_c is uniform on $[M]$, and Theorem 5.6 turns it into a uniform encoding. The search tries counter c exactly when counters $0, \dots, c - 1$ were rejected, so the expected count is $\sum_{c=0}^{2^s-1} (1 - \alpha)^c = (1 - (1 - \alpha)^{2^s}) / \alpha$. $\square$

Algorithm 7 CUTS+C selection: one counter, and the signer's search

```

1  SelectC(pk, rho, m, c):           // the verifier's check
2      if c >= 2^s:
3          return failure
4      bytes = H_msg(theta, c, pk, rho, m; r_b)
5      u = OS2IP(bytes) >> (8*r_b - r)
6      if u >= M:
7          return failure
8      (rkF, rkA) = divmod(u, D_w[L, S])
9      F = UnrankFrontier(1, h, L, rkF)
10     A = UnrankDecoration(L, S, rkA)
11     return (F, A)
12
13  GrindC(pk, rho, m):              // the signer's search
14     for c = 0, ..., 2^s-1:
15         encoding = SelectC(pk, rho, m, c)
16         if encoding is not failure:
17             return (c, encoding)
18     return failure

```

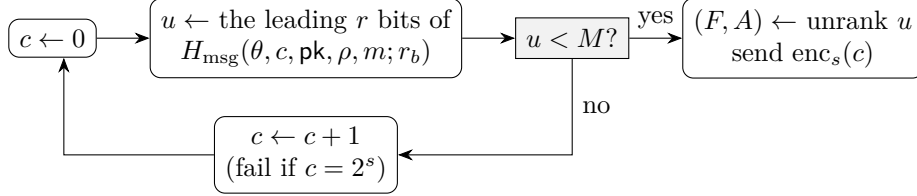


Figure 2: CUTS+C selection (Algorithm 7). Each counter is accepted with probability $p = M/2^r = 2^{-g}$, so the signer tries about 2^g counters (Theorem 5.24); the verifier checks only the counter carried in the signature.

5.4.3 Signing and verification

Algorithm 8 gives signing and verification. Chain values are computed and checked exactly as in Algorithms 2 and 3; only selection and the signature layout differ. The signature is

$$\sigma = \rho \parallel \text{enc}_s(c) \parallel Z_0 \parallel \cdots \parallel Z_{L-1},$$

of exactly $(L+1)n + s_b$ bytes. Signing is deterministic: the randomizer, and hence the accepted counter, is fixed by the seed and the message. The verifier checks only that the received counter is accepted, not that it is the first accepted one, so a message can have several valid signatures, one for each accepted counter; only the honest signer is restricted to the first.

Corollary 5.25 (CUTS+C correctness). *For any deterministic primitives of the stated types, valid CUTS+C parameters, key pair output by KeyGen, and message m for which Sign^C succeeds, $\text{Verify}^C(\text{pk}, m, \text{Sign}^C((\text{SK}, \text{pk}), m)) = 1$.*

Proof. The signature has the length the verifier expects, and parsing returns the signer's ρ and c , with $c < 2^s$. The verifier's call $\text{SelectC}(\text{pk}, \rho, m, c)$ repeats the signer's accepting call, so both sides hold the same encoding. The rest is the proof of Theorem 5.1. $\square$

5.4.4 Sizes and costs

The public key and stored secret key are as in Theorem 5.2, and a signature has $(L+1)n + s_b$ bytes. Key generation is unchanged and uses G tree primitive calls, as in (14). Successful

Algorithm 8 $\text{Sign}^C((\text{SK}, \text{pk}), m)$ and $\text{Verify}^C(\text{pk}, m, \sigma)$

```

1  SignC((SK, pk), m):
2    rho = F_rand(SK; theta, pk, m)
3    result = GrindC(pk, rho, m)
4    if result is failure:
5      return failure
6    (c, (F, A)) = result
7    for i = 0, ..., L-1:
8      z = Start(SK, F[i])
9      for j = 0, ..., A[i]-1:
10       z = H_chain(theta, F[i], j, z)
11     Z[i] = z
12    return (rho, enc_s(c), Z[0], ..., Z[L-1])
13
14  VerifyC(pk, m, sigma):
15    reject unless pk has n bytes and sigma has (L+1)*n + s_b bytes
16    parse sigma as (rho, C, Z[0], ..., Z[L-1]) // C has s_b bytes
17    c = OS2IP(C)
18    encoding = SelectC(pk, rho, m, c)
19    reject if encoding is failure
20    (F, A) = encoding
21    for i = 0, ..., L-1:
22      E[F[i]] = Advance(F[i], A[i], Z[i])
23    return (Recover(1) == pk)

```

verification uses the V tree primitive calls of (15) and exactly one H_{msg} call. The uncached signer uses $G - V$ tree primitive calls, as in (16), one F_{rand} call, and in expectation at most $2^g H_{\text{msg}}$ calls by Theorem 5.24. Larger r therefore costs the signer more grinding, and condition (33) caps it.

5.4.5 Security

The index u_c of (34) is the raw index function G^C of Section 4.4, evaluated at (pk, ρ, m, c) , and (18) is the bijection $\text{Unrank}_{\mathcal{E}}$. When H_{msg} is a random function, the leading r bits of its output form an independent uniform element of $[2^r]$ for each input, as the counter model requires (Theorem 5.24).

Corollary 5.26 (Few-time security of CUTS+C). *In the classical random-oracle model of Section 4.4, for every $b \geq 1$ and $Q \geq 0$,*

$$\text{Adv}_{\leq b}^{\text{EUF-CMA}, C}(Q) \leq \min \left\{ 1, (Q+1) 2^{-(\lambda_b + g)} + \frac{3(Q+V) + L}{2^{8n}} \right\}, \quad V = (L-1)(2w-1) - S.$$

Proof. Apply Theorem 4.10 to CUTS (Section 5.1), with $d = 2$ and $\alpha p_b = 2^{-(\lambda_b + g)}$. $\square$

For $b = 1$, Proposition 5.16 gives $\lambda_1 = \log_2 M$, so the coverage term is $(Q+1)2^{-r}$. For $b = 2$, every lower bound on λ_2 from Section 5.3 becomes a lower bound of $\lambda_2 + g$ on the exponent of the coverage term.

5.5 Example instances

This subsection records concrete parameter sets, gives their cost in hash-function calls, and compares signature length and verification cost with the one-time and few-time schemes those parameters are meant to sit beside. Every number in its tables is generated by the scripts

released with the reference implementation (github.com/p-11/cuts), and each λ_2 bound is certified in ball arithmetic, so it can be rechecked from them.

5.5.1 Parameter sets

Tables 1 and 2 list the parameter sets used in this paper. All use $n = 16$ and have a one-use coverage exponent of at least 128 bits ($\log_2 M \geq 128$ for CUTS, $r \geq 128$ for CUTS+C). CUTS+C uses a 16-bit counter and the largest r for which the signer exhausts the counter with probability at most 2^{-128} . Each set is named by its two-use design target in bits and the cost it favors: signature length (*smallest*), verification count V (*fastest*), or signature length subject to $G \leq 2^{14}$ (*cheap*). The column λ_2 is a lower bound on the two-use exponent of Section 5.3, proven with the congruence-filtered bound of Theorem 5.20. Each set reaches its design target with this bound, adding g for CUTS+C as Corollary 5.26 allows; both columns are rounded down.

