A note on SPHINCS$^+$ parameter sets

Stefan Kölbl[1], Jade Philipoom[1,2]

[1] Security Engineering Research, Google
kste@google.com, jadep@{google.com,opentitan.org}
[2] Open Titan

Keywords: Hash-based signatures, post-quantum, SPHINCS$^+$

Abstract. In this note, we explore parameter sets for SPHINCS$^+$ which support a smaller number of signatures than 2^{64}, but are otherwise compatible with the SLH-DSA specification. In practice, use cases for which a low number of signatures per key pair suffice are common, and as we will show this allows a significant reduction in signature size and verification speed for SPHINCS$^+$. For this we carry out a larger search through the SPHINCS$^+$ parameter space, comparing it with the current parameter sets and further showing that for carefully chosen parameter the security degrades slowly if one exceeds the limits. Finally, we provide a case study for firmware signing on OpenTitan to demonstrate the efficiency of these alternative parameters.

1 Introduction

The NIST call for post-quantum digital signature schemes had a requirement that every scheme must maintain its security when a single key pair is used for up to 2^{64} signatures. For most signature schemes this requirement comes with little additional costs. However for hash-based signatures supporting a large number of signatures for a single key pair increases both signature size and computation time, as it requires choosing parameters which lead to larger tree sizes.

In practice, an upper limit of 2^{64} is very conservative for some applications. Consider for instance the limit of 2^{50} signatures the original SPHINCS schemes targeted. This would allow issuing 1 million signatures per second for 30 years with a single key. There are many use cases in practice which will never need anywhere close to 2^{64} signatures, and could benefit greatly from parameter sets which are tailored towards those applications. To list a few:

- Firmware signing: Only a small number of firmware versions will exist in the lifetime of a key and a few thousand signatures would already be enough in those cases.
- Certificate signing: CAs sign a comparably small number of signatures compared to the 2^{64} target. This is especially true for root CAs, which sign intermediates. As today the certificate transparency log contains a total of around $\approx 2^{33}$ certificates.

In particular, in these use cases it is difficult for an adversary to trigger a large number of signatures being generated. For applications where this is possible, e.g. TLS handshakes, a conservative choice is preferable.

There are significant risks of targeting a lower number of signatures, as enforcing a query limit for a key in practice can be difficult. We therefore would not recommend to place these schemes alongside signature schemes which provide a virtually unlimited number of signatures per key as this could lead to misuse. In some environments this might not be easier than managing the state in a stateful hash-based signature scheme like LMS or XMSS.

However, a stateless scheme like SPHINCS+ would still have several advantages over stateful schemes in this scenario:

- Exceeding the limit is less severe. While stateful schemes fall quickly apart if state gets reused [BH17], the security of SPHINCS$^+$ degrades much more gracefully (see subsection 3.1). If the parameters are chosen carefully, security degrades surprisingly slowly. In practice this means that the limit can be exceeded by several orders of magnitude before it becomes a practical issue.
- Backing up the key becomes much easier, as no state has to be synchronized.
- Concurrently using the same key material in a distributed system is much simpler.

We think it would be particularly useful to select parameters that target the same number of signatures as the existing stateful hash-based signature standards. We can then directly compare the new parameter sets to stateful schemes like LMS/XMSS and can evaluate exactly how much it would cost to get rid of the stateful property.

2 Parameter space

In the following we use q to denote the maximum number of signatures which can be used, while the targeted security level is still guaranteed. We use the same approach as for the parameter search done for the original SPHINCS$^+$ parameter [BHK$^+$19], sampling a large number of parameters from a reasonable space in terms of signature size and expected signing costs.

We collected a large set of parameters and sorted them by signature size first, with signing speed as a second criteria. These were then filtered to only include parameters which are strictly better than others in the set, i.e. a smaller signature size or faster signing speed. The results of this search can be seen in Figure 1, Figure 2 and Figure 3. The parameters are limited to $< 10^9$ hash calls for signing (which corresponds roughly to 1 minute of computation time on a current system in our benchmarks). The complete data set of our parameter search are made available at `https://github.com/kste/spx-few/`, and include additional choices of q which are omitted from these plots for clarity.

