SPHINCS Interim Report

Vedanth Narayanan

March 18, 2016

Introduction

Over the past 10 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
- $s \overset{\$}{\leftarrow} \{0, 1\}^m$, seed for the PRNG
- PRNG $g : \{0, 1\}^m \rightarrow \{0, 1\}^n, m < n$
- Cryographic hash function $f_k : \{0, 1\}^n \rightarrow \{0, 1\}^n, k \in \mathcal{K}$

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, $c : \{0, 1\}^{n+m} \rightarrow \{0, 1\}^n$, 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

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

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 generate XOR values to help produce the following set of public keys. The concept is better understood once all the steps are explained.

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

$$\begin{aligned} sk_k &= ((x_1^{0,k}, x_1^{1,k}), ..., (x_j^{0,k}, x_j^{1,k})) \\ &= ((c_1^k(x_1^{0,k-1})||s_0, c_1^k(x^{1,k-1})||s_1), ..., (c_j^k(x_j^{0,k-1})||s_0, c_j^k(x_j^{1,k-1})||s_1)), \\ &\quad \text{where } s \overset{\$}{\leftarrow} \{0,1\}^m, 0 \leq k \leq l, 1 \leq j \leq n \end{aligned} \tag{6}$$

Like you could guess, there will no separate public key that will be generated. If the hash chain is properly generated, this is how it should look:

$$\begin{aligned}
&sk_0 = ((x_1^{0,0}, x_1^{1,0}), ..., (x_n^{0,0}, x_n^{1,0})) = ((c_1^0(x)||s_0, c_1^0(x)||s_1), ...(c_n^0(x)||s_0, c_n^0(x)||s_1))\\
&sk_1 = ((x_1^{0,1}, x_1^{1,1}), ..., (x_n^{0,1}, x_n^{1,1})) = ((c_1^1(x_1^{0,0})||s_0, c_1^1(x_1^{1,0})||s_1), ...(c_n^1(x_n^{0,0})||s_0, c_n^1(x_n^{1,0})||s_1))\\
&\quad\vdots\\
&sk_l = ((x_1^{0,l}, x_1^{1,l}), ..., (x_n^{0,l}, x_n^{1,l})) = ((c_1^1(x_1^{0,l-1})||s_0, c_1^1(x_1^{1,l-1})||s_1), ...(c_n^1(x_n^{0,l-1})||s_0, c_n^1(x_n^{1,l-1})||s_1))
\end{aligned} \tag{7}$$

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

$$\begin{aligned}
&\sigma = (\sigma_1, ..., \sigma_n) = (x_1^{M_1, l-1}, ..., x_n^{M_n, l-1}), M_i \in \{0, 1\}\\
&l = l - 1
\end{aligned} \tag{8}$$

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

$$pk = (y_1^l, ..., y_n^l) \overset{?}{=} (c_1^1(\sigma_1)||\sigma_{n+1,n+m}, ..., c_1^1(\sigma_n)||\sigma_{n+1,n+m}) \tag{9}$$

Lamport+ Hash Tree

The Lamport+ Hash Tree incorporates both the Lamport+ hash chains, and Merkle Hash Trees. The scheme shown here is loosely based on XMSS trees from [6]. The tree allows us sign $\left(\sum_{i=0}^{2^H-1} l_i\right)$ messages, with a single global public key. There are some caveats if the tree is going to be used effectively. Signatures are generated from the leftmost chain to the rightmost. To keep track of what the chain the next signature needs to be generated from, we introduce an index variable i. Once $i = 2^H - 1$, we can determine that signatures for all chains have been used once, and the tree needs to be rehashed using the next set of keys from each of the hash chains. Keep note of the following:

- H : Height of tree
- i : Index of a node starts from the left and goes to right
- $f_k : \{0, 1\}^{2n} \to \{0, 1\}^n$
- $N : \{0, 1\}^{n+m}$, stands for Node

Key Generation
Begin by choosing an $H \geq 1$ variable first. This will allow us to have 2^H number of

Lamport+ hash chains. The hash chains now need to be created to continue following the algorithm. The following equation shows to construct inner nodes. The leaf nodes will be the hashes of the public key of the hash chains initially. The leaves will change once signatures are generated.
Input: 2^H hashed public keys of Lamport+ hash chains
Output: Lamport+ Hash Tree

$$\begin{aligned} N_{i,h} &= f_k((N_{2i,h-1} \oplus g(s_{2i,h-1}))||(N_{2i+1,h-1} \oplus g(s_{2i+1,h-1})))||s_{i,h} \\ &\text{where } 0 \leq i < 2^H, 0 \leq h < H \end{aligned} \tag{10}$$

Signature Generation

A signature is first derived the lamport chain, and the tree is reconstructed to take into account the signature. Every time the tree is reconstructed, a new $s \overset{\$}{\leftarrow} \{0,1\}^m$ is generated. A is a list of sibling nodes on the path to the tree root. It's important to remember that once a signature is derived, another shouldn't be generated until the first one has been verified. If there is another generated, then the first signature will not properly verify.
Input: Hashed message M, lamport chain l_i
Output: Authentication path A, signature σ_i, index i

$$\begin{aligned} &\text{if } l_i > 0, \text{ continue, else } i = i+1, \text{ start over} \\ &\sigma_i = (x_1^{M_1, l-1}, ..., x_n^{M_n, l-1}) \\ &N_{i,h} = \begin{cases} f_k(\sigma_i||(N_{2i+1,h-1} \oplus g(s_{2i+1,h-1})))||s_{i,h} & \text{, left child} \\ f_k((N_{2i,h-1} \oplus g(s_{2i,h-1}))||\sigma_i)||s_{i,h} & \text{, right child} \end{cases} \\ &A = (A_0, ..., A_{H-1}) \end{aligned} \tag{11}$$