Table 1: CUTS+C parameter sets. $|\sigma|$ is the signature length, V and G are as in Theorem 5.2, $g = r - \log_2 M$ is as in Section 5.4, and λ_2 is a proven lower bound.

Set	n	h	w	L	S	r	s	$ \sigma $ (B)	V	G	$\log_2 M$	g	λ_2
48-cheap	16	10	5	32	64	129	16	530	215	10 231	119.5	9.5	41.2
48-smallest	16	14	5	31	62	128	16	514	208	163 831	118.5	9.5	42.7
80-smallest	16	15	5	47	118	190	16	770	296	327 671	180.6	9.3	70.8
80-cheap	16	11	4	55	103	200	16	898	275	16 377	190.6	9.4	72.3
80-fastest	16	15	1	73	0	137	16	1 186	72	65 535	127.9	9.1	72.0
128-smallest	16	15	5	81	162	335	16	1 314	558	327 671	325.5	9.5	119.7
128-cheap	16	11	4	92	138	338	16	1 490	499	16 377	328.8	9.1	118.9
128-fastest	16	15	1	123	0	227	16	1 986	122	65 535	218.2	8.8	120.1

Table 2: CUTS parameter sets. Columns as in Table 1.

Set	n	h	w	L	S	$ \sigma $ (B)	V	G	$\log_2 M$	λ_2
128-smallest	16	15	5	88	176	1 424	607	327 671	354.3	128.6
128-fastest	16	15	1	132	0	2 128	131	65 535	234.3	128.1

5.5.2 Hash-function costs

Table 3 gives the cost of each set in hash-function calls rather than timings: Keccak- $f[1600]$ permutations of SHAKE256, the hash of the reference implementation, and compression-function calls of SHA-256. Every input length is measured from the reference implementation, signing a 32-byte message. SHAKE256 absorbs 136-byte blocks, and every tree primitive input fits in one, so the SHAKE256 counts are G , $G - V$ and V of Theorem 5.2 plus the randomizer and message hashes. The reference implementation does not instantiate SHA-256. The SHA-256 counts hash the same serialized inputs, which take two compressions per tree primitive call, and expand outputs longer than 32 bytes with MGF1. The table caches no midstate. Every tree primitive input starts with a domain tag and the parameter identifier, which depend on the parameter set but not on the key: CUTS has no per-key public seed such as the PK.seed of SLH-DSA. A precomputed SHA-256 midstate saves one compression per call only where this prefix fills a 64-byte block. The prefix is 68–69 bytes for CUTS and 68–69 bytes for CUTS+C, so caching leaves 50–51% of the SHA-256 counts for CUTS and 50–53% for CUTS+C. Signing

is the expected cost of the uncached signer; for CUTS+C it includes about 2^g message hashes (Theorem 5.24).

Table 3: Hash-function cost of each parameter set: SHAKE256 permutations and SHA-256 compressions, for a 32-byte message. Signing is the expected cost.

Set	KeyGen		Sign		Verify	
	SHAKE256	SHA-256	SHAKE256	SHA-256	SHAKE256	SHA-256
cuts-c-48-cheap	10 231	20 462	10 752	22 239	216	433
cuts-c-48-smallest	163 831	327 662	164 359	329 455	209	419
cuts-c-80-smallest	327 671	655 342	328 046	656 764	297	595
cuts-c-80-cheap	16 377	32 754	16 795	34 283	276	553
cuts-c-80-fastest	65 535	131 070	66 015	132 583	73	147
cuts-c-128-smallest	327 671	655 342	327 848	657 899	559	1 121
cuts-c-128-cheap	16 377	32 754	16 458	34 655	500	1 003
cuts-c-128-fastest	65 535	131 070	65 874	132 210	123	247
cuts-128-smallest	327 671	655 342	327 067	654 137	609	1 220
cuts-128-fastest	65 535	131 070	65 407	130 816	133	267

5.5.3 Comparison with WOTS+C and FORS

Table 4 sets the parameter sets beside WOTS+C, as used in SHRINCS [6, 10], and FORS [3, 5]. Every row is counted as in Theorem 5.2: the signature length includes an n -byte randomizer, and V and G count tree primitive calls. WOTS+C and FORS also compress more than two values into their public key; the cost convention of Section 5.1.6 counts only binary merges, and we count each such compression as one call. The exponents λ_1 and λ_2 are those of Section 5.3 for one and two signatures, and λ_1 of a CUTS set is $\log_2 M$ (Proposition 5.16). They are coverage exponents, not security levels: the hash term of the bound limits security to about $8n$ bits whatever their size (Corollaries 5.3 and 5.26).

Both other schemes fit the same analysis. A WOTS+C key is a one-level tree whose frontier is every leaf, so two signatures cover a target exactly when its position on every chain is at least the smaller of theirs. Its covered triples are one coefficient of $K(z_1, z_2, u)^{32}$, and the argument of Theorems 5.19 and 5.20, which needs only nonnegative coefficients, bounds its λ_2 ; the bound is certified in ball arithmetic, as for the CUTS rows, and larger moduli do not change it. A FORS signature reveals one leaf secret per tree, so with k trees of height a , $\lambda_1 = ka$ and $\lambda_2 = k(a - \log_2(2 - 2^{-a}))$ exactly. The FORS rows are the FORS of SLH-DSA-128s and the smallest FORS with $\lambda_1, \lambda_2 \geq 128$ whose key generation costs no more than that of the costliest CUTS set. The signers of CUTS+C and of WOTS+C both grind a counter, and their gap g is listed separately; it is not included in λ_1 or λ_2 . For CUTS+C, Corollary 5.26 proves that g adds to the exponent of the coverage term. WOTS+C accepts a digest by its digit sum rather than by an index below M , so Theorem 4.10 is not stated for it, and its g rests on the heuristic argument of [10].

With a proven two-use exponent of at least 128 bits, both CUTS sets have shorter signatures than either FORS row, and the fastest needs fewer verification calls. The smallest CUTS set costs more key generation than the sized FORS, and the fastest costs less. The CUTS+C sets reach 128 bits for one use, and their two-use targets, only once g is added, as Corollary 5.26 allows. WOTS+C remains the cheapest key to generate by a wide margin, but two signatures leave it a small two-use exponent.

In SHRINCS, the stateful path signs with one WOTS+C key per message [6]. The set CUTS+C 48-cheap has the same length, 530 bytes against 530, so if its randomizer takes the place of the SHRINCS randomizer, the shortest stateful signature keeps its length. It verifies in

Table 4: Parameter sets beside WOTS+C (as in SHRINCS) and FORS. Lengths include an n -byte randomizer; V and G count tree primitive calls; λ_2 and g are rounded down.

