SPHINCS$^+$

Modifications for Round 3 submission to the NIST post-quantum project

Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens,
Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer,
Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl,
Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen,
Christian Rechberger, Joost Rijneveld, Peter Schwabe, Bas Westerbaan

September 29, 2020

1 New Parameter Sets

For Round 3 we carried out a broader search for alternative parameter sets for SPHINCS$^+$. When we considered alternative parameter sets for round 3 of SPHINCS$^+$, we used the following criteria:

- They had to be an improvement to the existing parameter sets in the dimension the parameter set is optimized for; for the F parameter sets, signature generation had to be faster on the large majority of the platforms and function families we tested, for the S parameter sets, the signatures had to be smaller
- They had to be not significantly worse in the dimension the parameter set is not optimized for; for the F parameter sets, the signatures should not be significantly larger; for the S parameter sets, the signature generation should not be significantly slower.
- The parameter sets should not require additional complexity to the implementation
- The parameter sets should not slow down the signature verification operation. This was considered especially critical for the S parameter sets, as the envisioned usage was for sign-once/verify-many-times applications.
- The parameter sets should not require additional security assumptions on the function family.

We ended up adjusting the h, d, $\log(t)$, k parameters to four of the parameter sets (128S, 128F, 192S, 256F). These changes improve things while leaving the implementation mostly unchanged, and security proofs completely unchanged.

For 128S, we shrink the signature size from 8080 bytes to 7856 bytes; performance tests showed that the signature generation time was largely unchanged (slightly slower, 3%-8%, on the SHA256 and SHAKE256 function families, and slightly faster on the Haraka function family). We also measured an approximate 8% speed up in signature verification.

For 128F, we sped up the signature generation time by approximately 24% (slightly less with SHAKE256, but significantly faster everywhere), and increase the signature size slightly to 17088 bytes. The signature verification time was slowed by about 10%, however for an F parameter set, this was not considered as critical.

For 192S, we shrink the signature size from 17,064 bytes to 16,224 bytes; performance tests showed that signature generation time was actually sped up by about 20% (slightly less on SHAKE256 parameter sets) – this speed up was not a goal of this search, but we could not find a parameter set with a smaller signature without slowing the signature generation process unacceptably. We also measured an approximate 10% speed up in signature verification.

For 256F, we sped up signature generation time by approximately 13% faster, and increase the signature size slightly to 49,856 bytes. The signature verification time is approximately the same as the original.

For the 192F and 256S scenarios, we could not find an alternative parameter set that performed better and met the goals, hence we retain the original parameter sets.

We did consider a number of different strategies (beyond just modifying the h, d, $\log(t)$, k parameters); we ultimately decided that the benefits that brought were not justified by the costs.

- We considered W=256 parameter sets; for the S parameter sets, we could find parameter sets that had rather smaller (circa 10%) signatures. However, this would significantly increase the signature verification time (by a factor of 6x-10x); this was deemed unacceptable.

- We considered other w values that were a power of 2 (32, 64, 128); for the S parameter sets, this would further shrink the signature (for the F parameter sets, $w = 16$ was generally optimal). This would lessen the increase in signature verification time (but there would still be an increase); in addition, this would increase the code complexity modestly (because the WOTS+ code would need to cross bytes when converting a hash into a series of indices). Because of these two costs, this was also considered unacceptable.

- We considered arbitrary w (for example, $w = 43$); this further improved performance moderately. However, this further increased the code complexity (because the WOTS+ code would now need to perform a base conversion); hence this was also rejected.

