Introduction to Cryptography 3/08/2018

Lecture 16: Digital Signatures

Instructor: Vipul Goyal *Scribe: Andrew Zigerelli*

1 Digital Signatures

Thus far, we've already seen message authentication codes (**MAC**) to deal with the problem of ciphertext malleability. The idea was that a Alice and Bob agree on a secret key, k to be used to choose a PRF f_k from an exponentially large family of PRFs $\{f_l\}_l$. Alice can send a both a message, m, and a MAC, $\sigma = f_k(m)$ to Bob. Bob can verify by calculating the MAC himself and see that it matches; if it doesn't, he should reject the message. Since PRFs take fixed-size inputs, we can construct a MAC by first hashing the message (using a cryptgraphic resistant hash function) and then using the hash as the input to the PRF; this scheme is known as hash then mac (**HMAC**). This gives us more flexibility, assuming the existence of CRHF h which takes as input strings of any length. We note, however, that only those in possesion of the private key k can verify the integrity of the message; the receiver and sender have the same cryptographic knowledge. We would like an authentication scheme in which those without knowledge of the private key could verify. Thus, we want a MAC like construction with PKE flavor, which is what a digital signature is.

Definition 1 ***Digital Signature*** *A digital signature consists of 3 PPT algorithms*

- $Gen(n) \to (sk, pk)$, *where n is the security parameter*
- $Sign(sk, m) \to \sigma$
- $Verify(pk, m, \sigma) = \begin{cases} 1, & \text{(accept)} \\ 0, & \text{(reject)} \end{cases}$

The scheme must satisfy completeness:

$$\Pr[Verify(pk, m, Sign(sk, m)) = 1] = 1 \tag{1}$$

For the digital signature scheme to be secure, we would like to guarantee that a receiver can be confident that message actually originated from the sender (as opposed to being created by an adversary), and that the message wasn't tampered with. We can view "tampering" as generating a new message m' from a valid message m. Thus, in both cases, we wish to prevent *forgery*. We formalize the notion below by defining a forgery game, and we want the chance of forgery to be negligile. That is,

$$\Pr[\mathcal{A} \text{ wins the forgery game}] \leq \text{negl(n)} \tag{2}$$

Definition 2 ***Forgery Game*** *The game is defined between a challenger,* $\mathtt{C}$*, and an adversary,* $\mathcal{A}$*.*

1. $\mathtt{C}$ *randomly generates* (pk, sk) *using* $Gen(n)$*, sends* pk *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *picks* $\{m_i\}_{i=1}^{poly(n)}$*, asks* $\mathtt{C}$ *to sign them, and receives* $\{\sigma_i\}_{i=1}^{poly(n)}$*, where* $\sigma_i = Verify(sk, m_i)$
3. $\mathcal{A}$ *outputs* (m, σ)*.* $\mathcal{A}$ *wins if:* $\forall i, m \neq m_i$ *and* $Verify(pk, m, \sigma) = 1$

2 One Time Signatures

We can also define a weaker notion of secure signatures by only allowing the adversary to ask for one signing. These are called **One Time Signatures (OTS)**. Leslie Lamport invented such a scheme in 1979.

Definition 3 ***Lamport's OTS*** *Given a OWF f, the signature scheme is defined as follows:*

- *Gen(n):*
$$sk = \begin{bmatrix} x_1^0 & x_2^0 & \dots & x_n^0 \\ x_1^1 & x_2^1 & \dots & x_n^1 \end{bmatrix} \tag{3}$$
where $\forall i \in \{1, 2, ..., n\}, \forall b \in \{0,1\}, x_n^b \overset{\$}{\leftarrow} \{0,1\}$
$$pk = \begin{bmatrix} y_1^0 & y_2^0 & \dots & y_n^0 \\ y_1^1 & y_2^1 & \dots & y_n^1 \end{bmatrix} \tag{4}$$
where $y_i^b = f(x_i^b)$

- $Sign(sk, m) : \{0,1\}^{2n} \times \{0,1\}^n \to \{0,1\}^n$
$$(sk, m) \mapsto \sigma = x_1^{m[1]}||x_2^{m[2]}||\dots||x_n^{m[n]} \tag{5}$$

- $Verify(pk, m, \sigma) = \begin{cases} 1, & \text{if } \forall i, y_i^{m[i]} = f(x_i^{m[i]}) \\ 0, & \text{o.w.} \end{cases}$

