CS 355 – Topics in Cryptography May 9, 2018

Lecture 12: Post-Quantum Cryptography and Hash-based Signatures

Instructors: Henry Corrigan-Gibbs, Sam Kim, David J. Wu

1 LWE in Hermite Normal Form

Lattice-based key exchange is an important topic that we did not cover in lecture. In problem set 4, we will study a key-exchange protocol. However, to prove security, we will need a simple variant of the original formulation of the LWE problem.

$\mathsf{LWE}_{\mathsf{HNF}}(n, m, q, \chi_B)$: Let $n, m, q, B \in \mathbb{N}$ be positive integers, and let χ_B be a B-bounded distribution over $\mathbb{Z}_q$. For a given adversary $\mathcal{A}$, we define the following two experiments:

Experiment b $(b = 0, 1)$:

- The challenger computes

$$\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{m \times n}, \quad \mathbf{s} \leftarrow \chi_B^n, \quad \mathbf{e} \leftarrow \chi_B^m, \quad \mathbf{b}_0 \leftarrow \mathbf{A} \cdot \mathbf{s} + \mathbf{e}, \quad \mathbf{b}_1 \xleftarrow{R} \mathbb{Z}_q^m,$$

  and gives the tuple $(\mathbf{A}, \mathbf{b}_b)$ to the adversary.

- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. Then, we define $\mathcal{A}$'s advantage in solving the LWE problem for the set of parameters n, m, q, χ_B to be

$$\mathsf{LWEAdv}_{n,m,q,\chi_B}[\mathcal{A}] := \Big| \Pr[W_0] - \Pr[W_1] \Big|.$$

The only difference between $\mathsf{LWE}(n, m, q, \chi_B)$ and $\mathsf{LWE}_{\mathsf{HNF}}(n, m, q, \chi_B)$ is that in $\mathsf{LWE}(n, m, q, \chi_B)$, the vector $\mathbf{s}$ is sampled uniformly at random, but in $\mathsf{LWE}_{\mathsf{HNF}}(n, m, q, \chi_B)$, the vector $\mathbf{s}$ is sampled from the B-bounded distribution χ_B. The two problem $\mathsf{LWE}(n, m, q, \chi_B)$ and $\mathsf{LWE}_{\mathsf{HNF}}(n, m, q, \chi_B)$ are known to be equivalent. Hence, if LWE is hard, then $\mathsf{LWE}_{\mathsf{HNF}}$ is hard, and vice versa.

2 Post Quantum Cryptography

Today, we will briefly take a look at post-quantum cryptography. Let's first take a look at the current landscape of cryptography.

Source		**Assumptions**		**Primitives**
Block Cipher Design	$\Rightarrow$	AES is secure	$\Rightarrow$	Symmetric Crypto:
Hash Function Design		SHA is secure		PRF, Hash Functions
CS Theory	$\Rightarrow$	DLog: CDH, DDH, ...	$\Rightarrow$	Public Key Crypto:
Mathematics		Factor: RSA, str-RSA, ...		PKE, DS, Key-Exchange

Let's see what happens to our assumptions if we had a quantum computer. The assumptions that we use for symmetric cryptography are generally considered secure even against quantum computers. However, the attacks on these assumptions do improve slightly. Therefore, we just need to increase the key space (for AES) or the output space (for SHA) suitably. For assumptions that we use for public key cryptography, they are completely broken by quantum computers. Therefore, we must find new assumptions.

AES. Let $\mathcal{K}$ be a key space for a version of AES (or any PRF/PRP). Then given any output of AES, we can recover the key $k \in \mathcal{K}$ using Grover's algorithm in time $|\mathcal{K}|^{1/2}$. Currently, to get $\lambda = 128$ bits of classical security, we set $\mathcal{K} = \{0,1\}^{128}$. If we want $\lambda = 128$ bits of quantum security, we can set $\mathcal{K} = \{0,1\}^{256}$.