- We considered replacing the current 'appended complimented checksum' WOTS+ encoding method with a constant sum method (where the sum of the WOTS+ digits was a predetermined value); this method would also ensure that any modification to the WOTS+ digits would require some digit to decrease (and hence such a forgery attempt would require the adversary to compute a second preimage to an F function in that WOTS+ chain). We considered two approaches to performing the hash to constant sum mapping:
  - We could do a direct translation between hash value and constant sum value. However, the algorithms we could find in the literature had significant amounts of complexity, and hence this was rejected.
  - We could do a randomized mapping with rejection sampling. That is, to find a constant sum mapping of WLEN digits, we could generate WLEN-1 random digits (seeded by the message hash we were signing), and check if there was a last digit that made the sum match the target value (and if not, try again with another set of WLEN-1 digits). It turns out that, for the values we would use with our parameter sets, the probability of there being a last digit was reasonably good, and hence the number of expected iterations was acceptable. And, if we model the random digit generation process as 'random', this is secure, and gave reasonably impressive performance. However, this assumption that our random digit generation process (which would be bits extracted from the PRF function) would require an additional security assumption, and hence we had to reject this possibility.

- We also considered including a deliberate slow hash while doing the initial message hash computation; the idea is that if the adversary could not hash as many test messages (we currently assume he could test 2^{seclevel} messages), we could modestly decrease the FORS parameters, which would allow us to decrease the signature size (or select better tradeoffs for the F parameter sets). However, the benefits turned out to be modest (circa 2%) and increased the signature verification time and increased the code complexity modestly. Hence, we rejected this as well.

2 Side-channels Attacks

In addition to the changes in the parameter sets, we also considered a change to improve the side channel resistance against electronic side channels (such as DPA); this change would involve changing how FORS and WOTS+ private values were generated by the signer; instead of using the PRF function of the seed repeatedly, we could use a tree structure (similar to the idea in eprint 2020/960). This tree structure would parallel the SPHINCS$^+$ structure, but values would go in the opposite direction (instead of the value of the parent being derived from the values of the children, the values of the children would be derived from the parent). This structure would ensure that no secret value would be used in more than two contexts, hence frustrating side channel attacks which require more. This could also be done in a way that would not significantly increase the signature time. However, when we put together the details, the complexity required (both in specification and code complexity) was larger than what we were comfortable with. Given this complexity, and also the possibility that an implementation could use a masked PRF implementation with the current method (which has higher cost, but would also address the problem), we left this idea off the table.

3 Security Evaluation

The 'Security Evaluation' section presented an attempt for a tight security reduction for SPHINCS$^+$ that turned out to be flawed; we kept it there for reference and added a disclaimer. The flaw is an artifact of the attempt to prove a tight security reduction for the variant of the Winternitz one-time signature scheme used by SPHINCS$^+$ taken from [2]. It should be noted that the non-tight proof for WOTS$^+$ from [1] still applies. Also, the flaw does not translate into an attack but just demonstrates that the proof made false assumptions. Indeed, at the time of writing we are positive that the problem can be circumvented. Hence, this does not influence our security estimates at all. As part of the disclaimer we briefly outline the flaw and a previous issue (that got fixed since version 2 of this specification) with the security reduction and outline a solution.

4 Bugfix in wots_sign for w = 256

We fixed a bug in the wots_sign function. The line

```
csum = csum << ( 8 - ( ( len_2 * lg(w) ) % 8 ));
```

got changed to

```
if( (lg(w) % 8) != 0) {
  csum = csum << ( 8 - ( ( len_2 * lg(w) ) % 8 ));
}
```

The old code ignored one base-w element of the checksum in case of $w = 256$. This is due to an unlucky interplay between this – in case of $w = 256$ – unnecessary shift and the way the integer to byte conversion is implemented in the function to_byte.

5 New Team Members

We are happy to announce that Ward Beullens and Bas Westerbaan join the SPHINCS$^+$ team.

References

[1] Andreas Hülsing. W-OTS+ – shorter signatures for hash-based signature schemes. In Amr Youssef, Abderrahmane Nitaj, and Aboul-Ella Hassanien, editors, *Progress in Cryptology – AFRICACRYPT 2013*, volume 7918 of *LNCS*, pages 173–188. Springer, 2013. 4

[2] Andreas Hülsing, Joost Rijneveld, and Fang Song. Mitigating multi-target attacks in hash-based signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016*, volume 9614 of *LNCS*, pages 387–416. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. 4