Lecture 16 - Digital Signatures

Boaz Barak

November 15, 2005

Definition of digital signatures. Recall that we had the following picture:

	Private Key	**Public Key**
Secrecy	Private Key Encryption	Public key Encryption
Integrity	Message Authentication Codes (MAC)	??

Digital signatures complete this picture by giving a public key analog of message authentication codes. Digital signatures were suggested by Diffie and Hellman in their seminal paper, but unlike the case of public key encryption schemes (where they had a key exchange protocol that could be made into a probabilistic encryption scheme) they did not have a reasonable candidate for such signatures until the RSA system was invented a year later by Rivest, Shamir and Adelman. However, even the RSA system was quickly seen not to have sufficient security and only later Goldwasser, Micali and Rivest gave what is now considered to be the "right" definition for digital signatures, and also a factoring-based construction meeting this definition. This definition is called *existential forgery under chosen-message attack* but we'll simply call it secure signature schemes.

Definition 1 (Digital Signatures)**.** A triplet of algorithms $\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}$ is called a (T, ϵ)*-secure signature scheme* if it satisfies the following properties:

Validity For every pair $(s, v) \leftarrow \mathsf{Gen}(1^n)$, and every $m \in \{0,1\}^n$, we have that

$$\mathsf{Ver}_v(m, \mathsf{Sign}_s(m)) = 1$$

Security For every T-time circuit A, we have that

$$\Pr[A^{\mathsf{Sign}_v(\cdot)}(v) = (m, \sigma) \text{ st } m \text{ wasn't queried by } A \text{ and } \mathsf{Ver}_v(m, \sigma) = 1] < \epsilon$$

Again, a scheme is simply secure if it is (T, ϵ)-secure for super-polynomial T and ϵ.

Applications. Digital signatures have many applications and are widely used today in the world. Some practical application include verifying websites such as **amazon.com** and verifying code such as drivers for Windows and upgrades for embedded devices. In fact, signature keys tend to be much "longer-lived" than encryption keys: they are often hardwired into various devices and there's no mechanism to replace them. Thus, there are fixed and well known public verification keys today whose corresponding secret keys are worth many millions of dollars. For this reason protecting the private signature keys is often more critical than protecting the private decryption keys, and time permitting, we may discuss some techniques for such protection in this class.

History For some time, people thought that it will be impossible to achieve signature schemes with a proof of security. The reasoning was that the proof of security will have to be an efficient way to transform a forged signature into, say, a factoring of n, where $n = pq$ is the public key. However, if there is such a way then the scheme cannot be secure under a chosen message attack. However, this reasoning is flawed and in 1984 this was demonstrated by Goldwasser, Micali and Rackoff who gave the first (albeit stateful) signature scheme based on the hardness of factoring. This was later improved by Goldreich to a stateless signature scheme. Surprisingly, results of Naor and Yung, and Rompel show that secure signature schemes can be constructed based on much weaker assumptions without using any number theory: either Axiom 1 (existence of pseudorandom generators) or Axiom 2 (existence of one-way permutations) suffice.

All these signature schemes are still not quite efficient enough for practical use. More efficient constructions were given by Gennaro, Halevi and Rabin , and by Cramer and Shoup, under some stronger variant of the RSA assumptions.

Plan We won't have time to present most of these. Rather our plan will be as follows:

1. We'll show a *one-time* signature scheme, that is only guaranteed to be secure if the adversary gets to ask a single query.
2. We'll use that to construct a *stateful* signature scheme, where the signer needs to maintain state between each signature.
3. Then we'll briefly explain how can we convert this signature scheme to a standard, stateless signature scheme.

In addition, we'll give two schemes that are closely related to ones used in practice:

1. We'll show a different construction for an *interactive* signature scheme where we do not need to maintain state but relax the assumption that the signature consists of a single message from signer to receiver.
2. Finally, we'll also present an efficient construction of a stateless, non-interactive signature scheme. However, we'll not be able to prove security for this construction under any reasonable assumption. Rather, we'll only give a *heuristic argument* why this scheme may be secure.

