

Shorter Few-Time Signatures from Hash Chains and Blockwise Forced Pruning

Lizheng Wang, Qi Liu, Hongrui Cui, Yuncong Hu, and Yu Yu

Shanghai Jiao Tong University, Shanghai, China

Abstract. SLH-DSA is a standardized stateless hash-based signature scheme based on SPHINCS⁺, but its signatures remain large. PORS+FP uses forced pruning to reduce the size of the bottom few-time signature (FTS) component, while BPORS+FP distributes repeated uses among independent PORS child keys. In BPORS+FP, each selected leaf reveals a fixed secret value, and a single pruning budget limits the total number of authentication nodes across all coordinates. We introduce BPORS-C+BFP, which replaces each secret at a child leaf with a short hash chain and assigns a separate authentication budget to each block of coordinates. The child Merkle tree authenticates each chain endpoint. Each selected chain still contributes one hash value to the signature, while its opening depth provides an additional encoding choice. For a fixed set of selected chains, a constant-sum constraint prevents forward hashing alone from converting one valid depth tuple into another.

Under repeated use, signatures may disclose different positions on the same chain. We derive a generating-function upper bound on coverage and an exact counting method for individual pruning blocks that accounts for these disclosures and target acceptance. Separate block budgets confine pruning-induced dependencies, reducing the number of coordinates counted jointly and making exact calculation practical for small blocks. Replacing FORS in SPHINCS⁺ while keeping the hypertree and one-time signature parameters fixed, BPORS-C+BFP reduces complete-signature sizes by 8.3%–18.1% across six standard settings, compared with 3.7%–13.4% for BPORS+FP. Across three limited-use settings, the reductions relative to FORS are 32.5%–36.5% for BPORS-C+BFP and 22.0%–26.0% for BPORS+FP.

Keywords: Hash-based signatures · SLH-DSA · SPHINCS · Post-quantum cryptography

1 Introduction

Hash-based signatures are among the standardized approaches to post-quantum digital signatures. Their appeal stems from their conceptual simplicity and minimal theoretical assumptions: in principle, they can be constructed from one-way functions alone [17,21]. Practical schemes use fast hash functions and pseudorandom functions to obtain efficient implementations [5]. When these primitives satisfy the required security properties against quantum adversaries,

the resulting constructions can provide post-quantum security [5,18]. This reliance on hash functions and pseudorandom functions makes hash-based signatures a conservative option for post-quantum digital signatures. Hash-based signatures also admit a wide range of parameter choices, allowing designers to trade signature size against signing and verification costs [11].

Early hash-based constructions, including Lamport and Winternitz-type signatures, were designed for one-time use [12,15]. Merkle-tree authentication extends these constructions to many-time signatures by authenticating many one-time public keys under a single root, as in XMSS and LMS [6,13]. In these stateful schemes, the signer must track previously used keys to prevent accidental reuse, which can compromise security. Maintaining this state reliably across backup restoration, system failures, and concurrent signing operations complicates deployment [7]. These difficulties motivate stateless constructions, which avoid the need to track key usage. SPHINCS demonstrated a practical stateless hash-based signature scheme [4], and its successor SPHINCS+ [5] formed the basis of SLH-DSA, standardized by NIST in 2024 [18].

A major practical limitation of SLH-DSA is its large signature size. For example, the 128s and 128f parameter sets both use a 128-bit security parameter and target NIST security category 1. Designed for smaller signatures (“s”) and faster signing (“f”), respectively, they produce signatures of 7,856 and 17,088 bytes [18]. This overhead is particularly relevant to applications that process many signatures. In blockchains, for example, including many large signatures increases block size and the bandwidth required for propagation [8]. Recent work has therefore explored alternative parameter choices, improved encodings, and compression techniques to reduce signature size while controlling signing and verification costs [11,10,1,19].

The SPHINCS+ framework combines three main components: a few-time signature (FTS) scheme, a one-time signature (OTS) scheme, and a Merkle hypertree [5,18]. The FTS, instantiated by FORS, signs the message digest. A Winternitz-type OTS signs the FTS public key, and the hypertree authenticates the corresponding OTS public key under the global public key. Each component contributes to the total signature size and offers opportunities for optimization. The hypertree and OTS layers are built from Merkle trees and Winternitz hash chains, respectively. Previous work has explored alternative tree parameters and more efficient chain encodings to improve the size–time tradeoffs of these two layers [5,22,10].

The FTS component offers further opportunities for structural optimization. FORS organizes secret values into separate Merkle trees and selects one leaf from each tree [5]. PORS instead selects multiple leaves from a single tree and uses Octopus to remove redundant nodes from their authentication paths [5,2]. The resulting signature size depends on the selected leaves. Abri and Katz apply forced pruning to PORS: the signer varies a counter until the number of authentication nodes for the selected leaves does not exceed a prescribed budget. With padding, this yields smaller fixed-length signatures at the cost of additional search [1]. Ren, Zhang, and Yu introduce BPORS+FP, which groups independent PORS

child keys into several coordinates [19]. Within each coordinate, an outer Merkle tree authenticates the roots of the child PORS trees. Each signature selects one bucket per coordinate and opens the corresponding PORS instance, together with the child authentication data and an outer authentication path. A global forced-pruning budget limits the total authentication data across all coordinates. By distributing repeated disclosures among the children, bucket thinning reduces the expected number of uses per child and can permit smaller child parameters, provided that the savings compensate for the additional outer authentication paths.

1.1 Our Contributions

We introduce BPORS-C+BFP, which extends BPORS+FP [19] with hash-chain openings and blockwise forced pruning (BFP). Within each child key, a short hash chain replaces each leaf secret, and its endpoint is authenticated by the child Merkle tree, as illustrated in Figure 1. Following the constant-sum encoding approach of [22], we sample the opening depths uniformly subject to a fixed-sum constraint. These depth choices enlarge the encoding space while each selected chain still contributes only one hash value to the signature.

An adversary may combine openings from multiple signatures to obtain all openings required by a fresh target encoding. Repeated selections of the same chain can disclose values at different positions, and an earlier disclosed value allows all later values to be computed by forward hashing. We characterize this accumulated information by the largest opening depth disclosed on each chain. Our coverage analysis therefore accounts for both the selected chain indices and the depths available from the entire history.

Forced pruning changes the distribution of the leaf sets selected in previous signatures, since only candidate encodings that meet the authentication budget are used. When several coordinates share a budget, their leaf selections can become dependent after pruning. BFP confines this dependence by partitioning the coordinates into blocks and assigning a separate authentication budget to each block. Global forced pruning is recovered when all coordinates form a single block.

We develop an exact counting method on the child Merkle trees to compute the probability that a fresh target encoding both passes pruning and is covered by openings from previous signatures. The method accounts for repeated chain selections, opening depths, the distribution of encodings used in previous signatures, and the target’s authentication budget. We perform this calculation separately for each block. Smaller blocks therefore reduce the number of coordinates that must be counted jointly. Given q previous signatures under the FTS key, the block events are independent, so the overall probability is the product of the block probabilities. We also derive a generating-function upper bound that applies to both global and blockwise pruning. It incorporates the opening-depth constraints but treats repeated selections of the same chain as distinct virtual copies, avoiding explicit tracking of overlap among chain indices.

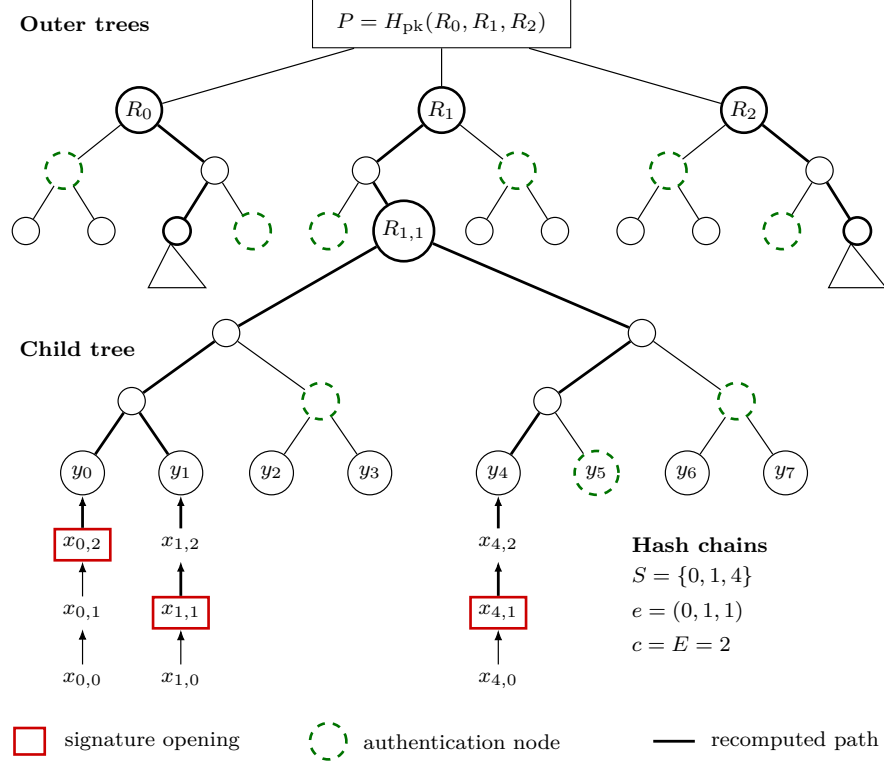


Fig. 1. The BPORS-C+BFP structure for $K = 3$ and $B = 2^b = 4$, with selected buckets $(j_0, j_1, j_2) = (2, 1, 3)$. The middle selected child is expanded into a PORS-style Merkle tree with hash chains below its leaves; its root is also an outer-tree leaf. Here $t = 8$, $k = 3$, $c = E = 2$, and $y_u = x_{u,3}$. Red boxes and green dashed circles mark the chain values and authentication nodes included in the signature, respectively. Bold edges mark paths recomputed during verification. Triangles abbreviate the other selected children; unselected child trees and chains are omitted.

We evaluate BPORS-C+BFP as a replacement for FORS in the SPHINCS+ framework, keeping the WOTS+ and hypertree parameters fixed. We consider six standard settings [18] and three limited-use settings [16] with a limit of 2^{24} signatures per key. In every setting, our selected configurations give smaller complete signatures than the corresponding FORS, PORS+FP, and BPORS+FP reference configurations [5,1,19]. Relative to FORS, complete-signature sizes decrease by 8.3%–18.1% across the standard settings and by 32.5%–36.5% across the limited-use settings. With costs measured in hash and PRF evaluations, our upper bounds on expected signing cost and our verification counts are also below those of the corresponding FORS-based SPHINCS+ configurations. Detailed parameters and comparisons are given in Section 5.

1.2 Overview of the Paper

Section 2 reviews Merkle authentication, the SPHINCS+ framework, and constant-sum encodings. Section 3 reviews previous FTS constructions, presents BPORS-C+BFP, and analyzes its costs. Section 4 defines the coverage experiment and develops the depth-sensitive analysis under repeated use. Section 5 describes parameter selection and evaluates the resulting configurations.

2 Preliminaries

2.1 Notation

For a positive integer t , write $[t] = \{0, \dots, t-1\}$ and let $\binom{[t]}{k}$ denote the family of k -element subsets of $[t]$. We write $x \leftarrow \mathcal{D}$ for sampling from a distribution $\mathcal{D}$, and we use the same notation for uniform sampling from a finite set. Deterministic assignment is denoted by $:=$, and $x \parallel y$ denotes concatenation. All logarithms are base 2. For a polynomial P , $[x^a z^d]P$ denotes the coefficient of $x^a z^d$ and $[z^{\leq L}]P := \sum_{m=0}^L [z^m]P$. We use uppercase letters for random variables and the corresponding lowercase letters for their values.

We use H_0 for hash-chain steps, H_1 for Merkle-tree internal nodes, H_2 for message encoding, and H_{pk} for public-key compression. For integration into the SPHINCS+ framework, we additionally use H_{addr} to generate a bottom FTS address jointly with its encoding. We model these functions as independent domain-separated random oracles. Let n denote the byte length of chain values, Merkle-tree nodes, and compressed public keys. We define the message-encoding function $H_2 : \{0, 1\}^* \rightarrow \mathcal{U}$, where $\mathcal{U}$ is the corresponding finite encoding space. For a hypertree of total height h , we define $H_{\text{addr}} : \{0, 1\}^* \rightarrow [2^h] \times \mathcal{U}$.

