SHRINCS Stateless Parameter Search Methodology

July 16, 2026

Contents

1 Introduction

In the case of Bitcoin, hash-based signature schemes are the most compatible among all quantum-resistant signature algorithms. Still, the smallest stateless hash-based signature proposed by NIST (SLH-DSA-128s) is *7,856* bytes [3], which is more than 100× larger than the current Schnorr signature. The question is whether we can have something smaller and more practical without significant deviation from the standard.

SHRINCS [4] proposes a hybrid signature consisting of a stateful and a stateless part, with signature size ranging from *324* bytes (stateful) to X bytes (stateless). This document attempts to answer the question of what the value of X might be while keeping other signature metrics (KeyGen, SigGen, SigVer, etc.) practical.

In this document, we present the methodology for searching parameters of SLH-DSA-compatible signatures: how changes in the hyper-tree structure and other parameters affect the signature metrics. As mentioned above, we deliberately stay within very conservative bounds and avoid any changes to the signature sub-algorithms in order to keep the construction as compatible as possible with the SLH-DSA standard.

Along the way, we will occasionally reference promising proposals such as hyper-tree pruning [1]

and the +C compression technique [2], which allow for even better shrinking results. Although some of them can be considered partially compatible, we do not rely on them in the evaluation framework. We still think it is useful to outline the potential of such techniques and to let the reader tweak the parameters accordingly.

The main contributions of this report are (1) a sweep script, (2) the filtering pipeline, (3) a recommended candidate tuple, and (4) a framework itself.

2 Parameters Description

2.1 Initial Bounds

As mentioned in the introduction, our objective is to find parameters that make the signature "better" than SLH-DSA. Before we can do so, we have to fix the strict boundaries that we will not cross:

1. We obviously do not want a signature larger than SLH-DSA - otherwise, why bother in the first place?
2. For Bitcoin, we require at least a 128-bit security level. None of the parameters we select may degrade security below this floor.
3. At least $q_s = 2^{40}$ supported signatures. SLH-DSA-128s defines the maximum number of signatures as $q_s = 2^{64}$, which is generous. For Bitcoin, we assume that $q_s = 2^{40}$ is a practical compromise, comfortably covering the foreseeable needs of on-chain operations and the Lightning Network.
4. Sub-algorithms, hash, and addressing functions remain unchanged.

2.2 SLH-DSA Parameters

SLH-DSA uses a hyper-tree of eXtended Merkle Signature Schemes (XMSS) to authenticate the underlying one-time and few-time signature keys. We formally parameterize each candidate configuration using the tuple (h, d, k, a, w), where each variable controls a distinct geometric trade-off within the overall structure.

Hyper-Tree Geometry (h and d). The hyper-tree configuration determines the capacity and layer-traversal costs of the signature scheme.

- h is the total height of the multi-layered hyper-tree. This dictates the maximum number of few-time signature instances supported at the base layer, yielding a total capacity of 2^h unique leaf nodes.
- d is the number of independent tree layers comprising the hyper-tree.

To ensure a symmetric construction across all internal boundaries, our search pipeline requires that the total height h be perfectly divisible by the layer count d. Consequently, each individual XMSS tree layer has a uniform height h':

$$h' = \frac{h}{d}.$$

Adjusting the ratio between h and d governs the balance between signature size and key-generation cost. Reducing the number of layers d shrinks the total number of intermediate public keys and authentication paths in the signature payload, saving valuable bytes. However, this exponentially increases the height h' of each constituent tree, requiring significantly more hash operations to compute the root node during key generation.

Few-Time Signature Layer (k and a). The lowest layer of the hyper-tree authenticates the Forest of Random Subsets (FORS) few-time signature scheme, which directly signs the message digest.

- k is the number of independent, parallel Merkle trees in the FORS layer.
- a is the height of each individual FORS tree, so that each constituent tree contains exactly $t = 2^a$ leaves.

Increasing k and a strengthens the scheme against multi-target subset-forgery attacks and allows the structure to support larger capacities safely. However, expanding these dimensions directly increases the byte footprint of the appended authentication paths.

One-Time Signature Base (w). Internal tree roots are signed using the Winternitz One-Time Signature Plus (WOTS+) scheme. The base parameter w defines the digit representation used to split the message digest during signing. Our framework evaluates candidate configurations across the standard baseline settings $w \in \{16, 32, 256\}$.

Selecting a larger base reduces the number of parallel hash chains ℓ, which in turn reduces the overall WOTS+ payload size. This spatial efficiency comes at the cost of local compute: larger base parameters require longer hash chains, resulting in higher execution costs during key and signature generation.

We sweep the ranges as follows:

$$(h, d, k, a, w) \in \{40, \ldots, 50\} \times \{2, \ldots, 25\} \times \{6, \ldots, 24\} \times \{8, \ldots, 20\} \times \{16, 32, 256\}.$$

The lower bound $k \geq 6$ discards only insecure configurations: with $a \leq 20$ and $h \leq 50$, any $k \leq 5$ falls short of the 128-bit security floor at $q_s = 2^{40}$, so smaller values of k never survive the security filter. The same grid is used by the sweep script (`slhdsa-2to40.sage`) and the interactive explorer, so the counts reported below are reproducible in both.