In this search we further include parameters with $w = 256$ for smaller values of q. While $w = 256$ can lead to parameters which are better in both signature size and signing speed, the resulting verification speed is worse. As verification

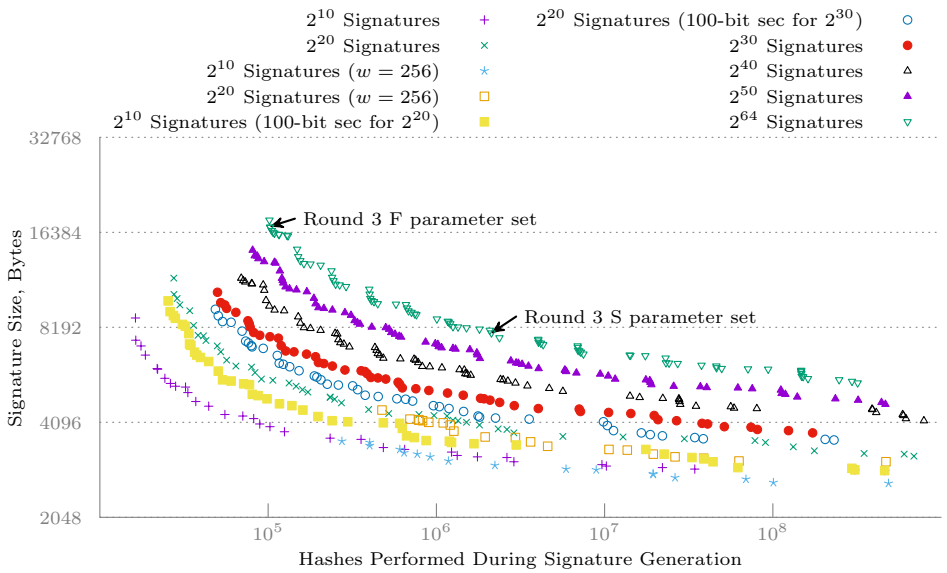

Fig. 1. Comparison of the size and signing speed for parameter sets providing 128-bit security, for different values of q.

speed is often critical to the applications which could utilize parameters with a small q, we do not think this trade-off offers a significant benefit. In addition, most of these parameters only have a single tree on top of the FORST layer, hence only a single OTS benefits from the signature size decrease by using a larger value w. Going from $w = 16$ to $w = 256$ saves 272 bytes for $n = 16$ in the OTS.

Applying the same search to the original requirements of $q = 2^{64}$, would give a minimal signature size of:

- 5 456 bytes ($h = 68, d = 4, b = 21, k = 6, w = 16$) for 128-bit security,
- 11 352 bytes ($h = 68, d = 4, b = 24, k = 8, w = 16$) for 192-bit security,
- 19 584 bytes ($h = 68, d = 4, b = 24, k = 11, w = 16$) for 256-bit security.

This already would give $\approx 33\%$ smaller signatures (at the cost of much slower key generation and signing). Using e.g. $q = 20$ pushes this to over 50% and further reduces verification time.

3 New parameter sets

We see the biggest benefit of these parameters in use cases where:

- Key generation time is not a concern. In particular we do not have to run key generation in an interactive protocol.

