6.875J/18.425J Cryptography and Cryptanalysis	November 14, 2001
Lecture 18	
Instructor: Anna Lysyanskaya	*Scribed by: Yogishwar Maharaj*

Today, we will introduce **digital signatures**.

1 Definition

One of the demands on modern cryptography, is that of a good digital signature scheme. Digital signatures should act in the same way as conventional signatures. We want the following properties:

- The signer Alice should be able to produce a signature for a document.
- Other users should be able to verify Alice's signature.
- No one besides Alice should be able to produce a signature that fools the verification process. That is, forgery is impossible.

We can use a public key system to achieve this. We then require the following:

1. Key generation Algorithm - A probabilistic polynomial time algorithm that can produce a public/private key pair.
2. Signature Algorithm - A probabilistic polynomial time algorithm that given a public/private key pair and a message m, produces as output a string σ called the signature.
3. Verification Algorithm - A probabilistic polynomial time algorithm that, given a public key, a signature σ and a message m decides whether or not $\sigma \in Sign(m)$. Let us suppose that it outputs a one for true.

A verification algorithm should accept signatures produced by the signature algorithm with probability one, however wee can sometimes relax this restriction if we want. We require a scheme in which signatures cannot be forged. The idea is similar to that of a MAC as seen in problem 3 of problem set 6. That is, signatures of chosen messages do not help the adversary to produce a forgery on a message not explicitly queried.

1.1 Definition of Security

Suppose that $(G, Sign, Verify)$ are a secure signature scheme.

$$\begin{aligned}
&\forall\, PPTF\{A_k\}\ \exists \nu(k) \text{ such that} \\
&\quad \Pr[(PK, SK) \leftarrow G(1^k); \\
&\qquad (Q, m, \sigma) \leftarrow A_k^{Sign(PK,SK,\cdot)}(PK) : m \notin Q \text{ and } Verify(PK, m, \sigma) = 1] \leq \nu(k)
\end{aligned}$$

This definition is due to Goldwasser, Micali, Rivest [2].

2 Digital Signature Schemes

Diffie and Hellman proposed a heuristic signature based on Trapdoor Permutations (TDP). As an example consider a public key cryptosystem (G, E, D) based on a trapdoor function. A message m can be signed with $D(m) = f^{-1}(m)$, where f is the user's trapdoor public function. The user is the only one who knows the trapdoor information. Thus no one else can make this signature. Verification is easy: $E(D(m)) = f(f^{-}(m)) = m$. RSA falls under the Diffie-Hellman category of signatures. Rabin has also proposed a scheme with factoring a large number as the hardness assumption.

We want to separate the notion of signing from encrypting, since some crypto systems have proved to be insufficient for digital signatures.

An example from the Goldwasser-Bellare lecture notes [1] illustrates this point. The RSA digital signature scheme is as follows:

The public key pair is the pair (n, e) where n is the product of two large primes and e is relatively prime to $\phi(n)$. The secret key is d such that $ed \cong 1 mod \phi(n)$. This is the usual RSA key pair. To sign a message, one computes $\sigma(m) \cong m^d \texttt{ mod } n$. To verify, one raises the signature to the power e and compares it to the original message. This system is forgable under a chosen message attack. See [1] for a proof.

Intuition: We want a signature system to be such that if $A_k(PK, m)$ is a forgery, then there is a reduction that uses it to break some hard problem. At first it was thought that if a reduction is able to answer $A_k' s$ queries, then it could not learn anything from a forgery, because it could just sign itself. However this is not true.

The right idea is that their are two ways to forge. The reduction knows only one way. If the adversary knows the another way to forge, then we can learn something from him.

3 Lamport's one-time signatures

We give as an example a one-time signature scheme developed in the 1970's by Lamport. Suppose that we are given an oneway function (OWF) f and security parameter k. Suppose also that we want to sign a binary string of length n. We chose some x_i^j with $j \in \{0, 1\}$ and $i \in \{1, \ldots n\}$. Put these into a table as in Table 1 and let this be the secret key. We let the public key be as in Table 2.