The SLH-DSA-128s baseline tuple (63, 7, 14, 12, 16) sits outside this range and is included for comparison

3 Search Methodology

3.1 Initial Search Conditions

Searching for the best parameters requires defining the conditions that the chosen parameter set must satisfy. Consequently, those conditions must rely on metrics that depend on the parameters. We use the following:

- **Signature size**: the size of the signature in bytes;
- **Key generation cost**: the number of SHA-256 compressions during the key-generation phase;
- **Signature generation cost**: the number of compressions executed during signing;
- **Signature verification cost**: the number of compressions during verification;
- **Compressions per byte**: the ratio of verification cost to signature size.

We choose SLH-DSA-128s as the baseline for the search; this gives us an intuitive way to compare custom parameter sets against the standardized scheme on equal footing.

3.2 Hash-Cost Convention

All compression counts in this report use the *midstate-cached* convention. The SHA-2 instantiation of FIPS 205 zero-pads $\mathsf{PK.seed}$ to a full 64-byte block precisely so that implementations can precompute the SHA-256 compression state after that block once per keypair and reuse it for every tweakable-hash call. Under this convention, each call therefore costs only its *marginal* compressions over the compressed address ADRS_c (22 bytes) plus the payload and padding: a WOTS chain step or FORS leaf ($22 + 16$ bytes) costs one compression, and — importantly — a two-child Merkle node hash ($22 + 32$ bytes $= 497$ bits) also costs exactly *one* compression. The message hash H_{msg} and $\mathsf{PRF}_{\mathsf{msg}}$ retain their uncached cost of two compressions, since their inputs begin with the randomizer R or are HMAC-structured, so the midstate trick does not apply to them.

Under this convention the SLH-DSA-128s baseline costs are: KeyGen 292,351, SigGen 2,218,483, and SigVer 2,155 (average) / 3,891 (worst-case) compressions. The original convention of the accompanying scripts, in which every call pays for its full input (a node hash then costs two compressions, giving the baseline SigVer 2,387), remains available via the `HASH_CONVENTION=uncached` environment flag.

3.3 Toolset

The methodology described here uses a set of SageMath scripts available at https://github.com/BlockstreamResearch/SPHINCS-Parameters.

3.4 Sweep Strategy

We start by enumerating all parameter tuples (h, d, k, a, w). Applying the initial bounds (security level, size $\leq$ SLH-DSA, at least 2^{40} supported signatures) reduces the search space from *25,935* candidates to *9,182* - a 65% cut. The remaining space is still too large to inspect by hand, so we need additional filters.

We introduce the following filters[1]:

- **Bound on signature size**: $\mathsf{target.size} \leq c_{\text{size}} \cdot \mathsf{std.size}$, where c_{size} is a size coefficient ($0 \leq c_{\text{size}} \leq 1$).
- **Bound on key-generation cost**: $\mathsf{target.KeyGen} \leq c_{\text{kg}} \cdot \mathsf{std.KeyGen}$, where $c_{\text{kg}} > 0$.
- **Bound on signature-generation cost**: $\mathsf{target.SigGen} \leq c_{\text{sg}} \cdot \mathsf{std.SigGen}$, where $c_{\text{sg}} > 0$.
- **Bound on signature-verification cost**: $\mathsf{target.SigVer} \leq c_{\text{sv}} \cdot \mathsf{std.SigVer}$, where $c_{\text{sv}} > 0$.
- **Bound on the compressions-per-byte ratio**: $\mathsf{target.C/B} \leq c_{\text{cb}} \cdot \mathsf{std.C/B}$, where $c_{\text{cb}} > 0$.

Note: All verification costs reported in this section are *average-case*: they reflect the expected number of compressions for a uniformly random message. The *worst-case* verification cost is approximately 1.81× the average for the standard parameter set (3,891 vs. 2,155 compressions): the WOTS chain-walk component alone grows by up to ≈1.94×, while the FORS and authentication-path costs are message-independent. The worst case depends solely on the message being signed — it cannot be influenced by the verifier. We also use an *average-case* signing cost, while the *worst-case* does not significantly increase the number of compressions required to generate a signature.

[1]The coefficient for each filter, except the signature size, may be greater than 1 (meaning we accept some regression on that metric in exchange for gains elsewhere).

These filters can be combined and parameterized in different ways to produce different result sets. Below, we consider several cases to illustrate how the choice of filter parameters affects the resulting candidate set.

3.4.1 Case 1: Best in Class

In this case, we find the most efficient signature for a single metric at a time, leaving the other metrics unbounded. The results are shown in Table 1.

Table 1: Absolute Global Extremes

Award Category	Tuple (h, d, k, a, w)	Size (B)	VS std, %	KeyGen (C)	VS std, ×	SigGen (C)	VS std, ×	SigVer (C)	VS std, ×	ρ (C/B)
STANDARD Baseline	$(63, 7, 14, 12, 16)$	7856	+0.0%	2.92×10^5	1.00	2.22×10^6	1.00	2155	1.00	0.27
Smallest Signature	$(50, 2, 6, 20, 256)$	3408	−56.6%	1.55×10^{11}	5.30×10^5	3.10×10^{11}	1.40×10^5	4780	2.22	1.40
Fastest Keygen	$(40, 10, 7, 20, 32)$	7488	−4.7%	1.45×10^4	0.05	2.22×10^7	9.99	4612	2.14	0.62
Fastest Signing	$(40, 8, 23, 8, 32)$	7552	−3.9%	2.90×10^4	0.10	2.49×10^5	0.11	3792	1.76	0.50
Fastest Verification	$(50, 2, 6, 20, 16)$	3952	−49.7%	1.92×10^{10}	6.55×10^4	3.83×10^{10}	1.73×10^4	724	0.34	0.18
Best Density Ratio (ρ)	$(40, 2, 18, 20, 16)$	7824	−0.4%	5.99×10^8	2.05×10^3	1.25×10^9	5.65×10^2	969	0.45	0.12

Table 1 confirms what one would suspect: each metric can be minimized impressively in isolation, but only at a brutal cost to the others, leaving these parameter sets unusable in practice. We therefore consider a different strategy.

