COMS W4995 Introduction to Cryptography **November 13, 2003**

Lecture 21: Multiple Use Signature Schemes

Lecturer: Tal Malkin *Scribes: M. Niccolai, M. Raibert*

Summary

In this lecture, we use the one time secure signature schemes discussed in lecture 20 to construct multiple use schemes. We describe Merkle signatures (which are provably secure), and Full Domain Hash signatures. Discussion of the security of the Full Domain Hash has two parts: First, we evaluate the security with respect to the normal definition of computationally secure signature schemes; Then, because we are unable to prove security, we introduce the Random Oracle Model (ROM) to support the (unproven) assertion that these schemes is secure.

1 Merkle Signatures

A signature scheme for use with multiple messages of arbitrary length; built using Collision Resistant Hash-Functions (CRHF) and one time signatures.

For this signature scheme, we must know in advance how many signatures the user will ever plan on sending. Assume that we have a one time signature scheme; generate n pairs (PK_i, SK_i) and build a tree using a colision resistant hash function, h, by hashing each pair of adjacent nodes recursively up the tree to the root. The root is the public key, r.

$SK = \{(PK_i, SK_i)\}_{i=1...n}$

Note: The public keys $\{PK_i\}_i$ are used to sign the messages, and so are formally specified as part of the secret key. However , they need not remain secret (and in fact they are published as part of the signature).

We build our tree from the n pairs of (public key, secret key).
Let the (PK_i, SK_i) pairs be the base of the tree (n leaves). Compute the next level up, and name the elements of the tree, as follows:

Figure 1: Merkle Signature tree

$$s_1^1 = h(PK_1, PK_2)\ ,\ s_2^1 = h(PK_3, PK_4)\ ,\ \ldots\ ,\ s_{n/2}^1 = h(PK_{n-1}, PK_n)$$

For constructing the second level from the first level, we compute:

$$s_1^2 = h(s_1^1, s_2^1)\ ,\ s_2^2 = h(s_3^1, s_4^1)\ ,\ \ldots\ ,\ s_{n/4}^2 = h(s_{n/2-1}^1, s_{n/2}^1)$$

Continuing this pattern up the tree, which has height of $logn$, we then use the signature in the current leaf to sign our message:

For the first message, we use SK_1 to sign m_1 and send

$(1, m1, sign_{SK_1}(m_1), PK_1, PK_2, s_1^1, s_2^1,$ and all hashes adjacent to route from PK_1 to $r)$

To verify check that $sign_{SK_1}(m_1)$ is correct by checking the hashes. In other words:

$h(h(h(h(PK_1, PK_2), s_1^1)s_2^1)...) = r$ (check hashing was computed correctly)

After sending the first signature, update to state 2.

For the second message: Send similar signature [state, message, signature, PK_i, all hashes adjacent to route from $PK_i \rightarrow r$]

and so on. For example, for message $(n-1)$: sign with

$(n-1, m_{n-1}, sign_{SKn-1}(m_{n-1}), PK_{n-1}, PK_n,$ and all hashes adjacent to route from PK_n to $r)$

The following can be thought of as an intuitive explanation of why this scheme works:

Even when the adversary gets the public keys, and the path from $PK \rightarrow r$, the adversary cannot forge any new signatures. If she were to succeed in trying to forge a new (PK, SK) pair that hashes correctly, this would be a collision contradicting the assumption that h is a CRHF. If she forged using the given tree for the signature scheme, that must mean that she can forge the one time signature $sign_{SK_i}(m)$ which we have assumed to be secure. The formal proof would be constructed as follows (note that this is not the formal proof): "assume f forges the signature scheme, either f can forge the signature scheme with high probability, or he would come up with a collision with high probability."