Table 1: Secret Key SK

x_1^0	x_2^0	$\ldots$	x_n^0
x_1^1	x_2^1	$\ldots$	x_n^1

Table 2: Public Key PK

$f(x_1^0)$	$f(x_2^0)$	$\ldots$	$f(x_n^0)$
$f(x_1^1)$	$f(x_2^1)$	$\ldots$	$f(x_n^1)$

Signing a message: Suppose we want to sign a message $m = m_1 m_2 \ldots m_n$. Then let $Sign(m, PK, SK) = \{x_i^{m_i} : 1 \leq i \leq n\}$. That is, if the ith bit of our message is a zero then we take the ith entry in the first row of our secret key. Otherwise we take the ith entry in the second row. We can simply apply f to **verify** the signature.

Why can't we forge a signature with this scheme?

We will give a reduction that takes input y and outputs $f^{-1}(y)$. Let us prepare PK as follows: Take the PK of a real signer, chose $b \in \{0,1\}$ and an $i \in \{1, \ldots n\}$ randomly. We place our y into that slot as in Table 3.

Table 3: Public Key for a reduction

$f(x_1^0)$	$f(x_2^0)$	$\ldots$		$\ldots$	$f(x_n^0)$
$f(x_1^1)$	$f(x_2^1)$	$\ldots$	**y**	$\ldots$	$f(x_n^1)$

This table is exactly the same as the real signer's except for one entry. We then receive a query on message m. There is a probability of $1/2$ that $m_i \neq b$ so m can be signed. Next, we receive a forgery m', σ' with $m' \neq m$. This means that some bit j of m' is different from the corresponding bit in m. That is, for some $m'_j \neq m_j$. With probability $1/n$, $i = j$. If an adversary forges a message with probability ϵ, then the reduction can invert f with probability $\epsilon/2n$ ■

4 Merkle trees

The preceding example shows us how to do a one time signature. Nowadays one usually wants to sign more than one document. We now present a method that allows us to sign several documents, as long as we know beforehand some upper bound on the number of documents to be signed. This method uses cryptographic hash functions.

Definition 1 *Suppose we have a function H_{PK} that takes two inputs x_1, x_2 of length k and outputs a y of length k. Then $H_{PK}(x_1, x_2) = y$ is a cryptographic hash function if it is hard to find $(x_1, x_2) \neq (x'_1, x'_2)$ with $H_{PK}(x_1, x_2) = H_{PK}(x'_1, x'_2)$.*

We want to develop a notion of ***collision-resistance***. Let p be a prime. Let $\mathbb{Z}_p^* = < g >$ $= < h >$. That is let g, h both be generators for the group $\mathbb{Z}_p^*$. Then let $H_{p,g,h}(a, b) = g^a h^b$ $\texttt{mod}\ p$. Suppose that we can find an $(a, b) \neq (a', b')$ $\texttt{mod}\ p-1$. such that $g^a h^b \cong g^{a'} h^{b'}$ $\texttt{mod}$ p. Since h is a generator, we have $h \cong g^\alpha$ $\texttt{mod}\ p$. Then $a + b\alpha \cong a' + b'\alpha$ $\texttt{mod}\ p-1$. Then we can solve for α. Knowing α, we can factor $\phi(p) = p - 1$.