We use a pseudorandom function (PRF), denoted by $F_{K_{\text{sk}}}$, to derive the secret starting values of the hash chains from a single secret key K_{sk} . Hash inputs include the relevant key and chain addresses, which we leave implicit in the notation. Throughout the paper, Q denotes the number of signatures under a complete SPHINCS+-style key, q denotes the number under one bottom FTS key, and r denotes the number selecting a particular child key within that FTS instance.

2.2 Merkle trees and Octopus

We assume the reader is familiar with Merkle trees [14,15] and introduce only the notation needed below. We also recall the Octopus algorithm [2], which removes redundant nodes from the authentication paths of multiple leaves.

Following the convention of previous works [1,19], we allow an arbitrary number $t \geq 2$ of leaves. Let $h = \lceil \log t \rceil$ and $u = t - 2^{h-1}$. Start with a complete binary tree of depth $h-1$ and expand its leftmost u leaves into pairs of children. The resulting full binary tree has exactly t leaves: the first $2u$ have depth h , and the others have depth $h-1$.

We identify each node by its binary path from the root. The root has position ϵ (the empty string), and an internal node at position p has left and right children at positions $p0$ and $p1$. A node at position p has depth $|p|$. Leaves are indexed from 0 to $t-1$ from left to right; we write $\text{leaf}(i)$ for the binary position of leaf i . A leaf lies in the subtree rooted at p exactly when its position has prefix p .

Given leaf values $y_0, \dots, y_{t-1} \in \{0, 1\}^{8n}$, set $v_{\text{leaf}(i)} = y_i$ for every $i \in [t]$. For a fixed tree, compute at each internal position p

$$v_p = H_1(p, v_{p0}, v_{p1}).$$

Applying this rule from the leaves to the root yields v_ϵ . We denote this computation by

$$\text{Merkle}_{H_1}(y_0, \dots, y_{t-1}).$$

To authenticate one leaf, a Merkle path supplies the sibling values needed to recompute the root.

When several leaves are opened, their authentication paths may contain repeated nodes. Octopus removes this redundancy [2]. For a nonempty support $S \subseteq [t]$, take the union of the paths from positions $\{\text{leaf}(i) : i \in S\}$ to the root. The authentication nodes are the roots of the maximal unselected subtrees adjacent to this union. Denote their positions by $\text{Oct}(S)$ and their number by

$$m(S) := |\text{Oct}(S)|.$$

Algorithm 6 in Appendix A.2 computes these positions in a fixed order. Both signer and verifier can derive them from S , so the signature includes the authentication-node values without their positions.

If $|S| = k$, the opening consists of k leaf values and $m(S)$ authentication values. It occupies $n(k + m(S))$ bytes and requires $k + m(S) - 1$ internal-node hashes to recover the root. The number of authentication nodes depends on the geometry of S : in a complete tree with four leaves, $S = \{0, 1\}$ needs one authentication value, whereas $S = \{0, 2\}$ needs two.

For the t -leaf tree rooted at ϵ , define

$$A_\epsilon(k, m) := \left| \left\{ S \in \binom{[t]}{k} : m(S) = m \right\} \right|.$$

For a uniformly sampled $S \in \binom{[t]}{k}$, we have $\Pr[m(S) = m] = A_\epsilon(k, m) / \binom{t}{k}$. Abri and Katz compute these counts exactly using dynamic programming on binary trees [1].

2.3 The SPHINCS+ framework

We recall only the aspects of the SPHINCS+ framework relevant to our replacement of its bottom FTS component [5, 18]. SPHINCS+ and SLH-DSA use FORS as their bottom FTS component [5, 18]. The FTS public key is authenticated under the global public key through the WOTS+ and hypertree layers. In our comparisons, we replace only FORS while keeping these surrounding layers fixed.

To sign a message m , the standard SPHINCS+ signing algorithm derives a randomizer R_{msg} and hashes it together with the global public key and m . For a hypertree of total height h , the hash output is parsed into a message digest M' and an address $i \in [2^h]$ selecting a bottom FORS instance [5,18]. FORS parses M' into the leaf indices used for signing. The complete signature contains R_{msg} , the bottom FTS signature, and the WOTS+ signatures and Merkle paths needed to authenticate the FTS public key.

2.4 Hash chains and constant-sum encodings

A hash chain of length ℓ consists of values $x_0, \dots, x_\ell \in \{0, 1\}^{8n}$ satisfying

$$x_{v+1} = H_0(v, x_v), \quad 0 \leq v < \ell,$$

where the edge index v separates the hash inputs at different steps. Given x_v , all later chain values can be computed by forward hashing. Under the random-oracle model, recovering an earlier chain value from x_v alone is computationally infeasible. With x_ℓ as the public endpoint, an opening x_v is verified by applying the remaining $\ell - v$ hashes and checking that the result equals x_ℓ .

Winternitz-type one-time signatures encode a message through the opening positions of multiple hash chains. For example, WOTS+ represents its input message as base- w digits and appends a checksum [5,18]. Each message or checksum digit specifies an opening position on a separate chain. The signature contains one value from each chain, and verification hashes these values forward to reconstruct the endpoints and the corresponding public key. The checksum ensures that a different valid encoding cannot be obtained solely by advancing the disclosed chain values.

Constant-sum encodings impose the same restriction by using tuples whose entries sum to a fixed value [22]. For integers $k \geq 1$, $c \geq 0$, and $0 \leq E \leq kc$, define

$$\begin{aligned} \mathcal{L}_{k,c,E} &= \left\{ e = (e_1, \dots, e_k) \in \{0, \dots, c\}^k : \sum_{i=1}^k e_i = E \right\}, \\ N_{k,c,E} &= |\mathcal{L}_{k,c,E}| = [z^E](1 + z + \dots + z^c)^k. \end{aligned} \quad (1)$$

If $e, e' \in \mathcal{L}_{k,c,E}$ satisfy $e'_i \geq e_i$ for every i , their equal sums imply $e' = e$. Distinct tuples in $\mathcal{L}_{k,c,E}$ are therefore incomparable under the componentwise order. When these tuples specify opening positions on hash chains, an adversary given the openings for one tuple cannot forge openings for a different tuple in the same set by forward hashing alone.

To sample tuples uniformly, we use the counts $N_{k,c,E}$. If the first entry is $d \in \{0, \dots, c\}$, the remaining $k-1$ entries must sum to $E-d$, giving $N_{k-1,c,E-d}$ possible tuples. Summing over the first entry yields the recurrence

$$N_{k,c,E} = \sum_{d=0}^c N_{k-1,c,E-d}, \quad N_{0,c,E} = \begin{cases} 1, & E = 0, \\ 0, & E \neq 0. \end{cases}$$

We set $N_{k,c,E} = 0$ when $E < 0$ or $E > kc$.

The encoding algorithm of [22] uses these counts to map each integer in $[N_{k,c,E}]$ to the corresponding tuple in lexicographic order. This mapping is a bijection, so a uniformly sampled integer yields a uniform tuple in $\mathcal{L}_{k,c,E}$. Algorithm 5 in Appendix A.1 gives the pseudocode.

3 The BPORS-C+BFP Construction

In this section, we present the construction of BPORS-C+BFP and analyze its costs. We begin by reviewing the sequence of prior work on which it builds.

3.1 Previous Few-Time Signature Schemes

The line of few-time signatures considered here begins with HORS, introduced by Reyzin and Reyzin [20]. This scheme generates t independent secret values $x_0, \dots, x_{t-1}$ and publishes their hashes as the public key. To sign a message, it interprets the digest as a sequence of k indices $(i_1, \dots, i_k) \in [t]^k$ and reveals the corresponding values $(x_{i_1}, \dots, x_{i_k})$. The verifier hashes these values and compares them with the corresponding public-key entries. The public key and signature have sizes nt and nk bytes, respectively.

HORST, introduced as the FTS component of SPHINCS, replaces the t public hashes of HORS with a single Merkle root [4]. A signature now includes the selected secret values and their Merkle authentication data. To reduce redundancy among the authentication paths, HORST includes all nodes at a fixed intermediate tree level and shortens each selected leaf's path to end at that level. The verifier checks the selected openings against these nodes and hashes the nodes upward to recover the root. The message digest is still interpreted as a sequence of indices, so the same leaf may be selected more than once, and a signature may reveal fewer than k distinct secrets.

FORS ensures that each signature uses k distinct secrets by selecting one leaf from each of k separate Merkle trees [5,18]. Each tree contains t leaves. The signature includes the selected secrets and their authentication paths. Verification hashes each secret to its leaf, reconstructs the k roots, and compresses them into the FORS public key. The signature size is $nk(1 + \log t)$ bytes. Since the selected leaves belong to different trees, their authentication paths are disjoint.

Distinct secrets can also be selected within a single Merkle tree with t leaves, as in PORS [2]. The message digest is mapped to a k -element subset $S \in \binom{[t]}{k}$, ensuring that the selected leaf labels are distinct. The shared tree allows Octopus to remove redundant nodes from their authentication paths. The resulting signature consists of k secret values and $m(S)$ authentication values, occupying $n(k + m(S))$ bytes. The amount of authentication data depends on the geometry of S , so the signature length varies with the selected leaves.

Abri and Katz control this variable authentication size by applying forced pruning to PORS, obtaining PORS+FP [1]. Given a message digest M' , the

signer tries successive counter values z , starting from zero. For each z , it computes

$$S_z = H_2(M' \parallel z) \in \binom{[t]}{k}$$

and determines the number $m(S_z)$ of authentication nodes required by Octopus. It stops at the first counter satisfying $m(S_z) \leq L$, where L is a public authentication budget, and includes that counter in the signature. The verifier recomputes S_z from M' and the received counter, then checks the same budget. Each signature reserves L authentication slots and pads any unused slots, giving a fixed size of $\ell_{\text{ctr}} + n(k + L)$ bytes with an ℓ_{ctr} -byte counter. The signing cost depends on how often a candidate passes the pruning test. For a uniform raw support, the acceptance probability is

$$\Pr[m(S) \leq L] = \frac{\sum_{m=0}^L A_\epsilon(k, m)}{\binom{t}{k}}.$$

A smaller threshold reduces the signature size but increases the expected number of counter attempts. It also changes the distribution of disclosed supports, since accepted signatures favor selections requiring fewer authentication nodes.

FTS parameter selection is also constrained by the coverage probability, namely the probability that all openings required by a fresh encoding can be obtained from previous signatures under the same key. BPORS+FP introduces a bucket structure that distributes signatures among independent PORS child keys, reducing the expected number of uses of each child [19]. This lowers the coverage probability and allows smaller child parameters for a fixed coverage bound.

In BPORS+FP, an FTS key contains K coordinates. Each coordinate has $B = 2^b$ buckets indexed by $j \in [B]$. Each bucket is associated with an independent PORS instance with t leaves, which we call a child PORS instance. The public key of this instance is its Merkle root, stored at leaf j of the coordinate's outer tree. The roots of the K outer trees are compressed into the FTS public key. For each signature, the message encoding selects a bucket label $j_a \in [B]$ and a support $S_a \in \binom{[t]}{k}$ in every coordinate $a \in [K]$. The label j_a identifies the selected child PORS instance, and S_a specifies the leaves opened in its Merkle tree. These openings require $m(S_a)$ authentication values within that tree and b values on the outer authentication path. The signature therefore contains Kk secret values, the authentication values from the selected child trees, and Kb outer authentication values.

BPORS+FP also uses forced pruning, with a single global budget L for the total number of child authentication nodes across all K coordinates. The signer varies the counter until the candidate encoding satisfies

$$\sum_{a \in [K]} m(S_a) \leq L.$$

For an accepted encoding, the signer appends $L - \sum_{a \in [K]} m(S_a)$ all-zero n -byte strings to the authentication data. The resulting authentication list has

$Kb + L$ entries, giving a fixed signature size of $\ell_{\text{ctr}} + n(Kk + Kb + L)$ bytes with an ℓ_{ctr} -byte counter. The global pruning test depends only on the supports, which are sampled independently of the uniform bucket labels. Each bucket is therefore still selected with probability $1/B$ after acceptance. For q signatures with independently sampled accepted encodings under the same FTS key, the number selecting a fixed child key follows $\text{Bin}(q, 1/B)$. The shared budget may nevertheless make the accepted supports dependent across coordinates.