The last scheme is probably the one most common in practice. Another common practical signature scheme is a non-interactive version of the interactive signature scheme that also has a heuristic argument of security.

To try to cover all this material in two lectures, the presentation will sometimes be sketchy. Complete proofs for the first three steps are available in Goldreich's book. A proof for the last scheme is available in the paper by Bellare and Rogaway on the random oracle methodology.

One time signature scheme We start by presenting a *one-time* signature scheme (due to Lamport) that remains secure if the attacker can only make a single query to the signing oracle. In fact, we'll consider an even simpler variant: a signature for a single bit. Thus, the attack is that the adversary chooses a bit $b \in \{0,1\}$, gets a signature for b and needs to forge a signature for $\overline{b} = 1 - b$. We'll base this on Axiom 2: the existence of a *one-way permutation* $f(\cdot)$ that is a one-to-one function $f : \{0,1\}^n \to \{0,1\}^n$ such that for every polynomial-time A, $\Pr_{x \leftarrow_{\mathrm{R}} \{0,1\}^n}[A(f(x)) = x] < n^{-\omega(1)}$.

Key generation $\mathsf{Gen}(1^n)$ chooses $x^0, x^1 \leftarrow_{\mathrm{R}} \{0,1\}^n$ and computes $y^b = f(x^b)$ for $b = 0, 1$. The private signing key is $s = (x^0, x^1)$ and the public verification key is $v = (y^0, y^1)$.

Signing algorithm To a sign a bit $b \in \{0,1\}$, $\mathsf{Sign}_s(b) = x^b$.

Verification $\mathsf{Ver}_v(b, x) = 1$ iff $f(x) = y^b$.

It is a simple exercise to verify that this scheme is secure under a single-query chosen message attack.

Extending to longer messages It is clear how to extend a single bit scheme into a scheme for signing ℓ bits: just generate ℓ independent public/private key pairs.

Signing messages longer than the key length. One drawback of that scheme (other than it is one-time) is that to sign a message of length ℓ, we need a key of length $n \cdot \ell$. This turns out to be a serious bottleneck in converting a one-time signature scheme into a standard (many-times) scheme. To overcome this, we'll need the notion of a *collision resistant hash function*. The idea is that this is a collection of functions H such that each function maps say $2n$ bit long strings into n bit long strings and so it's definitely *not* one-to-one but given such a function it is infeasible to demonstrate that it is not one-to-one (i.e., to find a *collision*: two values $x \neq x'$ such that $h(x) = h(x')$). The formal definition is the following:

Definition 2 (Collision-resistant hash functions)**.** A collection of functions $\{h_k\}_{k \in \{0,1\}^*}$, with $h_k : \{0,1\}^{2n} \to \{0,1\}^n$ for $k \in \{0,1\}^n$, is called (T, ϵ)-*collision resistant* if the function $(k, x) \mapsto h_k(x)$ is polynomial-time computable and for every T-time A we have that

$$\Pr_{k \leftarrow_{\mathrm{R}} \{0,1\}^n}[A(k) = (x, x') \text{ st } h_k(x) = h_k(x')] < \epsilon(n)$$

Collision resistant hash functions are known to exist based on the assumption that factoring is hard (Goldwasser, Micali and Rivest) and there are also several efficient candidates for collision resistant functions (e.g., SHA-256). Last year a team of researchers made headline news showing that the most commonly used hash function (MD-5) is *not* collision-resistant (they found collisions) and also gave some evidence that another widely used hash function (SHA-1) is much less collision resistant than people originally thought.

Naor and Yung defined a weaker notion of collision resistant that can be sufficient for the application of signature schemes, and showed that it can be achieved using Axiom 2 only, Rompel then improved this to use only Axiom 1. ‘

A scheme with message size $>$ key size : It is not hard to show that given a collision resistant hash function h mapping $\{0,1\}^{2n}$ to $\{0,1\}^n$ we can extend it to a function mapping say $\{0,1\}^{n^3}$ to $\{0,1\}^n$. Therefore we can have the following scheme:

Components: A signature scheme $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$ that uses n^2 long keys to sign n long messages. A hash function collection $\{h_k\}_{k \in \{0,1\}^*}$

where for $k \in \{0,1\}^n$, $h_k : \{0,1\}^{n^3} \to \{0,1\}^n$. For convenience we use h to denote both the function itself and its key k, thus we think of h as both a function from $\{0,1\}^{n^3}$ to $\{0,1\}^n$ and an n bit string.

