Shorter Hash-Based Signatures via Guarded Sharing in a DAG Model

Chongxu Ren and Hongbo Yu

Department of Computer Science and Technology,
Tsinghua University, Beijing, China
rcx23@mails.tsinghua.edu.cn, yuhongbo@tsinghua.edu.cn

Abstract. Stateless hash-based signatures are among the most important post-quantum signature schemes. Although they have been standardized by NIST, they still have large signatures. Within these constructions, the few-time signature component is receiving increasing attention. Recent work has introduced new techniques, including rejection sampling, forced pruning, bucket thinning, and hash chains.
In this work, we continue to improve few-time signatures with a new technique, *guarded sharing*. Our construction moves beyond the previous tree-based model to a DAG-based signature. By combining guarded sharing with bucket thinning, hash chains, and rejection sampling, we obtain a new signature scheme called BGS-CC.
Our construction further improves on the state-of-the-art few-time signature BPORS-C+BFP. Most notably, for 128-bit security and a small use quota of $Q = 2^{24}$, it yields complete signatures of 2,228 bytes. To our knowledge, this is the first SPHINCS-based stateless signature to be shorter than the 2,420-byte signatures of the standardized module-lattice scheme ML-DSA-44.

Keywords: Hash-based signatures · Few-time signatures · Directed acyclic graphs · Guarded sharing · SLH-DSA

1 Introduction

A digital signature allows a verifier to check that a message was authorized by a signer holding the secret key. In the post-quantum setting, stateless hash-based signatures are among the most established signature schemes. Their security rests on conservative assumptions about hash functions, and they can be instantiated using only hash functions and pseudorandom functions [6,25]. These properties make them an important approach to post-quantum signatures.

Hash-based signatures trace back to the one-time constructions of Lamport and Merkle, followed by a line of improvements to one-time signatures [19,22,13]. Merkle introduced hash-tree authentication to aggregate many one-time public keys under a single root, obtaining a *stateful* signature scheme [21,22]. The signer keeps track of which one-time keys have been used. Modern schemes, including XMSS and its multilayer variant XMSS$^{\mathrm{MT}}$, as well as LMS and its hierarchical

variant HSS, have been standardized for stateful hash-based signing [14,20,10]. SPHINCS combined a few-time signature with a hierarchy of one-time signatures and Merkle trees to obtain a practical *stateless* scheme [5]. Its successor SPHINCS+ introduced FORS as the few-time component and became the basis of SLH-DSA, standardized by NIST [6,25].

The SPHINCS+ framework has three main components: a one-time signature, a multilayer Merkle tree called a hypertree, and a few-time signature. While the first two have been studied extensively, the few-time component remains an important target for reducing signature size. Since SPHINCS+ was proposed, several techniques have improved this component, including rejection sampling, forced pruning, bucket thinning, and hash chains [16,1,28,30]. Despite these reductions, there is still room to shorten few-time signatures further.

The central difficulty is controlling the information revealed when a few-time key is reused. In the stateless framework, each signature selects a bottom key from a large address space. With Q historical signatures and 2^h possible addresses, the number of uses of a fixed bottom key follows a binomial distribution. The *coverage model* measures the probability that an independent new encoding passes the signing predicate and that its required openings can be computed from previous disclosures, averaged over this random use count [1,28,30]. This probability must remain below the chosen security term, such as 2^{-128}; Section 2.3 gives the model. Each historical signature should disclose as little information as possible, while a fresh encoding must require enough information to remain hard to cover. Requiring more values in each signature can make a fresh encoding harder to cover, but also lengthens the signature and adds to subsequent disclosure. This tradeoff is a central obstacle to shortening few-time signatures. We therefore ask: *Can we reduce the information disclosed by each signature while keeping the coverage probability small under repeated use?*

Our Contribution. In this work, we further improve few-time signatures by limiting the information disclosed across repeated uses. For the limited-use setting with 128-bit security and $Q = 2^{24}$, our few-time component uses 772 bytes, compared with 2,400 bytes for FORS and 1,140 bytes for BPORS-C+BFP [30]. Section 6 gives the corresponding signing and verification costs and the complete-signature comparisons.

FORS uses a forest of independent Merkle trees, with one selected leaf in each tree [6]. Our main idea is to arrange these choices along a sequence of smaller trees: a selected leaf in one layer authenticates a tree root in the next layer. The signer selects a route through the layers, and the verifier reconstructs the roots along that route in reverse order. Several predecessor leaves can share the same successor tree.

Sharing successor roots alone does not sufficiently restrict disclosure: a known root could be reused wherever that root appears. We therefore bind each use of a successor root to a separately addressed secret, called a *secret guard*. A leaf hashes the guard together with the successor root:

$$\text{leaf} = H_{\text{link}}\big(\text{secret guard}, \text{shared successor root}\big).$$

Opening the leaf requires its own guard secret, even when the successor root is already known. The shared roots form a layered directed acyclic graph (DAG). We call this technique *guarded sharing* (GS). Section 3 develops the basic construction.

Guarded sharing also combines with bucket thinning and hash-chain encoding [28,30]. Our main construction, *BGS-CC*, combines bucket thinning (B), guarded sharing (GS), chain encoding (the first C), and compression through rejection sampling following FORS+C (the second C) [16]. It further improves on the state-of-the-art BPORS-C+BFP construction [30]. Table 1 compares the few-time components at approximately equal signing time for the larger quota $Q = 2^{64}$, under the conditions listed with the table.

Table 1. Few-time signature components at 128-bit security and comparable signing work. Sizes are in bytes; signing and verification work are hash counts.

Scheme	Bytes	Sign	Verify	Fixed size	Tight cov.
HORST [5,1]	2 592	50 331 648	162	✓	×
PORS [2,1]	2 480	50 331 648	162	×	×
FORS [6]	2 464	44 040 187	156	✓	✓
PORS+FP [1]	1 700	47 400 108	113	✓	×
BPORS+FP [28]	1 428	40 817 474	101	✓	×
BPORS-C+BFP [30]	1 108	45 946 949	227	✓	×
Ours	**788**	54 085 639	216	✓	✓

Here $n = 16$, $Q = 2^{64}$ and $h = 64$; signing uses $4 \cdot 10^7$–$6 \cdot 10^7$ expected calls and verification at most 262. Bytes and Verify are worst-case bounds; signing counts round upward. Tight cov. indicates whether a tight coverage analysis is used in the reported parameter calculation. HORST and PORS use the per-leaf PRF and hashed-leaf presentations recalled in [1]. Prior rows count one call per source encoder invocation; Ours retains bounded-output charges (Section 6). Ours is BGS-CC, with coverage given by Corollary 1 and numerical evaluation in Section 6.

1.1 Related Work

FORS and PORS authenticate selected secrets within trees [6,2]. Forced pruning shortens the resulting authentication proofs [1], and bucket thinning reduces the histories reaching each child key [28]. Our closest predecessor, BPORS-C+BFP, combines bucket thinning with leaf-chain encodings and blockwise proof-size budgets [30]. BGS-CC uses a sequence of shared successor trees, with a private guard protecting each selected link. Its one-route authentication has fixed size, so it omits proof-size pruning. These constructions build on selection-based signatures such as BiBa and HORS [26,29]; Section 2.5 explains the mechanisms used here. The v-ORS approach instead changes subset generation by binding the message to the selected FORS instance [31].

FORS+C supplies the rejection-sampling approach that we extend with an adjustable threshold [16]. FORC and constant-sum WOTS encode choices in

chain positions [32,33]; here those choices occur on authentication edges and guards inside the shared graph. Recent work improves WOTS size–verification tradeoffs through non-injective encodings [18]. Qin et al. allow different hypertree heights and chain lengths across layers [27]. We keep WOTS+ and the hypertree fixed and change the bottom few-time component. Limited-use parameter tuning also shortens SPHINCS+ when the signing quota is smaller [11,23]; our evaluation includes these workloads.

DAG formulations of hash-based signatures go back to Bleichenbacher and Maurer [7,9,8], and underlie the verifiable-set treatment of constant-sum WOTS [33]. We use this formulation to describe guarded sharing and analyze pooled disclosures from several signatures. Hevia and Micciancio study the hash assumptions needed beyond the graph model [12]; later work develops the reductions and formal security analyses of XMSS and SPHINCS+ [17,15,3,4]. Our coverage model and its connection to these analyses are given in Section 2.3.

Organization. Section 2 recalls Merkle trees, the SPHINCS framework, coverage, the DAG model, and previous few-time signature ideas. Section 3 develops GS, and Section 4 gives the full BGS-CC construction. Section 5 analyzes its costs and coverage. Section 6 compares BGS-CC with previous schemes under standard and limited-use workloads.

2 Preliminaries

Notation. For a positive integer s, let $[s] = \{0, \ldots, s-1\}$. All logarithms are base two unless written ln. We write $x \leftarrow D$ for sampling from a distribution D, and use the same notation for uniform sampling from a finite set. The notation $[z^E]P(z)$ denotes a coefficient, and $x\|y$ denotes concatenation. Hash values have n bytes.

2.1 Merkle trees

A *Merkle tree* authenticates an ordered collection of values using a single root hash [21,22]. Given 2^ℓ leaf values $y_0, \ldots, y_{2^\ell-1}$, place them from left to right at the leaves of a complete binary tree of height ℓ. Each internal node stores the result of applying H_1 to its address and the ordered values of its left and right children. The address identifies the key, tree and node position. Computing these hashes from the leaves upward gives the *Merkle root*, denoted by $\mathrm{MerkleRoot}_\ell(y_0, \ldots, y_{2^\ell-1})$. For height zero, the root is the sole leaf value.

The *authentication path* of a leaf consists of the sibling value at each of the ℓ levels between that leaf and the root. To authenticate the leaf, the signer supplies this path. The verifier starts from the selected leaf, combines it with each sibling in the order determined by the leaf position, and compares the reconstructed root with the expected root.

When several leaves are opened, their paths can share nodes. Octopus removes the redundant authentication values [2]: take the union of the selected

paths and supply the roots of the unselected subtrees adjacent to that union. Both parties derive these positions from the selected leaves. For example, opening two sibling leaves in a four-leaf tree needs only the opposite pair's root, whereas opening one leaf from each pair needs two sibling values. Thus the authentication size depends on how the selected leaves are arranged.

2.2 The SPHINCS framework

SPHINCS combines a few-time signature with a hierarchy of one-time signatures and Merkle trees to obtain a stateless signature scheme [5]. SPHINCS+ refines this framework, and SLH-DSA standardizes its FORS-based instantiation [6,25]. We recall the interface needed to replace the bottom few-time component while keeping the surrounding layers fixed.