3.2 The construction of BPORS-C+BFP

We use the coordinate and bucket structure of BPORS+FP [19], with $K \geq 1$ coordinates, and extend it with hash-chain openings and blockwise forced pruning. Figure 1 illustrates the resulting construction.

For each coordinate $a \in [K]$, bucket $j \in [B]$, and chain $u \in [t]$, let $x_{a,j,u,0}$ be a secret starting value derived using $F_{K_{\text{sk}}}$. We replace each PORS secret value with a hash chain of $c + 1$ edges, where $c \geq 0$:

$$x_{a,j,u,v+1} = H_0(v, x_{a,j,u,v}), \quad 0 \leq v \leq c. \quad (2)$$

The endpoint $x_{a,j,u,c+1}$ forms leaf u of the corresponding PORS Merkle tree. Denote its root by $R_{a,j}$ and the root of coordinate a 's outer tree by R_a . The FTS public key is

$$P = H_{\text{pk}}(R_0, \dots, R_{K-1}).$$

For standalone signing under a fixed FTS key, a message digest M' and counter z determine a candidate encoding

$$y = H_2(M' \parallel z) = ((j_a, S_a, e_a))_{a \in [K]}.$$

In coordinate a , the label $j_a \in [B]$ selects a bucket, $S_a \in \binom{[t]}{k}$ selects k chains in its PORS tree, and $e_a \in \mathcal{L}_{k,c,E}$ specifies their opening depths, where $1 \leq k \leq t$ and $0 \leq E \leq kc$. Write $S_a = \{u_{a,1} < \dots < u_{a,k}\}$ and $e_a = (e_{a,1}, \dots, e_{a,k})$; depth $e_{a,\ell}$ belongs to chain $u_{a,\ell}$. The constant-sum set $\mathcal{L}_{k,c,E}$ and its cardinality $N = N_{k,c,E}$ are defined in Section 2.4. Before forced pruning, the encoding space is

$$\Omega = \left([B] \times \binom{[t]}{k} \times \mathcal{L}_{k,c,E} \right)^K. \quad (3)$$

The chain value specified by a candidate encoding is

$$\sigma_{a,\ell} = x_{a,j_a,u_{a,\ell},c-e_{a,\ell}}, \quad a \in [K], \quad 1 \leq \ell \leq k. \quad (4)$$

A larger depth specifies an earlier chain value. Starting from $\sigma_{a,\ell}$, the verifier applies $e_{a,\ell} + 1$ hashes to reach the committed endpoint. Since the entries of e_a sum to E , recovering the k selected endpoints takes exactly $k + E$ chain hashes per coordinate. For a fixed bucket and support, there are N possible depth tuples, while each selected chain still contributes one n -byte value to the signature.

The selected endpoints must also be authenticated through their PORs tree and the coordinate's outer tree. For support S_a , Octopus requires $m(S_a)$ authentication nodes to reconstruct R_{a,j_a} , as described in Section 2.2; the outer path from R_{a,j_a} to R_a adds b nodes. We use blockwise forced pruning to bound the variable number of PORs-tree authentication nodes. Let $G_1, \dots, G_g$ partition $[K]$ into nonempty blocks, let $s_i = |G_i|$, and assign each block a public budget L_i . A candidate is accepted exactly when every block meets its budget:

$$\text{Acc}_i(y) \iff \sum_{a \in G_i} m(S_a) \leq L_i, \quad \text{Acc}(y) = \bigwedge_{i=1}^g \text{Acc}_i(y). \quad (5)$$

We require $L_i \geq s_i \min_{S \in \binom{[t]}{k}} m(S)$ so that each block has positive acceptance probability. With one budget shared by different coordinates, an encoding is accepted based on their combined authentication-node count. This can make the selected supports dependent after acceptance. Separate block budgets confine this dependence to each block, limiting the size of the joint counting problems in Section 4.

The signer tests successive counter values, recomputing the entire candidate encoding each time, and uses the first one that satisfies $\text{Acc}(y)$. It outputs the counter, the Kk chain openings in (4), the PORs-tree authentication nodes, and the outer authentication paths. Each block's authentication data is padded to $L_i + s_i b$ values. The verifier recomputes y from the digest and counter, checks $\text{Acc}(y)$ and the padding, and hashes the openings to their endpoints. It then reconstructs each selected PORs root R_{a,j_a} , follows the outer paths to obtain the coordinate roots R_a , and checks their compression against P .

When $c = E = 0$, each chain has one edge and reveals its starting value. With one block $G_1 = [K]$ and the corresponding global budget, BPORS-C+BFP reduces to BPORS+FP [19]. The effect of the additional depth choices on coverage is analyzed in Section 4.

3.3 Signature Algorithms

The following algorithms describe the standalone interface for one fixed FTS key, with public key $PK = P$ and encoding function H_2 . Section 3.5 specifies how their opening and authentication procedures are used within a complete SPHINCS+-style signature.

Key generation. Algorithm 1 samples the secret derivation key K_{sk} , derives the chain endpoints, and computes the roots $R_{a,j}$, R_a , and P . It returns $PK = P$ and $SK = K_{\text{sk}}$. The PORs and outer trees use separate H_1 domains; their address inputs are implicit. When $B = 1$, the outer tree has height zero and $R_a = R_{a,0}$.

Signing. The ℓ_{ctr} -byte counter has $R = 2^{\ell_{\text{ctr}}}$ possible values $z = 0, \dots, R - 1$, each encoded in exactly ℓ_{ctr} bytes. For an accepted encoding, let A_i be the

Algorithm 1 BPORS-C+BFP key generation**Require:** Public parameters.**Ensure:** Public key PK and secret key SK .

```

1: function KEYGEN
2:   Sample  $K_{\text{sk}}$  uniformly from the PRF key space.
3:   for  $a \in [K]$  do
4:     for  $j \in [B]$  do
5:       for  $u \in [t]$  do
6:          $x_{a,j,u,0} \leftarrow F_{K_{\text{sk}}}(\text{start}, a, j, u)$ .
7:         for  $v = 0, \dots, c$  do
8:            $x_{a,j,u,v+1} \leftarrow H_0(v, x_{a,j,u,v})$ .
9:         end for
10:      end for
11:       $R_{a,j} \leftarrow \text{Merkle}_{H_1}(x_{a,j,0,c+1}, \dots, x_{a,j,t-1,c+1})$ .
12:    end for
13:     $R_a \leftarrow \text{Merkle}_{H_1}(R_{a,0}, \dots, R_{a,B-1})$ .
14:  end for
15:   $P \leftarrow H_{\text{pk}}(R_0, \dots, R_{K-1})$ .
16:   $PK \leftarrow P$ ;  $SK \leftarrow K_{\text{sk}}$ .
17:  return  $(PK, SK)$ .
18: end function

```

authentication values of block G_i in canonical order, padded with zeros to $L_i + s_i b$ hash-value slots. The signature is

$$\sigma_{\text{FTS}} = (z, (\sigma_{a,\ell})_{a \in [K], 1 \leq \ell \leq k}, (A_i)_{1 \leq i \leq g}).$$

The verifier derives the bucket labels, chain indices, depths, and authentication-node positions from M' and z ; the signature does not transmit them separately. If none of the R counters passes the budget test, signing returns $\perp$ without wrapping. The counter search and key derivation are deterministic, so repeated signing of the same digest under the same key returns the same result.

Verification. Verification does not require the supplied counter to be the first one that passes the budget test. For every nonfailure output of Algorithm 2, the chain openings recover the committed endpoints and the authentication values recover $P = PK$. The budget and padding checks also pass, so Algorithm 3 accepts.

3.4 Signature Size and Computational Cost

Signature size. An FTS signature contains Kk chain openings and Kb outer authentication values. The blocks reserve a total of $L_{\text{tot}} = \sum_{i=1}^g L_i$ child authentication values. Including the counter, its size in bytes is

$$S_{\text{FTS}} = \ell_{\text{ctr}} + n(Kk + Kb + L_{\text{tot}}). \quad (6)$$

Algorithm 2 BPORS-C+BFP signing

Require: Secret key SK and digest M' .
Ensure: An FTS signature σ_{FTS} , or $\perp$.

```

1: function SIGN( $SK, M'$ )
2:   Set  $K_{\text{sk}} \leftarrow SK$ .
3:   for  $z = 0, \dots, R - 1$  do
4:      $y \leftarrow H_2(M' \parallel z)$ .
5:     Write  $y = ((j_a, S_a, e_a))_{a \in [K]}$ .
6:     if Acc( $y$ ) then
7:       Recompute all chains and both tree levels using  $K_{\text{sk}}$ ; compute  $P$ .
8:       Collect the openings in (4) and the authentication values.
9:       for  $i = 1, \dots, g$  do
10:        Initialize  $A_i$  as an empty list.
11:        for  $a \in G_i$  in increasing order do
12:          Append the  $m(S_a)$  PORS-tree nodes to  $A_i$ .
13:          Append the  $b$  outer-path values for  $j_a$  to  $A_i$ .
14:        end for
15:        Append  $L_i - \sum_{a \in G_i} m(S_a)$  all-zero values to  $A_i$ .
16:      end for
17:      return  $(z, (\sigma_{a,\ell})_{a,\ell}, (A_i)_i)$ .
18:    end if
19:  end for
20:  return  $\perp$ .
21: end function

```

Key generation and signing. Following prior work [1,19], we model computational cost by counting hash and PRF evaluations and each evaluation of H_2 counts as one call. Each child requires t PRF evaluations to derive its chain starts, $(c+1)t$ evaluations of H_0 along the chains, and $t-1$ evaluations of H_1 for the child tree. The K outer trees require another $K(B-1)$ evaluations of H_1 , and the final public-key compression requires one evaluation of H_{pk} . The total key-regeneration cost is therefore

$$G = KB((c+3)t-1) + K(B-1) + 1. \quad (7)$$

After finding an accepted encoding, the signer regenerates the key once, storing the required chain openings and authentication nodes. Each counter attempt requires one evaluation of H_2 . Let Y be uniformly sampled from Ω , and let τ_g be the probability that Y satisfies all block budgets. Since the raw block encodings are independent and each budget is checked within its block,

$$\tau_g = \Pr[\text{Acc}(Y)] = \prod_{i=1}^g \Pr[\text{Acc}_i(Y)].$$

For independent uniform candidates, signing fails after R attempts with probability $(1 - \tau_g)^R$, and the expected number of counter attempts, including an

Algorithm 3 BPORS-C+BFP verification**Require:** Public key PK , digest M' , and signature σ_{FTS} .**Ensure:** accept or reject.

```

1: function VERIFY( $PK, M', \sigma_{\text{FTS}}$ )
2:   Check the fixed length and parse  $z, (\sigma_{a,\ell})_{a,\ell}, (A_i)_i$ ; return reject if malformed.
3:    $y \leftarrow H_2(M' \parallel z)$ .
4:   Write  $y = ((j_a, S_a, e_a))_{a \in [K]}$ .
5:   if  $\neg \text{Acc}(y)$  then
6:     return reject.
7:   end if
8:   for  $i = 1, \dots, g$  do
9:     for  $a \in G_i$  in increasing order do
10:      for  $\ell = 1, \dots, k$  do
11:         $w_{a,\ell} \leftarrow \sigma_{a,\ell}$ .
12:        for  $v = c - e_{a,\ell}, \dots, c$  do
13:           $w_{a,\ell} \leftarrow H_0(v, w_{a,\ell})$ .
14:        end for
15:      end for
16:      Use  $(w_{a,\ell})_{\ell=1}^k$  and the next  $m(S_a)$  values of  $A_i$  to reconstruct  $\hat{R}_{a,j_a}$ .
17:      Use  $\hat{R}_{a,j_a}$  and the next  $b$  values of  $A_i$  to reconstruct  $\hat{R}_a$ .
18:    end for
19:    return reject if any remaining value in  $A_i$  is nonzero.
20:  end for
21:   $\hat{P} \leftarrow H_{\text{pk}}(\hat{R}_0, \dots, \hat{R}_{K-1})$ .
22:  return accept if  $\hat{P} = PK$ , and reject otherwise.
23: end function

```

exhausted search, is

$$\sum_{z=0}^{R-1} (1 - \tau_g)^z = \frac{1 - (1 - \tau_g)^R}{\tau_g} \leq \tau_g^{-1}.$$

The expected signing cost therefore satisfies

$$T_{\text{unit}} \leq G + \tau_g^{-1}. \quad (8)$$

Smaller pruning budgets reduce the signature size but also lower the acceptance probability τ_g , increasing the expected number of counter attempts.