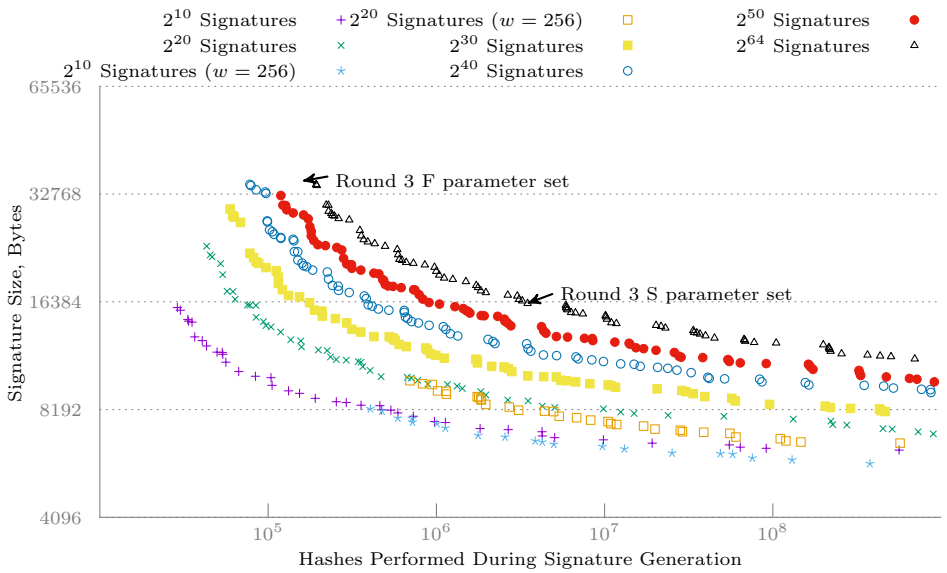

Fig. 2. Comparison of the size and signing speed for parameter sets providing 192-bit security, for different values of q.

- Signing time is not a concern. Signing happens infrequently, not in a resource constraint environment or an interactive protocol.
- Verification speed is important. For example in a secure boot scenario when the image has to be verified on start-up and the latency of signature verification directly contributes to start-up time.

For applications and protocols which require frequent signing, SPHINCS$^+$ is not an attractive choice and we don't expect this to change by alternative parameters. Compared to other post-quantum signature schemes like Dilithium, the signing speed will always be significantly worse unless very large signatures are acceptable (and if you need fast signing, having a large signature will likely be a major issue). Secondly, if you need to carry out frequent signing operations, having controls in place to ensure the maximum number of signatures may be harder to deploy which further discourages this approach.

We propose to consider a single parameter set (see Table 1) supporting $q = 2^{20}$ signatures for each security level, to keep the overall number of parameter sets small while still providing a significant performance improvement. Using $q < 2^{20}$ gives only a moderate improvement (for $q = 10$ signature size could be reduced by 10% for parameters targeting 128-bit security), while for larger values of q the signature size quickly grows. Additional parameter sets are given in Table 5 and Table 6 for comparison of this trade-off.

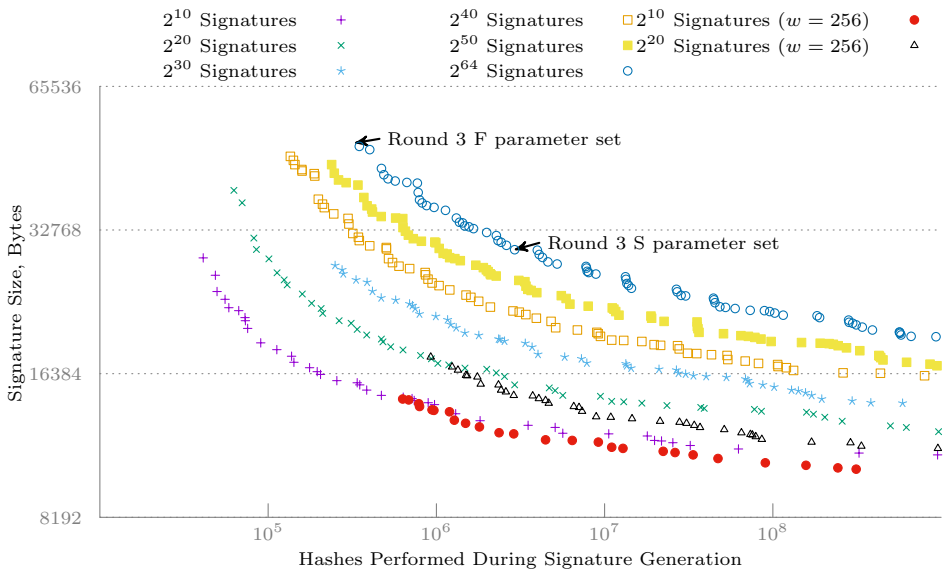

Fig. 3. Comparison of the size and signing speed for parameter sets providing 256-bit security, for different values of q.

