A note on SPHINCS$^+$ parameter sets

Stefan Kölbl[1]

Security Engineering Research, Google
kste@google.com

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

Abstract. In this note, we discuss using parameter sets for SPHINCS$^+$ which support a smaller number of signatures than the 2^{64} target. This includes a larger search through the SPHINCS$^+$ parameter space, comparing it with the current parameter sets and providing data on how the security degrades if one exceeds the limits.

1 Introduction

The NIST call for post-quantum digital signature schemes had a requirement that every scheme must maintain the full security level 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 parameter which lead to larger tree sizes.

In practice, a limit of 2^{64} seems 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 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 exists in the lifetime of a key and a few thousand signatures would already be enough in some 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 for instance managing the state in a stateful hash-based signature scheme.

However, a stateless scheme like SPHINCS+ would still have several advantages:

- Exceeding the limit is less severe. While stateful schemes fall quickly apart if state gets reused, the security of $\mathsf{SPHINCS}^+$ degrades much slower (see subsection 2.1). If the parameters are chosen carefully, security degrades surprisingly slow. In practice this means that the limit can be exceeded by several order of magnitudes 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 particular useful to have parameters, which target the same number of signatures as the existing stateful hash-based signature standards. These parameter sets then allow a direct comparison on how much it would cost to get rid of the stateful property.

2 Parameter search

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 used the same approach as for the parameter search done on the original $\mathsf{SPHINCS}^+$ 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, sorted them by signature size first and signing speed as a second criteria. These are then filtered to only include parameters which are strictly better, 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.

In this search we further include parameters with $w = 256$ for smaller q. While $w = 256$ can lead to parameters which are better in both signature size and signing speed, we expect the resulting verification speed to be significantly worse. As verification speed is often critical in the applications which parameter with small q would be used, we don't think these offer a significant benefit.

For estimating the signing speed, we simply count the number of hash operations. This only provides a rough estimate of the real performance, as we treat the costs for each hash operation the same. Depending on the platform and choice of hash function there might be an asymmetry in the costs, and better trade-offs could be possible.

In Table 1 we provide a comparison between the current $\mathsf{SPHINCS}^+$ parameters, and a selection of $q = 2^{20}$ parameters, which seem well-suited for the previously mentioned use cases. In particular, they reduce signature size by more than 50% and provide faster verification, while key generation and signing time is slower.

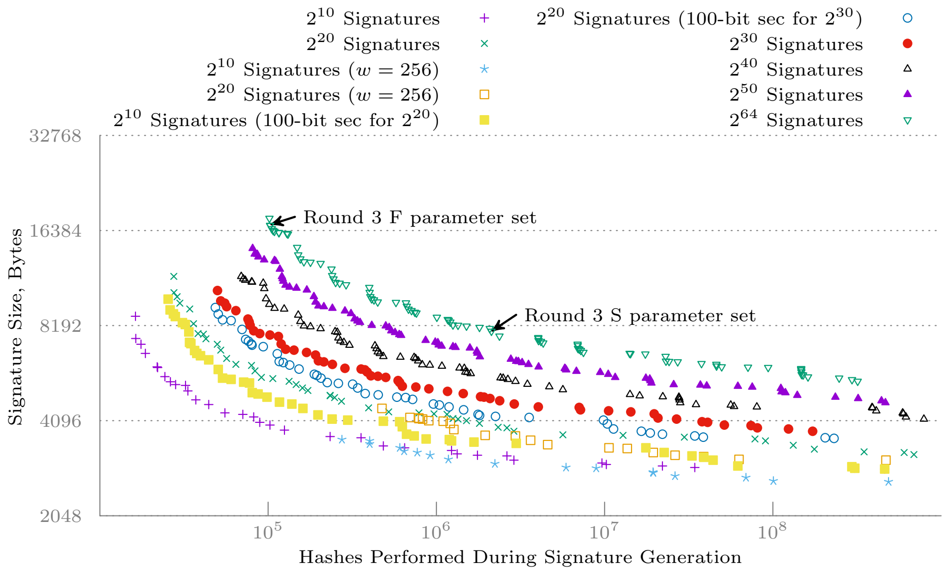

Fig. 1. Comparison of the size and signing speed for parameter sets providing 128-bit security, for different target of max signatures.