Verification. The verifier first evaluates $H_2(M' \parallel z)$ once to recover the encoding. The chain openings require $K(k + E)$ evaluations of H_0 . Let $m_{\text{tot}} = \sum_a m(S_a)$ be the total number of child authentication nodes. Reconstructing the child roots costs $K(k - 1) + m_{\text{tot}}$ evaluations of H_1 . The outer paths cost another Kb evaluations of H_1 , and public-key compression uses one evaluation of H_{pk} . Including the encoding hash, the verification count is

$$\begin{aligned} V_{\text{FTS}} &= K(k + E) + K(k - 1) + m_{\text{tot}} + Kb + 2 \\ &\leq K(2k + E - 1) + L_{\text{tot}} + Kb + 2. \end{aligned} \quad (9)$$

Every encoding that passes pruning satisfies $m_{\text{tot}} \leq L_{\text{tot}}$. Increasing c increases key-regeneration work, while increasing E increases verification work. The next section analyzes coverage under repeated use, providing the bounds needed to select these parameters.

3.5 Integration with SPHINCS+

Following [1], when integrating BPORS-C+BFP into SPHINCS+, we use a counter to select the bottom FTS address jointly with its encoding, while retaining the surrounding message randomization, WOTS+, and hypertree procedures.

Let PK_{SPX} denote the global SPHINCS+-style public key, and let P_I denote the public key of the bottom FTS instance at address $I \in [2^h]$. The signer first derives the message randomizer R_{msg} using the surrounding scheme’s randomization procedure and computes

$$M' = H_{\text{msg}}(R_{\text{msg}} \parallel PK_{\text{SPX}} \parallel m).$$

This digest remains fixed throughout the counter search. For $z = 0, \dots, R - 1$, the signer computes

$$(I_z, Y_z) = H_{\text{addr}}(M' \parallel z) \in [2^h] \times \Omega$$

and stops at the first z for which $\text{Acc}(Y_z)$ holds. If no candidate passes pruning, signing returns $\perp$. Using the selected encoding Y_z , the signer generates the chain openings and padded authentication data for the FTS instance at I_z using the opening and authentication procedures of Algorithm 2. The signer then authenticates P_{I_z} through the WOTS+ and hypertree layers. The complete signature contains R_{msg} , the FTS signature including z , and the WOTS+ and hypertree authentication data.

The verifier recomputes M' and $(I_z, Y_z) = H_{\text{addr}}(M' \parallel z)$ from the message, randomizer, and counter. It first checks $\text{Acc}(Y_z)$, the FTS signature format, and the zero padding, and reconstructs $\hat{P}_{I_z}$ from the supplied openings and authentication data. It then verifies the WOTS+ and hypertree authentication of $\hat{P}_{I_z}$ at address I_z against PK_{SPX} .

4 Depth-Sensitive Coverage Analysis

We analyze the probability that a fresh encoding both passes pruning and is covered by disclosures from previous signatures under one FTS key. We first specify the coverage experiment and characterize the chain openings available from a signing history. We then derive a general coverage upper bound and an exact calculation for individual blocks. The exact block probabilities are multiplied to obtain the overall coverage probability.

4.1 The Coverage Experiment

Following the coverage framework of PORS+FP and BPORS+FP [1,19], let $\mathcal{D}_{\text{raw}}$ be the uniform distribution on Ω before pruning. For $Y \leftarrow \mathcal{D}_{\text{raw}}$ and $\tau_g > 0$, the distribution conditioned on the acceptance predicate Acc in (5) is

$$\mathcal{D}_{\text{acc}}(y) = \Pr[Y = y \mid \text{Acc}(Y)] = \frac{\mathcal{D}_{\text{raw}}(y) \mathbf{1}[\text{Acc}(y)]}{\tau_g}.$$

Conditional on returning an encoding, rejection sampling from independent uniform candidates produces an encoding distributed according to $\mathcal{D}_{\text{acc}}$. The pruning tests depend only on the supports, so the bucket indices and depth tuples remain mutually independent and uniformly distributed, even conditioned on the accepted supports.

For a fixed history h and target encoding y , the predicate $\text{Cov}(h; y)$ holds when all openings required by y can be computed from the values disclosed in h . Let $H = (X_1, \dots, X_q)$ consist of q independent encodings sampled from $\mathcal{D}_{\text{acc}}$, representing the previous accepted signatures. Let the target encoding Y be sampled independently from $\mathcal{D}_{\text{raw}}$. We define

$$\Gamma_q := \Pr[\text{Acc}(Y) \wedge \text{Cov}(H; Y)]. \quad (10)$$

Thus Γ_q is the probability that a fresh candidate passes pruning and that all its required openings can be computed from the values revealed by q previous accepted signatures.

4.2 Coverage from Chain Openings

Fix a coordinate $a \in [K]$ and a bucket index $j \in [B]$. We consider the child key associated with this bucket. Let $h = ((S^{(1)}, e^{(1)}), \dots, (S^{(r)}, e^{(r)}))$ denote the history of the r previous signatures that selected this child key, with $S^{(i)} = \{u_1^{(i)} < \dots < u_k^{(i)}\}$ for $1 \leq i \leq r$. An opening at depth d reveals the chain value at position $c - d$. From this value, the adversary can compute every opening on the same chain with depth less than d by forward hashing. For each chain index $u \in [t]$, define its largest disclosed depth by

$$D_h(u) = \max(\{-1\} \cup \{e_\ell^{(i)} : u_\ell^{(i)} = u, 1 \leq i \leq r, 1 \leq \ell \leq k\}). \quad (11)$$

We set $D_h(u) = -1$ when no opening on chain u has been disclosed, since zero is already a valid opening depth. A depth-zero opening reveals $x_{a,j,u,c}$, which lies one hash evaluation before the public endpoint $x_{a,j,u,c+1}$. By definition, D_h records the accumulated information from all signatures in h . For example, openings on the same three chains with depth tuples $(2, 0, 0)$ and $(0, 2, 0)$ give the frontier $(2, 2, 0)$ and together cover the target tuple $(1, 1, 0)$. Neither signature alone covers that target.

Within this fixed child key, let $y = (S', e')$ be a target encoding, with $S' = \{u'_1 < \dots < u'_k\}$. The predicate $\text{Cov}(h; y)$ holds when all openings required by

y can be obtained from the values disclosed in h by evaluating chain hashes. Equivalently,

$$\text{Cov}(h; y) \iff e'_\ell \leq D_h(u'_\ell) \text{ for every } 1 \leq \ell \leq k. \quad (12)$$

Coverage alone does not require the target to satisfy the authentication budgets. For the complete FTS, Γ_q in (10) is the probability that the target both passes pruning and is covered.

Coverage for a fixed history. For $d \geq 0$, let $G_d(z) = 1 + z + \dots + z^d$. At a disclosed chain u , the available target depths are $0, \dots, D_h(u)$. We represent the possible choices for this chain by

$$1 + xG_{D_h(u)}(z) = 1 + x + xz + \dots + xz^{D_h(u)}.$$

The constant term 1 corresponds to excluding u from the target support. For each $0 \leq d \leq D_h(u)$, the term xz^d corresponds to selecting chain u and opening it at depth d . When these factors are multiplied over all disclosed chains, the exponent of x counts the selected chains, while the exponent of z records the sum of their opening depths. Thus, the coefficient of $x^k z^E$ counts the covered target encodings with exactly k selected chains and total depth E . For a target Y sampled uniformly from $\binom{[t]}{k} \times \mathcal{L}_{k,c,E}$, define

$$p(h) := \Pr[\text{Cov}(h; Y)] = \frac{[x^k z^E] \prod_{u: D_h(u) \geq 0} (1 + xG_{D_h(u)}(z))}{\binom{[t]}{k} N}. \quad (13)$$

The denominator counts all possible child encodings. Thus, $p(h)$ is the probability that a uniformly chosen child encoding can be covered using the openings disclosed in h , without imposing the block authentication budgets.

Comparison with subset coverage. Let $I_h = \{u : D_h(u) \geq 0\}$ be the set of disclosed chain indices. Discarding the depth requirements gives the familiar subset-coverage bound [1,19]:

$$p(h) \leq \frac{\binom{|I_h|}{k}}{\binom{[t]}{k}} \leq \frac{\binom{\min(rk, t)}{k}}{\binom{[t]}{k}}. \quad (14)$$

The first inequality discards opening depths, and the second also discards overlap among the historical supports.

4.3 A General Coverage Upper Bound

Averaging (13) over histories requires tracking repeated selections of each chain. We first derive an upper bound that avoids this tracking while accounting for only

the disclosed depths. We obtain this bound by treating each historical occurrence of a chain as a separate copy. If the same chain occurs at depths $d_1, \dots, d_v$, then

$$1 + xG_{\max_i d_i}(z) \preceq \prod_{i=1}^v (1 + xG_{d_i}(z)), \quad (15)$$

where $\preceq$ denotes an inequality between every pair of corresponding coefficients. The left-hand side represents a single chain that can either be omitted or opened at any depth up to $\max_i d_i$. On the right-hand side, this chain is replaced by v virtual copies, where copy i can be opened at any depth up to d_i . If all copies except one attaining the maximum depth are omitted, the remaining factor is exactly the left-hand side. The full product also counts additional choices, including selections of multiple copies of the same real chain.

For a depth tuple $e \in \mathcal{L}_{k,c,E}$, define

$$V_e(x, z) = \prod_{\ell=1}^k (1 + xG_{e_\ell}(z)), \quad P(x, z) = \sum_{e \in \mathcal{L}_{k,c,E}} V_e(x, z).$$

For a uniformly sampled depth tuple e , the average of V_e is P/N .

Theorem 1 (Average coverage bound for a child key). *Let H_{ch} be the random child history induced by $r \geq 1$ independent accepted BPORS-C+BFP signatures selecting a fixed child key. For an independent uniform raw child encoding Y , we have*

$$\Pr[\text{Cov}(H_{\text{ch}}; Y)] \leq \alpha_C(r) := \min \left\{ 1, \frac{[x^k z^E] P(x, z)^r}{\binom{t}{k} N^{r+1}} \right\}. \quad (16)$$

Set $\alpha_C(0) = 0$.

Proof. Fix a realization of H_{ch} , $h = ((S^{(1)}, e^{(1)}), \dots, (S^{(r)}, e^{(r)}))$. The probability $p(h)$ in (13) is over the raw target Y alone. Using (15) to bound the generating function in (13) and regrouping the resulting factors by signature gives

$$p(h) \leq \frac{[x^k z^E] \prod_{i=1}^r V_{e^{(i)}}(x, z)}{\binom{t}{k} N}.$$

Let $\mathbf{S} = (S^{(1)}, \dots, S^{(r)})$ be the random support history, and fix a value $\mathbf{s}$ with positive probability. Conditional on $\mathbf{S} = \mathbf{s}$, the historical depth tuples remain independent and uniform on $\mathcal{L}_{k,c,E}$. Each of the N^r possible combinations therefore has probability N^{-r} . Define $\bar{p}(\mathbf{s}) := \Pr[\text{Cov}(H; Y) \mid \mathbf{S} = \mathbf{s}]$ for the coverage probability conditional on the support history $\mathbf{s}$. Averaging the preceding

bound over these depth assignments gives

$$\begin{aligned}
\bar{p}(\mathbf{s}) &\leq \frac{[x^k z^E] \sum_{e^{(1)}, \dots, e^{(r)} \in \mathcal{L}_{k,c,E}} \prod_{i=1}^r V_{e^{(i)}}(x, z)}{\binom{t}{k} N^{r+1}} \\
&= \frac{[x^k z^E] \prod_{i=1}^r \left(\sum_{e^{(i)} \in \mathcal{L}_{k,c,E}} V_{e^{(i)}}(x, z) \right)}{\binom{t}{k} N^{r+1}} \\
&= \frac{[x^k z^E] P(x, z)^r}{\binom{t}{k} N^{r+1}}.
\end{aligned} \tag{17}$$

The first equality uses the distributive law, and the second follows from the definition of $P(x, z)$.