3.4.2 Case 2: Balanced Metrics

In the next experiment, we look for "average" candidates whose metrics all stay below a uniform threshold $X \cdot \mathsf{std}$ (the filter is applied to every metric, with the exception of the maximum signature size $\leq \mathsf{std}$). By gradually increasing X, we obtain the following picture.

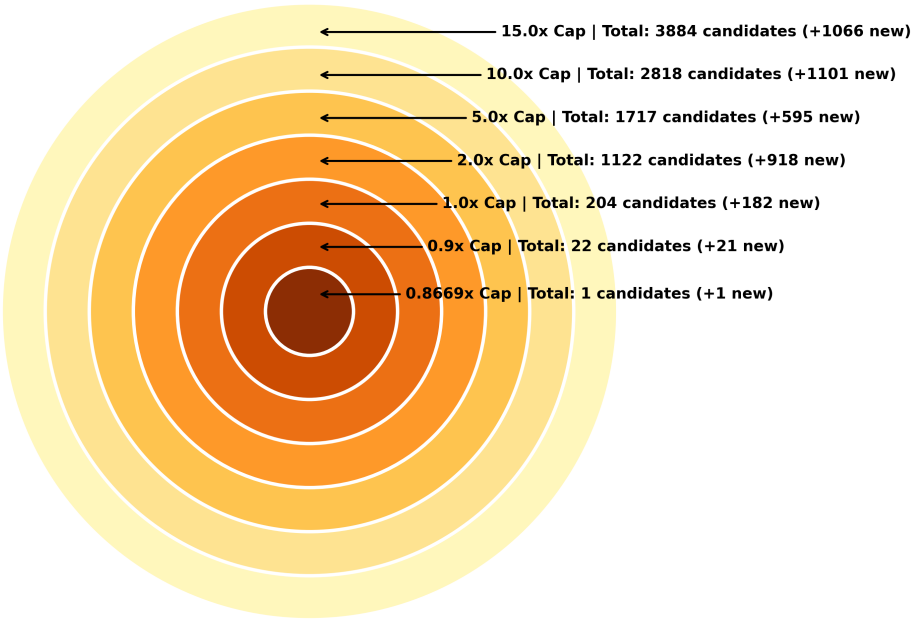


Figure 1: Global Minimax Expansion: Concentric Euler diagram illustrating the volume of candidates unlocked at expanding uniform multiplier thresholds.

Naturally, the smaller the multiplier X, the fewer candidates survive the filter. A clear winner emerges at $X \approx 0.8669$; let us see how compelling it actually is (Table 2).

Table 2: Performance Comparison between the Standard Baseline and the Balanced Candidate

Metric	**Standard Baseline**	**Optimal Candidate**	**Ratio**
Parameters (h, d, k, a, w)	$(63, 7, 14, 12, 16)$	$(40, 5, 19, 10, 16)$	–
Signature Size (Bytes)	7,856	6,800	0.8656×
Key Generation Cost	292,351	146,175	0.5000×
Signing Cost	2,218,483	789,234	0.3558×
Verification Cost	2,155	1,617	0.7503×
Compression per Byte (ρ)	0.2743	0.2377	0.8669×

One might call this a reasonable result. We are less impressed: a 12% saving on signature size is not particularly attractive on its own (the goal is to materially close the gap with Schnorr).

3.4.3 Case 3: Adaptive Search

The strategy is straightforward:

1. Decide which sacrifices you are willing to make on each metric.
2. Find the best candidate among those that remain.

For step (1) we apply the filters introduced in Section 3.4. This subsection is dedicated to step (2).

Each candidate is uniquely defined by its tuple of metric values, so the search for an optimal candidate can be cast as finding the point closest to the origin in a five-dimensional space (analogous to a shortest-vector problem, but mercifully confined to a small set of survivors), where each axis represents one of our metrics. That is almost the whole story.

To build intuition, suppose we care about only two metrics - signature size and signature generation cost. We require the signature size to be no larger than that of SLH-DSA, and we are willing to tolerate a signing cost of at most 2× the standard. In this case all admissible candidates fit inside the green rectangle in Figure 2, which makes brute-force inspection easy: the best candidate is simply the point closest to the origin $(0, 0)$.

We cannot directly visualize a 5-dimensional space, but we can show the filtering process in a different way.

First, we scatter the unbounded candidate set (subject only to the 128-bit security condition) on a plane, with positions chosen randomly - the layout carries no meaning, and each point is uniquely identified by its parameter tuple. After initialization, we apply the signature-size filter, visualized as a coloured "sieve". We then apply each remaining filter in turn until the last one.

The resulting set contains the candidates that survived every filter; this is the pool in which we search for the best one.

As an example, consider the following filter coefficients:

$$X_{\text{size}} = 0.85, \quad X_{\text{kg}} = 1.5, \quad X_{\text{sg}} = 2, \quad X_{\text{sv}} = 0.8, \quad X_{\text{cb}} = 1.$$

The filtering process is shown in Figure 3.