To prove Lamport's OTS is secure, we will proceed by contradiction, showing that we can use an adversary $\mathcal{A}$ which breaks the signature scheme to construct an adversary $\mathcal{B}$ to break the OWF f. How should we proceed? Recall that since $\mathcal{A}$ is a forger, given a $(m, \sigma), \mathcal{A}$ produces (m', σ') s.t. $m' \neq m$. For Lamport's scheme, more specifically, given

$$\begin{aligned} m &= m_1||m_2||\dots||m_n \\ \sigma &= x_1^{m[1]}||x_2^{m[2]}||\dots||x_n^{m[n]} \end{aligned} \tag{6}$$

$\mathcal{A}$ produces

$$\begin{aligned} m' &= m'_1||m'_2||\dots||m'_n \\ \sigma' &= x_1^{m'[1]}||x_2^{m'[2]}||\dots||x_n^{m'[n]} \end{aligned} \tag{7}$$

Because the messages differ, $\exists j$ s.t. $m[j] \neq m'[j]$. Further, by construction of sigma 5, $\mathcal{A}$ outputs $x_j^{m'[j]}$, and $f(x_j^{m'[j]}) = y_j^{m'[j]}$. If we could construct the scheme s.t. $y_j^{m'[j]} = y$, then we have inverted our OWF f given y!. Since we cannot control the message m $\mathcal{A}$ has signed nor the message m' that $\mathcal{A}$ produces, we cannot always invert. But fortunately, we can invert with a non-negligble probability, as we will see.

Proof. First, we create our adversary $\mathcal{B}$. Given a $OWF f$ and $y \in \text{Im}(f)$, $\mathcal{B}$ creates the following OTS scheme:

1. $\mathcal{B}$ generates (sk, pk) as in 3 and 4.

2. $\mathcal{B}$ samples $j \xleftarrow{\$} \{1, 2, ..., n\}, c \xleftarrow{\$} \{0, 1\}$

3. $\mathcal{B}$ modifies pk s.t $y_j^c = y$. For example, if $c = 0$,

$$pk = \begin{bmatrix} y_1^0 & y_2^0 & \dots & \mathbf{y} & \dots & y_n^0 \\ y_1^1 & y_2^1 & \dots & y_j^1 & \dots & y_n^1 \end{bmatrix} \tag{8}$$

4. By assumption to the contrary, $\exists$ PPT forger $\mathcal{A}$ which breaks this scheme with noticeable probability. $\mathcal{B}$ queries $\mathcal{A}$.

5. $\mathcal{A}$ generates $m = m_1 || \dots || m_n$ and asks for a signature from $\mathcal{B}$. $\mathcal{B}$ can has all the information to construct the signature as long as $m[j] \neq c$. By construction, if $m[j] = c$, $\mathcal{B}$ would have to produce x s.t. $f(x) = y$, which $\mathcal{B}$ doesn't know. For example, see 8 if $c = 0$. If $m[j] = c$, $\mathcal{B}$ outputs fail.

6. If $\mathcal{B}$ doesn't fail in the previous step, it will output m', σ'. If $m'[j] = c$, then $\sigma'[i] = x_i^{m'[j]} = x_i^c$, and by construction of the scheme, $f(x_i^c) = y$, so $\mathcal{B}$ inverts f. If $m'[j] \neq c$, $\mathcal{B}$ outputs fail.

Then,

$$\begin{aligned} \Pr[\mathcal{B} \text{ succeeds}] &= \Pr[\mathcal{A} \text{ succeeds} \wedge m[j] \neq c \text{ in Step 5} \wedge m'[j] = c \text{ in Step 6}] \\ &= \Pr[\mathcal{A} \text{ succeeds}]\Pr[m[j] \neq c \text{ in Step 5}]\Pr[m'[j] = c \text{ in Step 6}] \\ &> (\text{noticeable(n)})(\tfrac{1}{2})(\tfrac{\frac{1}{2}}{1-(\frac{1}{2})^n}) \geq \text{noticeable}(n) \end{aligned} \tag{9}$$