Let h be the total hypertree height and D its number of layers. Standard signing first derives a message randomizer and hashes it with the global public key and the message. The output determines a digest and an address $I \in [2^h]$ selecting a bottom FTS instance. That instance signs the digest; a WOTS+ signature authenticates its public key. A Merkle path authenticates the WOTS+ public key under the root of the bottom subtree. Further WOTS+ signatures and Merkle paths connect this root to the global public root. The complete signature contains the randomizer, the FTS signature, and these WOTS+ signatures and paths [6,25].

A stateless signer does not track which bottom addresses have already been used. Instead, the address space makes repeated use infrequent, and the FTS is designed to tolerate it. We write Q for the number of historical signatures under the complete key; an individual FTS key sees only those histories that select its address. For our modified bottom component, the encoder takes a message representative formed before the bottom address is known and jointly produces that address and an FTS encoding [30]. The bottom root is recovered from the signature and authenticated by the surrounding layers; neither that root nor the derived address is an extra FTS wire field. Section 4.4 specifies this encoder and its counter search.

Size and computation. We count signature size in bytes, including transmitted counters and all authentication values. Our model uses hash counts to measure theoretical time, with each hash or PRF call contributing one unit [28,30]. Signing time is expected time, including stateless regeneration and rejected encoding trials; verification time is the worst case over accepted signatures. Exact formulas for our construction appear in Section 5.2; full-signature comparisons also include the fixed surrounding layers in Section 6.

2.3 The coverage model

Coverage asks whether disclosures from earlier signatures suffice to supply the openings required by a new encoding. We use the accepted-history and raw-target experiment of prior FTS work [1,28,30]. Let D_{raw} be the distribution of

one encoding trial, let Acc be its acceptance predicate, and let D_{acc} be D_{raw} conditioned on acceptance. Independent rejection trials produce this accepted law whenever signing succeeds.

For one bottom key, take a history $H = (X_1, \ldots, X_q)$ of independent $X_i \leftarrow D_{\mathrm{acc}}$ and an independent target $Y \leftarrow D_{\mathrm{raw}}$. Write $\mathsf{Cov}(H; Y)$ when all openings required by Y can be computed from the historical disclosures and public values using the allowed forward hash operations. The coverage probability is

$$\Gamma_q = \Pr[\mathsf{Acc}(Y) \wedge \mathsf{Cov}(H; Y)]. \tag{1}$$

Repeated historical encodings are allowed. The target's acceptance is part of this probability; the target is not sampled from D_{acc}.

In the earlier subset model, coverage means that every selected target secret belongs to the union of previously disclosed secrets [1,28]. For leaf chains, an earlier opening also supplies all later values on that chain [30]. We retain the same experiment and extend this computability test to the DAG in Section 2.4, including all disclosed authentication values. Only the description of available openings changes.

Under the independent uniform bottom-address model, acceptance must leave the address uniform and independent of the accepted FTS encoding. The number of histories at an independent target address is then $N_h \sim \mathrm{Bin}(Q, 2^{-h})$, and the workload coverage is $\rho = \mathbb{E}[\Gamma_{N_h}]$. All components of the target key share this same N_h; its mean is not a replacement for the count distribution. Domain separation prevents other bottom keys from supplying this key's private openings.

Security interpretation. In the forward-computation model of Section 2.4, an accepted target can be signed from earlier disclosures precisely when its required openings are available. If the target is not covered, at least one required opening cannot be obtained from the disclosed and public values by forward computation. A small ρ therefore bounds the probability that previous signatures supply enough information to sign a new independent target.

For a concrete instantiation, PRF pseudorandomness and the appropriate hash preimage and collision guarantees must bound the probability of producing a missing opening or substituting a different computation with the same public commitment. The reduction must handle every signature accepted by the verifier, including those the honest signer would not produce [12]. The full stateless-signature proof framework is given by SPHINCS+C [16], with the few-time coverage analysis developed further in PORS+FP and BPORS+FP [1,28]. This framework separates the few-time covering term from the primitive-security and WOTS+/hypertree terms. Here we derive the DAG covering term ρ for independent accepted histories and a raw target in Section 5.3.

2.4 The DAG model

The graph-based viewpoint of Bleichenbacher and Maurer represents a hash-based signature by the values it reveals and the computations that connect them

to a public key [7,9,8]. We use the verifiable-set formulation recalled by Zhang et al. [33], and apply it to the pooled disclosures of several signatures.

Let $\mathcal{G} = (V, E)$ be a finite directed acyclic graph. Each vertex carries an addressed n-byte value, and edges point from inputs to outputs. Source values are secret; a unique public sink v_{pk} is the root commitment. Every other vertex has a specified public hash operation on an ordered list of predecessors. A join requires all its inputs. A shared value is one vertex used by several later computations, so a graph need not be a tree.

For a known set $A \subseteq V$, its *forward closure* $\mathrm{Cl}_{\mathcal{G}}(A)$ consists of A and every value obtained by repeatedly evaluating an operation whose inputs are all available. This rule neither creates secret sources nor inverts hashes. In particular, knowing a root does not reveal its children. A set A is *verifiable* if $v_{\mathrm{pk}} \in \mathrm{Cl}_{\mathcal{G}}(A)$, and minimal if no proper subset has this property. Write $A \preceq B$ when $A \subseteq \mathrm{Cl}_{\mathcal{G}}(B)$: the openings in B suffice to compute those in A.

An encoding selects the openings that a signature must provide. The verifier uses them and their authentication values to reconstruct the root and compares it with the expected root; the known root cannot replace that computation. For one-time use, an antichain of minimal verifiable sets prevents one allowed opening pattern from deriving a different one by forward computation [33]. Table 2 summarizes this correspondence.

Table 2. DAG objects and their signature roles, following the verifiable-set view [33]. An OTS code is a one-time encoding. Closure includes both the disclosed values and values computable from them.

DAG object	FTS role
Sources	Secrets
Sink	Public key
Verifiable set	Openings
$A \preceq B$	Reuse
Antichain	OTS code
Closure	Known data

For few-time use, we first pool all values disclosed by the historical signatures, including authentication nodes, and then take their closure together with the public values. Two signatures may supply different inputs to the same join, so this can reveal more than closing each signature separately. The predicate $\mathsf{Cov}(H; Y)$ from Section 2.3 holds when this pooled closure contains every opening required by the target. This recovers subset and chain coverage when the graph has the corresponding form.

In the construction, $F_{K_{\mathrm{sk}}}$ derives secret sources, H_0 advances chains, and H_1 joins ordered tree children. We additionally use H_{link} for the successor links of Section 3 and H_{enc} for encoding. All operations have separate domains and addresses identifying the key and relevant positions. Repeated uses of an address refer to the same value. The primitive assumptions underlying this symbolic model are discussed in Section 2.3.

2.5 Previous ideas on few-time signatures

From arrays to authenticated selections. HORS selects a small set of private preimages from a larger public array of hashes [29]. The short disclosures come with a large public key. HORST puts the secrets under a Merkle root, while FORS organizes them in separate trees and takes one secret from each [5,6]. The forest guarantees distinct selected secrets within a signature, at the price of separate authentication paths. These choices move cost between the public commitment and each message's signature.

Sharing authentication. PORS selects distinct positions in one collection, and Octopus removes repeated data from the Merkle proof [2]. Nearby openings share more of their routes to the root than scattered openings. Thus the geometry of the selection affects the byte count. It also affects disclosure: an authentication node is a graph value, not merely a line in a size formula. Its position may matter for what another target can compute.

Searching for cheaper signatures. FORS+C uses a counter to force a virtual index to a fixed value, eliminating its tree [16]. Section 4.4 applies this rejection-sampling idea with an adjustable acceptance threshold. Forced pruning searches for selections whose shared authentication proof is small enough [1]. Both trade search work for bytes, but only the second conditions directly on proof geometry. Selections within an accepted encoding need not retain their original independence. The covering analysis must follow the actual accepted law, rather than simply multiplying a former estimate by a rejection probability.

Thinning repeated disclosure. Bucket thinning uses an outer FORS-like forest whose leaves are public keys of another few-time signature scheme [28]. Each of its K trees contains B independent child keys. A message selects one child per tree; the signer supplies a signature under that child key and a Merkle path authenticating its public key. The verifier recovers the child public key from the child signature, then follows the outer path. Thus the child signature replaces the secret-leaf opening of ordinary FORS. BPORS+FP uses PORS children and applies forced pruning to their combined authentication size.

Information in chain positions. FORC permits leaf-chain positions to encode additional choices [32]. Constant-sum Winternitz patterns coordinate those positions so that advancing one valid pattern cannot produce a different valid pattern [33]. Two componentwise comparable depth vectors with the same sum must coincide. BPORS-C+BFP combines chain positions with bucket thinning and local proof budgets for blocks of coordinates [30]. After several signatures, different coordinates may have been opened early in different histories, so the one-time antichain intuition does not by itself establish a few-time guarantee.

3 Guarded Sharing: The Basic Construction

We first explain guarded sharing (GS) with ordinary Merkle paths and one private guard per selected leaf. We call each small Merkle tree an *owner*. Section 4 adds bucket thinning and chain-depth encoding.

Figure 1 uses four initial owners, three routing layers, and two leaves per tree to distinguish the roles of O, r, and t. Squares denote secret PRF sources; circles denote hash-derived values. Orange nodes are the values supplied by one signature; teal nodes are reconstructed from them.

3.1 The idea of the construction

Begin with several small Merkle trees. A message selects one leaf in a tree, then uses that choice to identify a tree in the next layer. Repeating the step produces a route. The goal is to authenticate the whole route under one public root without building an independent successor subgraph for every possible prefix of the route.

Share the successor trees. In Figure 1, the four layer-0 owners share the layer-1 roots $R_{1,0}$ and $R_{1,1}$. Slot 0 in every initial owner refers to $R_{1,0}$; slot 1 refers to $R_{1,1}$. Each shared root is computed once and is an input to four different first-layer leaves. The message does not change these edges: it chooses which owner and slot to open. This reuse is the source of the DAG structure. A separate copy of every successor tree would instead give a larger unfolded tree of trees.

Make each use require its own secret. Sharing a root must not make every leaf that refers to it freely openable. For each owner and slot, derive a distinct secret guard $s_{i,o,x}$. A nonfinal leaf hashes this guard together with the chosen successor root. Thus the four leaves referring to $R_{1,0}$ share that root but have different guards. A known successor root alone does not supply any of their guard secrets. If an authentication path discloses the leaf hash, it discloses the output of this combination, not its private inputs. At the final layer there is no successor, so the leaf is the hash of its private guard.

Choose the route forward; verify it backward. Each layer's leaf choice selects the next owner. Verification runs in the opposite direction: a recovered successor root and the preceding guard give the preceding leaf, whose Merkle path recovers its owner root. The signature therefore supplies guards and authentication siblings; successor roots are reconstructed along the route.