SHA. Let $H : \{0,1\}^* \rightarrow \mathcal{Y}$ be any hash function. Then, using Grover's algorithm, we can find a collision for H in time $|\mathcal{Y}|^{1/3}$. Therefore, if we want $\lambda = 128$ bits of quantum security, we can set $\mathcal{Y} = \{0,1\}^{384}$. Note that even in the classical setting, we have a birthday attack on H that can find a collision in time $|\mathcal{Y}|^{1/2}$. This is why we set $\mathcal{Y} = \{0,1\}^{256}$ (SHA-256) today.

DLog and Factor. Shor's algorithm completely solves these problems in (quantum) polynomial time. Hence, we must find new assumptions that can replace them in the quantum world.

2.1 Alternatives

Therefore, in general, symmetric cryptography is in good shape even in the quantum world. We just need to increase the parameters suitably. For public key cryptography, we need new alternatives for Public Key Encryption, Digital Signatures, and Key Exchange. In many scenarios, we can combine Key Exchange with symmetric cryptography to emulate Public Key Encryption. Therefore, a lot of the attention is given to constructing Digital Signatures and Key Exchange.

There are generally 5 families of assumptions that people currently studying for post-quantum cryptography.

Hash-based Cryptography.

- **Good**:
  - No additional algebraic assumption other than the fact that standard hash functions are collision-resistant.
- **Bad**:
  - Large signature sizes.
  - Cannot do key-exchange.

Lattice Cryptography.

- **Good**:
  - Well studied in the theory community
  - Simple and fast operations.
- **Bad**:
  - Things are big (for key-exchange).

Code-based Cryptography.

- **Good**:
  - Very old (as old as RSA).
  - Simple and fast operations.
- **Bad**:
  - Things are big.

Isogeny-based Cryptography.

- **Good**:
  - Based on elliptic curves (people are already very familiar with them).
  - Things are small.
- **Bad**:
  - Operations are slow.
  - Very new.

Multivariate Cryptography.

- **Good**:
  - Simple and fast operations.
- **Bad**:
  - Things are big.

3 Hash-based Signatures

Let's get a quick taste of hash-based signatures. The approach that we are going to take is as follows:

1. Construct one-time signatures (OTS).
2. Upgrade OTS to a full-fledged signature scheme.

4 Lamport One-Time Signatures

Fix a hash function $H : \{0,1\}^\lambda \to \{0,1\}^\lambda$ (we only need one-wayness of the hash function, so we can make the output space to be $\{0,1\}^\lambda$). Fix a message space $\mathcal{M} = \{0,1\}^n$.

- $\mathsf{KeyGen}(1^\lambda)$: Sample $2n$ uniformly random elements $x_{i,b} \xleftarrow{\text{R}} \{0,1\}^\lambda$ and set them to be the signing key:

$$\mathsf{sk} = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{n,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{n,1} \end{pmatrix}.$$

  Then, hash each of the elements in the signing key and set them to be the public key:

$$\mathsf{pk} = \begin{pmatrix} y_{1,0} = H(x_{1,0}) & y_{2,0} = H(x_{2,0}) & \cdots & y_{n,0} = H(x_{n,0}) \\ y_{1,1} = H(x_{1,1}) & y_{2,1} = H(x_{2,1}) & \cdots & y_{n,1} = H(x_{n,1}) \end{pmatrix}.$$

- $\mathsf{Sign}(\mathsf{sk}, m)$: Let $m = m_1 m_2 \cdots m_n \in \mathcal{M}$. Then, set the signature as follows

$$\sigma = (x_{1,m_1}, x_{2,m_2}, \ldots, x_{n,m_n}.$$

- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: Let $\sigma = (\sigma_1, \ldots, \sigma_n)$. For $i = 1, \ldots, n$, check whether

$$y_{i,m_i} = H(\sigma_i).$$

  If all check pass, then accept. Otherwise, reject.

Claim 4.1. *If an adversary gets access to 1 signature, then it cannot forge signatures.*

Proof Idea. The signatures consist of the preimages of the hash function. In order to forge a new signature, the adversary must come up with its own preimages that hashes to elements in the public key. Therefore, if the adversary could forge, then we can use it to invert the hash function H.

Claim 4.2. *If an adversary gets access to 2 signatures, then it can forge new signatures.*

Proof Idea. If the adversary gets access to the signatures for $m = 00 \cdots 0$, and $m' = 11 \cdots 1$, then it gets access to the whole signing key.