We start with *19,437* candidates. The signature-size filter trims the pool to *6,141*; the next filter (KeyGen) cuts it by more than 7×. Each subsequent filter prunes further, leaving *61* final candidates for inspection.

With the final 61 candidates in hand, we are ready to pick a winner.

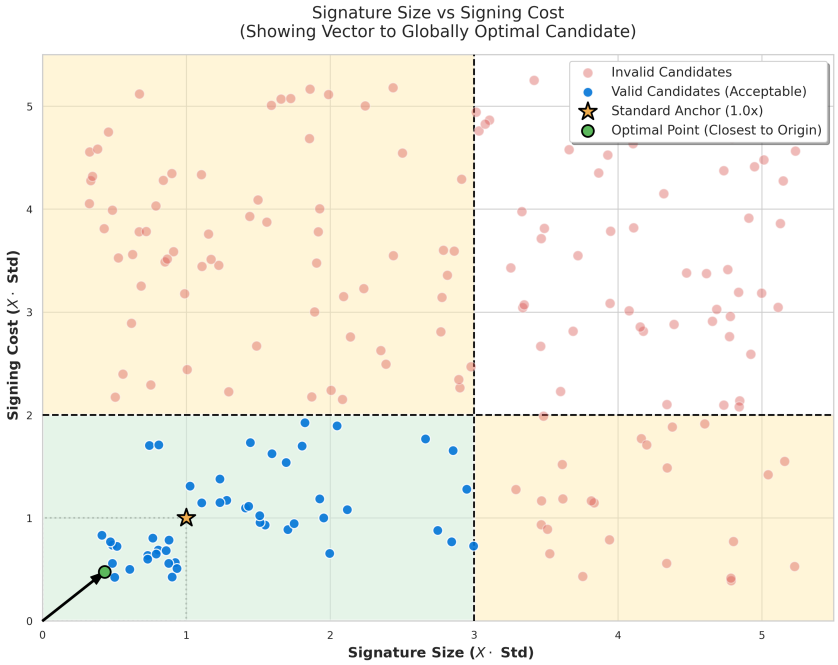
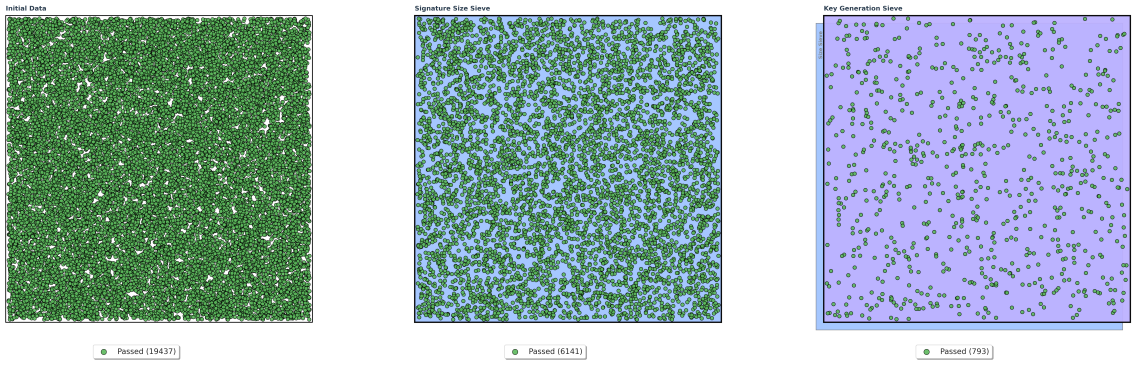
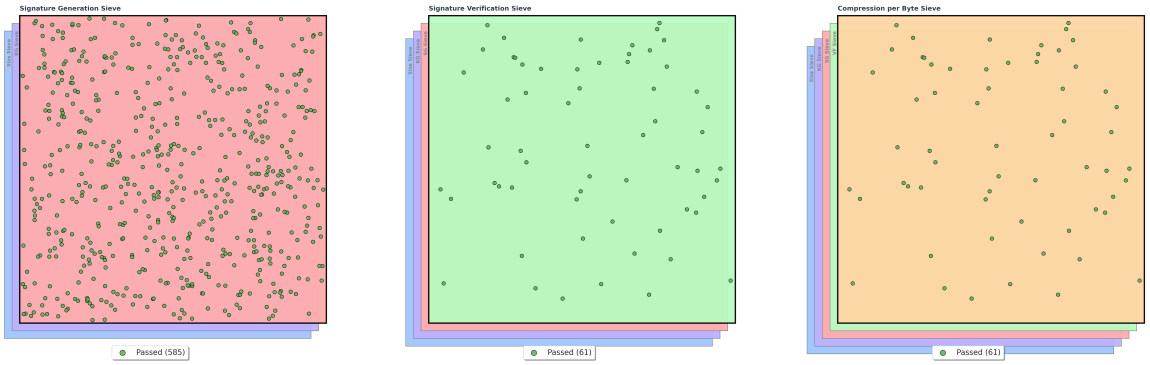


Figure 2: Visualization of Candidates with Prioritization of Size and Signing Cost Metrics.



