SPHINCS Interim Report

Vedanth Narayanan
March 13, 2016

Introduction

Over the past 7 weeks, we have been focusing on SPHINCS for our project. Most of the work leading up to the past couple weeks have been reading documents that explain the tools being used in SPHINCS. The single biggest challenge for us was primarily getting acquainted with the material. To properly, and throughly, understand SPHINCS we needed to get caught up with a lot of reading. There were multiple papers that required time and dedication to fully understand. Understanding the tools and technologies is crucial if we want to be successful. On top of this, we had the added challenge of figuring out how to piece together the technologies, and how SPHINCS uses them.
While we had a lot of catching up to do, we also did get a chance to work on something new. The actual focus is transcribed in this paper. With help from Professor Yavuz, our goal was to come up with a simpler signature scheme that withheld the security that can be incorporated into SPHINCS. The scheme we propose is called Lamport+.

Preliminaries

This section is utilized to briefly talk about existing signature schemes. The sole reason for this is so future references to the specific schemes are not ambiguous.

Lamport Signature Scheme

The Lamport Signature Scheme was created by Leslie Lamport in 1989, and it is the simplest signature scheme that exists. It is also the first One-Time signature that was invented. This scheme makes use of a cryptographic hash function that has already been predetermined.
To put it simply, the idea behind the scheme can be split into two separate pieces. Signer first generates the secret keys, public keys, hashes messages, and gets a signature that is passed off to the verifier. Now, the verifier's job more or less is to follow similar steps so they end up with a similar result. Thus, it can be argued that the message and signature could only come from the signer and no one else. The detailed version of the scheme is mentioned in the Lamport+ section.

WOTS

The primary idea behind the scheme is to break the messages into little blocks, that get processed together, and having an input run through a hash function several times. The

number of iterations entirely depends on the message that needs to be signed.
WOTS was built on top of the Lamport signature scheme, and the expectation is for it to be intuitive in its logic, but it's not the case. The complexity of the scheme is heavily influences by figuring out the number of iterations necessary for a value to go through the hash function.

WOTS+

WOTS+ is very similar to WOTS, expect for the addition of XORing random elements every time a value is iterated over hash function. In the key generation phase, WOTS+ generates a set of random numbers that will serve for XORing. Just like the keys are split into chunks, so are the random elements. They get incorporated in the following recursive chaining function

$$c_k^i(x, \mathbf{r}) = f_k(c_k^{i-1}(x, \mathbf{r}) \oplus r_i) \tag{1}$$

Th equation is strictly $i > 0$, but in the case of $i = 0$, $c_0^k(x, \mathbf{r}) = x$. The equation is clever in that it makes sure to XOR different values for every iteration.

Lamport+ Signature Scheme

Lamport+ Signature Scheme is the new scheme we are proposing. It not only brings the simplicity of the original Lamport scheme, but also pulls in elements of the WOTS+ scheme. Our hope is that the original scheme's security is withheld, if not enhanced. Please note that the security of the proposed scheme has not been proven, but it can very well be inferred from the previous. Similar to how WOTS+ introduces XORing of randomized elements to WOTS, the same principle is introduced to Lamport. The following is meant to give an idea of how Lamport+ would work as a One Time Signature, before we extrapolate it to hash chains. Take note of the following variables:

- $i \in \{0, 1\}$, denotes bit pair
- $n \in \mathbb{N}$, the security parameter
- $s \xleftarrow{\$} \{0, 1\}^m$, seed for the PRNG
- PRNG $g : \{0, 1\}^m \to \{0, 1\}^n, m < n$
- Cryographic hash function $f_k : \{0, 1\}^n \to \{0, 1\}^n, k \in \mathcal{K}$

Chaining Function

$$\begin{aligned} c^i(x) &= f_k(x_{1,n} \oplus g(x_{n+1,n+m})), i > 0 \\ c^0(x) &= x_{1,n} \end{aligned} \tag{2}$$

The idea for the chaining function, $c : \{0, 1\}^{n+m} \to \{0, 1\}^n$, is borrowed from WOTS+. Intuitively, the function just splits the seed from the passed in argument, and uses it

to XOR a PRN. The subscript, $x_{a,b}$ defines the specific bits used, i.e. $x_a, ..., x_b$. $a < b \leq (n+m)$ is assumed to be true.

Key Generation
Input: Security parameter n
Output: Secret key sk, Public key pk

$$\begin{aligned} sk &= ((x_1^0, x_1^1), ..., (x_n^0, x_n^1)) \\ &= ((c_1^0(x)||s_0, c_1^0(x)||s_1), ...(c_n^0(x)||s_0, c_n^0(x)||s_1)), x \in \{0,1\}^{n+m} \\ pk &= ((y_1^0, y_1^1), ..., (y_n^0, y_n^1)) \\ &= ((c_1^1(x_1^0)||s_0, c_1^1(x_1^1)||s_1), ...(c_n^1(x_n^0)||s_0, c_n^1(x_n^0)||s_1)) \end{aligned} \tag{3}$$

Signature Generation
Input: Secret key sk, hashed message M
Output: Signature σ

$$\sigma = (\sigma_1, ..., \sigma_n) = (x_1^{M_1}, ..., x_n^{M_n}), M_i \in \{0,1\} \tag{4}$$