2.1 Security degradation

It's important to point out that while SPHINCS$^+$ security will degrade when exceeding the maximum number of signatures, this happens slowly. Figure 4 and Figure 5 shows how security degrades for some of the parameters we found. With moderate additional costs it is possible to find parameter which guarantee that security does not drop below a certain level after exceeding the maximum number of signatures. In Figure 1, we explored parameter sets which:

- Support 2^{10} signatures, but security does not drop below 100 bits if one signs up to 2^{20} signatures.
- Support 2^{20} signatures, but security does not drop below 100 bits if one signs up to 2^{30} signatures.

These have been included in Figure 1 and can provide a more conservative approach, having a larger buffer before security would be affected in practice.

References

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.

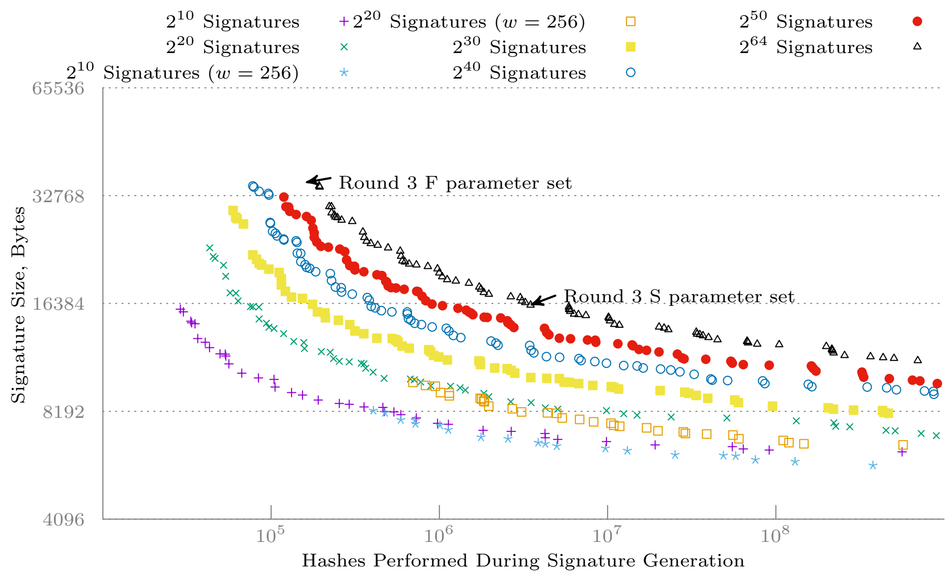

Fig. 2. Comparison of the size and signing speed for parameter sets providing 192-bit security, for different target of max signatures.

Table 1. Comparison of parameters for $q = 2^{20}$ with the round 3 SPHINCS$^+$ parameters.

	n	h	d	$\log(t)$	k	w	bitsec	sec level	sig bytes
SPHINCS$^+$-128s	16	63	7	12	14	16	133	**1**	7,856
SPHINCS$^+$-128s ($q = 2^{20}$)	16	17	1	20	8	16	132	**1**	3,424
SPHINCS$^+$-192s	24	63	7	14	17	16	193	**3**	16,224
SPHINCS$^+$-192s ($q = 2^{20}$)	24	26	2	16	12	16	192	**3**	7,992
SPHINCS$^+$-256s	32	64	8	14	22	16	255	**5**	29,792
SPHINCS$^+$-256s ($q = 2^{20}$)	32	24	2	16	17	16	257	**5**	14,336

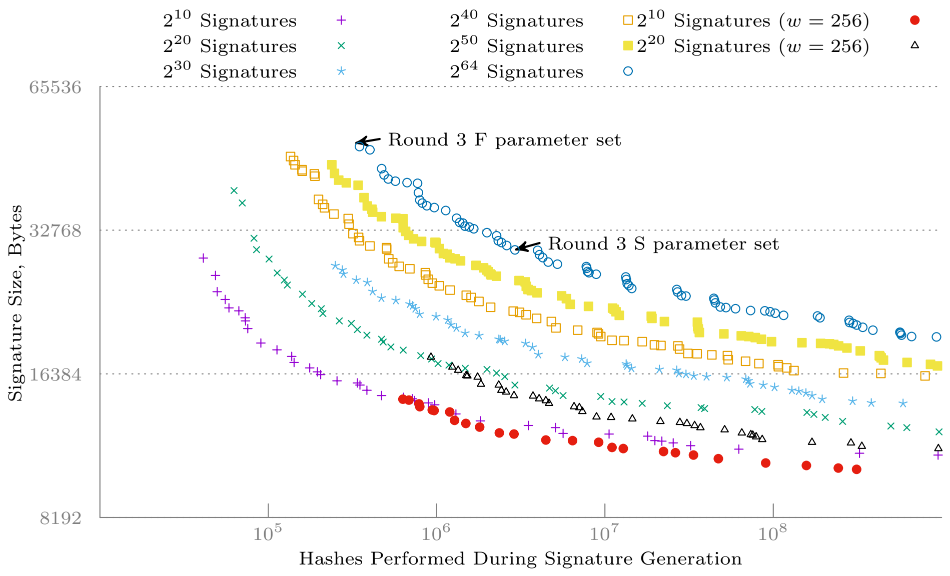

Fig. 3. Comparison of the size and signing speed for parameter sets providing 256-bit security, for different target of max signatures.

