University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 37

1 Signature Schemes for Multiple Messages

Before continuing to discuss the construction of signature schemes, we summarize the results of the last lecture:

Theorem 1 *A 1-time signature scheme can be constructed from any one-way function.*

(This theorem is evident from the Lamport 1-time signature scheme we have discussed.)

Theorem 2 *A 1-time signature scheme which can sign messages longer than the public key can be constructed from (any one-way function and) any collision-resistant hash function.*

(This theorem follows from the note at the end of the previous lecture. We note that any collision-resistant hash function is already a [weak] one-way function and therefore collision-resistant hash functions are a sufficient assumption for the theorem.)

Theorem 3 *For any (polynomial) M, a signature scheme for signing M messages can be constructed from any 1-time signature scheme.*

This theorem follows from the second-to-last note of the previous lecture. The given construction, however, had a number of drawbacks that we seek to remove in the next few lectures. These included:

- It was required to know in advance the number M of messages to sign.
- The public and secret key grow in size as $O(M)$.
- The signer was required to maintain and update state information.

1.1 A Path-Based Signature Scheme

We now give a scheme that avoids some of these problems. Our scheme will be based on any secure 1-time signature scheme $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ which signs messages longer than the public key, and works as follows:

1. To generate the public/secret key, simply run $\mathcal{K}$ to generate public key PK_0 and secret key SK_0.

2. We give a slightly informal description of the signing algorithm and assume the interested reader could make it more formal. To sign the first message m_1, the signer does the following: (1) run $\mathcal{K}$ to generate public key PK_1 and secret key SK_1 (we stress that $\mathcal{K}$ *must* be randomized; note further that $PK_0 \neq PK_1$ and $SK_0 \neq SK_1$ with all but negligible probability [why?]). (2) Compute $\sigma_1 \leftarrow \mathsf{Sign}_{SK_0}(m_1|PK_1)$ (where "|" denotes concatenation). Finally, (3) output signature $\langle PK_1, \sigma_1 \rangle$. The signer also stores the values $(PK_1, SK_1, m_1, \sigma_1)$ as part of the internal state.

   To sign the i^{th} message m_i, the signer will use PK_{i-1}. The signer does the following: (1) run $\mathcal{K}$ to generate PK_i and SK_i; (2) compute $\sigma_i \leftarrow \mathsf{Sign}_{SK_{i-1}}(m_i|PK_i)$; and (3) output signature $\langle PK_1, m_1, \sigma_1, \ldots, PK_{i-1}, m_{i-1}, \sigma_{i-1}, PK_i, \sigma_i \rangle$. The signer also stores the values $(PK_i, SK_i, m_i, \sigma_i)$ as part of the internal state.

3. We discuss verification in the general case. Remember that all the verifier has is the public key PK_0 (the "master" public key of the signer), a message m_i, and a purported signature $\langle PK_1, m_1, \sigma_1, \ldots, PK_{i-1}, m_{i-1}, \sigma_{i-1}, PK_i, \sigma_i \rangle$. The verifier checks that $\mathsf{Vrfy}_{PK_0}(m_1|PK_1, \sigma_1) \stackrel{?}{=} 1, \ldots, \mathsf{Vrfy}_{PK_i}(m_i|PK_i, \sigma_i) \stackrel{?}{=} 1$. Only if all of these are true does the verifier accept the signature as valid.

What is going on is that each time the signer signs a message m_i, the signer also "refreshes" its key by generating (PK_i, SK_i) which will be used to sign the next message. However, we do not want an adversary to be able to generate and use its own PK_i' (if this occurs, then the adversary can obviously forge a signature on any message it likes); to prevent this, the signer *certifies* each public key with respect to the previous public key PK_{i-1}. The verifier checks that each public key has been so certified back to the "root" public key PK_0. We emphasize that it is essential here that the original scheme $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ allow for signing messages longer than the public key (why?).

We do not give a rigorous proof of security here, although such a proof should largely be evident. Note that each public key generated by the signer is used to sign only *one* message; since $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ is assumed to be a secure 1-time signature scheme, the scheme as a whole is secure. A sketch of the proof is as follows: assume the adversary outputs signature $\langle PK_1', m_1', \sigma_1', \ldots, PK_{i-1}', m_{i-1}', \sigma_{i-1}', PK_i', \sigma_i' \rangle$ on message m_i' which was never signed by the real signer. In particular, we have $m_i' \neq m_i$ (assuming without loss of generality that the signer has signed at least i messages). Let $\langle PK_1, m_1, \sigma_1, \ldots, PK_{i-1}, m_{i-1}, \sigma_{i-1}, PK_i, \sigma_i \rangle$ be the i^{th} signature issued by the real signer on message m_i. If $PK_{i-1} = PK_{i-1}'$ then the adversary has forged a signature with respect to public key PK_{i-1} — but this is impossible by security of the 1-time signature scheme. If $PK_{i-1} \neq PK_{i-1}'$ but $PK_{i-2} = PK_{i-2}'$ then the adversary has forged a signature with respect to PK_{i-2} — which is again impossible. Continuing in this way, we see that the adversary must have forged a signature with respect to *some* public key $PK_0, \ldots, PK_{i-1}$; however, this should be impossible for any such key by the security of the 1-time scheme.

Let's see whether this scheme offers any improvements over previous schemes:

- The number of messages to be signed does not need to be known in advance. This scheme allows signing any arbitrary (polynomial) number of messages.