Scheme	Set	$ \sigma $ (B)	V	G	λ_1	λ_2	g
WOTS+C	SHRINCS	530	241	513	122.0	27.3	6.0
FORS	SLH-DSA-128s	2 928	183	172 019	168.0	154.0	–
FORS	$k = 11, a = 13$	2 480	155	270 326	143.0	132.0	–
CUTS+C	48-cheap	530	215	10 231	119.5	41.2	9.5
CUTS+C	48-smallest	514	208	163 831	118.5	42.7	9.5
CUTS+C	80-smallest	770	296	327 671	180.6	70.8	9.3
CUTS+C	80-cheap	898	275	16 377	190.6	72.3	9.4
CUTS+C	80-fastest	1 186	72	65 535	127.9	72.0	9.1
CUTS+C	128-smallest	1 314	558	327 671	325.5	119.7	9.5
CUTS+C	128-cheap	1 490	499	16 377	328.8	118.9	9.1
CUTS+C	128-fastest	1 986	122	65 535	218.2	120.1	8.8
CUTS	128-smallest	1 424	607	327 671	354.3	128.6	–
CUTS	128-fastest	2 128	131	65 535	234.3	128.1	–

215 tree primitive calls against 241. If the state fails and a key signs twice, it keeps $\lambda_2 \geq 41.2$ bits where WOTS+C keeps 27.3, before the gaps of 9.5 and 6.0 bits, of which only the first is proven. The cost is key generation: 10 231 calls per key against 513, paid for every key of the stateful tree. These counts are in tree primitive calls; SHRINCS states its costs in SHA-256 compressions, whose number per call depends on input length. Section 7.1 outlines this use; the composition is not specified here.

6 Quantum considerations

Theorems 4.5 and 4.10, their CUTS forms in Corollaries 5.3 and 5.26, and every other random-oracle statement in this paper, including Theorems 5.7 and 5.24, are proved for classical queries. This section gives the generic quantum costs of the two terms of the CUTS bound, for two signatures. They are estimates from quantum search, not theorems about CUTS, and they are not a quantum random-oracle reduction.

The term $(3(Q + V) + L)2^{-8n}$ charges events in which a query input or a supplied value equals an n -byte value the adversary has not been given: the secret seed or an unrevealed tree value. Finding such a value is a preimage search. Classically each query succeeds with probability about 2^{-8n} ; with Q quantum queries, Grover search succeeds with probability of order $Q^2 2^{-8n}$, so the term offers about $4n$ bits instead of $8n$. The bound has no collision term, so quantum collision search does not enter it.

The term $(Q + 1)2^{-\lambda_2}$ charges a query whose selected encoding is covered by the two signatures. Coverage can be tested from the disclosed values, so this is an unstructured search over selector inputs, and Grover search finds a covered encoding in about $2^{\lambda_2/2}$ queries. The grinding gap g that the CUTS+C parameter sets add to λ_2 comes from the same search, so it halves too. Generic quantum search therefore leaves about $4n$ bits in the first term and $\lambda_2/2$ in the second, plus $g/2$ for CUTS+C.

A proof in the quantum random-oracle model would have to program the oracles, answer quantum queries to the selector, and avoid rewinding the signer. Section 7 lists it as an open problem.

7 Future work

7.1 Applications

CUTS is a component, not a complete signature scheme: one key signs a few messages under one public root. This subsection outlines three settings in which such a key could be used. It specifies no composition and proves nothing about one.

Stateless. SLH-DSA signs with a FORS key at the bottom of a hyper-tree, chosen by hashing the message, so one FORS key may sign several messages and is sized for that [3, 5]. A CUTS key could take the place of FORS. At a proven two-use exponent of 128 bits, the CUTS sets of Section 5.5.3 have shorter signatures than the FORS rows there. A stateless scheme needs three further pieces: the hyper-tree that authenticates the CUTS roots, the schedule that maps a message to a key, and parameters sized for the number of signatures a key can expect to make. The last piece requires bounds on λ_b for $b > 2$, which are open for CUTS (Section 7.2).

Stateful. SHRINCS signs on its stateful path with one WOTS+C key per message, and reusing a state value can let an observer forge [6]. A CUTS+C key could take the place of the WOTS+C key. Section 5.5.3 compares one such set with the SHRINCS key: a signature of the same length, a larger proven lower bound on the two-use exponent, and a higher key-generation cost, paid for every key in the stateful tree. The case for the exchange is what remains after one accidental reuse: Corollary 5.26 bounds forgery against the CUTS+C key after two signatures, whereas the grinding gap of WOTS+C is not covered by our theorems.

Stateful with a stateless fallback. SHRINCS pairs its stateful tree with an SLH-DSA instance that signs when the state is lost [6, 23]. Both paths could be built from CUTS: a few-time CUTS key on the stateful path, which keeps the short signatures of the stateful case and limits what a repeated state value reveals, and a stateless CUTS-based scheme as the fallback. How the two paths share a public key, and the security of the combination, are not specified here.

7.2 Open problems

The problems below are open. This paper does not contain partial proofs of them.

Concrete selectors. Theorem 4.10 covers a counter search that accepts an index below M , which is how CUTS+C selects. Two selectors in this paper fall outside it. WOTS+C accepts a digest whose digits have a fixed sum, and its grinding gap rests on a heuristic argument [10]; the proof of the +C theorem uses only the acceptance probability, and extending the statement to other acceptance sets is open. The base theorem treats the selector as one ideal oracle, whereas the rejection sampler of CUTS (Theorem 5.7) can make several hash calls per selection; relating the two needs a query-accounting step.

Three or more signatures for CUTS. Union preservation, the coefficient bound and the unforgeability theorem hold for every number b of signatures (Section 4.3). For CUTS, the recurrence that evaluates the coverage polynomial is stated for $b = 2$ only. A recurrence for larger b has more cases at each node, and its cost is unknown. Without it, parameters for a key that may sign three or more times rest on sampling.

Optimal encoding families. Bleichenbacher and Maurer characterized optimal graph-based one-time signatures [12]. The analogous question for few-time use is open: among families of frontiers of a given tree, with a given signature length and verification cost, which

maximizes λ_2 ? CUTS fixes the frontier size and the decoration sum, and we do not know how far it is from the optimum.

Quantum random-oracle model. A proof of the few-time bound under quantum queries to the hash oracles. It would have to program the oracles, answer quantum queries to the selector, and avoid rewinding the signer. Section 6 gives the generic quantum search costs.

Mechanized proofs. A machine-checked proof of the unranking bijection, the cost identities for G , V and $G_{\text{sign}} = G - V$, union preservation, and the unforgeability theorem.

8 Conclusion

Signatures built from trees of hash chains share one structural property: the values derivable from several frontier disclosures are the union of those derivable from each. For every scheme in this class it reduces few-time security in the classical random-oracle model to a coverage exponent λ_b , the negative base-2 logarithm of the probability that b signatures reveal enough to complete an independent uniform encoding (Theorem 4.5), also when the signer grinds a counter (Theorem 4.10). Bounding that exponent is a counting problem on the scheme’s encoding family.

CUTS is one scheme in the class, with fixed signature length, fixed verification cost and a two-use exponent with certified lower bounds. Its encodings are frontiers of a single tree with fixed-sum chain positions, so a signature carries no authentication path, and the recurrence of Section 5.3 gives deterministic lower bounds on λ_2 . With these bounds, CUTS parameter sets reach a proven two-use exponent of 128 bits with shorter signatures than the FORS sets we compare against (Section 5.5.3).

Its counter variant CUTS+C adds a proven grinding gap of g bits to the coverage exponent (Corollary 5.26). The main limits are those of Section 7.2: the CUTS bounds are computed for two signatures only, the selector of the base scheme is modeled as an ideal oracle, and every security statement is classical.