■

We remark that the equality in 9 holds by independence. $\Pr[m'[j] = c \text{ in Step 6}] = \frac{\frac{1}{2}}{1-(\frac{1}{2})^n}$ for the following reason. $\Pr[m'[j] = c \text{ in Step 6}] = \Pr[m'[j] \neq m[j] | m \neq m']$, by Step 5 and the fact $\mathcal{A}$ produces a forgery $m' \neq m$. By Baye's Theorem,

$$\begin{aligned} \Pr[m'[j] \neq m[j] | m \neq m'] &= \tfrac{\Pr[m \neq m' | m[j] \neq m[j]]\Pr[m[j] \neq m'[j]]}{\Pr[m \neq m']} \\ &= \tfrac{(1)(\frac{1}{2})}{1 - \Pr[m = m']} \\ &= \tfrac{\frac{1}{2}}{1-(\frac{1}{2})^n} \end{aligned} \tag{10}$$

Remark 1 *Lamport's OTS is insecure if the adversary can ask the challenger just two messages to be signed.*

$\mathcal{A}$ just asks for $m_0 = 0^n$ and $m_1 = 1^n$ to be signed. $\mathtt{C}$ returns $\sigma_0 = x_1^0 || x_2^0 || \dots || x_n^0$ and $\sigma_1 = x_1^1 || x_2^1 || \dots || x_n^1$, so $\mathcal{A}$ now has the entire private key sk and can forge any message.

Remark 2 *Lamport's scheme works for fixed message length. We can modify it to work for arbitray length messages by using a CRHF*

$h : \{0, 1\}^* \to \{0, 1\}^n$. Then modify the signing algorithm to compute $\sigma = x_1^{h(m)[1]} || \dots || x_n^{h(m)[n]}$. This scheme is potentially less secure because the attack surface is larger. However, we give the proof intuition as follows. $\mathcal{A}$ generates m and receives σ_m. To break the scheme, $\mathcal{A}$ must either

1. Find m' s.t. $h(m) = h(m')$. $\mathcal{A}$ wouldn't need to ask for the signing in this case, as then $\sigma_{m'} = \sigma_m$, so $\mathcal{A}$ has found a forgery. However, this amounts to finding a collision in the CRHF h, which is assumed to be hard.

2. Produce an m' s.t. $h(m) \neq h(m') \implies \sigma_m \neq \sigma_{m'}$. Thus, $\mathcal{A}$ has produced a forgery without finding a hash collision. This violates Lamport's OTS security. (You can basically use the same proof as given before, except replace the signature scheme with the new signature scheme using the CRHF).

3 Multi Messsage Signature Schemes

We can construct a signature scheme for signing multiple messages using the OTS. The idea is before signing m under (pk, sk) in the OTS scheme, we can generate an additional pair (pk', sk') and append pk' to m. Then the recipient, when verifying m, also has a new verified public key for a future message. We then iterate the process, as shown below.

To sign n messages:

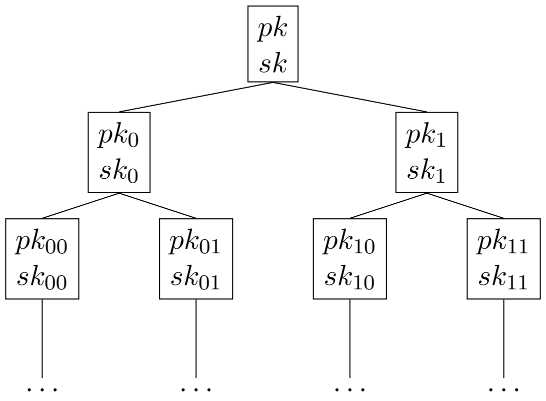
- $Gen(n) \to (sk_0, pk_0)$
- $Gen(n) \to (sk_1, pk_1), \sigma_0 = Sign(sk_0, m_0 || pk_1)$
- $Gen(n) \to (sk_2, pk_2), \sigma_1 = Sign(sk_1, m_1 || pk_2)$
- $\ldots$
- $Gen(n) \to (sk_n, pk_n), \sigma_{n-1} = Sign(sk_{n-1}, m_{n-1} || pk_n)$

Each message m_j and signature σ_j is stored. To verify message m_k:

- $Verify(pk_k, m_k || pk_{k+1}, \sigma_k)$. But must verify pk_k
- $Verify(pk_{k-1}, m_{k-1} || pk_k, \sigma_{k-1})$. But must verify pk_{k-1}
- $\ldots$
- $Verify(pk_1, m_1 || pk_2, \sigma_1)$. But must verify pk_1
- $Verify(pk_0, m_0 || pk_1, \sigma_0)$. Done since pk_0 is originally generated key, assumed to be valid.