(a) Initial Pool (b) SigSize (c) KeyGen



(d) SigGen (e) SigVerify (f) Comp/Byte

Figure 3: Full Parameter Filtering Process

For each surviving candidate P, let $\widetilde{X}_i$ denote the ratio of the candidate's value for metric i to the corresponding standard value, where $i \in \{\text{size}, \text{kg}, \text{sg}, \text{sv}, \text{cb}\}$. Working with ratios rather

than raw values keeps the five axes on a common, dimensionless scale. The Euclidean distance from the origin to P is then

$$d(P) = \sqrt{\widetilde{X}_{\text{size}}^2 + \widetilde{X}_{\text{kg}}^2 + \widetilde{X}_{\text{sg}}^2 + \widetilde{X}_{\text{sv}}^2 + \widetilde{X}_{\text{cb}}^2}\,, \tag{1}$$

and the optimal candidate is the one minimizing $d(P)$.

This treats every metric as equally important, which is rarely what we actually want. Signature size, for example, usually carries far more weight than key-generation cost in a Bitcoin context. The next subsection addresses this by attaching weights to each axis.

3.5 Decisional Weights

We can do better by applying the *weighted* Euclidean distance. The motivation is that weights give us a principled way to encode the relative priority of each metric. Let $w_{\text{size}}, w_{\text{kg}}, w_{\text{sg}}, w_{\text{sv}}, w_{\text{cb}} \in \mathbb{R}^+$ denote the raw weight coefficients representing the relative importance of signature size, key generation, signing, verification, and compressions per byte, respectively.

To keep the calculation proportionally consistent regardless of the absolute scale of the chosen weights, we normalize them into coefficients p_i (where $i \in \{\text{size}, \text{kg}, \text{sg}, \text{sv}, \text{cb}\}$) such that they sum to 1:

$$p_i = \frac{w_i}{w_{\text{size}} + w_{\text{kg}} + w_{\text{sg}} + w_{\text{sv}} + w_{\text{cb}}}, \quad \text{ensuring} \quad \sum p_i = 1 \tag{2}$$

Using these normalized coefficients, we calculate the Weighted Euclidean Distance (the L_2 norm) from the origin $(0, 0, 0, 0, 0)$ to the candidate's point P. The distance $d(P)$ is defined as:

$$d(P) = \sqrt{p_{\text{size}}\widetilde{X}_{\text{size}}^2 + p_{\text{kg}}\widetilde{X}_{\text{kg}}^2 + p_{\text{sg}}\widetilde{X}_{\text{sg}}^2 + p_{\text{sv}}\widetilde{X}_{\text{sv}}^2 + p_{\text{cb}}\widetilde{X}_{\text{cb}}^2} \tag{3}$$

As before, each $\widetilde{X}_i$ is the ratio of the candidate's metric to its standard counterpart, so the five axes remain on a common dimensionless scale.

To visualize the distances, we map each candidate's distance onto a colour scale: lower distances shade towards blue, higher distances towards red. As an example, consider the following two weight settings:

- Case 1: $w_{\text{size}} = w_{\text{kg}} = w_{\text{sg}} = w_{\text{sv}} = w_{\text{cb}} = 1$;
- Case 2: $w_{\text{size}} = 5,\ w_{\text{kg}} = 1,\ w_{\text{sg}} = 1.5,\ w_{\text{sv}} = 2,\ w_{\text{cb}} = 1$.

The result for the first case is shown below.

The parameters and metric values of the best candidate are shown in Table 3.

Table 3: Performance Comparison between the Standard Baseline and the Best Shortest Distance Candidate

Metric	**Standard Baseline**	**Best Candidate**	**Ratio to Std.**
Parameters (h, d, k, a, w)	$(63, 7, 14, 12, 16)$	$(40, 5, 19, 9, 16)$	–
Signature Size (Bytes)	7,856	6,496	0.8269×
Key Generation Cost	292,351	146,175	0.5000×
Signing Cost	2,218,483	760,050	0.3426×
Verification Cost	2,155	1,598	0.7415×
Compression per Byte (ρ)	0.2743	0.2460	0.8968×

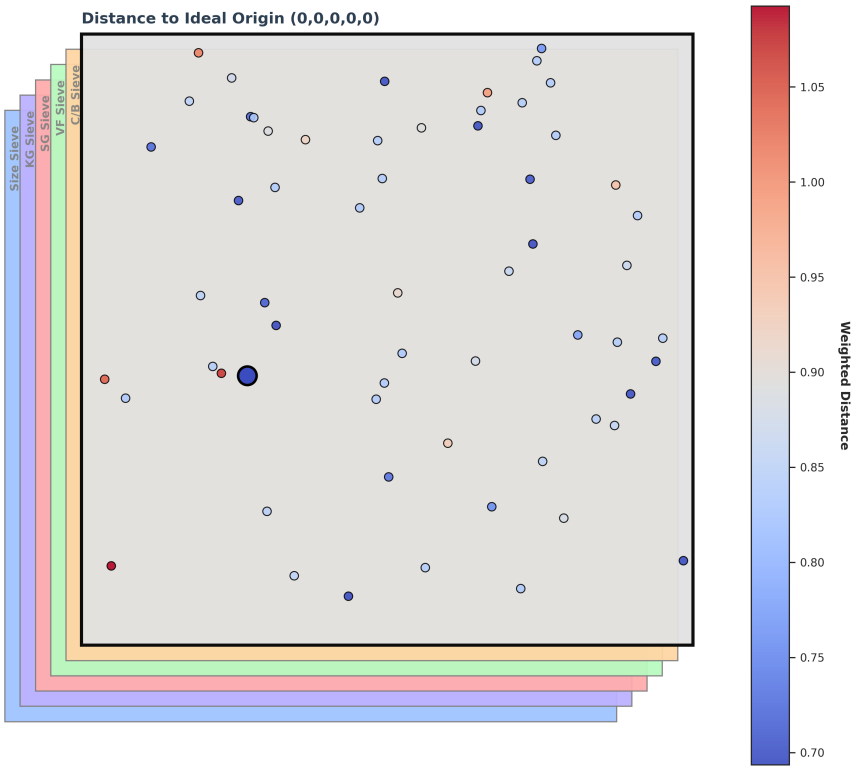


Figure 4: Visualization of Searching for the Case 1.

