Notes for Lecture 21

Scribed by Anand Bhaskar, posted May 1, 2009

Summary

Today we show how to construct an inefficient (but efficiently verifiable) signature scheme starting from a one-time signature scheme.

Next time we shall see how to make it efficient using a pseudorandom function.

From One-Time Signatures to Fully Secure Signatures

Assume we have a (t, ϵ)-secure one-time signature scheme (G, S, V) such that if m is the length of messages that can be signed by S, then the length of public keys generated by $G()$ is at most $m/2$.

(Lamport's signatures do not satisfy the second property, but in Lecture 20 we described how to use a collision-resistant hash function to turn Lamport's scheme into a scheme that can sign longer messages. We can arrange the parameters of the construction so that the hash-and-sign scheme can sign messages at least twice as long as the public key.)

We describe a scheme in which the key generation and signing have exponential complexity; later we will see how to reduce their complexity.

- Key Generation: run $G()$ $2^{m+1} - 1$ times, once for every string $a \in \{0,1\}^*$ of length at most m, and produce a public key / secret key pair (pk_a, sk_a).

  It is convenient to think of the strings a of length at most m as being arranged in a binary tree, with a being the parent of $a0$ and $a1$, and the empty string ϵ being the root.

  - Public Key: pk_ϵ (where ϵ is the empty string)
  - Secret Key: the set of all pairs (pk_a, sk_a) for all a of length $\leq m$.

- Sign: given a message M of length m, denote by $M_{|i}$ the string $M_1, \ldots, M_i$ made of the first i bits of M. Then the signature of M is composed of $m + 1$ parts:

- $pk_M, S(sk_M, M)$: the signature of M using secret key sk_M, along with the value of the matching public key pk_M
- $pk_{M_{|m-1}}, pk_{M_{|m-1}0}||pk_{M_{|m-1}1}, S(sk_{M_{|m-1}}, pk_{M_{|m-1}0}||pk_{M_{|m-1}1})$ the signature of the public keys corresponding to M and its sibling, signed using the secret key corresponding to the parent of M, along with the public keys
- $\ldots$
- $pk_{M_{|i}}, pk_{M_{|i}0}||pk_{M_{|i}1}, S(sk_{M_{|i}}, pk_{M_{|i}0}||pk_{M_{|i}1})$
- $\ldots$
- $pk_0, pk_1, S(sk_\epsilon, pk_0||pk_1)$

- Verify. The verification algorithm receives a public key pk_ϵ, a message M, and a signature made of $m+1$ pieces: the first piece is of the form (pk_m, σ_m), the following $m-1$ pieces are of the form $(pk_j, pk'_j, pk''_j, \sigma_j)$, for $j = 1, \ldots, m-1$, and the last piece is of the form $(pk'_0, pk''_0, \sigma_0)$.

  The verification algorithm:

  1. checks $V(pk_m, M, \sigma_m)$ is valid;
  2. For $j = 1, \ldots, m$, if $M_j = 0$ it checks that $pk'_{j-1} = pk_j$, and if $M_j = 1$ it checks that $pk''_{j-1} = pk_j$;
  3. For $j = 0, \ldots, m-1$, it checks that $V(pk_j, pk'_j||pk''_j, \sigma_j)$ is valid. (For the case $j = 0$, we take $pk_0 := pk_\epsilon$.)

We visualize the m-bit messages as labels for the leaf nodes of an m level complete binary tree. Each node a of the tree represents a public-secret key pair pk_a, sk_a. The above scheme signs a message M by first using the one-time signature function to sign M using the secret key sk_M at its corresponding leaf node, and releasing the public key pk_M for that node as part of the signature. Now the sender needs to convince the receiver that public key pk_M was really generated by the sender and not a forger. So the sender signs the message consisting of pk_M and its sibling, namely

$$pk_{M_{|m-1}}0||pk_{M_{|m-1}}1 \ ,$$

using the secret key of their parent node $sk_{M_{|m-1}}$, and releases these two public keys and the public key $pk_{M_{|m-1}}$ as part of the message. The sender now has to convince the receiver that $pk_{M_{|m-1}}$ was generated by the sender, and it can apply the previous procedure again to do this. This signing procedure moves up the tree from signing the message at the leaf node to signing messages of two public keys at each level of the tree until it gets to the root node. The root public key pk_ϵ doesn't have to be signed since this is the public key that is released by the sender at the very beginning for all future communication.

Each public-secret key pair node in this tree is used to sign only one message - either the message corresponding to the leaf node if the key is at a leaf node, or the message that is the concatenation of the public keys at its two children. Note that the public key length is $m/2$ and so there are only $2^{m/2}$ distinct public keys in this tree which has $2^{m+1}-1$ nodes. There will certainly be many copies (on average $2^{m/2+1}$) of each public key at different nodes of the tree. We might be concerned that an adversary might then see many signatures for the same public key and have a much higher chance of breaking the one-time signature scheme for some public key. But if this attack was feasible, then the adversary might as well have generated public-secret key pairs by calling $G()$ and checking if one of these matched some public key seen in the signature of some earlier message - thus, in this scheme, the adversary doesn't get any extra power from seeing multiple signatures using the same key pair.