Drawbacks of the Merkle scheme:

- it is stateful
- it uses CRHF (whereas we know how to construct CRHF from specific assumptions such as DLA or the hardness of factoring, we do not know how to construct them with the weaker assumption that OWF exist)
- we must know in advance the number of messages to be signed
- the length of the secret key is too long (it is proportional to the number of messages to be signed)

How do we improve efficiency? Instead of the Merkle scheme described above, we can build a tree with one-time signature schemes (PK, SK)s at the nodes (or a similar construction), instead of hashing. This scheme has a shorter key length because the tree grows from the top down, but we know that hashing less computationally expensive than the one-time signature scheme. Even though the key length is much shorter, this construction is worse in terms of time. Every time we need to sign, we need to perform the expensive task of generating a new pair.

Theorem 1 *There exist signature schemes which can be based on any OWF, are stateless, and have short keys (the length depends on the security parameter).*

2 Full-domain hash

If we start with signatures based on TDP, which we know to be not secure, and add hashing - do we get a secure scheme?

Recall signing with TDP:

- let f be a TDP (e.g. RSA), and consider the scheme with $PK = f$ nd $SK =$ trapdoor t.
- $sign_{SK}(m) = f^{-1}(m)$
- $ver(m, \sigma)$: check that $f(\sigma) = m$

Example 1 *Full Domain Hash Using RSA*

- $sign_{SK}(m) = f^{-1}(m) = m^d \pmod{n}$
- $ver_{PK} =$ *check that* $\sigma^e = m \pmod{n}$

This is completely not secure. Our adversary can choose $\sigma \in \mathbb{Z}_n^*$, compute $m = \sigma^e \pmod{n}$, and send (m, σ) as her forgery. This is valid signature that the verifier will accept.

So, we use this as a basis for hash and sign (see the discussion in Problem Set 5, question 1):

- Let f be a TDP. Also, let h be some hash function, which we will discuss later.
- $PK = (f, h)$ $SK =$ trapdoor t.
- $sign(m) = f^{-1}(h(m))$
- $ver(m, \sigma)$: check that $f(\sigma) = h(m)$

This is known as **Full Domain Hash signature scheme** and is the basis for many of the schemes that are actually used.

If we think of h as a totally random function at this point; $\sigma^e = h(m)$, but then how do we recover m? Hence, the above attack no longer works.

Note: As we see in homework 5, this is not secure when h is a CRHF. In fact, it can be proven that this does not work with *any* family $H = \{h_i\}_{i \in I}$ where h_i can be chosen in polynomial time. So then, how do we choose h?

We can prove the following theorem:

Theorem 2 *If we choose h to be a truly random function, the above hash-and-sign scheme is secure.*

Obviously it is not feasible to have a random function, but this fact will help support the case for the security of various signature schemes.

3 The Random Oracle Model

Because we do not know how to prove the security of multiple-use signature schemes we construct a weaker (infeasible) model in which we can prove security. The Random Oracle Model (ROM) is based on the idea that if all parties have access to a truly random function we can prove security for certain schemes.

Once we have proven that a scheme is secure under the ROM, this helps us support the case that the scheme that uses an h that "behaves like a random function"(ie SHA1 or MD5) may also be secure. If the scheme is secure under the Random Oracle model then it's probably secure with our h. It is important to note that this is just a heuristic and not a proof, since we know that a random oracle cannot be implemented in polynomial time (even the description of a truly random function takes exponential space).

Note: We cannot use pseudo random functions in the ROM because the function h needs to be public, and to make a pseudo random function public one must reveal the pseudo random functions's key, nullifying its pseudo random properties.

We may, therefore, restate the theorem from the previous section using the Random Oracle model terminology as follows:

Theorem 3 *Full-domain hash is a secure signature scheme in the Random Oracle model*

Proof omitted.

Example 2 *PKCS#1:* $sig_{RSA}(s \cdot SHA1(m))$*, where* s *is some specific byte string.*

Note: even though this is frequently used in practice, it is not provably secure even in the Random Oracle model (i.e. if we replace $SHA1$ with a random function).