For the second case, we have the following picture:

The resulting values are shown in Table 4.

Table 4: Performance Comparison between the Standard Baseline and the Best Shortest Distance Candidate

Metric	**Standard Baseline**	**Best Candidate**	**Ratio to Std.**
Parameters (h, d, k, a, w)	$(63, 7, 14, 12, 16)$	$(40, 5, 13, 12, 16)$	–
Signature Size (Bytes)	$7,856$	$6,160$	$0.7841\times$
Key Generation Cost	$292,351$	$146,175$	$0.5000\times$
Signing Cost	$2,218,483$	$890,614$	$0.4015\times$
Verification Cost	$2,155$	$1,575$	$0.7309\times$
Compression per Byte (ρ)	0.2743	0.2557	$0.9321\times$

Comparing the two cases, the resulting metrics differ noticeably. The first case yields more balanced costs across the board, while the second case clearly favours signature size and verification at the expense of a heavier signing operation - exactly the trade-off we asked the weights to enforce.

3.6 Metric-Normalized Weighted Distance

The weighted distance introduced in Section 3.5 correctly encodes relative priorities when the five ratio axes $\widetilde{X}_i$ vary over comparable ranges. In practice, however, the filter pipeline disturbs this assumption. Consider the signature-size filter: after it is applied with a tight coefficient $X_{\text{size}} \ll 1$, every surviving candidate has a ratio $\widetilde{X}_{\text{size}}$ that lies in a narrow band close to X_{size}. The range of $\widetilde{X}_{\text{size}}$ values across survivors is therefore small relative to the variation of cost metrics such as $\widetilde{X}_{\text{kg}}$ or $\widetilde{X}_{\text{sg}}$, which the filter may leave nearly unconstrained. As a result, the

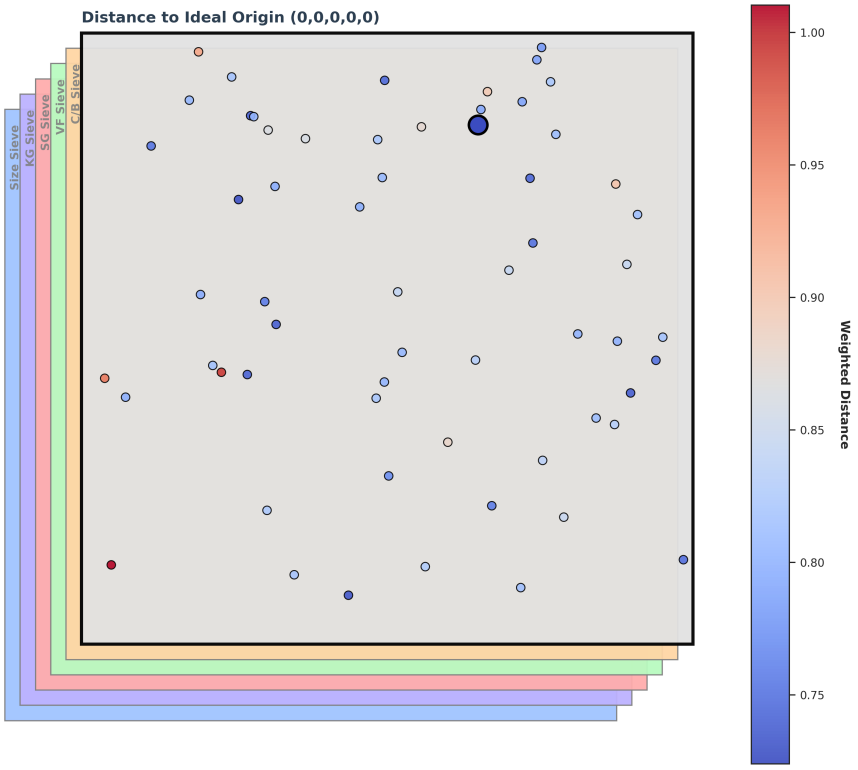


Figure 5: Visualization of Searching for the Case 2.

size term contributes almost the same value to $d(P)$ for every candidate in the pool, regardless of the weight p_{size} attached to it. The ranking is then dominated by the high-variance axes, and increasing w_{size} produces little to no change in which candidate is selected.

The remedy is to remove the scale imbalance before applying the weights. For each metric i and each surviving candidate P_j in the current pool, define the cross-pool minimum and maximum of the ratios:

$$m_i = \min_j \widetilde{X}_i(P_j), \qquad M_i = \max_j \widetilde{X}_i(P_j). \tag{4}$$

The *metric-normalized ratio* of candidate P on axis i is then

$$\widehat{X}_i(P) = \frac{\widetilde{X}_i(P) - m_i}{M_i - m_i}, \tag{5}$$

which maps every axis to $[0, 1]$ across the survivor pool, with $\widehat{X}_i = 0$ identifying the best-performing candidate on that axis and $\widehat{X}_i = 1$ the worst-performing one.