3.1 Security degradation

It's important to point out that while SPHINCS$^+$ security will degrade when exceeding the maximum number of signatures, this happens slowly, especially if parameters are picked carefully. We evaluated the security degradation for all the parameters we considered, and for the choices in Table 1 picked parameters which are on the conservative side.

The reason for this security degradation is that an attackers success probability to break the *interleaved target subset resilience* increases with the number of signatures obtained for a given key pair. Above q this probability is higher than the targeted security level for a parameter set. We use the same evaluation technique as in [BHK+19] to determine the security level for values $q' > q$.

Figure 4 shows a plot of all parameters and their security degradation, when targeting 128-bit security and $q = 2^{20}$ signatures compared to the parameter set proposed in Table 1. The best parameter set in terms of security degradation in this graph would be $n = 26$, $d = 2$, $b = 18$, $k = 7$, however this parameter set has a signature size of 3,680 bytes and verification time would be $\approx 33\%$ slower.

For higher security levels (see Figure 5 and Figure 6) the same holds true, and security only slowly degrades when exceeding 2^{20} signatures. For most parameters, the security would still be > 100 bits if 2^{30} signatures are issued under a single key pair.

Table 1. Comparison of parameters targeting small signatures with the round 3 SPHINCS$^+$ parameters.

	n	h	d	b	k	w	bitsec	sig bytes
SPHINCS$^+$-128s	16	63	7	12	14	16	128	7 856
SPHINCS$^+$-128s-q20	16	18	1	24	6	16	128	3 264
SPHINCS$^+$-192s	24	63	7	14	17	16	192	16 224
SPHINCS$^+$-192s-q20	24	20	1	21	10	16	192	7 008
SPHINCS$^+$-256s	32	64	8	14	22	16	255	29 792
SPHINCS$^+$-256s-q20	32	19	1	21	14	16	256	12 640

It's also worth nothing that due to the slow signing time it is difficult to exhaust a key pair quickly. With a signing time of $\approx$ 1 minute, it would take around 2 years of continuously signing to reach this limit in a single process.

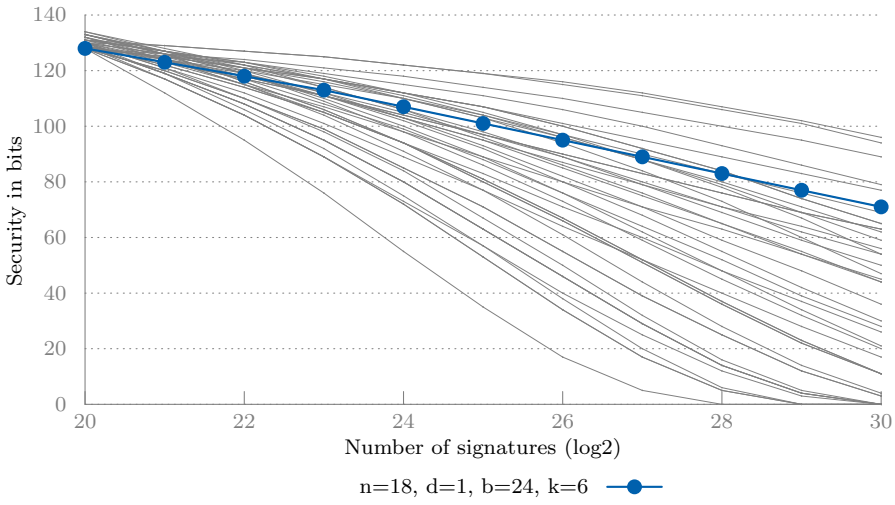

Fig. 4. Plot showing how the security degrades for all parameter sets targeting 128-bit security and 2^{20} signatures.

