Summer School on Post-Quantum Cryptography 2017, TU Eindhoven
Exercises on Hash-based Signature Schemes

1 Lamport

Consider Lamport's one-time signature scheme. Let messages be of length n and assume that Alice has published $2n$ hash values as her public key and knows $2n$ secret bit strings, representing her private key, which lead to those $2n$ hash results. Alice uses this signature system multiple times with the same key. Analyze the following two scenarios for your chances of faking a signature on a message M:

1. You get to see signatures on random messages.
2. You get to specify messages that Alice signs.

You may not ask Alice to sign M in the second scenario.
How many signatures do you need on average in order to construct a signature on M? How many signatures do you need on average to be able to sign any message? Answer these questions in both scenarios.

2 Optimized Lamport

Recall the optimized Lamport scheme. A detailed description can be found in Appendix A. Let message length $m = 16$ and internal hash length $n = 256$.

1. What is the length of the checksum in bits?
2. What is the bit size of signatures, secret, and public keys?
3. Let $M = 1001010011010101_2$. Which secret and public key elements become part of the signature?
4. Convince yourself (best by proof) that whenever at least one bit in the encoded message B flips from 1 to 0, at least one other bit flips from 0 to 1. Convince yourself that a single bit flip does not necessarily cause just a single bit flip in the other direction.

3 WOTS

Recall the Winternitz one-time signature scheme (WOTS). A detailed description can be found in Appendix B. Let Winternitz parameter $w = 16$ and message length $m = 16$. Assume we are internally using a hash function with $n = 256$.

1. What is the length of the checksum in base w representation?
2. What is the bit size of signatures, secret, and public keys?
3. How many calls to F are required for key generation, signing and verification?
4. How would speed and sizes change if we changed w to 8?
5. Let $M = 1001010011010101_2$. Draw the imaginary graph of a WOTS key pair with the above parameters ($w = 16$) and circle the nodes which become part of the signature.
6. Convince yourself (best by proof) that whenever at least one value in the encoded message B is increased, at least one other value in B is decreased. Convince yourself that an increase in one value might cause a decrease in several other values.

4 HORS

The HORS (Hash to Obtain Random Subset) signature scheme is an example of a few-time signature scheme. It has integer parameters k, t, and n, uses a hash function $H : \{0,1\}^* \to \{0,1\}^{k \cdot \log_2 t}$ and a length preserving one-way function $\mathrm{F} : \{0,1\}^n \to \{0,1\}^n$. For simplicity assume that H is surjective.

To generate the key pair a user picks t strings $\mathsf{sk}_i \in \{0,1\}^n$ and computes $\mathsf{pk}_i = \mathrm{F}(\mathsf{sk}_i)$ for $0 \le i < t$. The public key is $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1, \dots, \mathsf{pk}_{t-1})$; the secret key is $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1, \dots, \mathsf{sk}_{t-1})$.

To sign a message $M \in \{0,1\}^*$ compute $H(M) = (h_0, h_1, \dots, h_{k-1})$, where each $h_i \in \{0, 1, 2, \dots, t-1\}$. The signature on M is $\sigma = (\mathsf{sk}_{h_0}, \mathsf{sk}_{h_1}, \mathsf{sk}_{h_2}, \dots, \mathsf{sk}_{h_{k-1}})$.

To verify the signature, compute $H(M) = (h_0, h_1, \dots, h_{k-1})$ and $(\mathrm{F}(\mathsf{sk}_{h_0}), \mathrm{F}(\mathsf{sk}_{h_1}), \mathrm{F}(\mathsf{sk}_{h_2}), \dots, \mathrm{F}(\mathsf{sk}_{h_{k-1}}))$ and verify that $\mathrm{F}(\mathsf{sk}_{h_i}) = \mathsf{pk}_{h_i}$ for $0 \le i < t$.