3.2 A sequence of small trees

Fix a height $L \geq 1$, a routing length $r \geq 2$, and an initial-owner count $O = 2^u$. Write $t = 2^L$ for the number of leaves in each small tree. There are O owners in layer zero and t owners in each later layer. The root of owner o in layer i is $R_{i,o}$.

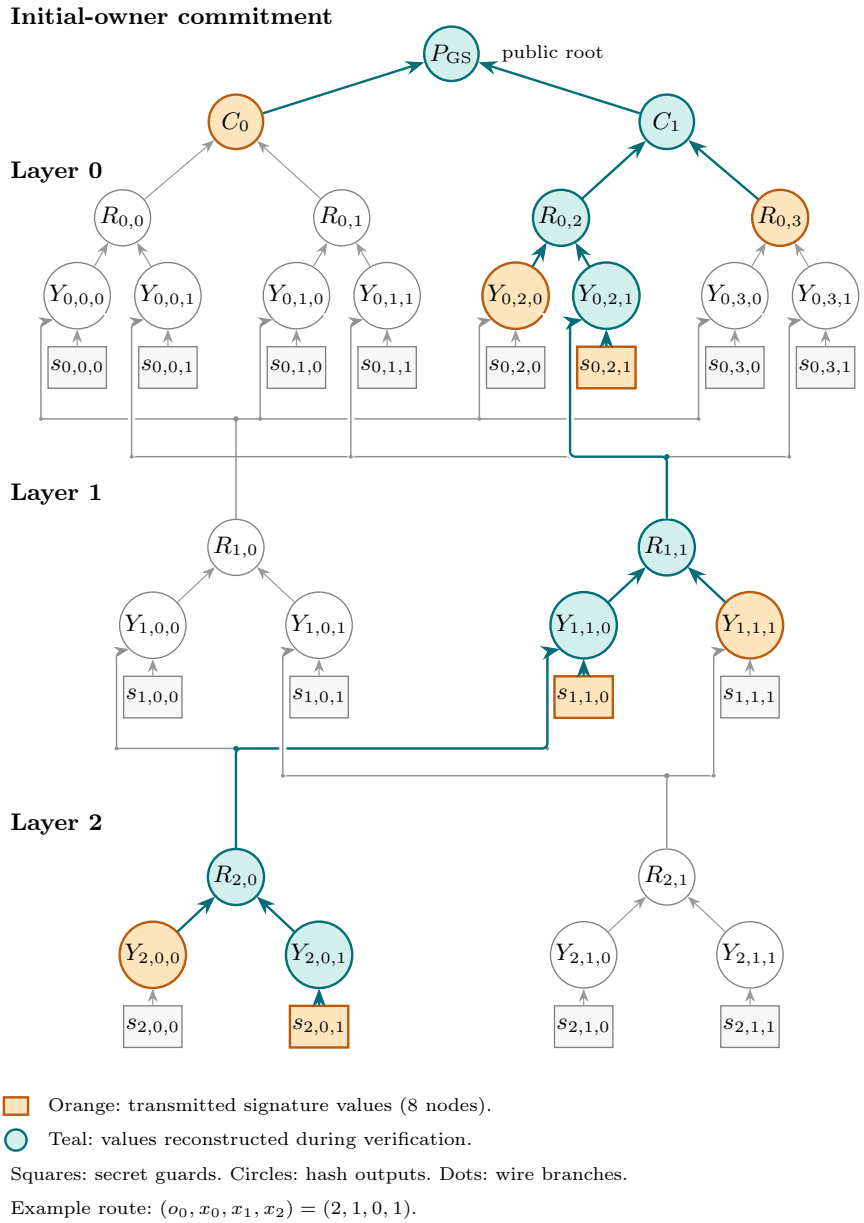


Fig. 1. Plain GS with $O = 4$ initial owners, $r = 3$ layers, and $t = 2$ leaves per tree ($L = 1$). Orange nodes form one signature; teal nodes are reconstructed. C_0 and C_1 commit to initial owners 0–1 and 2–3, respectively. Arrows show hash inputs; crossings without dots are not joins.

A message encoding specifies an initial owner and one leaf slot per layer:

$$z = (o_0, x_0, \ldots, x_{r-1}) \in [O] \times [t]^r.$$

The slot chosen in one layer is the owner chosen in the next:

$$o_{i+1} = x_i, \qquad 0 \le i < r - 1. \tag{2}$$

The route is selected from the first layer to the last, but its roots will be recovered in the opposite direction.

3.3 One private guard for each use of a shared root

Let K_{sk} be a secret seed. All hashes and pseudorandom-function calls are addressed and domain-separated; the bottom-key address and public seed are suppressed here. For each layer, owner and slot, derive

$$s_{i,o,x} = F_{K_{\mathrm{sk}}}(\mathsf{guard}, i, o, x).$$

At the final layer hash the guard to obtain the leaf:

$$Y_{r-1,o,x} = H_0(\mathsf{guardstep}, r-1, o, x, s_{r-1,o,x}). \tag{3}$$

At every preceding layer use

$$Y_{i,o,x} = H_{\mathrm{link}}(i, o, x, s_{i,o,x}, R_{i+1,x}). \tag{4}$$

The root $R_{i,o}$ is the height-L Merkle root of the ordered values $Y_{i,o,0}, \ldots, Y_{i,o,t-1}$, using Section 2.1's convention. Finally,

$$P_{\mathrm{GS}} = \mathrm{MerkleRoot}_u(R_{0,0}, \ldots, R_{0,O-1}). \tag{5}$$

For $O = 1$ this is just $R_{0,0}$. For $O = 2$ it is one addressed binary hash. A long-input hash of all O initial roots would also be a commitment, but would require the other $O - 1$ roots to authenticate one selection; the Merkle commitment needs only u values.

The only secret sources are the $s_{i,o,x}$; all other graph values follow from the displayed operations. Each shared root is one vertex with several outgoing edges, while its guards and link hashes have separate addresses.

3.4 Signing a route and checking it backward

Key generation works from the last routing layer toward the first, and then constructs the initial-owner commitment. It returns its compact root and retains only the ordinary seed material. The graph can be regenerated from the seed; it need not be retained between signing operations.

Given $z = (o_0, x_0, \ldots, x_{r-1})$, a signature supplies the selected private secret s_{i,o_i,x_i} and the L sibling values of its Merkle authentication path for each routing layer. It also supplies the u sibling values that connect R_{0,o_0} to P_{GS}. All route indices are determined by the message encoding, not sent as additional fields.

Verification evaluates (3) and then (4) backward along the route, using each layer's siblings to recover its owner root. The initial-owner path yields a freshly computed P_{GS}, which is compared with the public key.

A complete three-layer example. Take $L = 1$, $t = 2$, $r = 3$, and $O = 4$, as in Figure 1. Write $C_0 = H_{\text{top}}(R_{0,0}, R_{0,1})$ and $C_1 = H_{\text{top}}(R_{0,2}, R_{0,3})$ for the two intermediate commitment nodes. For $(o_0, x_0, x_1, x_2) = (2, 1, 0, 1)$, the signature supplies the eight orange nodes, listed with the bottom-up initial-owner path first:

$$\big(R_{0,3},\ C_0,\ s_{0,2,1},\ Y_{0,2,0},\ s_{1,1,0},\ Y_{1,1,1},\ s_{2,0,1},\ Y_{2,0,0}\big). \tag{6}$$

The verifier reconstructs the teal nodes from the final layer upward:

$$\begin{aligned}
\widehat{Y}_{2,0,1} &= H_0(s_{2,0,1}), & \widehat{R}_{2,0} &= H_1(Y_{2,0,0}, \widehat{Y}_{2,0,1}),\\
\widehat{Y}_{1,1,0} &= H_{\text{link}}(s_{1,1,0}, \widehat{R}_{2,0}), & \widehat{R}_{1,1} &= H_1(\widehat{Y}_{1,1,0}, Y_{1,1,1}),\\
\widehat{Y}_{0,2,1} &= H_{\text{link}}(s_{0,2,1}, \widehat{R}_{1,1}), & \widehat{R}_{0,2} &= H_1(Y_{0,2,0}, \widehat{Y}_{0,2,1}),\\
\widehat{C}_1 &= H_{\text{top}}(\widehat{R}_{0,2}, R_{0,3}), & \widehat{P}_{\text{GS}} &= H_{\text{top}}(C_0, \widehat{C}_1).
\end{aligned}$$

Addresses and fixed left/right ordering are implicit in these expressions. The signature contains three guard secrets, three within-owner siblings, and two initial-owner siblings. The successor roots $R_{1,1}$ and $R_{2,0}$ are computed, not transmitted. This example illustrates correctness; its small parameters are chosen for readability.

3.5 Disclosure across repeated uses

A signature may reveal a raw link leaf as an authentication sibling. That reveals the output of H_{link}, not its private guard input. To open a selected nonfinal leaf, the verifier needs the privately addressed guard secret as well as the successor root. At the final layer, a sibling may reveal a guard *endpoint*; this does not reveal an earlier guard value.

In the ideal forward-computation DAG, the selected private guard is therefore available only after a history has selected its exact layer, owner and slot. Such a history also supplies the authentication of that slot within its owner. Sharing a root does not create the guard capability at another owner. It does not follow that different historical routes cannot be spliced: one history may supply one layer and another history a later layer. The analysis must preserve that dependence rather than require the target route to have appeared intact.

Coverage in the running example. For $O = 4$, $t = 2$, and $r = 3$, fix the target route $(2, 1, 0, 1)$. A uniform history (o'_0, x'_0, x'_1, x'_2) supplies its three guards under the respective conditions

$$(o'_0, x'_0) = (2, 1), \qquad (x'_0, x'_1) = (1, 0), \qquad (x'_1, x'_2) = (0, 1).$$

Each guard must be supplied by at least one history; a history supplying it also supplies its authentication path. The probabilities that one history misses the three conditions are 7/8, 3/4, and 3/4. The probabilities of missing each pair are 11/16, 21/32, and 5/8, and that of missing all three is 9/16. Thus,

for q independent uniform histories, inclusion–exclusion gives the exact coverage probability

$$C(q) = 1 - \left(\frac{7}{8}\right)^q - 2\left(\frac{3}{4}\right)^q + \left(\frac{11}{16}\right)^q + \left(\frac{21}{32}\right)^q + \left(\frac{5}{8}\right)^q - \left(\frac{9}{16}\right)^q. \tag{7}$$

In particular, $C(1) = 1/32$ and $C(2) = 89/1024$. With two histories, the probability that the target route itself appears is only $1 - (31/32)^2 = 63/1024$; the remaining $26/1024$ comes from combining different routes. By symmetry the same calculation holds for every target. Section 5.3 extends it to arbitrary parameters and averages over the binomial use count.