The idea for proving this scheme is multi message secure is that each (pk, sk) is only used once, so to break this scheme, an adversary would have to break OTS.

Notice that for n messages, we require $O(n)$ storage. We can consider this storage as the size of the signature, since by construction, all of it is used for verifciation. So signature size grows linearly. The overhead is because we want to link each message to the originally generated (sk, pk) pair since we assume OTS is secure. Thus, for the current message, the last signature "attests" to current public key validity, and to prove the attester is valid, we must verify its public key; to verifiy the attester's public key we need another attester, which we should also verify, etc. We can change the attestation topology to be a tree, as seen below.



Each leaf in a balanced tree can be a message; a tree of length k has $n = 2^k$ nodes. Thus, to handle n messages, we verify by walking the tree from the leaf to the root, which requires $k = \log(n)$ verifications, so the signature size has logarithmic growth with the number of messages. Also, we "build" a tree from the leftmost leaf. We generate the k signatures needed to verify the path, as well as the siblings, since by construction, each node verifies its children. So, we only need to store the nodes of the tree that actually lead to message leaves, plus intermediate siblings of theses nodes.

We still would like to have a scheme in which the signature size is constant per message, no matter how many messages we need to sign. Fortunately, we can do so with trapdoor permutations.

4 RSA Signature Scheme

For encryption, the public key in RSA is usually used to send messages to a receiver, and the private key is used for decryption. However, in the mathematical construction, the public and private key are essentially used the same way; they are inverse to each other, mod $\phi(n)$. We simply just choose one to be public, and the other private.

Thus, we can also use this construction to sign messages. In this case, the encryption of the message under the private key is the signature, and anyone can decrypt the signature using the public key. The algorithms are described below:

- $Gen(n)$: Randomly choose large primes p, q s.t. $p \neq q$, set $N = pq$. Choose e s.t. $gcd(e, \phi(n)) = 1$. Find d s.t. $ed = 1 \mod \phi(n)$. $pk = (e, N), sk = d$
- $Sign(sk, m) : \sigma = m^d \mod N$
- $Verify(pk, m, \sigma)$: Check if $m = \sigma^e = (m^d)^e = m \mod N$

Remark 3 *RSA is a homomorphic signature scheme.*

This means it preserves some operations, in this case, multiplication. Thus, $\forall m1, m2$

$$Sign_{RSA}(sk, m_1 m_2) = Sign_{RSA}(sk, m_1) Sign_{RSA}(sk, m_1) \tag{11}$$

Such a scheme is not secure. An advesary needs only two queries to a signer to create a forgery (forge $m_1 m_2$). We can try to modify the scheme to not have this problem.

Definition 4 ***RSA Signature Scheme*** *Given a CRHF h, which is public,*

- *$Gen(n)$: Randomly choose large primes p, q s.t. $p \neq q$, set $N = pq$. Choose e s.t. $gcd(e, \phi(n)) = 1$. Find d s.t. $ed = 1 \mod \phi(n)$. $pk = (e, N), sk = d$*
- *$Sign(sk, m) : \sigma = h(m)^d \mod N$*
- *$Verify(pk, m, \sigma)$: Check if $h(m) = \sigma^e = (h(m)^d)^e = h(m) \mod N$*

Can we forge a new message, given two signatures of known messages, like we did before? More formally, given $m1, m2, \sigma_1 = h(m_1)^d \mod N, \sigma_2 = h(m_2)^d \mod N$, does the following hold?

$$h(m_1 m_2)^d = h(m_1)^d h(m_2)^d \mod N \tag{12}$$

If it holds, does it hold only with only a negligible probability? If we assume that h is a truly random function, then the above would hold with only a negligible probability, which would make our scheme secure. The issue is that a CHRF h is not truly random. This brings us to another issue in cryptography.

Assumption 1 ***Random Oracle Model*** *For a cryptographic scheme with CHRF h*

1. *Assume that h is truly random. Prove security for this case.*
2. *Replace h by the crytographic hash function. Assume the scheme is still secure.*

Some cryptographers do not like this model as it is not logically valid to assume that security in step 1 implies security in step 2. In fact, there exists constructions which are provably secure in step 1 and are provably insecure in step 2. However, in practice, some use the random oracle model to create efficient cryptoschemes. For example, NIST has a contest in which cryptographers submit hash function candidates, which require stronger assumptions then what we've seen thus far. Contestants try to attack other's submissions, such as showing the output is not random, which would compromise its security.