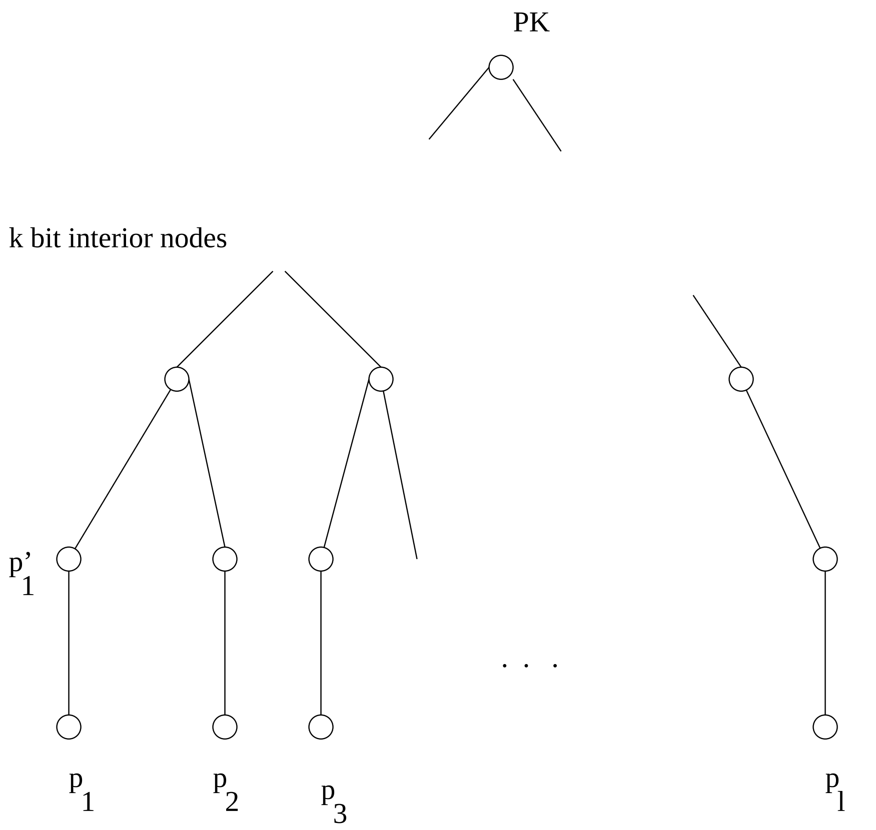
Proposition 2 *$H(H(a, b), H(c, d))$ is a collision-resistant hash function that takes $4k$ bits to k bits, if H is a collision resistant hash function that takes $2k$ bits to k bits.*

Sketch of Proof The general idea of a reduction is as follows: Suppose not. That is, suppose it is easy to find $(a', b', c', d') \neq (a, b, c, d)$ such that $H(H(a', b'), H(c', d')) =$ $H(H(a, b), H(c, d))$. We can suppose that if $(a, b) \neq (a', b')$ then $H(a, b) \neq H(a', b')$, since H is collision resistant; similarly if $(c, d) \neq (c', d')$. But at least one of these two must be different by hypothesis. Therefore $(H(a, b), H(c, d)) \neq (H(a', b'), H(c', d'))$. That is we have two pairs of k bit strings that are not equal, but the hashes of both are equal. This contradicts the hypothesis that H was collision resistant. ■

This is a "new" primitive for us. Some questions we should ask ourselves are where does this fit with the other primitives? Do we really need this primitive?

A signer has some bound ℓ on the number of messages to be sent. He prepares ℓ public/private key pairs (p_i, s_i) as in the one-time scheme above. The public keys are then hashed in a tree like way. The p_i are the leaves. We construct their parents, by hashing each enough times that we get a k bit string. Let these be p_i'. Then we hash p_1' with p_2', p_3' with p_4' and so on. This gives us the next set of parents. We proceed as described until we get to a single root node, which we publish as the public key. Note that all of the interior notes have length k. See Fig. 4