3.6 Size of the basic signature

Every route has the same number of graph-value words,

$$M_{\mathrm{GS}} = u + r(L+1), \qquad L_{\mathrm{GS}} = nM_{\mathrm{GS}}. \tag{8}$$

A counter or another encoder field, if used, is additional. The encoding space has size Ot^r and can be sampled without rejection because O and t are powers of two. Sections 5.2 and 5.3 give one cost and covering analysis for both this child and its chain-encoded extension; plain GS is the case with zero selectable chain depths. The latter analysis accounts for histories that supply different layers of the target route.

4 BGS-CC: The Combined Construction

We combine the GS child with constant-sum chain encoding (C) [33,30] and bucket thinning (B) [28], together with rejection sampling following FORS+C [16]. The second C denotes this compression through rejection sampling. We retain the BPORS-C+BFP symbols: K coordinates, $B = 2^b$ buckets per coordinate, and $t = 2^L$ leaves per owner. Here the $k = L + 1$ chain positions are one guard and L authentication edges, whereas BPORS-C+BFP uses k selected leaf chains. The $O = 2^u$ initial owners belong inside each child.

4.1 Parameters and the contribution of each mechanism

The construction parameters are

$$n \geq 4, \quad K \geq 1, \quad B = 2^b, \quad O = 2^u, \quad t = 2^L, \quad r \geq 2,$$
$$k = L + 1, \quad \boldsymbol{c} = (c_0, \ldots, c_L) \in \mathbb{Z}^k_{\geq 0}, \quad 0 \leq E \leq \textstyle\sum_{j=0}^{L} c_j.$$

The bottom-address length h and lifetime quota Q belong to the workload; finite counter and encoder parameters are given in Section 4.4. Table 3 separates the operations.

Table 3. Components of BGS-CC and their roles. Guarded sharing is defined in Section 3.

Component	Parameters	Role
Guarded sharing	$r, t = 2^L, O = 2^u$	Reuse roots
Chain encoding [30,32]	$k = L + 1, \boldsymbol{c}, E$	Encode depths
Bucket thinning [28]	$K, B = 2^b$	Reduce reuse
Rejection sampling [16]	Δ	Filter encodings

Guarded sharing binds successor roots to private guards. Chain encoding adds depth choices to the k values already sent per layer. Bucket thinning selects one of B independent children in each of K coordinates. Rejection sampling following FORS+C uses an independent threshold test to trade encoding work for smaller parameters.

Each c_d bounds the selectable depth at position d. Equal capacities $c_d = c$ recover the scalar notation (k, c, E) of BPORS-C+BFP. Unequal capacities permit the application parameter choices in Section 6.

4.2 Chain-encoded guarded children

An ordinary Merkle path supplies a sibling value at each join. We instead place a chain of c_d hashes on each edge entering a join at depth d. The verifier reconstructs the selected child and traverses all c_d steps on its edge. For the sibling, the encoding chooses a depth $0 \le e_d \le c_d$: the signature supplies the value at position $c_d - e_d$, leaving e_d hashes to reach the endpoint. Both endpoints are then hashed together. A guard opening similarly leaves e_L steps before the guard is used in its leaf. These depth choices encode additional information in the same $L + 1$ values sent per layer; we constrain their sum by (11) below.

Fix a bottom address I, coordinate $\nu \in [K]$, and bucket $j \in [B]$. Within this subsection, suppress these three indices in graph-value subscripts; they remain part of every primitive address. For each layer $i \in [r]$, owner o, and slot $x \in [t]$, derive a private guard chain from $F_{K_{\mathrm{sk}}}$:

$$\begin{aligned} a_{i,o,x,0} &= F_{K_{\mathrm{sk}}}(\mathsf{sourceaddr}), \\ a_{i,o,x,v+1} &= H_0(\mathsf{guardaddr}, v, a_{i,o,x,v}), \qquad 0 \le v < c_L. \end{aligned} \tag{9}$$

The leaf is

$$Y_{i,o,x} = \begin{cases} H_{\mathrm{link}}(\mathsf{linkaddr}, a_{i,o,x,c_L}, R_{i+1,x}), & i < r-1, \\ H_0(\mathsf{guardaddr}, c_L, a_{i,o,x,c_L}), & i = r-1. \end{cases} \tag{10}$$

Thus each nonfinal leaf binds its private guard prefix directly to the successor root; the final leaf hashes the prefix once more. Guard and link addresses contain (I, ν, j, i, o, x); tree-edge and join addresses contain (I, ν, j, i, o, d, v). Each operation also includes its role tag, relevant chain step, public seed, and BGS-CC suite identifier. Outer commitments use separate role tags and their tree positions. Every owner remains a complete height-L binary tree. On the edge from

a child at depth $d+1$ and position v to its parent at depth d, put an addressed chain $z_{i,o,d,v,0}, \ldots, z_{i,o,d,v,c_d}$, with its step-zero value equal to that raw child. The parent hashes the two endpoints in left/right order. Zero capacity means an identity edge, not a fresh vertex or a free secret. The resulting root is $R_{i,o}$. We write $\mathsf{Chain}(X; \alpha, \beta)$ for the value obtained from X by applying addressed chain steps α through $\beta - 1$.

We use the standard constant-sum code [33,30], allowing a separate capacity at each position:

$$\begin{aligned} \mathcal{L}_{k,\boldsymbol{c},E} &= \{e \in \mathbb{Z}_{\geq 0}^{k} : e_j \leq c_j \ (j \in [k]),\ \textstyle\sum_{j=0}^{k-1} e_j = E\}, \\ N &= |\mathcal{L}_{k,\boldsymbol{c},E}| = [z^E] \prod_{j=0}^{k-1} (1 + z + \cdots + z^{c_j}). \end{aligned} \tag{11}$$

The equal-sum antichain argument is unchanged. Lexicographic unranking uses suffix coefficients of this product to count completions of each prefix; a uniform rank in $[N]$ therefore gives a uniform codeword. Depths $e_0, \ldots, e_{L-1}$ specify authentication-edge openings; e_L is the guard depth. The corresponding values are at positions $c_d - e_d$ and $c_L - e_L$. After the e_L guard-chain steps, forming the leaf takes one further hash, using the successor root at a nonfinal layer.

With $\boldsymbol{c} = \mathbf{0}$ and $E = 0$, this child is exactly the GS core of Section 3.

4.3 Bucket thinning and the public root

Apply bucket thinning [28,30] to independent copies of this child. Restoring the coordinate and bucket indices, the child root and coordinate root are

$$\begin{aligned} C_{\nu,j} &= \mathrm{MerkleRoot}_u(R_{\nu,j,0,0}, \ldots, R_{\nu,j,0,O-1}), \\ R_{\nu}^{\mathrm{out}} &= \mathrm{MerkleRoot}_b(C_{\nu,0}, \ldots, C_{\nu,B-1}). \end{aligned}$$

A height-b path authenticates the selected child in each coordinate. A fixed binary reduction of the K coordinate roots gives the n-byte public key P_I: split their ordered interval recursively into two nonempty halves, with identity for $K = 1$. Its $K - 1$ addressed joins add work but no authentication values, because the signature reconstructs every coordinate root.

A candidate encoding independently chooses one child bucket and one route/code sequence in every coordinate:

$$\Omega = \left([B] \times [O] \times ([t] \times \mathcal{L}_{k,\boldsymbol{c},E})^r\right)^K, \qquad U = \left(BO(tN)^r\right)^K. \tag{12}$$

Coordinate ν visits owners $o_{\nu,0}, \ldots, o_{\nu,r-1}$ with $o_{\nu,i+1} = x_{\nu,i}$. All coordinates and buckets have independent private source addresses.

4.4 Encoding, algorithms and canonical bytes

We extend the rejection-sampling rule of FORS+C [16] by allowing a general threshold in place of a fixed number of zero bits. The signer searches over a

counter to trade additional encoding work for smaller signature parameters. Requiring r' leading bits of the message-and-counter hash to be zero gives a test that passes with probability $2^{-r'}$. We reserve these bits separately from the bottom-key address, so the accepted address remains uniform. A general threshold permits finer adjustment of the signing work than an integer number of zero bits.

Let $\ell = \lceil \log_2 U \rceil$. The public context ctx contains the suite identifier, parameters and public seed. Given a pre-address message representative $M \in \{0,1\}^*$ and counter κ, the encoder produces

$$W_\alpha = H_{\mathrm{enc}}(\mathsf{ctx}, M, \kappa, \alpha) \in \{0,1\}^{8n}, \qquad \alpha = 0, 1, \ldots.$$

Tuple encodings are unambiguous, and integer fields use the suite's fixed-width big-endian encoding. In the coverage experiment, outputs at distinct encoder inputs are independent uniform blocks. Parse $W_0 \| W_1 \| \cdots$ as a 32-bit rejection-sampling value y, an h-bit bottom address I, and an ℓ-bit rank z; ignore any remaining bits. Accept precisely when

$$y < \Delta, \qquad z < U, \qquad 1 \le \Delta \le 2^{32}. \tag{13}$$

The leading-zero rule is the special case $\Delta = 2^{32-r'}$ for an integer $0 \le r' \le 32$. The parameter sets in Section 6 use thresholds with at most three set bits, giving finer control of signing work while retaining simple public parameters. The rank test removes unused binary encodings when U is not a power of two, so accepted ranks are uniform on $[U]$. The signer tries successive counter values until both tests pass; the verifier checks only the supplied counter. Since $n \ge 4$, the threshold test requires only the first encoder block; the remaining blocks are requested only if $y < \Delta$. Define

$$p = \Delta / 2^{32}, \qquad \eta = U / 2^{\ell}, \qquad \tau = p\eta, \qquad q_{\mathrm{enc}} = \left\lceil \frac{32 + h + \ell}{8n} \right\rceil.$$

Here p and η are the threshold and rank acceptance probabilities, τ is the success probability of one trial, and q_{enc} counts the blocks in a complete encoding. Conditional on acceptance, I is uniform and independent of the uniform rank $z \in [U]$. Canonical mixed-radix unranking of (12), with lexicographic unranking of (11), gives the independently uniform coordinates. Rank rejection alone corresponds to $p = 1$.

The counter occupies four bytes and ranges from 0 to $2^{32}-1$, allowing at most 2^{32} trials. This is sufficient for every evaluated parameter set: the probability of exhaustion over its full signing quota is at most 2^{-73}, as calculated in Section 6.

The independent rejection-sampling test multiplies raw coverage by p, as shown in Section 5.3. Unlike forced pruning, it uses independent bits rather than a candidate's authentication size. The encoder's independently indexed blocks let the signer check $y < \Delta$ before computing the rank, with signing cost given in Section 5.2.