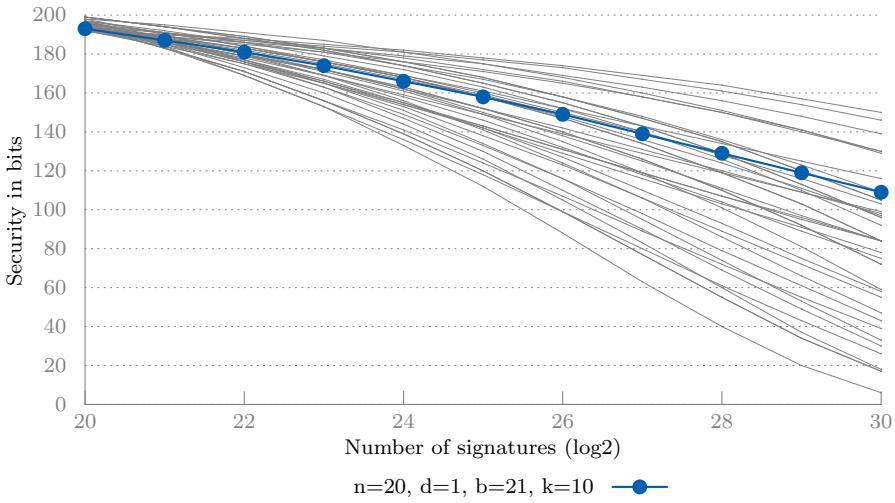

Fig. 5. Plot showing how the security degrades for all parameter sets targeting 192-bit security and 2^{20} signatures.

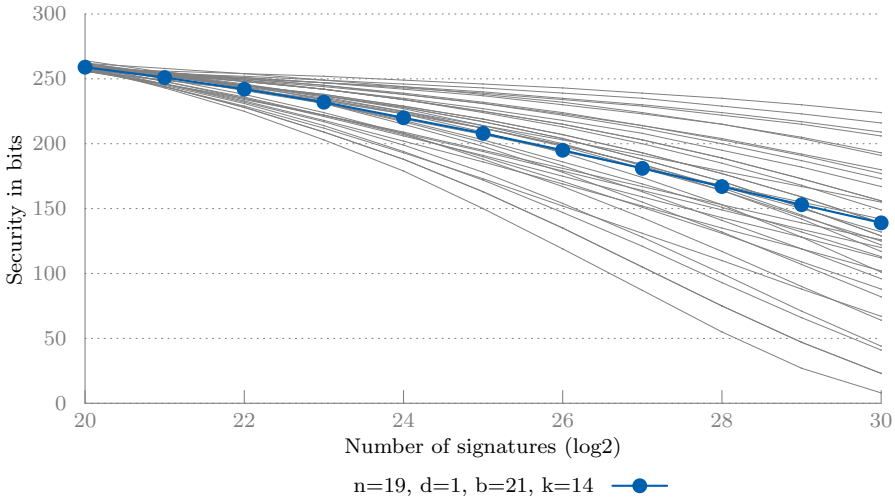

Fig. 6. Plot showing how the security degrades for all parameter sets targeting 256-bit security and 2^{20} signatures.

Table 2. Comparison of the performance for the new parameters and the original SPHINCS$^+$ parameters. Key generation, signing and verification time are given as number of cycles on an AMD EPYC 7B12.