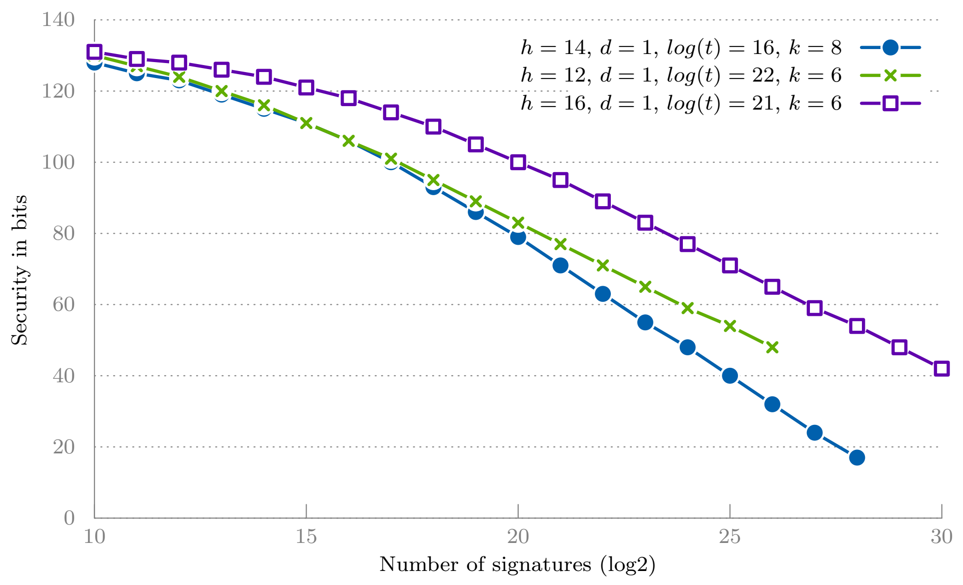

Fig. 4. Plot showing how the security degrades for different parameters targeting 2^{10} signatures with 128-bit security.

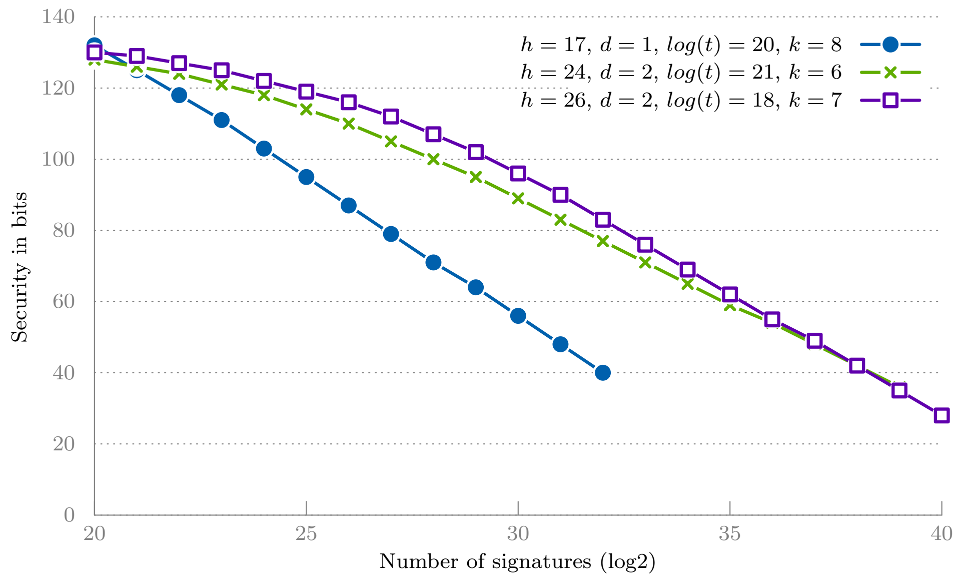

Fig. 5. Plot showing how the security degrades for different parameters targeting 2^{20} signatures with 128-bit security.