Replacing $\widetilde{X}_i$ with $\widehat{X}_i$ in Equation (3) yields the *metric-normalized weighted Euclidean distance*:

$$d(P) = \sqrt{p_{\text{size}}\,\widehat{X}^2_{\text{size}} + p_{\text{kg}}\,\widehat{X}^2_{\text{kg}} + p_{\text{sg}}\,\widehat{X}^2_{\text{sg}} + p_{\text{sv}}\,\widehat{X}^2_{\text{sv}} + p_{\text{cb}}\,\widehat{X}^2_{\text{cb}}}, \tag{6}$$

where the p_i are the normalized weight coefficients from Equation (2). The winner is still the candidate minimising $d(P)$, but now the weight p_i faithfully controls how much axis i influences the ranking: because every $\widehat{X}_i$ spans the same $[0, 1]$ interval, a coefficient $p_{\text{size}} = 0.5$ means that

signature size drives exactly half of the squared distance, independently of how tightly the size filter was set.

Note: The normalization is computed fresh from the survivors after each change to the filter coefficients. This means that the same weight vector $\mathbf{w}$ can produce a different winner when the filter is tightened, because the pool shrinks and the per-axis ranges change accordingly. This behaviour is intentional: the weights express relative importance *within the feasible region*, and that region is itself defined by the filter.

Consider the following filtering parameters and weights set for comparison of methods for finding the shortest vector with weights:

- Filtering parameters: $X_{\text{size}} = 0.75,\ X_{\text{kg}} = 5,\ X_{\text{sg}} = 5,\ X_{\text{sv}} = 1.5,\ X_{\text{cb}} = 1$;
- Weights coefficients: $w_{\text{size}} = 5,\ w_{\text{kg}} = 1.5,\ w_{\text{sg}} = 1.5,\ w_{\text{sv}} = 2,\ w_{\text{cb}} = 1$.

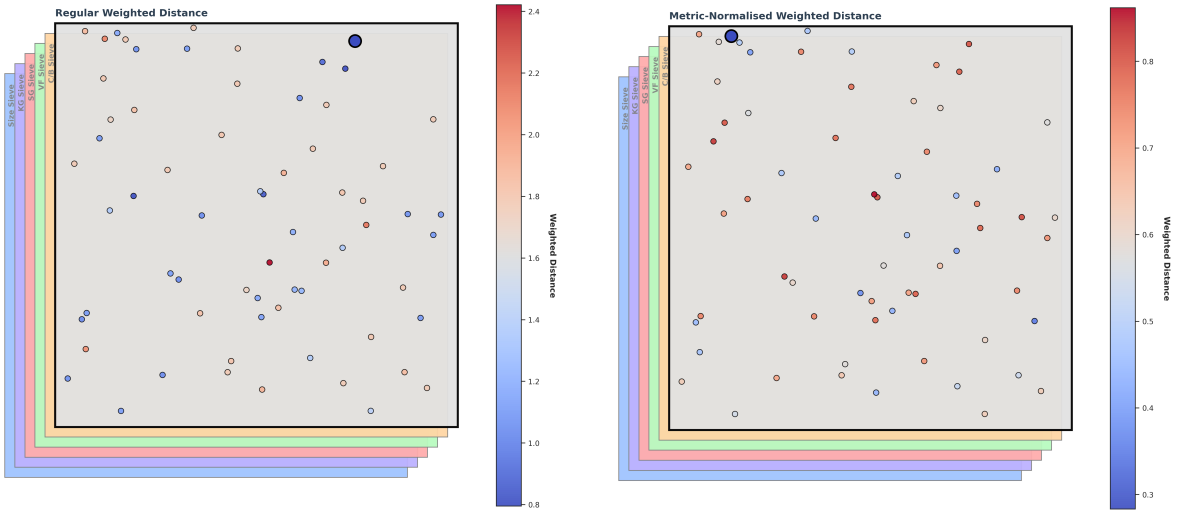
The resulting best candidates for each method shown in the Table 5.

Table 5: Comparison between the Best Candidates under the Regular Weighted and Metric-Normalised Weighted Distance Criteria

Metric	Standard Baseline	Metric-Normalized	Ratio to Std.	Regular Weighted	Ratio to Std.
Parameters (h, d, k, a, w)	$(63, 7, 14, 12, 16)$	$(40, 4, 9, 16, 16)$	–	$(45, 5, 12, 11, 16)$	–
Signature Size (Bytes)	7,856	5,344	0.6802×	5,840	0.7434×
Key Generation Cost	292,351	584,703	2.0000×	292,351	1.0000×
Signing Cost	2,218,483	4,108,282	1.8518×	1,535,479	0.6921×
Verification Cost	2,155	1,286	0.5968×	1,555	0.7216×
Compression per Byte (ρ)	0.2743	0.2406	0.8773×	0.2663	0.9707×

Comparing the two cases, the resulting metrics are different. The regular weighted distance chooses a more balanced best candidate, while the metric-normalized weighted distance selects a candidate with a noticeably smaller signature size, trading the key and signature generation costs.

The outcome of each distance method shown in the Figure 6.