1. Let $n = 256$, $t = 2^5$, and $k = 3$. How large (in bits) are the public and secret keys? How large is a signature? How many different signatures can the signer generate for a fixed key pair as $H(M)$ varies? Ignore that sk-values could collide.
2. The same public key can be used for $r+1$ signatures if H is r-subset-resilient, meaning that given r signatures and thus r vectors $\sigma_j = (s_{h_{j,0}}, s_{h_{j,1}}, s_{h_{j,2}}, \dots, s_{h_{j,k-1}})$, $1 \le j \le r$ the probability that $H(M')$ consists entirely of components in $\{h_{j,i} | 0 \le i < k, 1 \le j \le r\}$ is negligible.

   Even for $r = 1$, i.e. after seeing just one typical signature, an attacker has an advantage at creating a fake signature. What are the options beyond exact collisions in H?
3. Let $n = 256$, $t = 2^5$, and $k = 3$. Let M be a message so that $H(M) = (h_0, h_1, h_2)$ satisfies that $h_i \neq h_j$ for $i \neq j$. You get to specify messages that Alice signs. You may not ask Alice to sign M.

(a) State the smallest number of HORS signatures you need to request from Alice in order to construct a signature on M.

(b) How many calls to H does this require on average? You should assume that H and F do not have additional weaknesses beyond having too small parameters.

(c) Explain how you could use under 1000 evaluations of H if you are allowed to ask for two signatures.

5 The BDS Algorithm

The BDS algorithm is an algorithm that offers a time-memory trade-off for tree traversal which refers to authentication path generation for Merkle-tree signatures. It's basic building block is the TreeHash algorithm. Both were discussed during the lecture. You can find a description of Treehash in Appendix C. A description of BDS can be found in `http://www.cdc.informatik.tu-darmstadt.de/~dahmen/papers/hashbasedcrypto.pdf` on page 28.

1. Simulate the TreeHash algorithm for a tree of height 4 on paper. Consider the case where the whole tree is computed. Write down the state of the stack in each iteration of the loop.

2. Simulate the BDS algorithm for a tree of height 6 with parameter $k = 2$. Write down the state of all internal storage variables (Auth, Keep, Retain, Treehash.h - all internal variables).

6 Eliminate a State

Goldreich proposed a stateless version of a previous proposal by Merkle. In this scheme, one does not use a Merkle tree but a binary tree of one-time key pairs. Each one-time secret key on an inner node is used to sign the one-time public keys of its child nodes, The one-time key pairs on the leaf nodes are used to sign messages, the root node is the public key of the scheme. To sign hash values of length m, the scheme needs a tree of height $h = m$. To sign a message digest M, the M-th leaf node is used (taking M as integer). The signature contains all the one-time signatures on the path from the M-th leaf to the root. Assume the used one-time signature scheme is WOTS.

1. Compute signature and key size of the scheme for $m = 256$, $w = 16$.

2. What is the speed of key generation, signing and verification?

3. What is the trade-off you can achieve using a hyper-tree approach?

A Optimized Lamport Description

The optimized Lamport scheme uses a one-way function $\mathrm{F} : \{0,1\}^n \to \{0,1\}^n$, and signs m bit messages. The secret key consists of $\ell = m + \log m + 1$ random bit strings

$$\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$$

of length n. The public key consists of the ℓ outputs of the one-way function

$$\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell) = (\mathrm{F}(\mathsf{sk}_1), \ldots, \mathrm{F}(\mathsf{sk}_\ell))$$

when evaluated on the elements of the secret key. Signing a message $M \in \{0,1\}^m$ corresponds to first computing and appending a checksum to M to obtain the message mapping $\mathrm{G}(M) = B = M \| C$ where $C = \sum_{i=1}^{m} \neg M_i$. The signature consists of the secret key element if the corresponding bit in B is 1, and the public key element otherwise:

$$\sigma = (\sigma_1, \ldots, \sigma_\ell) \text{ with } \sigma_i = \begin{cases} \mathsf{sk}_i & \text{, if } B_i = 1, \\ \mathsf{pk}_i & \text{, if } B_i = 0. \end{cases}$$

