From: Joost Rijneveld <joost@joostrijneveld.nl>
Sent: Tuesday, March 13, 2018 10:03 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: SPHINCS+

Dear all,

It was brought to our attention that the SPHINCS+ reference code contains an error that causes a longer-than-necessary message digest to be generated and signed (contradicting the specification). We have fixed the code and updated it on our web page. Note that, as this also affects the leaf index selection, the fix causes incompatibility with the previous version of the code.

Thanks go to Dorian Amiet for spotting the bug and alerting us to it.

Cheers,
Joost Rijneveld, on behalf of the SPHINCS+ team

From: Thomas Prest <thomas.prest@ens.fr>
Sent: Friday, April 13, 2018 4:28 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: SPHINCS+

Dear all,

This mail is to inform you of a fault injection attack against SPHINCS+ and Gravity-SPHINCS.
In schemes of the SPHINCS family, the signing algorithm essentially consists of reconstructing a bunch of Merkle trees and signing their roots with one-time signatures (OTS).
Introducing one single random fault during the reconstruction of the penultimate Merkle tree has the effect that one OTS signs two different values, which can then be exploited to forge signatures for any message for a computational cost of computing about 2^34 hashes.

More detailed information can be found here:
- Article:
https://eprint.iacr.org/2018/102
- Slides: https://tprest.github.io/Slides/grafting-trees-pqcrypto.pdf
We have informed the submitters about this attack, and to the best of our knowledge they agree with the claims made in our article.

We want to emphasize that this is a fault injection attack: it does not question the security of these schemes when the signing algorithm is executed normally.

Best,
Laurent, Ange and Thomas

From: Christopher J Peikert <cpeikert@alum.mit.edu>
Sent: Thursday, May 24, 2018 3:16 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: SPHINCS+

This comment refers to the SPHINCS+ hash-based signature proposal at https://sphincs.org/

(specification document at https://sphincs.org/data/sphincs+-specification.pdf).

Summary:

1. The specification claims that Theorem 9.1 supports the concrete security of the submission.

2. This claim is incorrect, because the theorem apparently does not apply to the proposed instantiations: it requires the component function F to have a structural property that it almost certainly does not have. (Indeed, it seems hard to find *any* suitable instantiation having the property.)

3. Using a known non-tight (but presumably applicable) security reduction yields much weaker security bounds, by roughly 60 (classical) bits or more.

4. I'm not aware of (nor have I looked for) any attack that violates the claimed security levels. This comment is solely about what known reductions imply (or don't) about the concrete security of the proposed SPHINCS+ instantiations.

Details:

Section 9 claims that "the security of SPHINCS+ is based on standard properties of the used function families" (and a random-oracle assumption), and says that "we give a security reduction for SPHINCS+ underpinning the above claim." Specifically, Theorem 9.1 states a concrete security bound for SPHINCS+, which is tightly related to the security of its components.

In Section 7.1.2 (Table 3), Theorem 9.1 is used to derive the proposed
SPHINCS+ parameters, for classical bit-security levels of roughly 128,
SPHINCS+ 196,
256 (and corresponding quantum ones).

Unfortunately, Theorem 9.1 apparently does not apply to the proposed
SPHINCS+ instantiations -- in particular, the choices of F (Section 7.2)
SPHINCS+ --
because it requires F to have the structural property given in Equation (9), p. 42:

For every key k, every value in the image of F_k : {0,1}^n -> {0,1}^n has at least two preimages.

Even for relatively small n, this property is highly unlikely to hold for even a single random function, much less a huge collection. So it almost certainly does not hold for the concrete F derived from SHA-256, SHAKE-256, and Haraka. Indeed,

it seems difficult to guarantee the property without sacrificing F's practicality or second-preimage resistance (another requirement of Theorem 9.1).

The source of the reduction technique behind Theorem 9.1 is https://eprint.iacr.org/2015/1256 . As explained in Section 5.1 (page 24) the property from Equation (9) is essential to the reduction's tightness:

"This additional requirement is needed to not [have] to use the one-wayness of F. If we had to use the one-wayness of F... This would imply a security loss of roughly h bits."

According to this, such a loose reduction would yield security bounds that are about h >= 60 (classical) bits worse than the claimed security of the proposed SPHINCS+ parameters.

Sincerely yours in cryptography,
Chris

From: A. Huelsing <ietf@huelsing.net>
Sent: Tuesday, May 29, 2018 12:37 PM
To: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: SPHINCS+

Dear Chris, dear list

thanks for your comment. You are absolutely right with your analysis.
We made the same observation a few weeks ago. We currently have a scientific paper on SPHINCS+ in submission where we deal with this differently.

The paper should be online soonish (subject to vacation caused delay).

Best,

Andreas (for the SPHINCS+ team)

From: Santi Vives <santi.x256@gmail.com>
Sent: Friday, November 02, 2018 6:29 PM
To: pqc-comments
Subject: OFFICIAL COMMENT: SPHINCS

Hi all. Bellow is a possible optimization. It is a small change, and it can save some wastefull hash computations. I've already mailed the Gravity-Sphincs team, and it seems to apply to SPHINCS+ as well.

As you know, when using WOTS we pick a compression parameter w that determines the size.

We can have more efficient signatures if we allow two different compression parameters: one for the main part of the signature, and other for the checksum. I like to call this Unbalanced Winternitz since it results in chains of uneven heights.

Consider this:

Instead of defining a compression parameter (w) and computing the size of the signature from that, let's start by defining the final size, and compute the optimal compression parameters allowing different values for the main part and the checksum.

Let's start by defining a size k:
k = ka + kc
Here ka is the size of the main part, and kc is the size of the checksum.

The message is a random value in the range 0...2^256-1.

Since the main part has length ka, we need to encode the message as a number with ka digits. The smallest possible base for that number is:
ba = ceil ((2^256) ^ (1/ka)) = ceil (2^(256/ka))

Now, each chain can take a value in the range 0...ba-1. Then the checksum encodes a value in the range 0...(ba-1)*ka. That's (ba-1)*ka+1 possible values.
So the smallest base for the checksum is:
bc = ceil (((ba - 1) * ka + 1) ^ (1/kc))

Notice that from this we can compute the full cost of the ots (the total number of computed hashes from the bottom of the chains to the top):
ka * (ba - 1) + kc (bc - 1)

So, ka and kc are all we need to know the cost.

We can split a size k into a pair (ka, kc) in any of these ways:
(1, k-1), (2, k-2), ..., (k-1, 1)

So, just by having the size k we can compute the cost for each possible (ka, kc) pair and choose the one that's optimal.

Now, the numbers:

For WOTS+ and Unbalanced WOTS+ with size k=67 we have:
WOTS

w=16, k=67, cost=1005
UWOTS
ka=64, kc=3, ba=16, bc=10, cost=987
(we save 18 hashes per ots)

For WOTS+ and Unbalanced WOTS+ with size k=34 we have:
WOTS
w=256, k=34, cost=8670
UWOTS
ka=32, kc=2, ba=256, bc=91, cost=8340
(we save 330 hashes per ots)

Computing the checksum only takes 1 or 2 divmod extra computations (depending on the case) for the ots in use only, which is negligible compared to the hashes.

Below is some Python code. It uses 4 bases instead (2 for the main part, 2 for the checksum) which is better when the size of main part is not a power of 2.

https://jota.tuxfamily.org
https://jotasapiens.com/research

Best regards,
Santi J. (@jotasapiens)