References

- [1] Ralph C. Merkle. A Certified Digital Signature. In *Advances in Cryptology — CRYPTO ’89*, LNCS, pp. 218–238. doi:10.1007/0-387-34805-0_21.
- [2] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Ahmad Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018. <https://www.rfc-editor.org/rfc/rfc8391.html>.
- [3] National Institute of Standards and Technology. Stateless Hash-Based Digital Signature Standard. FIPS 205, August 2024. doi:10.6028/NIST.FIPS.205.
- [4] Andreas Hülsing. WOTS+ — Shorter Signatures for Hash-Based Signature Schemes. In *AFRICACRYPT 2013*, pp. 173–188, 2013. doi:10.1007/978-3-642-38553-7_10. Revised version: <https://eprint.iacr.org/2017/965>.
- [5] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ Signature Framework. In *ACM CCS 2019*, pp. 2129–2146, 2019. doi:10.1145/3319535.3363229. IACR ePrint 2019/1086.
- [6] conduction, Ethan Heilman, Mikhail Kudinov, Oleksandr Kurbatov, Jonas Nick, and remix7531. SHRINCS: A Compact Hash-Based Signature Scheme. Draft BIP, 2026. Commit a5891ec of <https://github.com/SHRINCS/shrinCS-bip>.

- [7] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In *Information Security and Privacy (ACISP 2002)*, LNCS, pp. 144–153, 2002. doi:10.1007/3-540-45450-0_11.
- [8] Kaiyi Zhang, Hongrui Cui, and Yu Yu. Revisiting the Constant-sum Winternitz One-time Signature with Applications to SPHINCS+ and XMSS. In *CRYPTO 2023*; IACR Cryptology ePrint Archive, Report 2023/850. <https://eprint.iacr.org/2023/850>.
- [9] Lucas Pandolfo Perin, Gustavo Zambonin, Ricardo Custódio, Lucia Moura, and Daniel Panario. Improved Constant-sum Encodings for Hash-based Signatures. *Journal of Cryptographic Engineering*, 11(4):329–351, 2021. doi:10.1007/s13389-021-00264-9.
- [10] Andreas Hülsing, Mikhail Kudinov, Eyal Ronen, and Eylon Yogev. SPHINCS+C: Compressing SPHINCS+ With (Almost) No Cost. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 1435–1453, 2023. doi:10.1109/SP46215.2023.10179381. IACR ePrint 2022/778.
- [11] Daniel Bleichenbacher and Ueli Maurer. Directed Acyclic Graphs, One-way Functions and Digital Signatures. In *CRYPTO '94*, LNCS 839, pp. 75–82, 1994. doi:10.1007/3-540-48658-5_9.
- [12] Daniel Bleichenbacher and Ueli Maurer. Optimal Tree-Based One-time Digital Signature Schemes. In *STACS '96*, LNCS 1046, pp. 361–374, 1996. doi:10.1007/3-540-60922-9_30.
- [13] Alejandro Hevia and Daniele Micciancio. The Provable Security of Graph-Based One-Time Signatures and Extensions to Algebraic Signature Schemes. In *ASIACRYPT 2002*, LNCS 2501, pp. 379–396, 2002. doi:10.1007/3-540-36178-2_24.
- [14] Jurjen N. E. Bos and David Chaum. Provably Unforgeable Signatures. In *Advances in Cryptology — CRYPTO '92*, LNCS, pp. 1–14. doi:10.1007/3-540-48071-4_1.
- [15] Jason Paul Cruz, Yoshio Yatani, and Yuichi Kaji. Constant-sum Fingerprinting for Winternitz One-time Signature. In *2016 International Symposium on Information Theory and Its Applications (ISITA)*, pp. 703–707, IEEE, 2016. <https://ieeexplore.ieee.org/document/7840516>.
- [16] Dmitry Khovratovich, Mikhail Kudinov, and Benedikt Wagner. At the Top of the Hypercube — Better Size-Time Tradeoffs for Hash-Based Signatures. In *Advances in Cryptology — CRYPTO 2025*, LNCS, pp. 93–123, 2025. doi:10.1007/978-3-032-01887-8_4. IACR ePrint 2025/889.
- [17] Jean-Philippe Aumasson and Guillaume Endignoux. Improving Stateless Hash-Based Signatures. In *Topics in Cryptology — CT-RSA 2018*, LNCS, pp. 219–242, 2018. doi:10.1007/978-3-319-76953-0_12. IACR ePrint 2017/933.
- [18] Mehdi Abri and Jonathan Katz. Shorter Hash-Based Signatures Using Forced Pruning. In *Advances in Cryptology — CRYPTO 2026*, LNCS, pp. 573–602, 2026. doi:10.1007/978-3-032-35398-6_19. IACR ePrint 2025/2069.
- [19] Mikhail Kudinov, Evgeniy Kiktenko, and Aleksey Fedorov. Round 3 Official Comment: SPHINCS+. NIST PQC forum, 23 July 2020. <https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/official-comments/Sphincs-Plus-round3-official-comment.pdf>.
- [20] Mikhail A. Kudinov, Evgeniy O. Kiktenko, and Aleksey K. Fedorov. Security Analysis of the W-OTS⁺ Signature Scheme: Updating Security Bounds. *Matematicheskie Voprosy Kriptografii*, 12(2):129–145, 2021. doi:10.4213/mvk362. arXiv:2002.07419.

- [21] Andreas Hülsing and Mikhail Kudinov. Recovering the Tight Security Proof of SPHINCS⁺. In *Advances in Cryptology — ASIACRYPT 2022*, LNCS, pp. 3–33, 2022. doi:10.1007/978-3-031-22972-5_1. IACR ePrint 2022/346.
- [22] Manuel Barbosa, François Dupressoir, Andreas Hülsing, Matthias Meijers, and Pierre-Yves Strub. A Tight Security Proof for SPHINCS⁺, Formally Verified. In *Advances in Cryptology — ASIACRYPT 2024*, LNCS, pp. 35–67, 2024. doi:10.1007/978-981-96-0894-2_2. IACR ePrint 2024/910.
- [23] Mikhail Kudinov and Jonas Nick. Hash-based Signature Schemes for Bitcoin. IACR Cryptology ePrint Archive, Paper 2025/2203, 2025. <https://eprint.iacr.org/2025/2203>.

A Proof of the few-time security theorem

Fix integers $b \geq 1$ and $Q \geq 0$, and write $\kappa = 8n$. The root is o and the secret seed is $\mathbf{SK} \in \Sigma$. The word alphabet and encoding family are finite; the message space is countable.

A.1 Adversaries and the real experiment

Let

$$\mathcal{I} = (\{\text{leaf}\} \times \mathcal{V}_{\text{leaf}}) \sqcup (\{\text{node}\} \times \mathcal{V}_{\text{int}}) \sqcup \{\text{rand}, \text{sel}\}$$

index the public oracles F_v, H_u, R, G , with their declared input and output spaces $\mathcal{X}_i, \mathcal{Y}_i$. Let $\text{DT}_{Q,b}$ consist of finite rooted, fully labeled decision trees of the following form:

- A node $\text{Query}(i, x)$, with $x \in \mathcal{X}_i$, has exactly one outgoing edge for each $y \in \mathcal{Y}_i$, labeled by y .
- A node $\text{Sign}(m)$, with $m \in \mathcal{M}$, has exactly one outgoing edge for each byte string of length $n(\ell + 1)$ with $1 \leq \ell \leq L$, labeled by that string.
- A leaf is Abort or $\text{Forge}(m, \sigma)$; candidate signatures may be arbitrary finite strings, with malformed strings rejected.
- Every root-to-leaf path has at most Q public-query nodes and b signing-query nodes.