The virtual-copy bound ignores overlap among the historical supports and depends only on the historical depth tuples, so the resulting upper bound on $\bar{p}(\mathbf{s})$ is the same for every support history $\mathbf{s}$. Averaging over the distribution of the support history $\mathbf{S}$ therefore gives the same bound on $\Pr[\text{Cov}(H; Y)]$. Since a probability is at most one, this proves (16). When $r = 0$, no chain openings have been disclosed, so the coverage probability is zero.

Fix the historical depth tuples $e^{(1)}, \dots, e^{(r)}$. There are $\binom{rk}{k}$ ways to select k of the rk virtual copies. Each selection can be paired with at most N target depth tuples, giving a total count of at most $\binom{rk}{k} N$. Summing over the N^r possible combinations of historical depth tuples gives $[x^k z^E] P(x, z)^r \leq N^{r+1} \binom{rk}{k}$. Consequently,

$$\alpha_C(r) \leq \min \left\{ 1, \frac{\binom{rk}{k}}{\binom{t}{k}} \right\} = \frac{\binom{\min(rk, t)}{k}}{\binom{t}{k}}.$$

The average bound $\alpha_C(r)$ is therefore no larger than the coarse subset bound in (14).

Bucket thinning and the complete FTS. We now return to the complete FTS history H and target Y of Section 4.1. Consider one coordinate $a \in [K]$, and let j_a be the bucket index selected by the raw target. Each of the q accepted historical signatures selects the same child key with probability $1/B$, independently of the others. Thus, the number r of signatures selecting bucket j_a in coordinate a follows $\text{Bin}(q, 1/B)$. Applying the coverage bound for the associated child key from Theorem 1 and averaging over r gives the coordinate coverage bound

$$\beta_C(q) = \sum_{r=0}^q \binom{q}{r} B^{-r} (1 - B^{-1})^{q-r} \alpha_C(r). \tag{18}$$

The shared pruning budgets can correlate the historical supports across coordinates. We therefore condition on these supports before combining the

coordinate bounds. Let $\mathcal{S}$ collect all historical supports, and let C_a denote the event that the raw target's required openings in coordinate a are covered. Fix a value $\mathbf{s}$ of $\mathcal{S}$ with positive probability. Since pruning depends only on the supports, conditioning on $\mathcal{S} = \mathbf{s}$ preserves the independence and uniformity of the bucket indices and depth tuples.

For any set of r historical signatures matching the target bucket in coordinate a , $\mathbf{s}$ fixes their child support history. By (17), the conditional coverage probability, taken over the historical depth tuples and the raw target, is at most $\alpha_C(r)$. The number of matching signatures still follows $\text{Bin}(q, 1/B)$. Averaging over this count therefore gives

$$\begin{aligned} \Pr[C_a \mid \mathcal{S} = \mathbf{s}] &\leq \sum_{r=0}^q \binom{q}{r} B^{-r} (1 - B^{-1})^{q-r} \alpha_C(r) \\ &= \beta_C(q), \quad a \in [K]. \end{aligned}$$

Given $\mathcal{S} = \mathbf{s}$, the historical bucket indices, depth tuples, and raw target choices are independent across coordinates, so

$$\Pr \left[\bigcap_{a \in [K]} C_a \mid \mathcal{S} = \mathbf{s} \right] = \prod_{a \in [K]} \Pr[C_a \mid \mathcal{S} = \mathbf{s}] \leq \beta_C(q)^K.$$

This bound holds for every possible support history. Averaging over the distribution of $\mathcal{S}$ therefore gives

$$\begin{aligned} p_{\text{raw}}(q) &:= \Pr \left[\bigcap_{a \in [K]} C_a \right] \\ &= \sum_{\mathbf{s}} \Pr[\mathcal{S} = \mathbf{s}] \Pr \left[\bigcap_{a \in [K]} C_a \mid \mathcal{S} = \mathbf{s} \right] \\ &\leq \beta_C(q)^K. \end{aligned}$$

Here $p_{\text{raw}}(q)$ is the probability that a raw target is covered by disclosures from q accepted historical signatures, without requiring the target to satisfy its authentication budgets. Requiring the target to satisfy those budgets can only decrease the probability, giving

$$\Gamma_q \leq p_{\text{raw}}(q) \leq \beta_C(q)^K. \quad (19)$$

The bound accounts for the constraints on opening depths but ignores overlap among historical supports.

4.4 Exact Coverage of a Pruning Block

We now analyze the coordinates within each pruning block jointly to account for support overlap. Consider a block of s coordinates with shared child-authentication budget L , and relabel its coordinates by $a \in [s]$. Its raw encoding

space is

$$\Omega_s = \left([B] \times \binom{[t]}{k} \times \mathcal{L}_{k,c,E} \right)^s,$$

Let $\mathcal{D}_{\text{raw}}^{(s)}$ be uniform on Ω_s , and let $\mathcal{D}_{\text{acc}}^{(s,L)}$ be uniform on the encodings satisfying the block budget. For q independent historical encodings $H = (X_1, \dots, X_q)$ drawn from $\mathcal{D}_{\text{acc}}^{(s,L)}$ and an independent target Y drawn from $\mathcal{D}_{\text{raw}}^{(s)}$, define

$$\gamma_{s,L}(q) = \Pr[\text{Acc}(Y) \wedge \text{Cov}(H; Y)]. \quad (20)$$

The histories are conditioned on acceptance. The target is drawn before pruning, and its acceptance remains part of the event in (20).

Coverage of a fixed target. Fix a target $y = ((j_a, S_a, e_a))_{a \in [s]} \in \Omega_s$, its full set of opening requirements is

$$\mathcal{U}^*(y) = \{(a, j_a, u_{a,\ell}, e_{a,\ell}) : a \in [s], 1 \leq \ell \leq k\}.$$

A historical encoding supplies a requirement (a, j_a, u, d) if, in coordinate a , it selects bucket j_a and opens chain u at depth at least d . For each $\xi \in \mathcal{U}^*(y)$, let $\mathcal{E}_\xi$ be the event that none of the q historical encodings supplies ξ . Coverage fails exactly when at least one such event occurs, so

$$\Pr[\text{Cov}(H; y)] = 1 - \Pr \left[\bigcup_{\xi \in \mathcal{U}^*(y)} \mathcal{E}_\xi \right]. \quad (21)$$

We apply inclusion–exclusion to the union in (21). For each subset $\mathcal{T} \subseteq \mathcal{U}^*(y)$, we consider the event that none of the q historical encodings supplies any requirement in $\mathcal{T}$. This event is $\bigcap_{\xi \in \mathcal{T}} \mathcal{E}_\xi$. For one accepted encoding $X \leftarrow \mathcal{D}_{\text{acc}}^{(s,L)}$, let $p_{\mathcal{T}}$ be the probability that X supplies none of these requirements:

$$p_{\mathcal{T}} = \Pr[X \text{ supplies no requirement in } \mathcal{T}].$$

Since the q historical encodings are independent samples from $\mathcal{D}_{\text{acc}}^{(s,L)}$, we have

$$\Pr \left[\bigcap_{\xi \in \mathcal{T}} \mathcal{E}_\xi \right] = p_{\mathcal{T}}^q. \quad (22)$$

Inclusion–exclusion now yields

$$\Pr[\text{Cov}(H; y)] = \sum_{\mathcal{T} \subseteq \mathcal{U}^*(y)} (-1)^{|\mathcal{T}|} p_{\mathcal{T}}^q. \quad (23)$$

For $\mathcal{T} = \emptyset$, the defining event is certain, so $p_{\emptyset} = 1$. When $q = 0$, the product over historical encodings is empty, and we interpret $p_{\mathcal{T}}^0 = 1$ for every $\mathcal{T}$, including when $p_{\mathcal{T}} = 0$.

Averaging over raw targets. The raw target Y is uniform on Ω_s . Averaging (23) over this space and requiring target acceptance gives

$$\gamma_{s,L}(q) = \frac{1}{|\Omega_s|} \sum_{y \in \Omega_s} \mathbf{1}[\text{Acc}(y)] \sum_{\mathcal{T} \subseteq \mathcal{U}^*(y)} (-1)^{|\mathcal{T}|} p_{\mathcal{T}}^q. \quad (24)$$

Here $|\Omega_s| = (B \binom{t}{k} N)^s$, and $\text{Acc}(y)$ holds exactly when $\sum_{a \in [s]} m(S_a) \leq L$.

To evaluate (24), we first simplify the target sum using bucket symmetry. All child keys within a coordinate have the same parameters and tree shape. Pruning depends only on supports, so the bucket tuple in each accepted historical encoding remains uniform on $[B]^s$ and independent of the supports and opening depths. For fixed target supports and depths, the coverage probability over H is therefore the same for every target bucket tuple. The target's acceptance also depends only on its supports, so it is unchanged across these bucket choices. The B^s target bucket tuples therefore contribute equally to the numerator of (24). Summing over them gives a factor B^s , which cancels the bucket factor in $|\Omega_s| = B^s (\binom{t}{k} N)^s$. We can thus fix the target bucket tuple and normalize the remaining sum by $(\binom{t}{k} N)^s$.

We choose $(0, \dots, 0)$ as the fixed target bucket tuple. For a requirement subset $\mathcal{T}$, write the requirements in coordinate a as

$$T_a = \{(u, d) : (a, 0, u, d) \in \mathcal{T}\}.$$

Thus $|\mathcal{T}| = \sum_{a \in [s]} |T_a|$. When counting historical encodings, we include all B possible bucket labels in each coordinate: label 0 matches the fixed target bucket, while the other $B - 1$ labels select different child keys.

The remaining calculation uses three counts. The number of accepted support tuples, multiplied by $(BN)^s$, gives the total number of accepted historical encodings and hence the denominator of $p_{\mathcal{T}}$. The number of accepted historical encodings that supply none of the requirements in $\mathcal{T}$ gives its numerator. Finally, the number of accepted targets with fixed buckets whose requirement sets contain $\mathcal{T}$ gives the multiplicity of $(-1)^{|\mathcal{T}|} p_{\mathcal{T}}^q$ when we group the target sum by $\mathcal{T}$. We track the number of authentication nodes in these counts to enforce the shared budget L across coordinates.

Counting accepted support tuples. We first count supports according to the size of their Octopus authentication sets, following [1]. We index the counts by the binary node positions of Section 2.2. For a node at position p , extend these counts to its subtree: $A_p(\ell, m)$ counts selections of ℓ leaves that require m authentication nodes within that subtree. The empty selection gives $A_p(0, 0) = 1$. At a leaf position, we also have $A_p(1, 0) = 1$, and all other counts are zero.

Consider an internal node at position p , with children at $p0, p1$. If selected leaves occur in both child subtrees, both child roots can be reconstructed. If they occur in only one, the other child's root must be supplied as an authentication node. If neither child contains a selected leaf, no authentication node is added within the subtree at p . We record this additional node by

$$\delta(\ell_L, \ell_R) = \mathbf{1}[\text{exactly one of } \ell_L, \ell_R \text{ is positive}].$$

The counts at p are therefore

$$A_p(\ell, m) = \sum_{\substack{\ell_L + \ell_R = \ell \\ m_L + m_R + \delta(\ell_L, \ell_R) = m}} A_{p0}(\ell_L, m_L) A_{p1}(\ell_R, m_R). \quad (25)$$

Set $A(z) = \sum_{m \geq 0} A_\epsilon(k, m) z^m$. The coefficient of z^m in $A(z)^s$ counts support tuples with a total of m authentication nodes across the block. Summing the coefficients for $m \leq L$ gives the number of accepted support tuples $W_{s,L}$ and hence the block acceptance probability $\tau_{s,L}$:

$$W_{s,L} = [z^{\leq L}] A(z)^s, \quad \tau_{s,L} = \frac{W_{s,L}}{\binom{t}{k}^s}. \quad (26)$$

Each support tuple allows $(BN)^s$ assignments of bucket indices and depth tuples. This factor is the same for every support tuple and cancels in the acceptance probability.

Counting historical encodings. We now count the accepted historical encodings that supply none of the requirements in $\mathcal{T}$. We first count the encodings within each coordinate according to the number of child authentication nodes they require. Then, we impose the shared block budget on the total number of these nodes across coordinates.