To verify a signature the verifier checks whether the full public key is obtained by hashing the elements of the signature that correspond to 1 bits in B:

$$\text{Return 1, iff } (\forall i \in [1, \ell]) : \mathsf{pk}_i = \begin{cases} \mathrm{F}(\sigma_i) & \text{, if } B_i = 1, \\ \sigma_i & \text{, if } B_i = 0. \end{cases}$$

B WOTS Description

WOTS uses a length-preserving (cryptographic hash) function $\mathrm{F} : \{0,1\}^n \to \{0,1\}^n$. It is parameterized by the message length m and the Winternitz parameter $w \in \mathbb{N}, w > 1$, which determines the time-memory trade-off. The two parameters are used to compute

$$\ell_1 = \left\lceil \frac{m}{\log(w)} \right\rceil, \quad \ell_2 = \left\lfloor \frac{\log(\ell_1(w-1))}{\log(w)} \right\rfloor + 1, \quad \ell = \ell_1 + \ell_2.$$

The scheme uses $w - 1$ iterations of F on a random input. We define them as

$$\mathrm{F}^a(x) = \mathrm{F}(\mathrm{F}^{a-1}(x))$$

and $\mathrm{F}^0(x) = x$.
Now we describe the three algorithms of the scheme:

Key generation algorithm ($\mathsf{kg}(1^n)$)**:** On input of security parameter 1^n the key generation algorithm choses ℓ n-bit strings uniformly at random. The secret key $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)$ consists of these ℓ random bit strings. The public verification key pk is computed as

$$\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell) = (\mathrm{F}^{w-1}(\mathsf{sk}_1), \ldots, \mathrm{F}^{w-1}(\mathsf{sk}_\ell))$$

Signature algorithm ($\mathsf{sign}(1^n, M, \mathsf{sk})$): On input of security parameter 1^n, a message M of length m and the secret signing key sk, the signature algorithm first computes a base w representation of M: $M = (M_1 \dots M_{\ell_1}),\ M_i \in \{0, \dots, w-1\}$. Next it computes the check sum

$$C = \sum_{i=1}^{\ell_1} (w - 1 - M_i)$$

and computes its base w representation $C = (C_1, \dots, C_{\ell_2})$. The length of the base-w representation of C is at most ℓ_2 since $C \leq \ell_1(w-1)$. We set $B = (B_1, \dots, B_\ell) = M \parallel C$. The signature is computed as

$$\sigma = (\sigma_1, \dots, \sigma_\ell) = (\mathrm{F}^{B_1}(\mathsf{sk}_1), \dots, \mathrm{F}^{B_\ell}(\mathsf{sk}_\ell)).$$

Verification algorithm ($\mathsf{vf}(1^n, M, \sigma, \mathsf{pk})$): On input of security parameter 1^n, a message (digest) M of length m, a signature σ and the public verification key pk, the verification algorithm first computes the B_i, $1 \leq i \leq \ell$ as described above. Then it does the following comparison:

$$\mathsf{pk} = (\mathsf{pk}_1, \dots, \mathsf{pk}_\ell) \stackrel{?}{=} (\mathrm{F}^{w-1-B_1}(\sigma_1), \dots, \mathrm{F}^{w-1-B_\ell}(\sigma_\ell))$$

If the comparison holds, it returns **true** and **false** otherwise.
Remark. The difference between the basic WOTS as described above and the advanced variants proposed in recent work is how F is iterated. For these exercises this is of no relevance. Hence, we stick to the easiest variant.

C TreeHash Description

TreeHash is a space efficient method to generate authentication paths if the are needed in order. In the following description, LeafCalc is a method that generates a leaf, e.g., in the case of MSS it will generate the respective one-time public key and hash it.

Input: Stack, leaf index ϕ
Output: Updated Stack

N= LeafCalc (ϕ);
while *top node on* Stack *has same height as* N **do**
  $N \longleftarrow \mathrm{H}((\mathsf{Stack}.pop()||N))$;
end
Stack.$push(N)$;
return Stack

Algorithm 1: TreeHash