The branches include all syntactically possible responses, even those having zero probability in a particular experiment. A deterministic adversary is a family $\mathbf{T} = (T_{\mathbf{pk}})_{\mathbf{pk} \in \Sigma}$ with $T_{\mathbf{pk}} \in \text{DT}_{Q,b}$. A randomized adversary is a distribution over these families, sampled independently of challenger randomness. It suffices to prove a uniform bound for deterministic families, since

$$\Pr[\text{Win}(\mathcal{A})] = \mathbb{E}_{\mathbf{T} \leftarrow \mathcal{A}}[\Pr[\text{Win}(\mathbf{T})]].$$

The experiment $\text{Real}(\mathcal{A})$ is:

1. Sample the independent random oracles, run $(\mathbf{SK}, \mathbf{pk}) \leftarrow \text{KeyGen}()$, independently sample $\mathbf{T} \leftarrow \mathcal{A}$, and start $T_{\mathbf{pk}}$ with $\mathcal{M}_{\text{signed}} = \emptyset$.
2. At a public-query node, return the corresponding oracle answer. At a signing-query node with message m , return $\text{Sign}(\mathbf{SK}, \mathbf{pk}, m)$ and add m to $\mathcal{M}_{\text{signed}}$. Follow the edge labeled by the response.
3. At Abort , return 0. At $\text{Forge}(m^*, \sigma^*)$, return 1 exactly when $m^* \notin \mathcal{M}_{\text{signed}}$ and verification accepts. A forgery on a previously signed message is rejected without further verification.

A.2 The simulated experiment

The experiment $\text{Sim}(\mathcal{A})$ uses the same decision tree, parsing rules, freshness test, and verification algorithm, with the handlers below. Its tables and private bookkeeping are part of the challenger state.

Initialization. Independently sample $\mathbf{SK}$ and all canonical labels x_u uniformly from Σ ; set $\mathbf{pk} = x_o$ and $\mathcal{C} = \{o\}$. Independently sample

$$E_1, \dots, E_b, E'_1, \dots, E'_{Q+1} \leftarrow \mathcal{E}$$

uniformly with replacement. Initialize empty tables for F_v, R, G, H_u , an empty signature record, $\mathcal{M}_{\text{signed}} = \emptyset$, and an independent private random function $m \mapsto \rho_m \in \Sigma$. Independently sample the adversary family and run $T_{\mathbf{pk}}$ as in Real . Lazy sampling of these objects is equivalent. The closure $\mathcal{C}$ changes only through signing disclosures. For an internal node u , $x_{\text{ch}(u)}$ denotes the ordered tuple of its children's canonical labels.

Public F_v and R queries. Return the cached answer if defined; otherwise sample and store an independent uniform word. These public tables do not consult SK , the canonical labels, or the private function $m \mapsto \rho_m$.

Selector queries. On $G(\text{pk}', \rho, m)$, return the current table entry if defined. Otherwise store and return the next unused E'_j . The final verifier uses the same handler, so there are at most $Q + 1$ such assignments.

Signing queries. For a previously signed message, return its recorded signature. Otherwise let $i \in \{1, \dots, b\}$ be its ordinal among distinct signed messages, obtain ρ_m , and assign

$$G(\text{pk}, \rho_m, m) \leftarrow E_i$$

even if an entry already exists. Return and record the signature byte string consisting of ρ_m followed by the labels $(x_u)_{u \in E_i}$ in the order $\text{ord}(E_i)$, add m to $\mathcal{M}_{\text{signed}}$, and update $\mathcal{C} \leftarrow \text{cl}(\mathcal{C} \cup E_i)$. Signing does not insert internal-node hash cache entries.

Internal-node hash queries. For a call $H_u(y)$, where $y \in \Sigma^{d_u}$:

- (i) If y is cached at node u , return its cached answer.
- (ii) Otherwise, if every child of u lies in $\mathcal{C}$ and $y = x_{\text{ch}(u)}$, store and return x_u .
- (iii) Otherwise, sample an independent uniform $z \in \Sigma$, store it at (u, y) , and return it.

The final verifier uses the same internal-node hash handler. In case (ii), closure implies $u \in \mathcal{C}$. Thus every response depends only on exposed canonical labels and independent randomness. Cache hits return consistent answers; only the specified signing assignment can overwrite a selector entry.

Coverage and recorded events. Let $\mathcal{C}_{\text{final}}$ be the signing closure at the final output. If final verification invokes G , let E^* denote its answer and define

$$\text{Cov} = \{m^* \notin \mathcal{M}_{\text{signed}} \text{ and } E^* \subseteq \mathcal{C}_{\text{final}}\}.$$

Otherwise Cov is false. Define the following private flags throughout the simulation, including final verification where specified:

- B_s An adversarial F_v query has input SK , or an adversarial R query has first argument SK .
- B_f At the first signing of a message m , the input (pk, ρ_m, m) was previously queried publicly to G .
- B_{in} At an adversarial or final-verifier H_u call, some child slot for $p \notin \mathcal{C}$ contains x_p . Use $\mathcal{C}$ at the time of the call, and inspect slots even on cache hits.
- B_{out} A fresh sample in internal-node hash handler case (iii), at node u , equals x_u . Deliberate returns in case (ii) are excluded.
- B_{pay} After the final payload passes the format and target-size checks, some supplied word at a selected $u \notin \mathcal{C}_{\text{final}}$ equals x_u . This flag is false when these checks fail.

Set

$$\Delta = B_s \cup B_f \cup B_{\text{in}}, \quad \text{Bad} = \Delta \cup B_{\text{out}} \cup B_{\text{pay}}.$$

Recording a flag never changes a response, aborts the game, or enlarges $\mathcal{C}$. The simulated win event is the ordinary fresh-message acceptance event using these handlers, including on flagged executions.

A.3 Comparison of the real and simulated experiments

Lemma A.1 (Real–simulation comparison). *For every admissible adversary,*

$$\left| \Pr_{\text{Real}}[\text{Win}] - \Pr_{\text{Sim}}[\text{Win}] \right| \leq \Pr_{\text{Sim}}[\Delta]. \quad (35)$$

Proof. Fix a deterministic adversary family. An equivalent way to generate the real key and oracles is to sample SK and all x_u independently uniformly, program

$$F_v(\text{SK}) = x_v, \quad H_u(x_{\text{ch}(u)}) = x_u,$$

and fill all other entries independently uniformly. Bottom-up key generation has exactly this distribution: each node function contributes one fresh uniform output at its canonical input. Likewise sample independent ρ_m and program $R(\text{SK}, \text{pk}, m) = \rho_m$, leaving the other entries uniform.

Couple the games by sharing these samples, the public key, and the adversary family. Until the first Δ flag, couple fresh unprogrammed answers with the same sample. A new selector input during signing receives the next source E_i in both games; a new public or final-verifier selector input receives the next target E'_j in both games. Their marginals are uniform as required. The following checks show that the exposed transcripts agree before a flag:

- A seed-domain query outside B_s avoids every real seed-programmed entry, so both games use the same cached or fresh independent answer.
- Public selector tables agree. At a new signing, absence of B_f means the signing input is undefined: an earlier signing of another message cannot have assigned it. Both games assign E_i and return identical signatures. Repeated signings return identical recorded responses.
- For a new internal-node hash input, a canonical tuple with a hidden child triggers B_{in} . A canonical tuple with all children in $\mathcal{C}$ returns x_u in both games. A noncanonical tuple outside the flag uses a shared fresh random word. Cached internal-node hash answers already agree; if such an input had been canonical while a child was hidden, its first query would already have triggered B_{in} .