The theorem below shows that if it is hard for an adversary to forge signatures for the one-time signature scheme (G, S, V), then it will be also be hard to forge signatures under this tree-scheme.

Theorem 1 *Suppose that the scheme described in this section is not (t, ϵ) existentially unforgeable against a chosen message attack.*

Then (G, S, V) is not a $(t \cdot O(r \cdot m), \epsilon/(2tm+1))$-secure one time signature scheme, where r is the maximum of the running time of S and G.

PROOF: If the tree-scheme is not (t, ϵ) existentially unforgeable against a chosen message attack, then there exists an algorithm A with complexity $\leq t$ such that

$$\mathbb{P}[A^{Sign()}(pk) \text{ forges}] \geq \epsilon$$

A makes $\leq t$ queries to the signing oracle before outputting a fresh message and its forged signature. Hence, A can only see $\leq 2tm+1$ public keys (and signatures using them) generated by the key generation algorithm G. Using A as a subroutine, we will construct an algorithm A' which given as input a public key pk' of the signature scheme (G, S, V) and one-time access to the signature function $S(sk', \cdot)$ will forge a signature for a fresh message with probability $\geq \epsilon$.

A' picks a random integer i^* in $\{1, ..., 2tm+1\}$ and using the key generation algorithm $G()$, generates $2tm$ key pairs

$$(pk^1, sk^1), ..., (pk^{i^*-1}, sk^{i^*-1}), (pk^{i^*+1}, sk^{i^*+1}), ..., (pk^{2tm+1}, sk^{2tm+1})$$

For notational convenience, set $pk^{i^*} = pk'$.

A' now simulates A on input pk^1. Whenever A makes a call to $Sign()$ with a given message, A' performs the signing algorithm of the tree-scheme by using the public-secret key pairs it randomly generated at the beginning. A' will keep track of which

nodes of the tree were already assigned key pairs from it's cache of $2tm+1$ key pairs. Since at worst $2tm+1$ key pairs are needed for performing the t signatures requested by A, A' can satisfy all these signature queries using its generated key pairs. If A' needs to sign using $S(sk', \cdot)$, it will use its one-time access to $S(sk', \cdot)$ to perform this action. A' won't have to call $S(sk', \cdot)$ twice with different messages since a public key is never used to sign more than one message in the tree-scheme, unless coincidentally pk' is present as another $pk^j, j \neq i$ in the list of $2tm$ key-pairs generated, in which case A' would have the secret key sk' corresponding to pk' and can completely break (G, S, V). The view of A being run in the simulation by A' is exactly the same as if A had been run on a random public key as input. Hence, the probability A produces a forgery is $\geq \epsilon$.

If A produces a fresh message M and its valid signature

$$\{pk'_{M_{|i}}, pk'_{M_{|i}0}||pk'_{M_{|i}1}, \sigma'_{M_{|i}}\}_{i=0}^{m-1}, pk'_{M_{|m}}, \sigma'_{M_{|m}}$$

then let j be the largest integer such that $pk'_{M_{|j}}$ was seen by A as part of the signature of some message at position $M_{|j}$ in the virtual signature tree. Hence, $pk'_{M_{|j}}$ must be one of the $2tm+1$ keys pk^i generated by A'. Such a value of j must exist because certainly 0 is a candidate for j (since the public key $pk'_\epsilon = pk'_{M_{|0}} = pk^1$ was given as input to A).

Based on the value of j, there are two cases:

- $j \in \{0, ..., m-1\}$. Hence, $pk'_{M_{|j}} = pk^i$ for some i, and if $i = i^*$, then A' will output the message-signature pair $pk'_{M_{|j}0}||pk'_{M_{|j}1}, \sigma'_{M_{|j}}$ as a forgery. $V(pk'_{M_{|j}}, pk'_{M_{|j}0}||pk'_{M_{|j}1}, \sigma'_{M_{|j}}) = 1$ because this was part of the valid tree-scheme signature of message M output by A. By the definition of j, A has never seen the signature of $pk'_{M_{|j}0}||pk'_{M_{|j}1}$ before. Since the position i^* was chosen randomly, the event $i = i^*$ has probability $1/(2tm+1)$.

- $j = m$. Here, all the intermediate public keys in the forged signature of M match those seen by A (and hence match the keys generated by A'), but the signature of M at the last level of the tree itself has not been seen. Hence, $pk'_{M|m} = pk^i$ for some i and $V(pk'_{M|m}, M, \sigma'_{M_{|m}}) = 1$ because M is a valid forge produced by A. If $i = i^*$, then A' outputs the forged message-signature pair $M, \sigma'_{M_{|m}}$. Again, since the position i^* was chosen randomly, the event $i = i^*$ has probability $1/(2tm+1)$.

Conditioned on algorithm A outputting a forge to the tree scheme, in both cases algorithm A' produces a forge to the original scheme (G, S, V) with probability $1/(2tm+1)$. Hence, the probability that A' produces a forge to (G, S, V) is $\geq \epsilon/(2tm+1)$. The running time of the simulation A' is dominated by having to

generate $2tm$ key pairs and performing m signatures using S for each of the t signing queries made by A, and is $t \cdot O(r \cdot m)$.

□