- The public key size is constant (i.e., $O(1)$), independent of the total number of messages signed. Unfortunately, although the initial secret key is also constant size, the state maintained by the signer grows as $O(M)$, where this is the number of messages signed *thus far*. We may note, however, that most of the state information (i.e., all the $\{PK_i, m_i, \sigma_i\}_{1 \le i \le M}$) need not be *secret*. This may save on the cost of storing this information.

- The signer still needs to maintain state when signing messages.

- We also note a new disadvantage of the present scheme: both the signature length and the verification time grow as $O(M)$. This is in contrast to the previous scheme which had essentially $O(1)$ signature length and verification time. It is worth mentioning, though, that the signing time in the current scheme is essentially $O(1)$.

1.2 A Tree-Based Signature Scheme

We may note that the signature scheme of the previous section essentially constructs a path from a root PK_0 to a "current key" PK_i; this path is certified using the 1-time signature scheme and the current leaf PK_i is used to sign the next message m_{i+1}. This led to signature size/verification time $O(M)$. It seems natural that we might do better by considering a tree-based approach instead of a path-based approach.

We sketch this generalization here. We now denote the signer's public key by PK and the initial secret key by SK. We will construct a tree in which every node has two children: the children of PK are PK_0, PK_1 and the children of any node PK_w (for any binary string w) are PK_{w0} and PK_{w1}. We use binary notation to represent signed messages; thus, the first message is denoted m and the second, third, and fourth messages are denoted m_0, m_1, m_{01}, respectively. To sign the first message m, the signer generates two public keys PK_0, PK_1, computes $\sigma \leftarrow \mathsf{Sign}_{SK}(m|PK_0|PK_1)$, and outputs (PK_0, PK_1, σ_0). The signer also stores associated secret keys SK_0 and SK_1, as well as m and σ, as part of the internal state. To sign the second message m_0, the signer will use key SK_0 and sign with respect to PK_0. Thus, the signer generates public keys PK_{00}, PK_{01}, computes $\sigma_0 \leftarrow \mathsf{Sign}_{SK_0}(m_0|PK_{00}|PK_{01})$, and outputs $\langle PK_0, PK_1, m, \sigma, PK_{00}, PK_{01}, \sigma_0 \rangle$.

Now, to sign the third message m_1, the signer will use SK_1 and sign with respect to PK_1. Thus, the signer generates PK_{10}, PK_{11}, computes $\sigma_1 \leftarrow \mathsf{Sign}_{SK_1}(m_1|PK_{10}|PK_{11})$, and outputs $\langle PK_0, PK_1, m, \sigma, PK_{10}, PK_{11}, \sigma_1 \rangle$. Note that *the length of the signature does not grow linearly with the number of messages signed*! Instead, the signature length grows logarithmically. It should also be clear that verification time grows logarithmically as well.

Thus, the scheme has the following properties:

- The number of messages to be signed does not need to be known in advance.

- The public key is of size $O(1)$. The internal state grows as $O(M)$, but as mentioned in the previous section most of this state does not need to be kept secret. (Using an improved traversal of the tree it is possible to reduce the state to $O(\log M)$; however, since we show below how to reduce the state to $O(1)$, we do not go into the details.) The signature length and verification time grows as $O(\log M)$. The signing time is essentially $O(1)$.

- The signer still must maintain state, although we discuss below how this can be avoided.

For completeness, we mention how the signer can reduce the size of its (total) storage to $O(1)$, at the expense of increasing the signing time to $O(\log M)$. In the Lamport 1-time signature scheme, the secret key is just a sequence of random values. Also, the public key is a deterministic function of the secret key, and signatures are also deterministic functions of the message and the secret key. So, to avoid storing all the public keys, secret keys, and signatures we can do the following: Imagine a full binary tree as before, except that we now make two changes: (1) we imagine that the tree is fixed in advance (and thus assume for simplicity that the number of messages to be signed is known in advance) and (2) we sign actual messages only using the leaves of the tree (as opposed to before, where every node of the tree was used to sign messages). Note that using only the leaves still gives a scheme with signature length/verification time $O(\log M)$.

In a mental experiment, we can imagine that the signer stores the complete tree (that is, for every node in the tree, the signer stores SK_w; as mentioned above, given this the signer can generate PK_w and a signature σ_w on $PK_{w0}|PK_{w1}$ deterministically). It should be clear that the signer can then sign messages as they come in, using the corresponding leaf of the tree (the signer will need to maintain state to remember the number of messages it has signed so far). However, we note an easy way to avoid storing the entire tree: simply have the signer store a random seed s for a PRF F. Instead of choosing secret key SK_w at random, simply set $SK_w = F_s(w)$. Now, the secret keys can be dynamically generated *as needed*. And the storage has been reduced to $O(1)$ — simply a seed for a PRF. The secret keys are now no longer random, but pseudorandom; however, since F is a PRF, this will not affect the security of the scheme.

We mention that in the above description, the signer must know the number of messages to be signed in advance and must also maintain state (in particular, the number of messages signed thus far). It is possible to remove all of these drawbacks (if you are interested, please ask me); we state this as in a theorem summarizing what we have so far.

Theorem 4 *Based on any 1-time signature scheme which can sign messages longer than the public key, we can construct a signature scheme for signing any (polynomial) number of messages M with the following properties (k represents a security parameter for the scheme):*

- *The number of messages to be signed does not need to be known in advance.*
- *The public key and secret key have size $O(1)$. The signature length, verification time, and signing time are $O(\log^2 k)$.*
- *The signer is not required to maintain any state.*

Since we know how to construct 1-time signature schemes that can sign messages longer than the public key based on any collision-resistant hash function, this shows that CRHFs are sufficient to construct a signature scheme with the above properties. We will see in the next class how to base the construction on any one-way function.