Apply the internal-node hash and selector checks also to final verification. At the first flag and thereafter, let each game follow its own transition rules. This preserves both marginals. If no flag occurs, the two games reach the same leaf and their win indicators agree. On the coupled probability space,

$$|\mathbf{1}_{\text{Win}_{\text{Real}}} - \mathbf{1}_{\text{Win}_{\text{Sim}}}| \leq \mathbf{1}_{\Delta_{\text{Sim}}}.$$

Taking expectations proves (35). Sharing a sampled adversary family gives the same conclusion for randomized adversaries. $\square$

A.4 Forgery decomposition and adaptive coverage

Proof of Lemma 4.6. First work entirely in the simulation. Suppose a fresh forgery is accepted and is not covered. Choose $u \in E^* \setminus \mathcal{C}_{\text{final}}$. If the supplied word at u is x_u , then B_{pay} holds. Otherwise follow the verifier's reconstruction from this incorrect word toward the root. Completeness of the frontier implies that every ancestor on this path is evaluated. Since the root is accepted, there is a first node on the path whose computed value is its canonical label. Its input tuple is noncanonical but its output is canonical. That answer cannot originate in internal-node hash handler case (ii); it was first sampled in case (iii), possibly on an earlier query that the verifier now repeats. Its creation triggered B_{out} . Hence

$$\text{Win}_{\text{Sim}} \subseteq \text{Cov} \cup B_{\text{out}} \cup B_{\text{pay}}.$$

Under the preceding coupling, a real win outside Δ_{Sim} is also a simulated win. Consequently, on the joint space,

$$\text{Win}_{\text{Real}} \subseteq \text{Cov}_{\text{Sim}} \cup \Delta_{\text{Sim}} \cup (B_{\text{out}})_{\text{Sim}} \cup (B_{\text{pay}})_{\text{Sim}}.$$

Take probabilities and use the simulated marginal. $\square$

Proof of Lemma 4.7. A fresh final message cannot use a signing-programmed selector input, because the input includes the message. Its target encoding is therefore some E'_J , whether that input was first queried publicly or by the final verifier. Moreover,

$$\mathcal{C}_{\text{final}} \subseteq \text{cl}\left(\bigcup_{i=1}^b E_i\right).$$

If fewer than b distinct messages were signed, the unused independent sources can only enlarge this right-hand side. Thus, pathwise,

$$\text{Cov} \subseteq \bigcup_{j=1}^{Q+1} \left\{ E'_j \subseteq \text{cl}\left(\bigcup_{i=1}^b E_i\right) \right\}.$$

For every fixed j , the tuple $(E_1, \dots, E_b, E'_j)$ is independent uniform, so its event has probability p_b . The union bound proves the claim. In particular, we never condition on the adaptively selected J , on the number of signings, or on the absence of bad events. No tail bound on conditional coverage is used. $\square$

A.5 Exceptional-event charges

Proof of Lemma 4.8. All probabilities in this proof refer to the non-aborting simulation. Write N_s, N_m, N_g for its actual numbers of adversarial seed-domain (F_v, R) , selector (G) , and internal-node hash (H_u) calls. Pathwise, $N_s + N_m + N_g \leq Q$.

Information invariant. Augment the public history by the canonical labels at all nodes currently in $\mathcal{C}$, but not by the private flags or unused source and target samples. Conditional on this augmented history, SK and all labels outside $\mathcal{C}$ remain independent uniform words; the private ρ_m for messages not yet signed remain independent uniform as well. Initially this holds by construction. Public seed-domain tables ignore SK ; selector answers use independent randomness; internal-node hash responses consult only labels in $\mathcal{C}$ or return independent samples. A signing chooses an unused independent source encoding and exposes labels in the enlarged closure. Adaptive exposure of independent coordinates leaves the other coordinates independent uniform. Cache hits add no new information. Private flags do not alter execution, so this argument does not condition away earlier hits. During final verification, include its selector answer and computed intermediate values in the history. These are obtained from the supplied payload and the same handlers, so the invariant continues to hold.

Seed guesses. At each actual seed-domain call the proposed seed is fixed by the history and matches SK with probability $2^{-\kappa}$. Summing the conditional charges gives

$$\Pr[B_s] \leq \mathbb{E}[N_s] 2^{-\kappa}.$$

Selector prequeries. Just before the first signing of a message m_i , its private ρ_{m_i} is still uniform. Let a_i count distinct randomizers previously queried at $(\text{pk}, \cdot, m_i)$. Conditional on the augmented history, the probability of a prior match is $a_i 2^{-\kappa}$. The messages m_i at distinct first signings are different. Each candidate counted in some a_i therefore has a distinct earlier

public selector query as a witness; in particular, pathwise $\sum_i a_i \leq N_m$. Taking conditional expectations at the at most b first signings and then applying the union bound gives

$$\Pr[B_f] \leq 2^{-\kappa} \mathbb{E} \left[\sum_i a_i \right] \leq 2^{-\kappa} \mathbb{E}[N_m].$$

There is no factor b : a selector query concerns one message only.

Hidden internal-node hash inputs. There are at most $N_g + V$ internal-node hash calls, each with at most d child slots. For a slot whose corresponding child node is outside $\mathcal{C}$, the matching probability is $2^{-\kappa}$, including on cache hits. Therefore

$$\Pr[B_{\text{in}}] \leq d(\mathbb{E}[N_g] + V)2^{-\kappa}.$$

This is where the fan-in bound d is used.

Random internal-node outputs. Each fresh case-(iii) answer is uniform independently of the canonical label at its output node. There are at most $N_g + V$ such samples, so

$$\Pr[B_{\text{out}}] \leq (\mathbb{E}[N_g] + V)2^{-\kappa}.$$

Here the target label may already be known; a fresh uniform answer still hits that specified value with probability $2^{-\kappa}$.

Final payload. After the target and format checks are determined, every supplied word at a selected node outside $\mathcal{C}_{\text{final}}$ matches its still-hidden canonical label with probability $2^{-\kappa}$. There are at most L words, giving $\Pr[B_{\text{pay}}] \leq L2^{-\kappa}$.

Combining these bounds in the same simulation yields

$$\begin{aligned} \Pr[\text{Bad}] &\leq 2^{-\kappa} (\mathbb{E}[N_s + N_m + (d+1)N_g] + (d+1)V + L) \\ &\leq ((d+1)(Q+V) + L)2^{-\kappa}. \end{aligned}$$

The expectations account for adaptively varying query counts. No event bound conditions on the complement of an earlier bad event. $\square$

Scope. The theorem counts access to an ideal uniform selector as one query. Connecting a concrete hash-based selector to this interface requires a separate reduction and query accounting. Section 4.4 gives a separate raw-index model for $+C$. Quantum oracle access is outside both statements.

B Proof of counter-mode security

Proof of Theorem 4.10. We give the changes to the simulator of Appendix A.2, then verify the three inequalities. Retain its independent labels, private randomizers, internal-node hash handlers, and b independent uniform source disclosures. Replace its candidate targets by independent uniform raw indices $U_1, \dots, U_{Q+1} \in [2^r]$, independent of all sources. A public or final index call returns its current table entry, or installs the next U_j .