For coordinate a , we distinguish whether the historical encoding selects the target bucket 0 or one of the other $B - 1$ buckets. We first consider the matching case. To supply none of the requirements in T_a , the encoding must, for each $(u, d_0) \in T_a$, either omit chain u or open it at a depth strictly less than d_0 .

For a node at position p , let $F_p(\ell, f, m; T_a)$ denote the number of encodings in its subtree that select ℓ leaves, have total opening depth f , require m authentication nodes, and supply none of the requirements in T_a . For a set T of requirement pairs, write $T|_p$ for the pairs $(u, d) \in T$ whose leaf positions $\text{leaf}(u)$ have prefix p . At a leaf u , omitting chain u gives the state $(0, 0, 0)$, while selecting it at opening depth d gives $(1, d, 0)$. If no requirement in T_a refers to u , any depth $0 \leq d \leq c$ is allowed. If $(u, d_0) \in T_a$, only depths $0 \leq d < d_0$ are allowed. Each allowed state has count one, and all other states have count zero.

At an internal position p , combine the counts for $p0$ and $p1$ as follows:

$$F_p(\ell, f, m; T_a) = \sum_{\substack{\ell_L + \ell_R = \ell, f_L + f_R = f \\ m_L + m_R + \delta(\ell_L, \ell_R) = m}} F_{p0}(\ell_L, f_L, m_L; T_a|_{p0}) \cdot F_{p1}(\ell_R, f_R, m_R; T_a|_{p1}). \quad (27)$$

At the root position ϵ , a child encoding must select k leaves with total depth E . Let $C_{T_a, m} = F_\epsilon(k, E, m; T_a)$ denote the number of these encodings that supply none of the requirements in T_a and require m authentication nodes. Algorithm 4 evaluates this recurrence in **history** mode.

Write $C_{T_a}(z) = \sum_m C_{T_a, m} z^m$. For any of the other $B - 1$ buckets, every support and depth tuple avoids the selected requirements. These buckets contribute

$(B-1)NA(z)$, while the target bucket contributes $C_{T_a}(z)$. Multiplying these coordinate polynomials and summing the coefficients of z^m for $m \leq L$ counts the historical encodings that avoid all requirements in $\mathcal{T}$ and pass the shared pruning budget. There are $W_{s,L}(BN)^s$ block encodings whose total number of child authentication nodes is at most L , so

$$p_{\mathcal{T}} = \frac{[z^{\leq L}] \prod_{a \in [s]} ((B-1)NA(z) + C_{T_a}(z))}{W_{s,L}(BN)^s}. \quad (28)$$

Counting target encodings. For the targets with fixed buckets, the term indexed by $\mathcal{T}$ appears once for each accepted target y satisfying $\mathcal{T} \subseteq \mathcal{U}^*(y)$. Its sign and probability factor $p_{\mathcal{T}}^q$ are the same in every occurrence. We therefore group these terms and count the accepted targets containing $\mathcal{T}$.

Let $B_{T_a,m}$ denote the number of child targets that contain every requirement in T_a and require m authentication nodes. At a leaf u with $(u, d_0) \in T_a$, the target must select chain u at depth d_0 , so the only allowed state is $(1, d_0, 0)$. At a leaf u with no requirement in T_a , the target may either omit chain u , giving $(0, 0, 0)$, or select it at any depth $0 \leq d \leq c$, giving $(1, d, 0)$. Each allowed leaf state has count one, and all other states have count zero. Let $F'_p(\ell, f, m; T_a)$ denote the corresponding subtree counts obtained using the target leaf rules. At an internal node, the target counts satisfy (27) with F replaced by F' . At the root position ϵ , we obtain $B_{T_a,m} = F'_\epsilon(k, E, m; T_a)$. Algorithm 4 evaluates this in **target** mode.

Collect the target counts in the polynomial

$$B_{T_a}(z) = \sum_m B_{T_a,m} z^m. \quad (29)$$

Multiplying these polynomials and summing the coefficients up to degree L gives the number of accepted block targets containing $\mathcal{T}$, with all target buckets fixed to 0:

$$B_{\mathcal{T}}^{\text{acc}} = [z^{\leq L}] \prod_{a \in [s]} B_{T_a}(z). \quad (30)$$

The coefficient cutoff in (28) enforces the historical budget; the cutoff in (30) enforces the target budget. The inclusion–exclusion sign remains explicit in the final sum.

Theorem 2 (Exact coverage of a pruning block). *Let q historical encodings be sampled independently from the accepted distribution of a block with $s \geq 1$ coordinates and $W_{s,L} > 0$. Let $\mathcal{M}$ be the family of all subsets of $\mathcal{U}^*(y)$, where y ranges over raw targets with all bucket labels fixed to 0. The probability that an independent uniform raw target is both accepted and covered is*

$$\gamma_{s,L}(q) = \frac{1}{\binom{t}{k} N^s} \sum_{\mathcal{T} \in \mathcal{M}} (-1)^{|\mathcal{T}|} B_{\mathcal{T}}^{\text{acc}} p_{\mathcal{T}}^q. \quad (31)$$

Algorithm 4 Counting historical and target encodings on a child tree

Require: A node position p in the fixed child tree; a set T of requirement pairs (u, d) in its subtree, with at most one depth in $\{0, \dots, c\}$ per leaf; mode $\text{mode} \in \{\text{history}, \text{target}\}$; parameters k, c, E .

Ensure: Integer count table $F(\ell, f, m)$ for the chosen mode, with $0 \leq \ell \leq k$ and $0 \leq f \leq E$.

```

1: function COUNTMARKED( $p, T, \text{mode}, k, c, E$ )
2:   Initialize an empty count table  $F$  with default value 0.
3:   if  $p$  is a leaf position, with leaf index  $u$  then
4:     if  $T = \emptyset$  then
5:        $F(0, 0, 0) \leftarrow 1$ .
6:       for  $d = 0, \dots, \min(c, E)$  do
7:          $F(1, d, 0) \leftarrow 1$ .
8:       end for
9:     else
10:      Let  $(u, d_0)$  be the unique requirement pair in  $T$ .
11:      if  $\text{mode} = \text{history}$  then
12:         $F(0, 0, 0) \leftarrow 1$ .
13:        for each integer  $d$  with  $0 \leq d < d_0$  and  $d \leq E$  do
14:           $F(1, d, 0) \leftarrow 1$ .
15:        end for
16:      else if  $\text{mode} = \text{target}$  and  $d_0 \leq E$  then
17:         $F(1, d_0, 0) \leftarrow 1$ .
18:      end if
19:    end if
20:    return  $F$ .
21:  end if
22:   $F_L \leftarrow \text{COUNTMARKED}(p_0, T|_{p_0}, \text{mode}, k, c, E)$ .
23:   $F_R \leftarrow \text{COUNTMARKED}(p_1, T|_{p_1}, \text{mode}, k, c, E)$ .
24:  for each  $(\ell_L, f_L, m_L)$  with  $F_L(\ell_L, f_L, m_L) > 0$  do
25:    for each  $(\ell_R, f_R, m_R)$  with  $F_R(\ell_R, f_R, m_R) > 0$  do
26:       $\ell \leftarrow \ell_L + \ell_R$ ;  $f \leftarrow f_L + f_R$ .
27:      if  $\ell \leq k$  and  $f \leq E$  then
28:         $m \leftarrow m_L + m_R + \delta(\ell_L, \ell_R)$ .
29:         $F(\ell, f, m) \leftarrow F(\ell, f, m) + F_L(\ell_L, f_L, m_L)F_R(\ell_R, f_R, m_R)$ .
30:      end if
31:    end for
32:  end for
33:  return  $F$ .
34: end function

```

Proof. By bucket symmetry, the B^s target bucket tuples contribute equally to (24). Fixing all target buckets to 0 therefore replaces the normalization $|\Omega_s|^{-1}$ by $\binom{t}{k}N^{-s}$. For each $\mathcal{T} \in \mathcal{M}$, there are $B_{\mathcal{T}}^{\text{acc}}$ targets y with all bucket labels fixed to 0 that pass pruning and satisfy $\mathcal{T} \subseteq \mathcal{U}^*(y)$. Each such target contributes $(-1)^{|\mathcal{T}|}p_{\mathcal{T}}^q$, so their contributions sum to $B_{\mathcal{T}}^{\text{acc}}(-1)^{|\mathcal{T}|}p_{\mathcal{T}}^q$. Summing over $\mathcal{T} \in \mathcal{M}$ and dividing by $\binom{t}{k}N^s$ gives (31).

This counting method applies to any block size, but the number of requirement sets in the exact sum grows with s . Using small blocks makes the exact calculation practical for the reported parameter sets.

Combining pruning blocks. Raw encodings are independent across blocks. Since each pruning test depends only on its own block, conditioning on acceptance preserves this independence. For q independent accepted historical encodings and an independent raw target, the probability that the target passes pruning and is covered is

$$\Gamma_q = \prod_{i=1}^g \gamma_{s_i, L_i}(q). \quad (32)$$

4.5 Coverage Analysis for SPHINCS+ Integration

Following PORS+FP and BPORS+FP [1,19], we average the fixed-key coverage probability over the number of signatures under a bottom FTS key in SPHINCS+. We use the joint address and FTS encoding interface of Section 3.5, which randomizes the message before the counter search.

For the coverage calculation, we consider Q signing operations with fresh, distinct inputs to H_{addr} and condition on every operation returning a signature. In the random-oracle model, each fresh counter query returns an independent uniform pair $(I_z, Y_z) \in [2^h] \times \Omega$. The pruning predicate depends only on Y_z . Rejection sampling therefore leaves the selected bottom address uniform and independent of the accepted FTS encoding, which follows $\mathcal{D}_{\text{acc}}$. The returned pairs are independent across signing operations.

Consequently, the number q of signatures under a fixed bottom FTS key follows $\text{Bin}(Q, 2^{-h})$. Given q , the corresponding historical encodings are independent samples from $\mathcal{D}_{\text{acc}}$, as required by Section 4.1. A fresh encoding query selects a uniform bottom address and an independent raw FTS encoding. Its probability of passing pruning and being covered by the history at that address is

$$\bar{\gamma}(Q, 2^h) = \sum_{q=0}^Q \binom{Q}{q} 2^{-hq} (1 - 2^{-h})^{Q-q} \Gamma_q, \quad (33)$$

where Γ_q is the block product in (32). All pruning blocks share the same bottom-key use count q , so their probabilities are multiplied at fixed q before averaging over this count. Section 5 uses this averaged coverage term for parameter selection. Counter exhaustion is bounded separately by $Q(1 - \tau_g)^R$ over Q signing operations.

The coverage term is the probability that a fresh raw encoding passes pruning and that all its required openings can be obtained by forward hashing from previous disclosures. The PRF distinguishing advantage and the probabilities of recovering undisclosed chain values or breaking Merkle authentication are separate primitive-security contributions to a complete security bound. These contributions are outside the coverage term analyzed here. For the surrounding message hashing, WOTS+, and hypertree authentication, we refer to the existing SPHINCS+ security analyses [5,9,3].

5 Evaluation

Few-time signatures are rarely used alone, and their ultimate purpose is to serve as building blocks for stateless hash-based signature schemes that support many signatures under one public key. Following prior work [1,19], we first evaluate BPORS-C+BFP as a replacement for the bottom few-time component of SPHINCS+, while keeping the surrounding parameters fixed.

5.1 Evaluation setup

We consider the six standard SLH-DSA configurations 128s/128f, 192s/192f, and 256s/256f [18], with a limit of $Q = 2^{64}$ signatures per key. For these three parameter families, we set the target coverage exponent to $\lambda = 128, 192, 256$, respectively, as in [19]. The corresponding hash lengths are $n = 16, 24, 32$ bytes. We keep the prescribed hypertree height h and number of layers d . All six configurations use the WOTS+ Winternitz parameter $w = 16$.

We also consider the three limited-use configurations proposed in the initial public draft of NIST SP 800-230 [16], with a limit of $Q = 2^{24}$ signatures per key. For the same three values of λ , the SPHINCS+ parameters (h, d, w) are $(22, 1, 4)$, $(21, 1, 8)$, and $(21, 1, 4)$, respectively.