Parameters	size	key generation	signing	verification
SPHINCS$^+$-SHA2-128f	17 088	679 151	16 020 636	1 330 842
SPHINCS$^+$-SHA2-128s	7 856	42 915 167	325 201 745	481 106
SPHINCS$^+$-SHA2-128s-q20	3 264	21 777 216 922	66 414 818 220	155 002
SPHINCS$^+$-SHA2-192f	35 664	1 007 932	27 332 156	1 885 107
SPHINCS$^+$-SHA2-192s	16 224	64 971 780	622 697 429	753 233
SPHINCS$^+$-SHA2-192s-q20	7 008	128 538 607 582	140 291 427 167	318 803
SPHINCS$^+$-SHA2-256f	49 856	2 636 584	55 860 356	1 897 032
SPHINCS$^+$-SHA2-256s	29 792	42 439 089	555 910 269	1 038 173
SPHINCS$^+$-SHA2-256s-q20	12 640	85 194 574 723	102 103 271 534	434 148

3.2 Benchmarks

We evaluated the new parameter sets using the SPHINCS$^+$ reference implementation from `https://github.com/sphincs/sphincsplus/`[3]. The environment for our benchmarks is a single core of a AMD EPYC 7B12, running Debian 11 (Kernel version 5.10.0-26) and GCC 10.2.1 (with -O3 flag). The resulting cycle counts for the original SPHINCS$^+$ and the alternative parameter sets are given in Table 2.

As can be seen from these benchmarks, key generation and signing time are significantly slower for these parameters, while the signature size and verification time improve at least by a factor of 2 across all security levels.

4 Firmware signing

As a case study for the firmware signing use case, we evaluate the performance of SPHINCS$^+$ parameter sets on OpenTitan, an open-source silicon root of trust which can be configured to run secure boot with SPHINCS$^+$.

The OpenTitan benchmarks all use simple SHAKE instantiations of SPHINCS$^+$, matching the existing secure boot implementation. The hardware design includes a SHAKE accelerator, which makes this configuration particularly efficient. Full replication instructions for the benchmarks can be found here: `https://github.com/jadephilipoom/opentitan/tree/spx-benchmark/spx-benchmark`

Table 3 shows that the new parameter sets produce a 4.5-4.9x speedup for signature verification on this platform.

[3] Commit `035b39429d96ca554402b78f296f0de181674abd`.

Table 3. Comparison of signature size and verification speed for new parameters to the current SPHINCS$^+$ parameters on OpenTitan.

Parameters	signature size (bytes)	verification speed (cycles)
SPHINCS$^+$-SHAKE-128s	7 856	1 298 047
SPHINCS$^+$-SHAKE-128s-q20	3 264	277 852
SPHINCS$^+$-SHAKE-192s	16 224	2 089 772
SPHINCS$^+$-SHAKE-192s-q20	7 008	462 991
SPHINCS$^+$-SHAKE-256s	29 792	3 390 932
SPHINCS$^+$-SHAKE-256s-q20	12 640	695 937

Table 4. Comparison of SPHINCS$^+$ signature public key size, signature size, and verification speed to classical cryptography on OpenTitan at a 128-bit security level.

Scheme	pk bytes	sig bytes	verification cycles
RSA-3072	416	384	236 057
ECDSA-P256	64	64	449 164
SPHINCS$^+$-SHAKE-128s	32	7 856	1 298 047
SPHINCS$^+$-SHAKE-128s-q20	32	3 264	277 852

The speedup from the new parameters brings SPHINCS$^+$ verification speed close to that of classical cryptography at a similar security level. Table 4 shows how SPHINCS$^+$ compares to RSA and ECDSA for a 128-bit security level in terms of public key size, signature size, and verification speed on OpenTitan.

The RSA numbers in table 4 follow OpenTitan's implementation regarding pre-computed constants, in order to be a realistic comparison point. The Montgomery multiplication constant $-(N^{-1}) \bmod R$, for $R = 2^{256}$ in this case, is precomputed and stored along with the public key, while the constant $R^2 \bmod N$ is computed on the fly. An alternative would be precomputing and storing R^2, which would significantly improve verification speed (about 100K cycles faster), but would require 384 additional bytes of storage per public key. Public keys are stored in limited ROM space, and OpenTitan uses several of them for redundancy, so the aggregate storage cost of storing the constants made the speed tradeoff worthwhile in this context.