(a) Regular Weighted Distance Result (b) Metric-Normalised Weighted Distance Result

Figure 6: Visualization of each Distance Method Result for Finding the Shortest Vector

4 Recommended Candidate Tuple

To demonstrate the full pipeline on a concrete parameter set we selected, we fix the following filtering coefficients:

$$X_{\text{size}} = 0.74, \quad X_{\text{kg}} = 2.0, \quad X_{\text{sg}} = 2.0, \quad X_{\text{sv}} = 0.8, \quad X_{\text{cb}} = 1.0.$$

These coefficients say that we really care about the signature size (to make at least 25% win) and the verification complexity. At the same time having the key and signature complexity $2\times$ at most is an acceptable trade-off because (1) it's not very critical taking into account even hardware wallet takes 2-3 seconds to produce SLH-DSA signature, (2) we can apply pruning technique [1] to reduce it additionally.

Applying these constraints to the full parameter grid leaves exactly two surviving configurations, whose metrics are compared against the standard baseline in Table 6.

Table 6: Performance Comparison between the Standard Baseline and the Two Evaluated Candidates

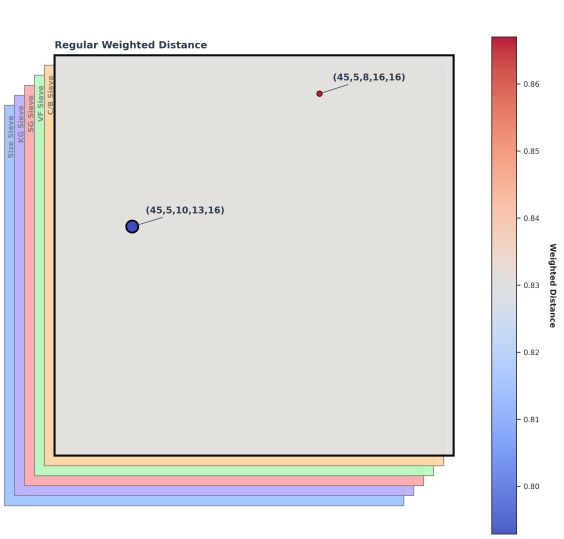
Metric	**Standard Baseline**	**Candidate 1**	**Ratio to Std.**	**Candidate 2**	**Ratio to Std.**
Parameters (h, d, k, a, w)	$(63, 7, 14, 12, 16)$	$(45, 5, 10, 13, 16)$	–	$(45, 5, 8, 16, 16)$	–
Signature Size (Bytes)	7,856	5,776	0.7352×	5,712	0.7271×
Key Generation Cost	292,351	292,351	1.0000×	292,351	1.0000×
Signing Cost	2,218,483	1,707,512	0.7697×	3,034,618	1.3679×
Verification Cost	2,155	1,550	0.7193×	1,546	0.7174×
Compression per Byte (ρ)	0.2743	0.2684	0.9783×	0.2707	0.9867×

As the table makes clear, the two survivors represent distinct operational trade-offs.

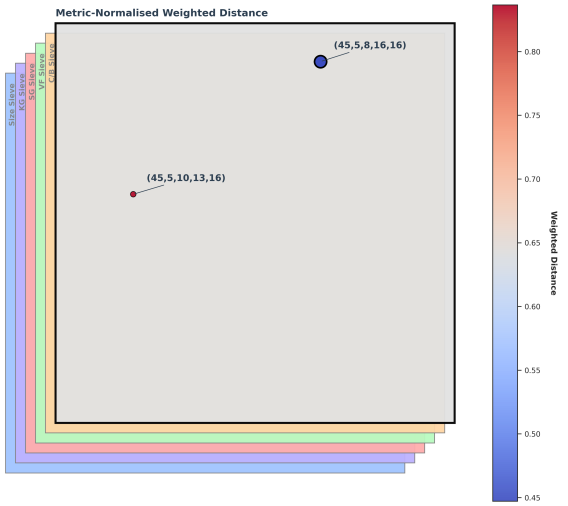
- Candidate 1 offers a well-rounded improvement: signature size, signing cost, and verification cost all decrease relative to the standard.
- Candidate 2 pushes further on size and verification, but at a concrete price - its signing cost exceeds the standard by roughly $1.37\times$, making it suitable only in contexts where signing latency is secondary.

Under the regular weighted distance of Equation (3), Candidate 1 attains the smallest distance for any weight vector that does not entirely discount signing cost: Candidate 2's advantages are marginal (64 B of size, 4 compressions of verification) while its signing cost is roughly $1.8\times$ higher. Note that the metric-normalized distance of Section 3.6 is not informative in this specific case: with only two survivors, every axis collapses to its endpoints $\{0, 1\}$, and the ranking degenerates into a bare comparison of weight sums. The results of the search are presented in Figure 7.

All figures and tables in this report are generated by the accompanying pipeline (`make generate`). An interactive explorer that reproduces these searches — including the filter coefficients, decisional weights, and distance criteria used above — is publicly available at https://blockstreamresearch.github.io/SPHINCS-Parameters/site/.



(a) Regular weighted distance: Candidate 1 wins



(b) Metric-normalised distance degenerates to Candidate 2

Figure 7: The Two Surviving Candidates under Both Distance Criteria

References

[1] Conduition. Pruning hypertrees for the lax and lazy. Conduition Cryptography Blog, April 2026. https://conduition.io/cryptography/hypertree-pruning/.

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

[3] National Institute of Standards and Technology. FIPS 205: Stateless hash-based digital signature standard. https://csrc.nist.gov/pubs/fips/205/final, August 2024.

[4] Jonas Nick. SHRINCS: 324-byte stateful post-quantum signatures with static backups. Delving Bitcoin Forum, December 2025. https://delvingbitcoin.org/t/shrincs-324-byte-stateful-post-quantum-signatures-with-static-backups/2158.