The signature format. For coordinate ν, let A_ν be the bucket authentication path of length b, U_ν the initial-owner path of length u, and $s_{\nu,i,d}$ the selected sibling-edge opening at depth d of layer i. The selected guard opening is $s_{\nu,i} = a_{\nu,j_\nu,i,o_{\nu,i},x_{\nu,i},c_L-e_{\nu,i,L}}$. The byte format is

$$\sigma = \kappa \parallel \Big\|_{\nu=0}^{K-1} \Big(A_\nu \| U_\nu \| \Big\|_{i=0}^{r-1} (s_{\nu,i} \| s_{\nu,i,0} \| \cdots \| s_{\nu,i,L-1}) \Big) . \tag{14}$$

The bucket path A_ν and initial-owner path U_ν are stored bottom-up. Layer records follow the forward route order. Within each owner, sibling-edge openings are stored root-to-leaf, in increasing depth d; the verifier consumes them in reverse order. Each graph value occupies n bytes, and the four-byte counter is big-endian. There are no separately transmitted indices or depths. The verifier rejects malformed lengths, failed rejection-sampling tests, and inconsistent repeated addresses. It checks the supplied counter, not whether it was the first accepted one.

Regeneration returns the root together with a map $\mathcal{D}$ from requested vertex addresses to their values. Its collector records every requested source, intermediate chain point and Merkle value as it is visited, including step-zero aliases. Signing serializes this map in the order of (14). Algorithm 3 recovers the bottom address and root from the signature. For a standalone public key (I, P_I), verification accepts exactly when recovery returns (I, P_I). In the SPHINCS framework, the recovered pair is passed to the WOTS+/hypertree verification described in Section 2.2.

4.5 Does forced pruning reduce this signature?

Forced pruning helps PORS-type schemes because different selections can yield authentication proofs of different sizes [1,30]. Nearby selected leaves share more authentication data, so the signer can search for a shorter proof.

BGS-CC opens one route per coordinate through complete trees. Every route therefore contributes the same b bucket-path values, u initial-owner-path values, and L sibling openings plus one guard opening at each of its r layers. Sharing successor roots does not eliminate these openings: each sibling branch and each selected leaf has its own private guards, which the other supplied values cannot recover. The initial-owner and bucket paths likewise authenticate unselected private subtrees, and separate addresses prevent one coordinate from supplying another's private values. Chain depths change the positions of the openings, but not their number. Thus every encoding has the same signature length, leaving no shorter candidates for proof-size pruning to select. We therefore omit forced pruning.

Algorithm 1 Key generation and deterministic full regeneration

Require: Public parameters, public seed, indexed bottom address I

function KeyGen(I)
  Sample secret seed K_{sk}
  $(P_I, \varnothing) \leftarrow$ Regenerate($K_{\mathrm{sk}}, I, \varnothing$) ▷ root only
  return (K_{sk}, P_I)
function Regenerate($K_{\mathrm{sk}}, I, \mathcal{A}$)
  Initialize $\mathcal{D} \leftarrow \varnothing$; record each visited (a, v) with $a \in \mathcal{A}$ as $\mathcal{D}[a] \leftarrow v$
  for $\nu = 0, \ldots, K-1$ **do**
    for $j = 0, \ldots, B-1$ **do**
      for $i = r-1, \ldots, 0$ **do** ▷ successor roots first
        $m_i \leftarrow O$ if $i = 0$, and $m_i \leftarrow t$ otherwise
        for $o = 0, \ldots, m_i - 1$ **do**
          for $x = 0, \ldots, t-1$ **do**
            $s \leftarrow F_{K_{\mathrm{sk}}}(\mathsf{sourceaddr})$; collect the source if requested
            $\xi \leftarrow \mathsf{Chain}(s; 0, c_L)$; collect requested guard values
            if $i = r-1$ **then** ▷ no successor root
              $V_{L,x} \leftarrow H_0(\mathsf{guardaddr}, c_L, \xi)$
            else ▷ guard the successor root
              $V_{L,x} \leftarrow H_{\mathrm{link}}(\mathsf{linkaddr}, \xi, R_{\nu,j,i+1,x})$
          for $d = L-1, \ldots, 0$ **do** ▷ build the owner tree
            for $v = 0, \ldots, 2^d - 1$ **do**
              $Z_0 \leftarrow \mathsf{Chain}(V_{d+1,2v}; 0, c_d)$; $Z_1 \leftarrow \mathsf{Chain}(V_{d+1,2v+1}; 0, c_d)$
              $V_{d,v} \leftarrow H_1(\mathsf{nodeaddr}, Z_0, Z_1)$
          $R_{\nu,j,i,o} \leftarrow V_{0,0}$
        $C_{\nu,j} \leftarrow \mathrm{MerkleRoot}_u(R_{\nu,j,0,0}, \ldots, R_{\nu,j,0,O-1})$ ▷ initial owners
      $R^{\mathrm{out}}_{\nu} \leftarrow \mathrm{MerkleRoot}_b(C_{\nu,0}, \ldots, C_{\nu,B-1})$ ▷ buckets
  $P_I \leftarrow$ the fixed binary reduction of $(R^{\mathrm{out}}_0, \ldots, R^{\mathrm{out}}_{K-1})$
  return $(P_I, \mathcal{D})$

Algorithm 2 Bottom signing

Require: K_{sk}, public context, pre-address message representative M
Ensure: (I, P_I, σ), or $\perp$; only σ is the FTS byte string

for $\kappa = 0, \ldots, 2^{32} - 1$ **do**
  Query $W_0 = H_{\mathrm{enc}}(\mathsf{ctx}, M, \kappa, 0)$ and parse its 32-bit value y
  if $y \geq \Delta$ **then**
    continue
  Query the remaining $q_{\mathrm{enc}} - 1$ blocks; parse $(y, I, z, \mathsf{unused})$
  if $z \geq U$ **then**
    continue
  Unrank z into the Cartesian space (12)
  Derive every $o_{\nu,i+1} = x_{\nu,i}$ and the transmitted-value address set $\mathcal{A}$
  $(P_I, \mathcal{D}) \leftarrow$ Regenerate($K_{\mathrm{sk}}, I, \mathcal{A}$)
  Serialize κ and $\mathcal{D}$ as (14) to obtain σ
  return (I, P_I, σ)
return $\perp$

Algorithm 3 Bottom-address and root recovery

Require: Public context, M, received σ
Ensure: Recovered (I, P'), or $\perp$
if $|\sigma| \neq 4 + nK(b + u + rk)$ **then**
  return $\perp$
Parse all fixed-width fields and recompute the received counter's indexed blocks
Decode (y, I, z); return $\perp$ if $y \geq \Delta$ or $z \geq U$
Unrank the encoding; compute all route and transmitted-value addresses
for $\nu = 0, \ldots, K-1$ **do**
  for $i = r-1, \ldots, 0$ **do** ▷ X is the successor root when $i < r-1$
    $Z \leftarrow \mathsf{Chain}(s_{\nu,i}; c_L - e_{\nu,i,L}, c_L)$
    if $i = r-1$ **then**
      $X \leftarrow H_0(\mathsf{guardaddr}, c_L, Z)$
    else
      $X \leftarrow H_{\text{link}}(\mathsf{linkaddr}, Z, X)$
    for $d = L-1, \ldots, 0$ **do**
      $\chi \leftarrow \lfloor x_{\nu,i} / 2^{L-d-1} \rfloor \bmod 2$; $v \leftarrow \lfloor x_{\nu,i} / 2^{L-d} \rfloor$
      $Z_\chi \leftarrow \mathsf{Chain}(X; 0, c_d)$ on edge $(d, 2v + \chi)$
      $Z_{1-\chi} \leftarrow \mathsf{Chain}(s_{\nu,i,d}; c_d - e_{\nu,i,d}, c_d)$ on edge $(d, 2v + 1 - \chi)$
      $X \leftarrow H_1(\mathsf{nodeaddr}_{d,v}, Z_0, Z_1)$
  $X \leftarrow \text{RecoverMerkle}_u(o_{\nu,0}, X, U_\nu)$
  $R_\nu^{\text{out}} \leftarrow \text{RecoverMerkle}_b(j_\nu, X, A_\nu)$
$P' \leftarrow$ the fixed binary reduction of $(R_0^{\text{out}}, \ldots, R_{K-1}^{\text{out}})$
Return $\perp$ for inconsistent repeated-address values; otherwise return (I, P')

5 Analysis of Our Algorithm

5.1 Correctness

Every chain opening in an honest signature is a label in the deterministically generated DAG. Extending the final selected guard yields its generated leaf. Induction through its ordered path joins yields the final owner root. At each preceding layer, the recovered successor root and its separately reconstructed private guard prefix give the generated link leaf, and the path gives that owner's root. The two ordinary paths then recover the child-coordinate commitment. Finally, the binary combination of the reconstructed K coordinate roots equals P_I. Recovery therefore returns the generated address and public root. Thus every nonfailure honest signature is accepted by the matching key.

5.2 Sizes and work bounds

Generating one owner uses t source-derivation PRFs, tc_L guard-chain hashes, t leaf hashes, and $t-1$ tree joins. At depth d, the 2^{d+1} child edges each require c_d hashes. Summing these contributions gives the same generation count for every owner:

$$J = t(c_L + 3) + \sum_{d=0}^{L-1} 2^{d+1} c_d - 1.$$

The child regeneration count and the full FTS generation count are

$$G_{\text{child}} = \big(O + (r-1)t\big)J + O - 1, \tag{15}$$

$$G = KB(G_{\text{child}} + 1) - 1. \tag{16}$$

Equation (15) includes the initial-owner commitment. In (16), every coordinate contributes $B-1$ bucket joins, and the final combination contributes $K-1$ joins. No regeneration is subsidized by persistent private graph state.

The complete component costs are

$$\mathcal{L} = 4 + nK\big(b + u + rk\big), \tag{17}$$

$$\mathcal{S}^{\text{ub}} = G + \frac{1 + (q_{\text{enc}} - 1)p}{\tau} = G + \frac{1}{\tau} + \frac{q_{\text{enc}} - 1}{\eta}, \tag{18}$$

$$\mathcal{V} = q_{\text{enc}} + K\Big(b + u + r\big(E + \textstyle\sum_{d=0}^{L-1} c_d + L + 1\big)\Big) + K - 1. \tag{19}$$

The plain GS graph uses $K = B = 1$, $\boldsymbol{c} = \mathbf{0}$, $E = 0$ and no rejection: $J = 3t - 1$ and graph recovery costs $u+r(L+1)$ calls. Every accepted codeword has ordinary sum E, so (19) is a worst accepted count rather than a depth average. Each rejection-sampling trial queries one block unconditionally and $q_{\text{enc}} - 1$ further blocks with probability p; the renewal equation gives (18). All graph regeneration occurs after a successful encoding.