Signature Verification

There are two main steps to look out for here. First, verifying the Lamport+ OTS using the signature that is passed in. Secondly, traversing the hash tree to help check the authenticity for the signature. This algorithm was created with help from [6] and [8].
Input: Authentication path A, signature σ, index i, hashed message M, root node of hash tree pk
Output: Verification pass or fail

$$\begin{aligned} &sk_l = (y_1^l, ..., y_n^l) \overset{?}{=} (c_1^1(\sigma_1)||\sigma_{n+1,n+m}, ..., c_1^1(\sigma_n)||\sigma_{n+1,n+m}) \\ &P_h = \begin{cases} f_k((P_{h-1} \oplus g(s))||(A_{h-1} \oplus g(s))) \oplus s, & \text{if} \lfloor \frac{i}{2^h} \rfloor \equiv 0 \bmod 2; \\ f_k((A_{h-1} \oplus g(s))||(P_{h-1} \oplus g(s))) \oplus s, & \text{if} \lfloor \frac{i}{2^h} \rfloor \equiv 1 \bmod 2; \end{cases}, \\ &\text{where } 0 \leq h < H \\ &P_0 = N_{i,0} \\ &pk \overset{?}{=} P_{H-1} \end{aligned} \tag{12}$$

Security Considerations

The idea started by looking at the differences between WOTS and WOTS+. They are very similar, so the difference between them is really what makes WOTS+ more secure than the other. The difference all comes down to the fact that WOTS+ performs an Exclusive OR (XOR) operation before a value is hashed. Our job was to take that idea and apply it to Lamport, and ideally the security would be enhanced.

One of the most *elusive* questions in the project was why the authors of WOTS+ XORed random values before hashing values. Even after thoroughly reading the paper multiple times the reason for XORing random values and the security enhancement are not explained. We found out that the paper on XMSS was written only a couple of years before WOTS+, and the author of WOTS+ also coauthored the XMSS paper. It is in XMSS that the idea for XORing random values came from for WOTS+. This operation, once again, is not properly explained in XMSS. The only reference to the bitmasks points to "Collision-resistant hashing: Towards making UOWHFs practical"[7], written by Mihir Bellare and Phillip Rogaway. It was clear that the XOR trick came from them. They have collectively written multiple papers on collision resistant hashing, and key hashing functions. Upon looking through some of the papers, it seemed to first come about in "Keying Hash Functions for Message Authentication" in 1996. This idea was suggested to them by Adi Shamir, one of the cofounders of RSA. Despite the origins of the idea, the thing to take note of is the variety of ways it gets used.

There are two reasons why random values are XORed in the collision-resistant hashing paper. The first reason being able to bring down the key sizes. The more interesting reason is for key scheduling, especially when it comes to compression functions. From our understanding, setting up keys for compression functions can be tedious, so the XOR operations helps with fixing the keys for the "rounds" in a compression function. The security proof was arduous to understand, but the gist is captured here. Essentially, they prove that by using the XOR trick, the compression function is Target-Collision Resistant (TCR). TCR is second preimage resistant, which essentially the Birthday Paradox.

The following is just a small example. The adversary essentially expects the second collision to be just like the first one, but that is not the case. A new K is introduced in the first collision, and the only way the adversary will get it is by brute forcing it. The adversary needs something to serve as a base, and the adversary does not get that here.

$$\begin{cases} XLH_{K,K_1,...,K_j}(M_1,...,M_j) & = XLH_{K,K_1,...,K_j}(M'_1,...,M'_j) \\ XLH_{K,K_1,...,K_{j-1}}(M_1,...,M_{j-1}) & \neq XLH_{K,K_1,...,K_{j-1}}(M'_1,...,M'_{j-1})||M'_j \end{cases} \tag{13}$$

Like mentioned, WOTS+ makes use of XORing random values. From our understanding, it didn't need to incorporated at all, however it does make the scheme more secure. While there is a small overlapping piece, having to work with hashing, we don't see a significant necessity for the trick other than to make the scheme more secure.

Conclusion

While an implementation for Lamport+ Hash Tree was created, it definitely can be made better. Given enough time, the algorithm could be optimized to not only be faster, but cut down on costs, and even make a smaller memory footprint. This isn't entirely a promise, but a potential future work that would be worth investing time in.

We presented the Lamport+ Signature Scheme along with hash chains and hash trees in this report. The scheme itself is much simpler than both WOTS and WOTS+. The complications arrive when trying to integrate Lamport+ with the Merkle hash trees. The primary reason is

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. Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. In Johannes Buchmann and Jintai Ding, editors, Post-Quantum Cryptography, volume 5299 of Lecture Notes in Computer Science, pages 109–123. Springer Berlin / Heidelberg, 2008.

9. Johannes Buchmann, L. C. Coronado Garcia, Erik Dahmen, Martin Doring, and Elena Klintsevich. CMSS – an improved Merkle signature scheme. In INDOCRYPT, volume 4329 of Lecture Notes in Computer Science, pages 349–363. Springer, 2006.