Signature Verification
Input: Public key pk, hashed message M, and signature σ
Output: Verification pass or fail

$$pk = (y_1^i, ..., y_n^i) \stackrel{?}{=} (c_1^1(\sigma_1)||\sigma_{n+1,n+m}, ..., c_1^1(\sigma_n)||\sigma_{n+1,n+m}) \tag{5}$$

Lamport+ Hash Chain

Hash chains aid One-Time signatures to be used multiple times with a single key. This can be applied to the Lamport+ signature scheme. The underlying idea here is that after the public key is generated in the scheme, another set of keys are generated based on the previous public key getting hashed. As you can guess, expect the last public key of the chain, all keys before act as a private key.
There are two important things to make note of here. First off, we introduce a new parameter, which we call $l \in \mathbb{N}$, $l > 0$. This parameter helps keep track of how long the chain is, or how many messages can be signed. Secondly, the public keys will hold new PRNG seeds. These seeds will help produce XOR values for the following set of public keys, and this is not used to verify the signature at that level. The concept is better understood once all the steps are explained.

Key Generation
Input: Security parameter n

Output: Secret key sk, Public key pk

$$\begin{aligned} sk_l &= ((x_1^{0,l}, x_1^{1,l}), ..., (x_n^{0,l}, x_n^{1,l})) \\ &= ((c_1^l(x_1^{0,l-1})||s_0, c_1^l(x^{1,l-1})||s_1), ..., (c_n^l(x_n^{0,l-1})||s_0, c_n^l(x^{1,l-1})||s_1)), \\ &\quad \text{where } s \xleftarrow{\$} \{0,1\}^m, \text{ and } l, n \in \mathbb{N} \end{aligned} \tag{6}$$

Signature Generation
Input: Secret key sk, hashed message M, Lamport parameter l
Output: Signature σ, Updated Lamport parameter l

$$\sigma = (\sigma_1, ..., \sigma_n) = (x_1^{M_1}, ..., x_n^{M_n}), M_i \in \{0,1\} \tag{7}$$

Signature Verification
Input: Public key pk, hashed message M, and signature σ
Output: Verification pass or fail

$$pk = (y_1^i, ..., y_n^i) \stackrel{?}{=} (c_1^1(\sigma_1)||\sigma_{n+1,n+m}, ..., c_1^1(\sigma_n)||\sigma_{n+1,n+m}) \tag{8}$$

Security Considerations

Why do we XOR?

Lamport+ Hash Tree

Note that this section hasn't been fully developed yet, and it is still in the works. If Lamport+ is going to get incorporated into SPHINCS, then it needs to be in a tree fashion. The idea we have is that the leafs of a Binary hash tree have the public keys of the hash chains. If there are four leaves, then there are 4 hash chain associated with each of the leaves. The parent node is a hash of the children hash nodes concatenated with each other, and XORed with a set of random elements for every level. The root node serves is the global public key.
When a signature is used from one of the leaves, then the tree needs to be computed again. Note that the whole tree does not need to be recomputed, but only the path to the specific leaf that used a signature. This idea hasn't been thought over entirely yet, but we believe that it should carry over.

Conclusion

In this report, we presented the Lamport+ Signature Scheme, which is much simpler than WOTS and WOTS+. WOTS+ is currently being used in SPHINCS. SPHINCS is indeed stateless, but if Lamport+ were to be integrated instead, the system would become stateful. We went through an example or two to make sure that the scheme worked and

it was possible, but have not spent more time on it. We want to go over it again, and see how it fairs against edge cases and such. The plan is also to try to implement it, and integrate it with SPHINCS to see the results.

References

1. Merkle, Ralph C. "A certified digital signature." Advances in Cryptology-CRYPTO'89 Proceedings. Springer New York, 1989.
2. Buchmann, Johannes, et al. "On the security of the Winternitz one-time signature scheme." Progress in Cryptology-AFRICACRYPT 2011. Springer Berlin Heidelberg, 2011. 363-378.
3. Hulsing, Andreas. "W-OTS+ Shorter signatures for hash-based signature schemes." Progress in Cryptology-AFRICACRYPT 2013. Springer Berlin Heidelberg, 2013. 173-188.
4. Bernstein, Daniel J., et al. "SPHINCS: practical stateless hash-based signatures." Advances in Cryptology-EUROCRYPT 2015. Springer Berlin Heidelberg, 2015. 368-397.
5. Buchmann, Johannes, et al. "CMSS - An improved Merkle signature scheme." Progress in Cryptology-INDOCRYPT 2006. Springer Berlin Heidelberg, 2006. 349-363.
6. Buchmann, Johannes, Erik Dahmen, and Andreas Hulsing. "XMSS-a practical forward secure signature scheme based on minimal security assumptions." Post-Quantum Cryptography. Springer Berlin Heidelberg, 2011. 117-129.
7. Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In Burton Kaliski, editor, Advances in Cryptology - CRYPTO '97, volume 1294 of Lecture Notes in Computer Science, pages 470-484. Springer Berlin / Heidelberg, 1997. 10.1007/BFb0052256.
8. https://github.com/joostrijneveld/SPHINCS-py
9. https://github.com/CertiVox/MIRACL