Stream child instances and coordinates in sequence. Adjacent routing layers need two t-entry root buffers; the initial-owner and bucket trees use stacks. The output signature and fixed parameter-only unranking tables are additional. Integer unranking, address formation and primitive-state storage are not counted as hash calls.

Signing reliability. Counter exhaustion returns a signing failure. With $R = 2^{32}$ trials, one signing call fails with probability $(1 - \tau)^R$, and its unconditional expected cost is $(1 - (1 - \tau)^R)\mathcal{S}^{\text{ub}}$. Over the signing quota Q, the union bound gives

$$\Pr[\text{at least one signing failure}] \le Q(1 - \tau)^R. \tag{20}$$

We assess this probability separately from the coverage requirement.

5.3 Covering guarded sharing and applying bucket thinning

We use the coverage experiment of Section 2.3 with the DAG computability rule of Section 2.4, pooling all transmitted values before closure. Histories follow the accepted law, and the independent target follows the raw law. The new steps are to account for the guard and internal-edge openings and to preserve dependence along a route. Once the child probability is known, the outer calculation follows bucket thinning [28,30].

Private openings and shared roots. A required guard opening lies on its private prefix, at position $c_L - e_L$. It can be obtained only from that chain's source or an earlier disclosed point. At a nonfinal layer the prefix is an input to the link hash; disclosing the link leaf does not disclose this private input. A history selecting the leaf supplies a guard opening and also reconstructs its successor root, its link leaf, and the ancestor path. Hence combining such a guard prefix with a shared root adds no private link leaf absent from that history's computation. At the final layer, an authentication value can reveal the leaf at position $c_L + 1$, but cannot reveal a required guard opening at position at most c_L.

Within an owner, a history selecting a descendant already reconstructs all raw ancestors on its path. If no history selects below a private subtree, external authentication can disclose that subtree's root or a later chain point, not its hidden predecessor guards. Induction from leaves to root therefore reduces the availability of the target's private requirements to the union of individual disclosures and their forward chain successors. This argument is specific to the private guards; it is not an assumption that union and closure commute for arbitrary DAGs.

Local covering probability. We follow the missing-requirement inclusion–exclusion approach used for chain-based few-time signatures [30]. Here requirements include internal authentication edges, so their avoidance counts depend on where a historical path first diverges from the target path. For a target codeword $e \in \mathcal{L}_{k,\boldsymbol{c},E}$, let $S \subseteq [k]$ be a set of requirements; the guard is coordinate L. Define

$$A_e(S) = \big|\{f \in \mathcal{L}_{k,\boldsymbol{c},E} : f_j < e_j \text{ for every } j \in S\}\big|, \tag{21}$$

$$D_e(S) = \sum_{\substack{0 \le j < L \\ j \notin S}} 2^{L-j-1} A_e(S \cap \{0, \dots, j-1\}). \tag{22}$$

A same-leaf history avoids S in $A_e(S)$ ways. A history first diverging at depth j supplies sibling requirement j, may supply higher requirements, and does not supply a lower requirement or the selected guard. The number of leaves with that first divergence is 2^{L-j-1}, giving (22). If m histories select the current owner and v select the exact target slot, the local covering probability is $f(m, 0) = 0$, and for $m \ge v \ge 1$,

$$f(m, v) = \frac{1}{N} \sum_{e \in \mathcal{L}_{k,\boldsymbol{c},E}} \sum_{S \subseteq [k]} (-1)^{|S|} \left(\frac{A_e(S)}{N}\right)^v \left(\frac{D_e(S)}{(t-1)N}\right)^{m-v}. \tag{23}$$

The two powers count independent same-slot and different-slot histories that avoid every requirement in S. Inclusion–exclusion and averaging over the target codeword give (23). The antichain property gives $f(1, 1) = 1/N$.

To evaluate this sum, group pairs (e, S) with the same avoidance counts. Let $h_{a,d}$ be the sum of $(-1)^{|S|}$ over all pairs satisfying $A_e(S) = a$ and $D_e(S) = d$. Then

$$f(m, v) = \frac{\sum_{a,d} h_{a,d}\, a^v d^{m-v}}{N^{m+1}(t-1)^{m-v}}, \tag{24}$$

where the sum runs over the attained count pairs (a, d). For a fixed codebook, the coefficients $h_{a,d}$ can be precomputed and reused for all (m, v). This gives an exact evaluation procedure in principle, with the numerator computed in integer arithmetic before division.

Coupling consecutive layers. Fix q histories in the selected child. If j visit the current target owner, let v choose the target slot. These v histories also enter the next target owner; among the remaining $q - j$ histories, each enters it with probability $1/t$. Conditional on the current count j, let $M_q(j, j')$ be the joint probability that the current layer's requirements are covered and that j' histories enter the next target owner. Thus

$$M_q(j, j') = \sum_{v=1}^{j} \Pr[\mathrm{Bin}(j, 1/t) = v]\, f(j, v)\, \Pr[\mathrm{Bin}(q - j, 1/t) = j' - v]. \tag{25}$$

Here $j, j' \in \{1, \ldots, q\}$ and infeasible binomial probabilities are zero. Let $b_q(j) = \Pr[\mathrm{Bin}(q, 1/O) = j]$.

Lemma 1 (Child covering probability). *For $q \geq 1$ independent uniform histories in one GS-C child and an independent uniform target in $[O] \times ([t] \times \mathcal{L}_{k,\boldsymbol{c},E})^r$, the forward-DAG covering probability is*

$$C(q) = b_q^{\mathsf{T}} M_q^r \mathbf{1}, \qquad C(0) = 0. \tag{26}$$

Proof. The initial-owner count has distribution b_q. Conditional on the current count j and exact-slot count v, the local requirements are covered with probability $f(j, v)$, and the next count is $v + \mathrm{Bin}(q - j, 1/t)$. Independent slot choices and depth vectors in later layers give the same conditional transition at every step. Multiplying r transitions therefore accounts for all layers. The final $\mathbf{1}$ sums an auxiliary outgoing count; the last layer requires no successor root. With no history, the target guard is unavailable, giving $C(0) = 0$. □

For plain GS, $\boldsymbol{c} = \mathbf{0}$ and $E = 0$ give $N = 1$ and $f(m, v) = 1$ for $v \geq 1$. Thus the same recurrence covers the basic construction; in particular $C(1) = 1/(Ot^r)$, since one history covers only its own route.

Instantiating bucket thinning. The standard bucket-thinning calculation [28,30] averages a child covering term over the number of histories reaching the target bucket. Here the child term is $C(j)$ from Lemma 1, giving

$$D_B(q) = \sum_{j=0}^{q} \binom{q}{j} B^{-j} (1 - B^{-1})^{q-j} C(j). \tag{27}$$

Corollary 1 (Bucket thinning of GS-C). *For BGS-CC with the sampler of Section 4.4, independent accepted histories, and an independent raw target in the forward-DAG experiment,*

$$\rho = \tau \sum_{q=0}^{Q} \binom{Q}{q} 2^{-hq} (1 - 2^{-h})^{Q-q} D_B(q)^K. \tag{28}$$

Proof. Conditioned on acceptance, the sampler is uniform on the product space in (12). At a fixed bottom-key count q, each coordinate therefore has $\mathrm{Bin}(q, 1/B)$ matching-bucket histories, and its coverage probability is $D_B(q)$. Coordinate independence gives $D_B(q)^K$. The raw target passes both rejection-sampling tests with probability τ; conditioned on passing, it has the same uniform product law. Multiplying by τ and averaging over $q \sim \mathrm{Bin}(Q, 2^{-h})$ proves the claim. □

The power is taken before averaging over the common history count, and target acceptance is counted once. For $K = 1$, binomial thinning simplifies this to

$$\rho = \tau\, \mathbb{E}_{N_b \sim \mathrm{Bin}(Q, 2^{-h}/B)}[C(N_b)].$$

The computation uses this simplification only for $K = 1$.

6 Evaluation

We evaluate the signature-size reductions of BGS-CC and the signing and verification work they require. We first compare few-time components at comparable signing costs, then compare complete SPHINCS-based signatures under nine standard and limited-use workloads while keeping WOTS+ and the hypertree fixed. Size is measured in bytes and work in hash calls, following Section 2.2.

For the component comparison in Table 1, we use $n = 16$, $Q = 2^{64}$, $h = 64$, and a coverage target of 2^{-128}. Expected signing costs lie between $4 \cdot 10^7$ and $6 \cdot 10^7$ calls, with at most 262 verification calls. BGS-CC uses 788 bytes with 54.1 million signing calls and 216 verification calls, compared with 1,108 bytes for BPORS-C+BFP. PORS has variable size, and its entry gives the maximum.

The previous-work rows give the smallest signatures found within this signing band among the source-supported parameter families. They retain their source encoder interfaces, charging one logical hash call per invocation, including variable-output encoders. BGS-CC includes the bounded-output charges from Section 5.2.

For complete signatures, Table 4 lists the fixed WOTS+ and hypertree parameters. The six standard workloads have $Q = 2^{64}$; the three limited-use workloads have $Q = 2^{24}$ and follow the draft profiles considered in prior work [23,30]. Their mean bottom-key reuse counts are $2, 2, 1$ for the small-signature standard rows, $1/4, 1/4, 1/16$ for the fast rows, and $4, 8, 8$ for the limited-use rows. Every comparison uses the full finite-binomial use-count distribution from Section 2.3.

Table 5 gives the BGS-CC construction and encoder parameters. Component signing is capped at 1.05 times FORS (1.2 for 128f), and component verification at 1.75 times FORS. All thresholds use at most three set bits, and every row uses a four-byte counter. For limited use, $\tau \geq 10^{-8}$ keeps the expected rejection search at most 10^8 trials.

Table 4. Fixed surrounding parameters and FORS tree counts and heights (k_F, a_F) for the application points; w is the Winternitz parameter. Standard and limited-use reference profiles follow [25,23,30].

Setting	n	Q	h	D	w	Wrapper bytes	(k_F, a_F)
128s	16	2^{64}	63	7	16	4 944	$(14, 12)$
192s	24	2^{64}	63	7	16	10 104	$(17, 14)$
256s	32	2^{64}	64	8	16	19 232	$(22, 14)$
128f	16	2^{64}	66	22	16	13 392	$(33, 6)$
192f	24	2^{64}	66	22	16	28 536	$(33, 8)$
256f	32	2^{64}	68	17	16	38 656	$(35, 9)$
128-24	16	2^{24}	22	1	4	1 456	$(6, 24)$
192-24	24	2^{24}	21	1	8	2 136	$(9, 25)$
256-24	32	2^{24}	21	1	4	4 960	$(12, 25)$

Table 5. BGS-CC application parameters. Every row uses a 32-bit rejection value and a four-byte counter; $B = 2^b$ and $O = 2^u$. The notation $c^{[m]}$ means m consecutive capacities of value c. Total acceptance probability $\tau = p\eta$ is rounded to three significant digits.