4.1 From One-Time to Many-Time Signatures

We can construct a q-time secure signature scheme from any one-time signature scheme $(\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ as follows.

- $\mathsf{KeyGen}(1^\lambda)$: Generate q one-time signature secret-public key pairs $(\mathsf{sk}_1, \mathsf{pk}_1), \ldots, (\mathsf{sk}_q, \mathsf{pk}_q) \leftarrow \mathsf{KeyGen}'(1^\lambda)$. It sets

$$\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_q)$$
$$\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_q).$$

- $\mathsf{Sign}(\mathsf{sk}, m, \mathsf{st} = i)$: The signing algorithm keeps a local state st (initially set to 1) that keeps track of how many messages it signed. On input $\mathsf{st} = i$, it signs $\sigma_i \leftarrow \mathsf{Sign}'(\mathsf{sk}_i, m)$. It sets $\sigma = (\sigma_i, i)$. It also updates the state $\mathsf{st} = \mathsf{st} + 1$.

- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: Let $\sigma = (\sigma_i, i)$. The verification algorithm accepts if $\mathsf{Verify}'(\mathsf{pk}_i, m, \sigma_i)$ accepts. Otherwise, it rejects.

The downside of the construction above is that (i) public key size is large, (ii) the signing algorithm is stateful and (iii) the signature size is large. We can remove the downside (i) by using a Merkle tree.

4.2 Merkle Tree Construction

A Merkle tree is a way of hashing a set of strings $S = \{\mathsf{str}_1, \ldots, \mathsf{str}_n\}$ into one small digest h^* such that given any string str_i, one can give a short proof that certifies that str_i was contained in the original set of strings S that was used to create h^*.

$$P(h^*, \mathsf{str}_i, S) \qquad \xrightarrow{\pi_i} \qquad V^*(h^*, \mathsf{str}_i)$$

Using Merkle Trees, we can construct a digital signature scheme as follows.

- $\mathsf{KeyGen}(1^\lambda)$: Generate q one-time signature secret-public key pairs $(\mathsf{sk}_1, \mathsf{pk}_1), \ldots, (\mathsf{sk}_q, \mathsf{pk}_q) \leftarrow \mathsf{KeyGen}'(1^\lambda)$. It sets
$$\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_q, \mathsf{pk}_1, \ldots, \mathsf{pk}_q), \qquad \mathsf{pk} = h^*.$$

- $\mathsf{Sign}(\mathsf{sk}, m, \mathsf{st} = i)$: The signing algorithm keeps a local state st (initially set to 1) that keeps track of how may messages it signed. On input $\mathsf{st} = i$, it signs $\sigma_i \leftarrow \mathsf{Sign}'(\mathsf{sk}_i, m)$. It also computes a proof π_i that certifies that pk_i with respect to h^*. It sets $\sigma = (\sigma_i, \mathsf{pk}_i, \pi_i)$.

- $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$: Let $\sigma = (\sigma_i, \mathsf{pk}_i, \pi_i)$. The verification algorithm verifies the signature σ_i with respect to pk_i, and also verifies the proof π_i for pk_i with respect to h^*. If both of these checks go through, then accept. Otherwise, reject.

Security intuition. Let $\tilde{\sigma} = (\tilde{\sigma}, \tilde{\mathsf{pk}}, \tilde{\pi})$ be a forgery that an adversary submits.

- By the security of the Merkle tree hashing, $\tilde{\mathsf{pk}}$ must correspond to some public key pk_i that was originally generated by $\mathsf{KeyGen}(1^\lambda)$:
$$\tilde{\mathsf{pk}} = \mathsf{pk}_i.$$

- By the security of the original signature scheme in Section 4.1, $\tilde{\sigma}$ must an invalid signature with respect to $\tilde{\mathsf{pk}} = \mathsf{pk}_i$.

Remarks. This is the general flavor of hash-based signature. It is possible to make upgrade one-time signatures to a full-fledged signature scheme where the number of signatures it can support is unbounded and the signing algorithm is also not stateful. We can also shrink the size of the signatures significantly with additional bags of tricks. The textbook Boneh-Shoup Chapter 14 has a very nice description of how this can be done, so please take a look if you are interested.