We use the averaged coverage probability $\bar{\gamma}(Q, 2^h)$ defined in (33). For each fixed q , we bound the probability that each block is both accepted and covered using the bounds from Section 4, using exact block values where available. We then multiply these bounds across blocks and average the product over q . Following [1,19], we truncate the sum in (33) at $q = 200$. Since $\Gamma_q \leq 1$, the remaining contribution satisfies

$$\sum_{q=201}^Q \Pr[X = q] \Gamma_q \leq \sum_{q=201}^Q \Pr[X = q] = \Pr[X > 200],$$

where $X \sim \text{Bin}(Q, 2^{-h})$. We add this tail probability to the upper bound computed from the terms with $q \leq 200$.

Signature sizes are measured in bytes and include the pruning counter in the FTS component. Following [1,19], we measure computational cost by the number of hash and PRF evaluations. Signing costs are expectations over the counter searches. For BPORS-C+BFP, we report the bounds in Section 3.4 on expected

signing cost under full key regeneration and on verification cost, rounding signing values upwards. Complete-signature sizes and costs include the contributions of message randomization, WOTS+, and the hypertree. Appendix A.3 specifies the counting conventions.

Candidate parameter sets must have positive acceptance probability and a computed upper bound on (33) of at most $2^{-\lambda}$. We choose the counter width so that the probability of any counter exhaustion over Q signing operations is also at most $2^{-\lambda}$. For each setting, we select the smallest FTS signature among the evaluated parameter sets that satisfy these requirements and strictly improve on FORS in signature size, upper bound on expected signing cost, and verification count.

5.2 Standard SPHINCS+ parameter sets

Table 1 lists the parameters of FORS, PORS+FP, BPORS+FP, and BPORS-C+BFP for the six standard settings. For BPORS+FP and BPORS-C+BFP, the column b gives the outer-tree height, with $B = 2^b$ child keys per coordinate. The block-size tuple $(s_1, \dots, s_g)$ satisfies $\sum_{i=1}^g s_i = K$. Each entry in the budget tuple bounds the number of child authentication nodes in the corresponding block. The tuple (K) denotes one global budget. The column *Cover exp.* gives the negative base-two logarithm of the computed upper bound on (33).

Table 1. Parameters for the six standard SPHINCS+ settings: reference parameters (top) and BPORS-C+BFP parameters (bottom).

Setting	Common FORS					PORS+FP					BPORS+FP				
	h	d	w	k	a	t	k	$m_{\max}$	τ	K	b	t	k	$m_{\max}$	τ
128s	63	7	16	14	12	$14 \cdot 2^{12}$	14	135	$3.66 \cdot 10^{-4}$	5	6	$33 \cdot 2^2$	4	96	$2.37 \cdot 10^{-5}$
128f	66	22	16	33	6	$33 \cdot 2^6$	33	156	$1.85 \cdot 10^{-2}$	8	1	2^7	5	151	$4.39 \cdot 10^{-3}$
192s	63	7	16	17	14	$16 \cdot 2^{14} + 2^{12}$	17	200	$4.99 \cdot 10^{-4}$	6	8	2^7	5	141	$7.10 \cdot 10^{-6}$
192f	66	22	16	33	8	$33 \cdot 2^8$	33	217	$8.76 \cdot 10^{-3}$	9	3	$7 \cdot 2^4$	5	178	$1.01 \cdot 10^{-3}$
256s	64	8	16	22	14	$21 \cdot 2^{14} + 2^{13}$	22	260	$1.27 \cdot 10^{-4}$	7	7	$35 \cdot 2^3$	5	194	$3.66 \cdot 10^{-6}$
256f	68	17	16	35	9	$35 \cdot 2^9$	35	261	$2.58 \cdot 10^{-3}$	9	3	$29 \cdot 2^3$	5	218	$2.93 \cdot 10^{-4}$

BPORS-C+BFP										
Setting	K	b	t	k	c	E	Blocks	Budgets	τ	Cover exp.
128s	6	7	16	4	6	7	(3, 3)	(13, 14)	$4.54 \cdot 10^{-5}$	128.547
128f	11	2	26	3	2	2	(3, 4, 4)	(22, 28, 28)	$2.43 \cdot 10^{-3}$	128.044
192s	8	9	15	4	6	8	(2, 2, 2, 2)	(9, 9, 9, 9)	$4.93 \cdot 10^{-6}$	192.290
192f	14	4	16	3	4	4	(4, 5, 5)	(22, 27, 27)	$4.42 \cdot 10^{-3}$	192.347
256s	10	9	18	4	6	9	(2, 2, 2, 2, 2)	(11, 11, 11, 11, 12)	$4.65 \cdot 10^{-6}$	256.765
256f	17	5	12	3	5	5	(1, 4, 4, 4, 4)	(5, 17, 18, 18, 18)	$8.73 \cdot 10^{-4}$	257.209

All BPORS-C+BFP counters occupy 4 bytes. The BPORS+FP threshold $m_{\max}$ includes outer authentication paths; in the notation of Section 3.1, $m_{\max} = L + Kb$. The BPORS-C+BFP budgets count only child authentication nodes.

Table 2 reports complete-signature sizes and computational costs, together with the FTS signature sizes. Percentages are relative to FORS in the same setting. Across the six settings, the selected FTS signatures are 39.4%–48.8% shorter than FORS, giving an 8.3%–18.1% reduction in complete-signature size. The reduction in complete-signature size is smaller because the WOTS+ and hypertree components are fixed.

Table 2. Complete-signature sizes and hash counts for the standard parameter sets.

Setting	FTS scheme	Total bytes	FTS bytes	Expected sign calls	Verify hashes
128s	FORS	7,856	2,912	2,186,222	2,214
	PORS+FP	7,332 (-6.7%)	2,388 (-18.0%)	2,188,971 (+0.1%)	2,195 (-0.9%)
	BPORS+FP	6,804 (-13.4%)	1,860 (-36.1%)	2,183,140 (-0.1%)	2,168 (-2.1%)
	BPORS-C+BFP	6,436 (-18.1%)	1,492 (-48.8%)	2,146,809 (-1.8%)	2,186 (-1.3%)
128f	FORS	17,088	3,696	105,196	6,481
	PORS+FP	16,420 (-3.9%)	3,028 (-18.1%)	105,282 (+0.1%)	6,472 (-0.1%)
	BPORS+FP	16,452 (-3.7%)	3,060 (-17.2%)	105,257 (+0.1%)	6,481 (+0.0%)
	BPORS-C+BFP	15,524 (-9.2%)	2,132 (-42.3%)	105,015 (-0.2%)	6,428 (-0.8%)
192s	FORS	16,224	6,120	3,767,275	3,183
	PORS+FP	15,316 (-5.6%)	5,212 (-14.8%)	3,732,432 (-0.9%)	3,162 (-0.7%)
	BPORS+FP	14,212 (-12.4%)	4,108 (-32.9%)	3,662,422 (-2.8%)	3,129 (-1.7%)
	BPORS-C+BFP	13,468 (-17.0%)	3,364 (-45.0%)	3,687,458 (-2.1%)	3,157 (-0.8%)
192f	FORS	35,664	7,128	169,260	9,363
	PORS+FP	34,540 (-3.2%)	6,004 (-15.8%)	169,406 (+0.1%)	9,349 (-0.1%)
	BPORS+FP	33,892 (-5.0%)	5,356 (-24.9%)	169,123 (-0.1%)	9,334 (-0.3%)
	BPORS-C+BFP	32,716 (-8.3%)	4,180 (-41.4%)	169,250 (-0.0%)	9,325 (-0.4%)
256s	FORS	29,792	10,560	3,280,869	4,692
	PORS+FP	28,260 (-5.1%)	9,028 (-14.5%)	3,264,215 (-0.5%)	4,666 (-0.6%)
	BPORS+FP	26,564 (-10.8%)	7,332 (-30.6%)	3,225,264 (-1.7%)	4,626 (-1.4%)
	BPORS-C+BFP	25,188 (-15.5%)	5,956 (-43.6%)	3,243,819 (-1.1%)	4,669 (-0.5%)
256f	FORS	49,856	11,200	345,839	9,549
	PORS+FP	48,132 (-3.5%)	9,476 (-15.4%)	346,261 (+0.1%)	9,530 (-0.2%)
	BPORS+FP	47,076 (-5.6%)	8,420 (-24.8%)	345,628 (-0.1%)	9,507 (-0.4%)
	BPORS-C+BFP	45,444 (-8.8%)	6,788 (-39.4%)	345,467 (-0.1%)	9,531 (-0.2%)

Hash counts cover the complete signature, including the FTS encoding query; percentages are relative to FORS. BPORS-C+BFP signing entries bound the expected cost. Verification uses the WOTS+ approximation of Appendix A.3.

For example, in the 128f setting, the FTS signature decreases from 3,696 to 2,132 bytes, and the complete signature decreases from 17,088 to 15,524 bytes. For the complete scheme, the expected signing cost is at most 105,015 calls, compared with 105,196 when using FORS, and the verification cost decreases from 6,481 to 6,428 calls.

5.3 Parameter sets for limited signing workloads

Table 3 lists the limited-use parameters, and Table 4 reports the corresponding complete-signature costs and FTS signature sizes.

Table 3. Parameters for the limited-use settings: reference parameters (top) and BPORS-C+BFP parameters (bottom).

Setting	Common			FORS		PORS+FP				BPORS+FP						
	h	d	w	k	a	t	k	$m_{\max}$	τ	K	b	t	k	$m_{\max}$	τ	
128-24	22	1	4	6	24	$3 \cdot 2^{25}$	6	110	$1.59 \cdot 10^{-7}$	2	13	$23 \cdot 2^5$	7	77	$2.20 \cdot 10^{-8}$	
192-24	21	1	8	9	25	$9 \cdot 2^{25}$	9	183	$6.84 \cdot 10^{-8}$	2	13	$17 \cdot 2^9$	8	134	$1.20 \cdot 10^{-8}$	
256-24	21	1	4	12	25	$3 \cdot 2^{27}$	12	253	$1.09 \cdot 10^{-7}$	3	13	$27 \cdot 2^7$	8	185	$1.29 \cdot 10^{-8}$	

BPORS-C+BFP											
Setting	K	b	t	k	c	E	Blocks	Budgets	τ	Cover	exp.
128-24	2	15	152	7	16	32	(2)	(27)	$2.52 \cdot 10^{-8}$	128.019	
192-24	3	16	152	7	16	32	(3)	(47)	$6.03 \cdot 10^{-9}$	192.001	
256-24	5	17	50	6	14	25	(5)	(45)	$1.78 \cdot 10^{-9}$	256.125	

Counter lengths in bytes: 128-24: 4, 192-24: 5, 256-24: 5. The BPORS+FP threshold $m_{\max}$ includes outer authentication paths; in the notation of Section 3.1, $m_{\max} = L + Kb$. The BPORS-C+BFP budgets count only child authentication nodes.

The selected FTS signatures are 48.7%–52.5% shorter than FORS, reducing complete-signature size by 32.5%–36.5%. The larger reduction in complete-signature size reflects the greater share contributed by the FTS in these settings. For example, at the 128-bit coverage target, the FTS signature decreases from 2,400 to 1,140 bytes and the complete signature from 3,856 to 2,596 bytes.

Table 4. Complete-signature sizes and hash counts for limited signing workloads. Reference sizes and signing totals are from [19, Table 5].

Setting	FTS scheme	Total bytes	FTS bytes	Expected sign calls	Verify hashes
128-24	FORS	3,856	2,400	1,451,229,180	311
	PORS+FP	3,316 (-14.0%)	1,860 (-22.5%)	1,457,500,226 (+0.4%)	283 (-9.0%)
	BPORS+FP	2,916 (-24.4%)	1,460 (-39.2%)	1,230,807,030 (-15.2%)	266 (-14.5%)
	BPORS-C+BFP	2,596 (-32.7%)	1,140 (-52.5%)	1,378,187,117 (-5.0%)	309 (-0.6%)
192-24	FORS	7,752	5,616	2,034,237,433	526
	PORS+FP	6,748 (-13.0%)	4,612 (-17.9%)	2,048,864,458 (+0.7%)	493 (-6.3%)
	BPORS+FP	5,740 (-26.0%)	3,604 (-35.8%)	1,639,311,819 (-19.4%)	458 (-12.9%)
	BPORS-C+BFP	4,925 (-36.5%)	2,789 (-50.3%)	1,861,952,200 (-8.5%)	523 (-0.6%)
256-24	FORS	14,944	9,984	2,327,838,710	602
	PORS+FP	13,444 (-10.0%)	8,484 (-15.0%)	2,336,980,369 (+0.4%)	567 (-5.8%)
	BPORS+FP	11,652 (-22.0%)	6,692 (-33.0%)	1,452,127,379 (-37.6%)	523 (-13.1%)
	BPORS-C+BFP	10,085 (-32.5%)	5,125 (-48.7%)	2,237,453,399 (-3.9%)	601 (-0.2%)

Hash counts cover the complete signature, including the FTS encoding query; percentages are relative to FORS. BPORS-C+BFP signing entries bound the expected cost. Verification uses the WOTS+ approximation of Appendix A.3.