For the i -th distinct signing request m , independently sample $J_i \in \{0, \dots, C_{\text{max}} - 1, \infty\}$ with

$$\Pr[J_i = c] = (1 - \alpha)^c \alpha, \quad \Pr[J_i = \infty] = (1 - \alpha)^{C_{\text{max}}}.$$

If $J_i = c < \infty$, overwrite the row $(\mathbf{pk}, \rho_m, m, \cdot)$ at counters $0, \dots, c-1$ with independent uniform indices in $[2^r] \setminus [M]$, and at counter c with $\text{Unrank}_{\mathcal{E}}^{-1}(E_i)$. Return the corresponding signature and enlarge the closure by E_i . If $J_i = \infty$, overwrite all C_{max} entries with independent uniform rejected indices and return $\perp$, leaving the closure unchanged. When $\alpha = 1$, only $J_i = 0$ is possible, so the empty rejected range is never sampled. Record the message and response in

either case; repeats return that response. Entries beyond a finite stopping counter are left untouched. No signing call modifies an internal-node hash cache.

Define B_f^C at a first signing to mean that *any* counter in its row was queried publicly beforehand. Keep $B_s, B_{\text{in}}, B_{\text{out}}, B_{\text{pay}}$ from the base proof, with $\Delta^C = B_s \cup B_f^C \cup B_{\text{in}}$ and $\text{Bad}^C = \Delta^C \cup B_{\text{out}} \cup B_{\text{pay}}$. Flags never affect execution. Let Cov^C require a fresh forged message, an accepted final index, and a target in the final signing closure.

Comparison and decomposition. Outside B_f^C a signing row is unqueried. The programmed prefix has exactly the law of independent uniform indices stopped at the first accepted one: a specified prefix of c rejected values followed by a specified accepted value has probability $(1 - \alpha)^c \alpha (2^r - M)^{-c} M^{-1} = 2^{-r(c+1)}$ when $\alpha < 1$; the case $\alpha = 1$ is immediate. The all-rejected case has the correct law as well. Unqueried tail entries remain fresh uniform. The coupling of Lemma A.1 therefore applies up to Δ^C . The root-reconstruction argument is identical, giving

$$\Pr_{\text{Real}^C}[\text{Win}] \leq \Pr_{\text{Sim}^C}[\text{Cov}^C] + \Pr_{\text{Sim}^C}[\text{Bad}^C].$$

Coverage. A fresh forged message has no signing-programmed row, so its index is some U_J . The actual closure is contained in $\text{cl}(E_1 \cup \dots \cup E_b)$, including unused sources and sources assigned to exhausted requests. Define

$$B := \{u \in [M] : \text{Unrank}_{\mathcal{E}}(u) \subseteq \text{cl}(E_1 \cup \dots \cup E_b)\}.$$

Then

$$\text{Cov}^C \subseteq \bigcup_{j=1}^{Q+1} \{U_j \in B\}.$$

Each event has probability αp_b , since U_j and all E_i are independent. This yields $(Q + 1)\alpha p_b$ without conditioning on success, the stopping counters, or the adaptive choice of J .

Exceptional events. Before a first request for m_i , its private ρ_{m_i} remains uniform in the simulator, even given the earlier transcript. Let a_i count the distinct randomizers ρ in prior public queries $(\text{pk}, \rho, m_i, c)$, over all counters. The conditional probability of B_f^C at this request is $a_i 2^{-8n}$. Because the m_i are distinct, $\sum_i a_i \leq N_m$ pathwise, giving $\Pr[B_f^C] \leq 2^{-8n} \mathbb{E}[N_m]$. Programmed rows do not consult hidden label values, so the invariant and other charges in Appendix A.5 are unchanged. Thus

$$\Pr[\text{Bad}^C] \leq 2^{-8n} (\mathbb{E}[N_s + N_m + (d + 1)N_g] + (d + 1)V + L).$$

Use the shared budget, combine the inequalities, and cap at one. The same mixture argument covers randomized adversaries, with signing-response alphabet $\{\perp\} \sqcup \bigsqcup_{\ell=1}^L (\Sigma \times [C_{\max}] \times \Sigma^\ell)$. $\square$

C Serialization and domain requirements

This appendix gives the exact byte layout of the reference implementation; an implementation that follows it produces the same keys and signatures. Appendix E gives test vectors.

Every primitive is SHAKE256: $\text{SHAKE256}(x, \ell)$ denotes the first ℓ bytes of SHAKE256 output on the byte string x . Let $\text{enc}(a)$ be the eight-byte unsigned big-endian encoding of $a \in [2^{64}]$ and, for CUTS+C, $\text{enc}_s(c)$ the unsigned big-endian encoding of $c \in [2^8]$ in exactly $s_b = \lceil s/8 \rceil$ bytes. OS2IP reads a byte string as an unsigned big-endian integer. Nodes are numbered in heap order: the root is 1, and the children of v are $2v$ and $2v + 1$. The parameter identifier is the 56-byte string

$$\text{PID} = \text{enc}(n) \parallel \text{enc}(h) \parallel \text{enc}(w) \parallel \text{enc}(L) \parallel \text{enc}(S) \parallel \text{enc}(r) \parallel \text{enc}(s),$$

with $r = s = 0$ for CUTS. A CUTS key and a CUTS+C key with the same (n, h, w, L, S) therefore never share a tree.

Each primitive hashes an ASCII tag followed by its fields, as in Table 5. The tags have no terminator, and none is a prefix of another. PID has the same length in both modes and fixes the length of every later field except the message m , which comes last, so each input parses uniquely.

Table 5: Inputs to SHAKE256, in order, and output lengths in bytes. **SK** is the secret seed.

Primitive	Input	Output
$F_{\text{leaf}}(\text{SK}; \theta, v)$	CUTS/LEAF/v0 $\parallel$ PID $\parallel$ enc(v) $\parallel$ SK	n
$F_{\text{rand}}(\text{SK}; \theta, \text{pk}, m)$	CUTS/RAND/v0 $\parallel$ PID $\parallel$ SK $\parallel$ pk $\parallel$ m	n
$H_{\text{merge}}(\theta, v, x_L, x_R)$	CUTS/MERGE/v0 $\parallel$ PID $\parallel$ enc(v) $\parallel$ x_L $\parallel$ x_R	n
$H_{\text{chain}}(\theta, v, j, x)$	CUTS/CHAIN/v0 $\parallel$ PID $\parallel$ enc(v) $\parallel$ enc(j) $\parallel$ x	n
$H_{\text{msg}}, \text{CUTS}$	CUTS/HMSG/v0 $\parallel$ PID $\parallel$ enc(c) $\parallel$ pk $\parallel$ ρ $\parallel$ m	β
$H_{\text{msg}}, \text{CUTS+C}$	CUTS/HMSG/v0 $\parallel$ PID $\parallel$ enc _{s} (c) $\parallel$ pk $\parallel$ ρ $\parallel$ m	$\lceil r/8 \rceil$

Selection. In CUTS, $\beta = \lceil (\lceil \log_2 M \rceil + 256)/8 \rceil$. For $c = 0, 1, \dots$ the selector computes $u = \text{OS2IP}(H_{\text{msg}})$, accepts the first $u < T_\beta = \lfloor 2^{8\beta}/M \rfloor M$, and unranks $u \bmod M$, as in Algorithm 6. In CUTS+C, the counter $c \in [2^s]$ is the one the signature carries, $u = \lfloor \text{OS2IP}(H_{\text{msg}})/2^{8\lceil r/8 \rceil - r} \rfloor$ keeps the leading r bits, and c is accepted if $u < M$, in which case u is unranks (Section 5.4). Unranking is as in Section 5.2.