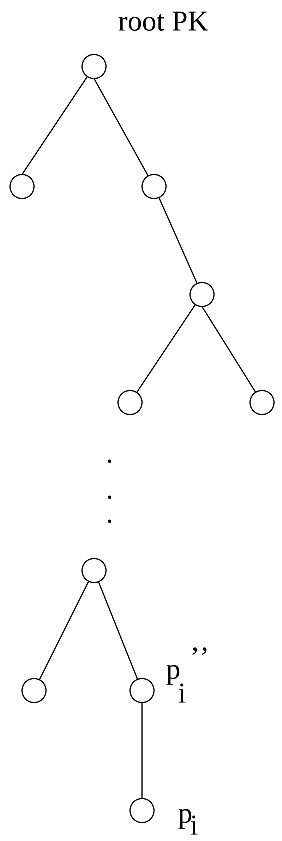
Figure 1: Merkle's multi-use Public Key



Merkle Public Key Tree

Merkle Signature: Suppose that we have issued $i - 1$ signatures so far. Take (p_i, s_i) and produce $Sign(p_i, m_i)$ according to Lamport's one-time scheme described above. Along with this, we give some *proof* that p_i was hashed into PK. For this, we give c_i^j's such that these interior nodes are either parents or siblings of parents of p_i. See Fig. 4

Figure 2: Proof that the key is in the Tree



The "proof" that the chosen key is in the tree.

Verification: First we verify the onetime signature as described above. Next we verify that p_i is in the tree. We do this, by repeating the process of tree generation. That is, we first hash p_i to p_i'. Then together with the appropriate siblings, with which we are provided (see fig. 4). In so doing, we can trace backwards to the root node. If we fail to do so, then we reject the signature.

Why is this Merkle-Lamport scheme secure?

Suppose that some adversary can forge signatures with probability ϵ. Then one of the following statements is true:

1. Type 1 forger: With probability $\geq \epsilon/2$ he can compute a forgery that reuses the same p_i.
2. Type 2 forger: With probability $\geq \epsilon/2$ he can compute a forgery where no p_i is reused.

Claim 3 *If a Type 1 forger exists, then we can break the Lamport scheme.*

Sketch of Proof We construct a reduction with input p. We query on any m. The goal is to forge a signature on any $m' \neq m$.

Chose i at random. Let $j \neq i$, p_j, s_j be as in Merkle except $p_i = p$. With probability $1/\ell$ we can forge the Lamport scheme. ■

Claim 4 *If a Type 2 forger exists, then we can break the hash function.*

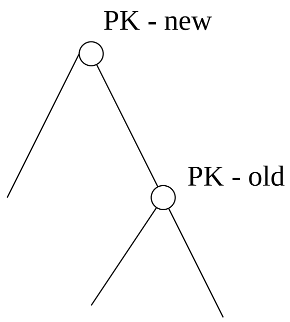
Proof We give a reduction. Let the reduction take input some hash function H. The goal is to find a collision in H. We prepare PK in the usual manner. Then we answer the adversary's queries, obtaining a forgery in which no p_i is reused. Let $\sigma = (\sigma_1, \sigma_2) = (p, c_1, \ldots, c_w)$ be the signature. We note that since verify accepts this signature,

$H(H(\ldots(H(p), c_1), c_2)\ldots), c_w) = PK.$

Suppose that $H(H(\ldots), c_{w-1})$ is not a child of $PK.$ or c_w is not a child of $PK.$ Then we are done. Otherwise look one level down. ■

Limitations: One of the major limitations of this scheme is that we are required to know ℓ beforehand. Suppose that we do not. We can still sign an $\ell + 1$th message. However new signatures will be signatures on a new public key (or root value).

Figure 3: Signing more messages



Adding new keys.

This method is quite inefficient. It has huge space requirements. This problem was solved by Goldwasser, Micali, Rivest. One is also required to keep state information. One must know how many messages have already been sent. This problem was solved with a number theoretic argument.

References

[1] S. Goldwasser, M Bellare, Lecture Notes on Cryptography, August 2001.

[2] S. Goldwasser, S. Micali, R. Rivest. A digital signature scheme against adaptive chosen-message attacks. SIAM J Computing, 17(2):281-308, April 1988.