6 Discussion and Conclusion

In this work, we propose BPORS-C+BFP, which extends BPORS+FP [19] with hash-chain openings and blockwise forced pruning (BFP). Replacing leaf secrets with hash-chain openings provides additional encoding choices while each selected chain still contributes one hash value to the signature. Repeated signatures may disclose different positions on the same chain, so the coverage analysis must account for their combined effect. We derive a general coverage bound that is inexpensive to evaluate and develop an exact counting method that accounts for support overlap and authentication budgets. BFP limits the number of coordinates involved in each joint counting problem, reducing the computational cost of the exact analysis.

We evaluate BPORS-C+BFP as a replacement for FORS in the standard and limited-use SPHINCS+ parameter sets. In all evaluated settings, BPORS-C+BFP gives smaller signatures, lower upper bounds on expected signing cost, and lower verification costs than FORS. With the outer parameters unchanged, the complete signature size is reduced by up to 18.1% and 36.5% in the standard and limited-use settings, respectively.

Acknowledgments

The main results of this work were developed primarily by the authors. However, we also engaged in extensive discussions with GPT-5.6 Sol Pro and GPT-6 Astra to validate the result and assist in drafting the manuscript. The authors independently reviewed all AI-assisted text and retain full responsibility for the paper.

References

1. Abri, M., Katz, J.: Shorter hash-based signatures using forced pruning. Cryptology ePrint Archive, Paper 2025/2069 (2025), <https://eprint.iacr.org/2025/2069>, full version; published in CRYPTO 2026
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
3. 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, Part IV. Lecture Notes in Computer Science, vol. 15487, pp. 35–67. Springer (2024). https://doi.org/10.1007/978-981-96-0894-2_2
4. 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, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 368–397. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_15, <https://eprint.iacr.org/2014/795>

5. 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>
6. Buchmann, J., Dahmen, E., Hülsing, A.: XMSS: A practical forward secure signature scheme based on minimal security assumptions. In: Post-Quantum Cryptography. Lecture Notes in Computer Science, vol. 7071, pp. 117–129. Springer (2011), <https://eprint.iacr.org/2011/484>
7. 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>, <https://csrc.nist.gov/pubs/sp/800/208/final>
8. Ethereum Foundation Post-Quantum Team: Post-quantum ethereum (2026), <https://pq.ethereum.org/>, accessed September 24, 2026
9. Hülsing, A., Kudinov, M.: Recovering the tight security proof of SPHINCS+. In: Advances in Cryptology—ASIACRYPT 2022, Part IV. Lecture Notes in Computer Science, vol. 13794, pp. 3–33. Springer (2022). https://doi.org/10.1007/978-3-031-22972-5_1
10. Khovratovich, D., Kudinov, M., Wagner, B.: At the top of the hypercube—better size-time tradeoffs for hash-based signatures. Cryptology ePrint Archive, Paper 2025/889 (2025), <https://eprint.iacr.org/2025/889>, full version of CRYPTO 2025 paper
11. Kölbl, S., Philipoom, J.: A note on SPHINCS+ parameter sets. Cryptology ePrint Archive, Paper 2022/1725 (2022), <https://eprint.iacr.org/2022/1725>
12. Lamport, L.: Constructing digital signatures from a one way function. Tech. Rep. SRI-CSL-98, SRI International (1979)
13. McGrew, D., Curcio, M., Fluhrer, S.: Leighton-micali hash-based signatures. RFC 8554, Internet Research Task Force (2019). <https://doi.org/10.17487/RFC8554>, <https://www.rfc-editor.org/rfc/rfc8554>
14. 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
15. 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
16. Moody, D., Dang, Q.: Additional SLH-DSA parameter sets for limited signature use cases. Special Publication 800-230, National Institute of Standards and Technology (2026). <https://doi.org/10.6028/NIST.SP.800-230.ipd>, <https://csrc.nist.gov/pubs/sp/800/230/ipd>, initial Public Draft
17. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: Proceedings of the 21st Annual ACM Symposium on Theory of Computing. pp. 33–43. ACM (1989). <https://doi.org/10.1145/73007.73011>
18. National Institute of Standards and Technology: Stateless hash-based digital signature standard. FIPS 205, National Institute of Standards and Technology (2024). <https://doi.org/10.6028/NIST.FIPS.205>, <https://csrc.nist.gov/pubs/fips/205/final>
19. Ren, C., Zhang, K., Yu, H.: Shorter hash-based signatures via bucket thinning. Cryptology ePrint Archive, Paper 2026/1328 (2026), <https://eprint.iacr.org/2026/1328>

20. Reyzin, L., Reyzin, N.: Better than BiBa: Short one-time signatures with fast signing and verifying. Cryptology ePrint Archive, Paper 2002/014 (2002), <https://eprint.iacr.org/2002/014>
21. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing. pp. 387–394. ACM (1990). <https://doi.org/10.1145/100216.100269>
22. Zhang, K., Cui, H., Yu, Y.: Revisiting the constant-sum Winternitz one-time signature with applications to SPHINCS+ and XMSS. Cryptology ePrint Archive, Paper 2023/850 (2023), <https://eprint.iacr.org/2023/850>, full version of CRYPTO 2023 paper

A Encoding and Cost Conventions

A.1 Uniform sampling of subsets and depth tuples

Within a selected child key, an encoding specifies a set of chain indices $S \in \binom{[t]}{k}$ and an ordered tuple of opening depths $e \in \mathcal{L}_{k,c,E}$. Writing $S = \{u_1 < \dots < u_k\}$, the entry e_i gives the opening depth on chain u_i . We describe how to sample these two objects using independent uniform integer ranks.

Subset decoding. We order the k -element subsets of $[t]$ lexicographically by their increasing lists of indices. Given a uniform rank $r_S \in [\binom{t}{k}]$, start with an empty list and set $r = r_S$. Suppose j indices remain to be selected. A candidate next index $u \in [t]$ must be larger than the last selected index, if any, and satisfy $u \leq t - j$. We consider these candidates in increasing order. For a fixed candidate u , the remaining $j - 1$ indices must be chosen from $\{u + 1, \dots, t - 1\}$. Hence the group of subsets whose next index is u has size $\binom{t-u-1}{j-1}$. If r is at least this size, we skip the group and subtract its size from r . Otherwise, we select u and continue with the remaining $j - 1$ indices. The updated rank identifies the desired completion of the selected prefix.

Depth-tuple decoding. Independently, choose a uniform rank $r_e \in [N_{k,c,E}]$. This rank selects an ordered tuple from $\mathcal{L}_{k,c,E}$, with all such tuples listed in lexicographic order. At position i , let f be the depth sum still to be assigned. Choosing $e_i = d$ leaves $k - i$ entries with sum $f - d$, so the number of possible completions is $N_{k-i,c,f-d}$. We consider candidate values $d = 0, \dots, c$ in increasing order at each position and use these counts to locate the group containing the current rank. Algorithm 5 implements this procedure [22], using the counts from Section 2.4. As defined there, infeasible sums have count zero.

Each k -element subset of $[t]$ corresponds to exactly one rank in $[\binom{t}{k}]$. Likewise, each depth tuple in $\mathcal{L}_{k,c,E}$ corresponds to exactly one rank in $[N_{k,c,E}]$. Choosing the two ranks independently and uniformly therefore produces a uniform pair $(S, e) \in \binom{[t]}{k} \times \mathcal{L}_{k,c,E}$ before forced pruning.

Algorithm 5 Lexicographic unranking of constant-sum tuples

Require: Integers $k \geq 1$, $c \geq 0$, $0 \leq E \leq kc$, an input rank $r \in [N_{k,c,E}]$, and precomputed counts $N_{j,c,f}$.

Ensure: The tuple $e \in \mathcal{L}_{k,c,E}$ at the input rank in lexicographic order.

```

1: function UNRANKCONSTANTSUM( $k, c, E, r$ )
2:    $f \leftarrow E$ .
3:   for  $i = 1, \dots, k$  do
4:      $d \leftarrow 0$ .
5:     while  $r \geq N_{k-i,c,f-d}$  do
6:        $r \leftarrow r - N_{k-i,c,f-d}$ .
7:        $d \leftarrow d + 1$ .
8:     end while
9:      $e_i \leftarrow d$ .
10:     $f \leftarrow f - d$ .
11:  end for
12:  return  $(e_1, \dots, e_k)$ .
13: end function

```

A.2 Trees and authentication data

We use the tree and binary node positions defined in Section 2.2. For a nonempty support $S \subseteq [t]$, the call `OctopusPositions( $\epsilon, S$ )` in Algorithm 6 lists the authentication-node positions required by Octopus [2], processing the left subtree before the right subtree. At a recursive call with position p , the set S_p contains the selected leaf indices whose positions have prefix p . Within the algorithm, $\parallel$ denotes list concatenation.

The traversal outputs the binary position of each required authentication node once. Verification follows the same order, using the supplied values at these positions and reconstructed chain endpoints at the selected leaf positions.

A.3 Cost conventions for the comparison tables

For the fixed SPHINCS+ components, write $u = h/d$ and let ℓ_W be the number of WOTS+ chains, including the checksum chains, for hash length n and Winternitz parameter w [5]. The contributions of message randomization, WOTS+, and the hypertree to signature size, signing cost, and verification cost are

$$\begin{aligned}
S_{\text{wrap}} &= n(1 + d\ell_W + h), \\
W_{\text{wrap}} &= d[2^u(\ell_W w + 1) + (2^u - 1)] + 2, \\
V_{\text{wrap}} &= d(\ell_W w/2 + 1 + u) + 1.
\end{aligned} \tag{34}$$

Following [1,19], we use $\ell_W w/2$ as an approximation to the number of chain evaluations in each WOTS+ verification. The reported complete verification costs combine this approximation with the FTS verification upper bounds from Section 3.4. The leading n bytes are the message randomizer. Signing regenerates the WOTS+ public keys and internal nodes of the d XMSS subtrees.

Algorithm 6 Octopus authentication-node positions in canonical order

Require: A node position p in the fixed tree and the selected leaf indices S_p in its subtree.

Ensure: An ordered list of binary authentication-node positions for that subtree.

```

1: function OCTOPUSPOSITIONS( $p, S_p$ )
2:   if  $S_p = \emptyset$  then
3:     return  $[p]$ .
4:   end if
5:   if  $p$  is a leaf position then
6:     return  $[\ ]$ .
7:   end if
8:    $S_L \leftarrow \{u \in S_p : \text{leaf}(u) \text{ has prefix } p0\}$ ;  $S_R \leftarrow S_p \setminus S_L$ .
9:    $P_L \leftarrow \text{OCTOPUSPOSITIONS}(p0, S_L)$ .
10:   $P_R \leftarrow \text{OCTOPUSPOSITIONS}(p1, S_R)$ .
11:  return  $P_L \parallel P_R$ .
12: end function

```

The FTS signing counts for BPORS-C+BFP and FORS include one final coordinate-root compression. The bounds for BPORS-C+BFP follow Section 3.4. In the integrated construction, each counter attempt contributes one H_{addr} evaluation to the FTS signing count, and FTS verification includes one H_{addr} evaluation. These joint queries already supply both the bottom address and the FTS encoding. Equation (34) separately includes two calls for message randomization and hashing.

For the standard and limited-use SPHINCS+ settings, we reproduce both the complete-signature signing and verification totals in [19, Tables 3 and 5] for FORS, PORS+FP, and BPORS+FP without changing the authors' counting conventions. The counts for BPORS-C+BFP are computed from Section 3.4 and (34).