Keys and signatures. The public key is the n -byte root value **pk**. The stored secret key is **SK** $\parallel$ **pk**, $2n$ bytes. A CUTS signature is $\rho \parallel Z_0 \parallel \dots \parallel Z_{L-1}$, exactly $(L+1)n$ bytes, where Z_i is the revealed value of the i -th frontier node in the order unranking returns it. A CUTS+C signature inserts enc _{s} (c) after ρ , for exactly $(L+1)n + s_b$ bytes. The verifier checks the length before parsing.

Implementation requirements. Parameters are supplied externally and must be validated before arithmetic or allocation. Rank counts can exceed machine words; the product $L(w-1)$, loop bounds, and allocation sizes must not be evaluated with unchecked overflow. Although $h = 63$ permits the maximum 64-bit node address, algorithms never form child addresses for a leaf. The CUTS selector’s exhaustion case is explicit in Algorithm 6; its counter never wraps to zero.

D A worked encoding

This appendix works one small instance through by hand, $(h, w, L, S) = (2, 3, 3, 2)$: the counting tables, the unranking of one rank into a frontier and a decoration, and the values a signature reveals. Every number comes from the reference implementation. Appendix E gives byte-level test vectors on the same family.

Counting. Table 6 gives the frontier counts $N[t, \ell]$ for heights $t \leq 2$ and the decoration counts $D_3[k, s]$ of Section 5.2. The tree of height two has $N[2, 3] = 2$ frontiers of size three, and $D_3[3, 2] = 6$ decorations sum to two, so $M = 12$.

Unranking. Table 7 unranks $rk = 9$ by (18) and Algorithms 4 and 5. Each call lists the blocks it considers, in order, with their sizes; it skips each block the rank is not below, subtracting its

Table 6: Counting tables for $(h, w, L, S) = (2, 3, 3, 2)$: frontier counts $N[t, \ell]$ (left) and decoration counts $D_3[k, s]$ (right).

t	$\ell = 1$	$\ell = 2$	$\ell = 3$	k	$s = 0$	$s = 1$	$s = 2$
0	1	0	0	0	1	0	0
1	1	1	0	1	1	1	1
2	1	1	2	2	1	2	3
				3	1	3	6

size, and descends into the first block the rank is below. The result is

$$\text{Unrank}_{\mathcal{E}}(9) = ((4, 5, 3), (1, 0, 1)).$$

Table 7: Unranking $rk = 9$, one row per call. A block is a left subtree size a for `UnrankFrontier` and a first position a for `UnrankDecoration`.

Call	Blocks (left size or position: count)	Result
<code>divmod(9, D₃[3, 2] = 6)</code>		$(rk_F, rk_A) = (1, 3)$
<code>UnrankFrontier(1, 2, 3, 1)</code>	$a = 1: 1, a = 2: 1$	split $a = 2$, ranks $(0, 0)$
<code>UnrankFrontier(2, 1, 2, 0)</code>	$a = 1: 1$	split $a = 1$, ranks $(0, 0)$
<code>UnrankFrontier(4, 0, 1, 0)</code>		stop at node 4
<code>UnrankFrontier(5, 0, 1, 0)</code>		stop at node 5
<code>UnrankFrontier(3, 1, 1, 0)</code>		stop at node 3
<code>UnrankDecoration(3, 2, 3)</code>	$a = 0: 3, a = 1: 2$	position 1
<code>UnrankDecoration(2, 1, 0)</code>	$a = 0: 1$	position 0
<code>UnrankDecoration(1, 1, 0)</code>	$a = 1: 1$	position 1

Disclosure. A signature whose message selects this encoding reveals $(X_{4,1}, X_{5,0}, X_{3,1})$, in the order unranking returns the frontier (Figure 3). Table 8 lists what verification computes from them: it advances each revealed value to the end of its chain, merges nodes 4 and 5 into the start of node 2’s chain and advances that chain, and merges nodes 2 and 3 into `pk`. This takes $V = 8$ tree primitive calls, as in (15), besides the message hash.

Table 8: Verification of the disclosure above.

Node	From	Computes	Tree calls
4	revealed $X_{4,1}$	$X_{4,2}$	1
5	revealed $X_{5,0}$	$X_{5,2}$	2
2	merge of 4 and 5	$X_{2,0}$, then $X_{2,2}$	3
3	revealed $X_{3,1}$	$X_{3,2}$	1
1	merge of 2 and 3	<code>pk</code>	1
		Total	8

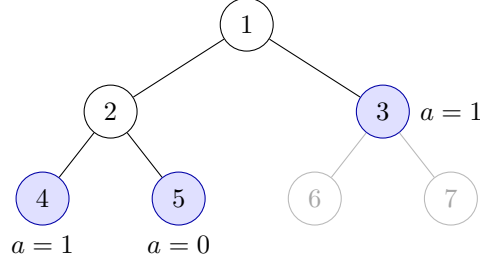


Figure 3: A size-three frontier, shown in blue; chains are suppressed in the node diagram. The disclosure is $(X_{4,1}, X_{5,0}, X_{3,1})$ in canonical order. Verification reconstructs node 2 and then the chainless root.

E Test vectors

This appendix gives one test vector for each mode, on the family $(n, h, w, L, S) = (16, 2, 3, 3, 2)$ of Appendix D. Both were generated and verified with the reference implementation (github.com/p-11/cuts), which instantiates every primitive with SHAKE256. The CUTS+C vector uses a 16-bit counter and the largest r for which the signer exhausts the counter with probability at most 2^{-128} .

The seed SK is the first n bytes of SHAKE256 applied to the ASCII string “CUTS test vector seed: ” followed by the vector’s name, `toy-cuts` or `toy-cuts-c`. Nodes are numbered as in the algorithms of Section 5.1: the root is 1 and the children of v are $2v$ and $2v + 1$. Byte strings are hexadecimal. The signature is $\rho \parallel Z_0 \parallel Z_1 \parallel Z_2$ for CUTS. For CUTS+C, the counter c , as two big-endian bytes, follows ρ . Vectors for every parameter set of Section 5.5 are distributed with the reference implementation.

CUTS

Parameters	$(n, h, w, L, S) = (16, 2, 3, 3, 2)$
Seed SK	5b3aa980072a50f22eb84da994c7c888
Message m	“CUTS test vector” (ASCII)
Public key pk	734f8a6c069ce11f335308c1da132d7e
Randomizer ρ	24ab52aff264c39d86b3cddfd2aa0555
Encoding	$F = (4, 5, 3), A = (0, 1, 1)$
Z_0	0b715aeeb32bd2c27667827575250a6e
Z_1	8f767cd718a4ea5fa647a1d8a61b8e23
Z_2	cce60078260dc61b120b3ebb6e33b8b8

CUTS+C

Parameters	$(n, h, w, L, S) = (16, 2, 3, 3, 2), (r, s) = (13, 16)$
Seed SK	c81a7cbe50b6a65da0b7a06c16dc3bab
Message m	“CUTS test vector” (ASCII)
Public key pk	3273f7f61427ea1dd30e727612009c07
Randomizer ρ	67cf9fd799af504c758b0ad90ea0063c
Counter c	112
Encoding	$F = (2, 6, 7), A = (1, 0, 1)$
Z_0	bf7fd568690872ea495af4807129661c
Z_1	81d0cd286b28c0d2e77fece1a0ace87a
Z_2	8803d04329ba0d75a611a97e9e19bfe3