We can also evaluate the practicality of a lower maximum number of signatures for this use case example:

- Signing is only required for firmware updates, which are expected to be rare. One signature per month is a realistic estimate, but $q = 2^{20}$ would allow even one per day for 2,872 years. Even $q = 2^{10}$ would allow weekly updates for 19 years.

– A slow signing speed, even minutes, is tolerable, since signing happens rarely. Key generation is even rarer and can also be slow.
– Memory is limited; space to store the firmware and its signature is on the order of 100kB, for potentially multiple signed boot stages. Large code signatures leave significantly less space for the firmware itself.
– Stateless schemes are appealing because the signing infrastructure is less complex to safely maintain, evidenced by the fact that OpenTitan chose SPHINCS$^+$ over LMS even with existing parameter sets, accepting the computation time and signature size penalty in order to get the stateless property.

References

BH17. Leon Groot Bruinderink and Andreas Hülsing. "Oops, I Did It Again" - Security of One-Time Signatures Under Two-Message Attacks. In *SAC*, volume 10719 of *Lecture Notes in Computer Science*, pages 299–322. Springer, 2017.

BHK+19. Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS$^+$ Signature Framework. In *CCS*, pages 2129–2146. ACM, 2019.

5 Appendix

Table 5. Comparison of parameters targeting small signatures with the round 3 SPHINCS$^+$ parameters.

	n	h	d	b	k	w	bitsec	sig bytes
SPHINCS$^+$-128s	16	63	7	12	14	16	128	7 856
SPHINCS$^+$-128s-q10	16	15	1	18	7	16	128	2 944
SPHINCS$^+$-128s-q20	16	18	1	24	6	16	128	3 264
SPHINCS$^+$-128s-q30	16	32	2	19	7	16	128	3 888
SPHINCS$^+$-192s	24	63	7	14	17	16	192	16,224
SPHINCS$^+$-192s-q10	24	19	1	23	8	16	192	6 312
SPHINCS$^+$-192s-q20	24	20	1	21	10	16	192	7 008
SPHINCS$^+$-192s-q30	24	36	2	21	9	16	192	8 088
SPHINCS$^+$-256s	32	64	8	14	22	16	256	29 792
SPHINCS$^+$-256s-q10	32	18	1	25	10	16	256	11 072
SPHINCS$^+$-256s-q20	32	19	1	21	14	16	256	12 640
SPHINCS$^+$-256s-q30	32	36	2	20	13	16	256	14 208

Table 6. Comparison of parameters targeting fast signatures with the round 3 SPHINCS$^+$ parameters. These parameters have a signing speed comparable to the current SPHINCS$^+$-Xf parameter sets.

	n	h	d	b	k	w	bitsec	sig bytes	sig speed
SPHINCS$^+$-SHA2-128f	16	60	20	9	30	16	128	16 976	16 020 636
SPHINCS$^+$-SHA2-128f-q10	16	12	2	10	15	16	128	3 968	17 440 489
SPHINCS$^+$-SHA2-128f-q20	16	20	4	10	17	16	128	5 568	18 598 462
SPHINCS$^+$-SHA2-128f-q30	16	30	6	9	19	16	128	6 896	20 454 319
SPHINCS$^+$-SHA2-192f	24	66	22	8	33	16	192	35 664	27 332 156
SPHINCS$^+$-SHA2-192f-q10	24	12	2	10	24	16	192	9 096	30 118 839
SPHINCS$^+$-SHA2-192f-q20	24	25	5	9	25	16	192	12 744	27 068 467
SPHINCS$^+$-SHA2-192f-q30	24	30	5	10	26	16	192	13 782	54 835 110
SPHINCS$^+$-SHA2-256f	32	68	17	10	30	16	256	49 216	55 860 356
SPHINCS$^+$-SHA2-256f-q10	32	12	2	11	29	16	256	15 840	54 274 466
SPHINCS$^+$-SHA2-256f-q20	32	24	4	9	36	16	256	20 896	52 813 844
SPHINCS$^+$-SHA2-256f-q30	32	30	5	10	36	16	256	24 384	73 163 909