Key generation $\mathsf{Gen}(1^n)$ chooses a pair (s', v') of a signature scheme for messages of length n, and a hash function $h : \{0,1\}^{n^3} \to \{0,1\}^n$. The public key is (h, v') while the private key is s'.[1] Note that the length of the keys is $n^2 + n \ll n^3$.

Signing algorithm To a sign a message $m \in \{0,1\}^{n^3}$, compute $m' = h(m)$ and output $\sigma = \mathsf{Sign}'_{s'}(m')$.

Verification $\mathsf{Ver}_v(m,\sigma) = 1$ iff $\mathsf{Ver}_{v'}(h(m),\sigma) = 1$.

This transformation applies equally well to one time and many time schemes. In both cases the new scheme inherits the security of the old scheme. The idea of the proof is that as long as the adversary doesn't find a collision in the hash function, we can convert an attack on the new scheme to an attack on the underlying scheme.

From one-time to many-times scheme. Our next step is to have a scheme that will be secure for more than one signature. However, our scheme will have the drawback that it will preserve state, and also that the signature size will grow with time: the t^{th} signature made will be of size about $t \cdot n$.

The idea is simple: first we observe that we can easily convert a one-time signature for messages of length ℓ to a two-time signature for messages of length $\ell/2$ and so we can assume that we have a two-time signature scheme with n-bit public key length for, say, $10n$ (although n will suffice) long messages.

We'll let the public and private keys of our schemes be the keys of the two-time scheme, and denote them by (s^0, v^0). At time i, to sign a message m, we first generate a pair of new keys s^i, v^i, and then use s^{i-1} to sign the pair v^i and obtain a signature σ^i and in addition sign m using v^i to obtain a signature σ. The signature is the list $\sigma_1, \ldots, \sigma_i$ and σ. To verify it using the key v^0 the verifier checks for all $j > 0$ that σ^j is a signature of v^j that passes verification w.r.t. v^{j-1} and that in addition σ^i also contains a signature for m.

The idea of signing one verification key using another is called a *certificate* and is widely used in cryptography.

[1] When specifying a private key we always ignore the parts that are already present in the corresponding public key.

Getting shorter signatures. The fact that the signature size grows linearly with time is a very serious drawback of the previous scheme. We'll now present a scheme where the t^{th} signature is only of size $(\log t) \cdot n$. Since we can always assume $t \leq 2^n$, this means that the signature size is never bigger than n^2.

The idea (due to Merkle) is to use a certificate *tree* instead of a path as was used before. That is, we maintain a binary tree where each node is associated with a verification/signing key pair and each no-leaf node is also associated with a signature. Initially the tree consists of only one node that is both root and a leaf and is associated with the "main" pair s^0, v^0 and no signature. When we want to sign a message m we find the "shallowest" (closest to root) leaf, denoted x, and if (s, v) are the keys associated with x, we generate two new pairs, (s_1, v_1) and (s_2, v_2) and sign $v_1 \circ v_2$ using s. We associate this signature with x and create two children for it with the corresponding key pairs (s_1, v_1) and (s_2, v_2).

The signature on m is the list of signatures on the path from the root to x along with a signature on m using s. Note that this path is of length at most $\log t$.

Making this stateless Goldreich suggested a way to make this signature scheme stateless. The idea is the following: if the signing algorithm could just recompute the tree (or even just the path) "on demand" every time it needed a new signature then it would not need to keep the tree in memory. However, the problem is that it needs to record its random choices for all the keys generated when computing the tree. Fortunately, this can be resolved using pseudorandom functions: the private key will contain a key k for a PRF family $\{f_k\}$ and all the randomness needed during the creation of a node x in the tree will be obtained by running $f_k(x)$.

This of course requires proof, which we skip here, but can be found in Section 6.4.2.3 of Goldreich's book, Vol II.