Setting	K	L	r	b	u	$\boldsymbol{c}$	E	Δ	τ
128s	1	5	10	0	3	$(3^{[5]}, 6)$	8	$19 \cdot 2^{14}$	$5.83 \cdot 10^{-5}$
192s	1	6	12	0	6	$(3^{[6]}, 6)$	9	$13 \cdot 2^{14}$	$2.63 \cdot 10^{-5}$
256s	1	6	17	0	3	$(1, 3^{[6]})$	8	$273 \cdot 2^{8}$	$1.12 \cdot 10^{-5}$
128f	7	2	5	1	2	$(1, 1, 2)$	2	$7 \cdot 2^{21}$	$3.42 \cdot 10^{-3}$
192f	1	3	34	0	4	$(2, 2, 2, 4)$	4	$69 \cdot 2^{16}$	$9.17 \cdot 10^{-4}$
256f	6	3	6	1	3	$(1, 3, 3, 3)$	4	$41 \cdot 2^{15}$	$2.27 \cdot 10^{-4}$
128-24	1	7	5	5	3	$(4^{[8]})$	12	82	$1.59 \cdot 10^{-8}$
192-24	1	7	8	6	7	$(4^{[8]})$	12	104	$1.57 \cdot 10^{-8}$
256-24	1	7	11	8	9	$(4^{[8]})$	12	208	$2.45 \cdot 10^{-8}$

BGS-CC reduces the complete signature size in all nine settings relative to both FORS and the state-of-the-art BPORS-C+BFP (Table 6). At 128-bit security with $Q = 2^{24}$, the complete signature is 2,228 bytes, compared with 2,596 bytes for BPORS-C+BFP [30] and 2,420 bytes for ML-DSA-44 [24]. To our knowledge, BGS-CC is the first SPHINCS-based stateless signature to cross this size threshold. The table also reports $-\log_2 \bar{\rho}$, where $\bar{\rho}$ is the computed upper bound on the coverage probability. Every row satisfies $\bar{\rho} \le 2^{-128}$, 2^{-192}, or 2^{-256} for its respective security level. The numerical evaluation below describes how these bounds are computed.

Table 7 gives full-signature hash counts for all five schemes, including the fixed WOTS+ and hypertree costs. The 128f and 256f rows use $(K, B) = (7, 2)$ and $(6, 2)$, saving 112 and 1,408 bytes over the published BPORS-C+BFP rows. Relative to those rows, full-signature signing counts increase by 1.17% and 0.03%, and verification counts by 1.49% and 2.50%, respectively. The other seven rows use $K = 1$; the limited-use rows retain bucket thinning with $B = 32, 64, 256$.

Table 6. Complete-signature bytes and BGS-CC coverage bits $-\log_2 \bar{\rho}$, rounded down to three decimals. Prior-construction values are the published source rows [30]; BGS-CC uses the parameters in Table 5.

Setting	FORS	PORS+FP	BPORS+FP	C+BFP	BGS-CC	Cover bits
128s	7 856	7 332	6 804	6 436	5 956	128.032
192s	16 224	15 316	14 212	13 468	12 268	192.110
256s	29 792	28 260	26 564	25 188	23 140	256.025
128f	17 088	16 420	16 452	15 524	15 412	128.027
192f	35 664	34 540	33 892	32 716	31 900	192.042
256f	49 856	48 132	47 076	45 444	44 036	256.013
128-24	3 856	3 316	2 916	2 596	2 228	128.018
192-24	7 752	6 748	5 740	4 925	3 988	192.014
256-24	14 944	13 444	11 652	10 085	8 324	256.063

C+BFP denotes BPORS-C+BFP.

Across the three limited-use settings, Table 7 shows fewer signing calls and more verification calls than BPORS-C+BFP.

Full-signature cost accounting. For D equal-height hypertree layers, Winternitz parameter w, and WOTS+ chain count ℓ_W, the fixed byte contribution, including the message randomizer, is

$$L_{\text{wrap}} = n(1 + D\ell_W + h). \tag{29}$$

The corresponding signing and verification contributions are [30]

$$\begin{aligned} \mathcal{S}_{\text{wrap}} &= D\big[2^{h/D}(\ell_W w + 1) + (2^{h/D} - 1)\big] + 2, \\ \mathcal{V}_{\text{wrap}} &\approx D(\ell_W w/2 + 1 + h/D) + 1. \end{aligned} \tag{30}$$

We use the same $\ell_W w/2$ approximation for WOTS+ verification in every scheme. Full-signature counts add these contributions to the few-time component counts. For BGS-CC, these are $\mathcal{S}_{\text{wrap}} + \mathcal{S}^{\text{ub}}$ and $\mathcal{V}_{\text{wrap}} + \mathcal{V}$ from Section 5.2.

Component parameters. In Table 1, the previous schemes use their source parameter notation. The pairs (t, k) are $(2^{24}, 7)$ for HORST and PORS, and $(2^{21}, 7)$ for FORS. The HORST cut $x = 3$ is the depth of the intermediate authentication level, measured from the root: all 2^x nodes at that level are transmitted, and the selected leaf paths stop there. PORS+FP uses $(t, k, m_{\max}) = (5\,767\,168, 7, 99)$. BPORS+FP uses $(K, b, t, k, m_{\max}) = (2, 10, 1\,792, 6, 77)$, where 2^b is the number of buckets. Here $m_{\max}$ bounds the number of authentication hash values, excluding disclosed secrets and the counter; for BPORS+FP it also includes the outer bucket paths. BPORS-C+BFP uses $(K, b, t, k, c, E) = (3, 12, 41, 5, 48, 48)$ and one block with an authentication budget of 18 nodes.

Table 7. Full-signature hash counts and percentage changes from FORS: signing above verification in each cell. Panels give standard small-signature, standard fast, and limited-use workloads. Prior work follows [30]; Ours uses Table 5 and Section 5.2. Signing counts round upward; verification uses (30). Percentages use unrounded counts.

(a) Standard small-signature workloads

Scheme	128s	192s	256s
FORS	2 186 222 (0%)	3 767 275 (0%)	3 280 869 (0%)
	2 214 (0%)	3 183 (0%)	4 692 (0%)
PORS+FP	2 188 971 (+0.1%)	3 732 432 (−0.9%)	3 264 215 (−0.5%)
	2 195 (−0.9%)	3 162 (−0.7%)	4 666 (−0.6%)
BPORS+FP	2 183 140 (−0.1%)	3 662 422 (−2.8%)	3 225 264 (−1.7%)
	2 168 (−2.1%)	3 129 (−1.7%)	4 626 (−1.4%)
BPORS-C+BFP	2 146 809 (−1.8%)	3 687 458 (−2.1%)	3 243 819 (−1.1%)
	2 186 (−1.3%)	3 157 (−0.8%)	4 669 (−0.5%)
Ours	2 171 383 (−0.7%)	3 701 722 (−1.7%)	3 069 680 (−6.4%)
	2 326 (+5.1%)	3 343 (+5.0%)	4 893 (+4.3%)

(b) Standard fast workloads

Scheme	128f	192f	256f
FORS	105 196 (0%)	169 260 (0%)	345 839 (0%)
	6 481 (0%)	9 363 (0%)	9 549 (0%)
PORS+FP	105 282 (+0.1%)	169 406 (+0.1%)	346 261 (+0.1%)
	6 472 (−0.1%)	9 349 (−0.1%)	9 530 (−0.2%)
BPORS+FP	105 257 (+0.1%)	169 123 (−0.1%)	345 628 (−0.1%)
	6 481 (0%)	9 334 (−0.3%)	9 507 (−0.4%)
BPORS-C+BFP	105 015 (−0.2%)	169 250 (0%)	345 467 (−0.1%)
	6 428 (−0.8%)	9 325 (−0.4%)	9 531 (−0.2%)
Ours	106 242 (+1.0%)	168 295 (−0.6%)	345 569 (−0.1%)
	6 524 (+0.7%)	9 547 (+2.0%)	9 769 (+2.3%)

(c) Limited-use workloads

Scheme	128-24	192-24	256-24
FORS	1 451 229 180 (0%)	2 034 237 433 (0%)	2 327 838 710 (0%)
	311 (0%)	526 (0%)	602 (0%)
PORS+FP	1 457 500 226 (+0.4%)	2 048 864 458 (+0.7%)	2 336 980 369 (+0.4%)
	283 (−9.0%)	493 (−6.3%)	567 (−5.8%)
BPORS+FP	1 230 807 030 (−15.2%)	1 639 311 819 (−19.4%)	1 452 127 379 (−37.6%)
	266 (−14.5%)	458 (−12.9%)	523 (−13.1%)
BPORS-C+BFP	1 378 187 117 (−5.0%)	1 861 952 200 (−8.5%)	2 237 453 399 (−3.9%)
	309 (−0.6%)	523 (−0.6%)	601 (−0.2%)
Ours	1 243 947 910 (−14.3%)	1 317 113 176 (−35.3%)	2 037 457 146 (−12.5%)
	410 (+31.8%)	690 (+31.2%)	836 (+38.9%)

All three pruning-based rows use four-byte counters. Their coverage bounds are averaged over the finite-binomial history law. For BPORS-C+BFP, the parameters use the general coverage upper bound [30], evaluated through exact finite-binomial factorial moments.

The BGS-CC row in Table 1 uses $K = 1$, $B = 16$, $O = 32$, $L = 7$, $r = 5$, $c_j = 3$ for every j, and $E = 12$. Its encoder uses two hash outputs and a 32-bit rejection-sampling value with threshold $112 = 2^6+2^5+2^4$. The four-byte counter and 49 hash values give 788 bytes; verification takes 216 calls.

Numerical evaluation. We evaluate the numerator of (24) with integers before division. The evaluator computes $f(m, v)$ exactly for $m \le 24$ ($m \le 28$ for 256f) and grants all local coverage for larger counts with $v \ge 1$, while retaining $f(m, 0) = 0$. The subsequent positive probabilities and matrix operations use 768-bit upward rounding. We retain counts through $q_{\max} = 120$, 110 for 128f and 256f, or 100 for the limited-use rows, and bound the remaining tail by the factorial moment

$$\Pr[N_\theta > q_{\max}] \le \binom{Q}{q_{\max}+1}\theta^{q_{\max}+1} \quad \text{for } N_\theta \sim \mathrm{Bin}(Q, \theta). \tag{31}$$

For $K > 1$, the power across coordinates is taken before the common outer binomial average. A paired lower-bound evaluation bounds the relative numerical slack by 2^{-268} for Table 1 and by 2^{-91} for every application row. Thus the reported bounds closely enclose the exact coverage probabilities.

