SPHINCS Interim Report

Daniel Kirkpatrick
Vedanth Narayanan
February 18, 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. We needed to revisit older papers multiple times to get a better picture of it. Despite that, the actual focus is transcribed in this paper. With help from Professor Yavuz, our goal was to come up with something 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 the new scheme before we extrapolate it to hash chains.

Key Pair Generation: n is the number of random pairs to be generated. Random element $r \in \{0,1\}^n$ is chosen. The secret key is $sk = ((sk_{0,0}, sk_{0,1}), (sk_{1,0}, sk_{1,1}), ..., (sk_{n,0}, sk_{n,1}))$. Each key is n-bits long. Let $f_k : \{0,1\}^n \to \{0,1\}^n | k \in \mathcal{K}_n$. The cryptographic hash function f_k outputs a n-bit value. Public keys are derived by $pk = f_k(sk_{0,1} \oplus r)$. The whole set of sk is run through the hash function, so get a bijective public key set. The public key is the following

$$\begin{aligned} pk &= ((pk_{0,0}, pk_{0,1}), (pk_{1,0}, pk_{1,1}), ..., (pk_{n,0}, pk_{n,1})) \\ &= ((f_k(sk_{0,0} \oplus r), f_k(sk_{0,1} \oplus r)), (f_k(sk_{1,0} \oplus r), f_k(sk_{1,1} \oplus r)), ..., \\ &\quad (f_k(sk_{n,0} \oplus r), f_k(sk_{n,1} \oplus r))) \end{aligned} \tag{2}$$

Signature Generation: The message to be signed is m. This messaged is hashed by $h(m) = f_k(m \oplus r)$. For consistency purposes, we assume that the message size and r are the same. The output of the hash is now n-bits. Based on every single bit (0 or 1), the corresponding key from a bit pair is selected from the secret keys. These keys make

up the signature of the message.

Signature Verification: At this point, the verification is almost trivial. The verifier first obtains the hash of the message (once again, w). Based on the bits of the hash, they pick out the corresponding keys from public key of the signer. Now the signature that the signer gave the verifier is hashed, and ideally the values should equal the values that the verifier picked out of the public key. If and when the values don't match is when we know something is wrong. Also, make a note that when something is hashed, the random element is XORed in. Thus, the random element r is passed off by the signer.

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 underlining 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.
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 many times new public keys have been derived. The second change involves generating a new randomized element for every new hashed public key. The random element from the previous level should not be used again. Now $\mathbf{r}$ is a set, $\mathbf{r} = (r_1, r_2, ..., r_l)$. Once again, this concept is borrowed from WOTS+, where the random element changes based on the iteration number.
To properly use this hash chain, it's important to know the end of the chain. Imagine the length of the chain is 10. This chain can sign 10 messages. To verify a message, the 9th key down the chain is released. The l parameter should then be decremented. When another message needs to be signed, then the 8th key should be released, and l should be decremented. Once l hits 0, there are no more messages that can be used. The random elements set does not need to be modified, because the subset $r_{1,...,l}$ automatically changes.

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.

Challenges

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.

Conclusion

References