We check counter exhaustion separately using (20). For the BGS-CC row in Table 1 and all nine application rows, its probability over the respective signing quota is at most 2^{-73} with four-byte counters throughout.

7 Discussion and Conclusion

The new mechanism is guarded sharing: a selected private guard binds each use of a shared successor root. This permits successor subgraphs to be reused while requiring a separate private opening for the link selected by the message. Verification follows the route backward, and the covering analysis retains the dependence between histories supplying a current guard and those entering the next owner.

Combining guarded sharing with bucket thinning, constant-sum chain encoding, and rejection sampling gives BGS-CC. It reduces complete-signature size in all nine evaluated settings relative to both FORS and the state-of-the-art BPORS-C+BFP, with the signing and verification tradeoffs in Table 7. At 128-bit security and a quota of $Q = 2^{24}$, the complete signature is 2,228 bytes, down from 2,596 bytes for BPORS-C+BFP [30]. This brings SPHINCS-based stateless signatures below the 2,420-byte size of ML-DSA-44 [24].

References

1. Abri, M., Katz, J.: Shorter hash-based signatures using forced pruning. In: Advances in Cryptology – CRYPTO 2026. Lecture Notes in Computer Science, vol. 16803, pp. 573–602. Springer (2026). https://doi.org/10.1007/978-3-032-35398-6_19, https://eprint.iacr.org/2025/2069, full version: Cryptology ePrint Archive, Report 2025/2069
2. Aumasson, J.P., Endignoux, G.: Improving stateless hash-based signatures. In: Topics in Cryptology – CT-RSA 2018. Lecture Notes in Computer Science, vol. 10808, pp. 219–242. Springer (2018). https://doi.org/10.1007/978-3-319-76953-0_12, https://eprint.iacr.org/2017/933, full version: Cryptology ePrint Archive, Report 2017/933
3. Barbosa, M., Dupressoir, F., Grégoire, B., Hülsing, A., Meijers, M., Strub, P.Y.: Machine-checked security for XMSS as in RFC 8391 and SPHINCS+. In: Advances in Cryptology – CRYPTO 2023. Lecture Notes in Computer Science, vol. 14085, pp. 421–454. Springer (2023). https://doi.org/10.1007/978-3-031-38554-4_14
4. Barbosa, M., Dupressoir, F., Hülsing, A., Meijers, M., Strub, P.Y.: A tight security proof for SPHINCS+, formally verified. In: Advances in Cryptology – ASIACRYPT 2024. Lecture Notes in Computer Science, vol. 15487, pp. 35–67. Springer (2024). https://doi.org/10.1007/978-981-96-0894-2_2
5. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: Practical stateless hash-based signatures. In: Advances in Cryptology – EUROCRYPT 2015. Lecture Notes in Computer Science, vol. 9056, pp. 368–397. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_15
6. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS+ signature framework. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2129–2146. ACM (2019). https://doi.org/10.1145/3319535.3363229, https://eprint.iacr.org/2019/1086, full version: Cryptology ePrint Archive, Report 2019/1086
7. Bleichenbacher, D., Maurer, U.: Directed acyclic graphs, one-way functions and digital signatures. In: Advances in Cryptology – CRYPTO '94. Lecture Notes in Computer Science, vol. 839, pp. 75–82. Springer (1994). https://doi.org/10.1007/3-540-48658-5_9
8. Bleichenbacher, D., Maurer, U.: On the efficiency of one-time digital signatures. In: Advances in Cryptology – ASIACRYPT '96. Lecture Notes in Computer Science, vol. 1163, pp. 145–158. Springer (1996). https://doi.org/10.1007/BFb0034843
9. Bleichenbacher, D., Maurer, U.: Optimal tree-based one-time digital signature schemes. In: STACS 96. Lecture Notes in Computer Science, vol. 1046, pp. 361–374. Springer (1996). https://doi.org/10.1007/3-540-60922-9_30
10. Cooper, D.A., Apon, D.C., Dang, Q.H., Davidson, M.S., Dworkin, M.J., Miller, C.A.: Recommendation for stateful hash-based signature schemes. Special Publication 800-208, National Institute of Standards and Technology (2020). https://doi.org/10.6028/NIST.SP.800-208
11. Fluhrer, S., Dang, Q.: Smaller SPHINCS+. Cryptology ePrint Archive, Report 2024/018 (2024), https://eprint.iacr.org/2024/018
12. Hevia, A., Micciancio, D.: The provable security of graph-based one-time signatures and extensions to algebraic signature schemes. In: Advances in Cryptology – ASIACRYPT 2002. Lecture Notes in Computer Science, vol. 2501, pp. 379–396. Springer (2002). https://doi.org/10.1007/3-540-36178-2_24

13. Hülsing, A.: W-OTS+: Shorter signatures for hash-based signature schemes. In: Progress in Cryptology – AFRICACRYPT 2013. Lecture Notes in Computer Science, vol. 7918, pp. 173–188. Springer (2013). https://doi.org/10.1007/978-3-642-38553-7_10
14. Hülsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: eXtended Merkle signature scheme. RFC 8391 (2018). https://doi.org/10.17487/RFC8391
15. Hülsing, A., Kudinov, M.: Recovering the tight security proof of SPHINCS+. In: Advances in Cryptology – ASIACRYPT 2022. Lecture Notes in Computer Science, vol. 13794, pp. 3–33. Springer (2022). https://doi.org/10.1007/978-3-031-22972-5_1
16. Hülsing, A., Kudinov, M.A., Ronen, E., Yogev, E.: SPHINCS+C: Compressing SPHINCS+ with (almost) no cost. In: IEEE Symposium on Security and Privacy. pp. 1435–1453. IEEE (2023). https://doi.org/10.1109/SP46215.2023.10179381, https://eprint.iacr.org/2022/778, full version: Cryptology ePrint Archive, Report 2022/778
17. Hülsing, A., Rijneveld, J., Song, F.: Mitigating multi-target attacks in hash-based signatures. In: Public-Key Cryptography – PKC 2016. Lecture Notes in Computer Science, vol. 9614, pp. 387–416. Springer (2016). https://doi.org/10.1007/978-3-662-49384-7_15, https://eprint.iacr.org/2015/1256, full version: Cryptology ePrint Archive, Report 2015/1256
18. Khovratovich, D., Kudinov, M., Wagner, B.: At the top of the hypercube—better size–time tradeoffs for hash-based signatures. In: Advances in Cryptology – CRYPTO 2025. Lecture Notes in Computer Science, vol. 16005, pp. 93–123. Springer (2025). https://doi.org/10.1007/978-3-032-01887-8_4, https://eprint.iacr.org/2025/889, full version: Cryptology ePrint Archive, Report 2025/889
19. Lamport, L.: Constructing digital signatures from a one way function. Tech. Rep. CSL-98, SRI International (1979)
20. McGrew, D., Curcio, M., Fluhrer, S.: Leighton–Micali hash-based signatures. RFC 8554 (2019). https://doi.org/10.17487/RFC8554
21. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Advances in Cryptology – CRYPTO '87. Lecture Notes in Computer Science, vol. 293, pp. 369–378. Springer (1988). https://doi.org/10.1007/3-540-48184-2_32
22. Merkle, R.C.: A certified digital signature. In: Advances in Cryptology – CRYPTO '89. Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer (1990). https://doi.org/10.1007/0-387-34805-0_21
23. Moody, D., Dang, Q.: Additional SLH-DSA parameter sets for limited-signature use cases. NIST Special Publication 800-230 ipd, National Institute of Standards and Technology (April 2026). https://doi.org/10.6028/NIST.SP.800-230.ipd, Initial Public Draft
24. National Institute of Standards and Technology: Module-lattice-based digital signature standard. Federal Information Processing Standards Publication 204, National Institute of Standards and Technology (2024). https://doi.org/10.6028/NIST.FIPS.204
25. National Institute of Standards and Technology: Stateless hash-based digital signature standard. Federal Information Processing Standards Publication 205, National Institute of Standards and Technology (2024). https://doi.org/10.6028/NIST.FIPS.205

26. Perrig, A.: The BiBa one-time signature and broadcast authentication protocol. In: Proceedings of the 8th ACM Conference on Computer and Communications Security. pp. 28–37. ACM (2001). https://doi.org/10.1145/501983.501988
27. Qin, Z., Sun, S., Zheng, F.: Extending the SPHINCS+ framework: Varying the tree heights and chain lengths. Cryptology ePrint Archive, Report 2025/2236 (2025), https://eprint.iacr.org/2025/2236, revised version of the ASIACRYPT 2026 paper
28. Ren, C., Zhang, K., Yu, H.: Shorter hash-based signatures via bucket thinning. Cryptology ePrint Archive, Report 2026/1328 (2026), https://eprint.iacr.org/2026/1328
29. Reyzin, L., Reyzin, N.: Better than BiBa: Short one-time signatures with fast signing and verifying. In: Information Security and Privacy – ACISP 2002. Lecture Notes in Computer Science, vol. 2384, pp. 144–153. Springer (2002). https://doi.org/10.1007/3-540-45450-0_11, full version: Cryptology ePrint Archive, Report 2002/014
30. Wang, L., Liu, Q., Cui, H., Hu, Y., Yu, Y.: Shorter few-time signatures from hash chains and blockwise forced pruning. Cryptology ePrint Archive, Report 2026/2374 (2026), https://eprint.iacr.org/2026/2374
31. Yehia, M., AlTawy, R., Gulliver, T.A.: Verifiable obtained random subsets for improving SPHINCS+. In: Information Security and Privacy – ACISP 2021. pp. 694–714. Springer (2021). https://doi.org/10.1007/978-3-030-90567-5_35, https://eprint.iacr.org/2021/1495
32. Zhang, K., Cui, H., Yu, Y.: SPHINCS-α: A compact stateless hash-based signature scheme. Cryptology ePrint Archive, Report 2022/059 (2022), https://eprint.iacr.org/2022/059, version dated 8 June 2023; subsumed by [33]
33. Zhang, K., Cui, H., Yu, Y.: Revisiting the constant-sum Winternitz one-time signature with applications to SPHINCS+ and XMSS. In: Advances in Cryptology – CRYPTO 2023. Lecture Notes in Computer Science, vol. 14085, pp. 455–483. Springer (2023). https://doi.org/10.1007/978-3-031-38554-4_15, https://eprint.iacr.org/2023/850, full version: Cryptology ePrint Archive, Report 2023/850