University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

# Lecture 35

## 1 Signature Schemes

Let's review what cryptographic protocols we have covered thus far. In the context of symmetric-key (or private-key) cryptography, we have discussed encryption protocols and message authentication schemes. The former provide *secrecy* while the latter provides *integrity*. We next discussed public-key encryption which is the direct counterpart of private-key encryption, although it allows many people to secretly communicate with one person (who has published a public key) rather than "just" allowing two people (who have shared a key in advance) to secretly communicate. What is the counterpart of message authentication?

In the case of message authentication, we saw that two people who had shared a key in advance can use a message authentication scheme to determine whether future messages they receive are indeed from each other (and not maliciously inserted by someone else). The public-key analogue is a *digital signature scheme.* Here, one person (called the *signer*) publishes a public key which, as usual, is then assumed to be known by everyone else. The signer keeps an associated secret key which he can then use to sign messages. The resulting signature on the message is then verifiable *by anyone who knows the signer's public key.* We give a formal definition and then a representative example.

**Definition 1** *A triple of poly-time algorithms* $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ *is a signature scheme (over some message space* $\mathcal{M}$*) if the following hold:*

1. $\mathcal{K}$ *is a randomized algorithm which outputs a public key* $pk$ *and a secret key* $sk$*.*

2. $\mathsf{Sign}$ *is a (possibly) randomized algorithm which takes a secret key* $sk$ *and a message* $m \in \mathcal{M}$ *and outputs a signature* $\sigma$*. We denote this by* $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(\mathsf{m})$*.*

3. $\mathsf{Vrfy}$ *is a deterministic algorithm which takes a public key* $pk$*, a message* $m \in \mathcal{M}$*, and a (purported) signature* $\sigma$ *and outputs 1 or 0. We denote execution of the algorithm by* $\mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{m}, \sigma)$*. A 1 indicates that the signature is* valid *and a 0 indicates that the signatures is* invalid.

*We require that for all* $(pk, sk)$ *output by* $\mathcal{K}$*, all* $m \in \mathcal{M}$*, and all* $\sigma$ *output by* $\mathsf{Sign}_{\mathsf{sk}}(\mathsf{m})$*, we have* $\mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{m}, \sigma) = 1$.

The final condition just means that all signatures issued by the legitimate signer are recognized as valid by anyone with access to the public key for the signer.

We give a brief example of how a signature scheme can be used. Say a bank $B$ has a public key $PK_B$ (they can publicize this key in one of the ways we discussed earlier, like putting it in the newspaper). Assume this bank then wants to certify a statement $m$ of the form "As of Jan. 1, 2002, $X$ has \$100 in her account". They can use their signature scheme to generate a signature $\sigma$ on this message. The bank can give this signature to $X$ who can now take it and use it to convince *anybody else* (as long as they know the bank's public key $PK_B$) that the bank indeed certified statement $m$.

Of course, the preceding example doesn't make much sense unless we also define a notion of security for a signature scheme. (In particular, if $X$ could have forged the bank's signature, then showing a signature proves nothing.) Our definition of security will be analogous to the one we gave for message authentication schemes, and represents a very strong notion of security. Informally, it should be the case that regardless of what messages are signed by the signer, no one else (without the secret key) should not be able to forge a valid signature on *any* new message in the name of the signer. We model this by defining a *signing oracle* $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$ with which an adversary can interact; when the adversary sends any message to this oracle the oracle responds with the signature on that message. The adversary can interact with this oracle as often as he likes, and the adversary's goal is to come up with a valid signature $\sigma$ on a message $m$ *that was never explicitly queried to the oracle* (representing the fact that this message was never signed by the real signer). More formally:

**Definition 2** *A signature scheme* $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* $(t, \epsilon)$*-secure if for all adversaries* $A$ *running in time* $t$ *we have:*

$$\Pr[(pk, sk) \leftarrow \mathcal{K}; (m, \sigma) \leftarrow A^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(pk) : \mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{m}, \sigma) = 1 \wedge \mathsf{m} \notin \mathsf{M}] \leq \epsilon,$$

*where* $M$ *denotes the set of messages submitted by* $A$ *to* $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$.

Note that the adversary is explicitly given the public key and, in particular, can always verify whether a particular $\sigma$ is indeed a valid signature on a message $m$. This is in contrast to the case of message authentication where the adversary never learns the secret key and therefore does not even know if what he outputs is a forgery or not.

## 1.1 Digital Signatures vs. Message Authentication

It is worth stressing two important differences between digital signatures and message authentication:

**Public verifiability** Digital signatures are *publicly verifiable.* That is, anyone who knows a user's public key can verify whether that user has signed a particular message. In contrast, message authentication schemes only allow the recipient to verify that the message originated from the sender; others have no way of verifying this fact.

**Non-repudiation** Digital signatures provide *non repudiation* in the sense that they bind the signer to the signed message. The signer himself cannot later deny (say, to a judge) signing the message; his signature "binds" him just as a real signature would on a legal document. On the other hand, message authentication does not provide this *at all.* For one thing, there is no way to even convince a judge that a message was authenticated without revealing the secret key. Even at that point, there is no way to know that this secret key was actually the agreed-upon key that the sender and receiver used. Finally, even if a judge definitively knows the secret key used all this proves is that *either* the sender or receiver authenticated the message (since both these parties have the same secret information, there is no way to tell which party actually authenticated the message).

### 1.2 Perfectly-Secure Digital Signatures?

Can perfectly-secure signature schemes exist? We saw that in the case of message authentication, perfectly-secure schemes could be constructed as long as we limit the number of authenticated messages the adversary can request. Might something similar work in the case of signatures?

We show that *no* sort of perfectly-secure signature schemes can exist. In particular, an unbounded adversary can always forge a valid signature on any message of his choice even without seeing any previously signed messages. To see this, consider an adversary $A$ who is given the public key $pk$ and wants to forge a signature on message $m$. If signatures are, say, $k$ bits long then the adversary can simply try every possible string $\sigma$ of length $k$ until it finds one for which $\mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{m}, \sigma) = 1$. Then $\sigma$ is a valid signature on $m$! Note a crucial difference between the case of digital signatures and message authentication is that the adversary can verify correctness of the signature $\sigma$ before it outputs $\sigma$; this cannot be done for message authentication schemes, which is why perfectly-secure schemes are possible in that setting.

### 1.3 The Suggestion of Diffie and Hellman

Diffie and Hellman suggested the following way to implement digital signature schemes based on any trapdoor permutation. Let $(\mathcal{K}, f, f^{-1})$ be a trapdoor permutation. Their suggestion was the following:

- Key generation consists of running $\mathcal{K}$ to generate key $\mathsf{k}$ and trapdoor $\mathsf{td}$. The public key is $\mathsf{k}$ and the secret key is $\mathsf{td}$.
- To sign message $m \in \mathcal{D}_{\mathsf{k}}$, the signer (who has the trapdoor) computes $\sigma = f_{\mathsf{k}}^{-1}(m)$.
- To verify signature $\sigma$ on message $m$, check whether $f_{\mathsf{k}}(\sigma) \stackrel{?}{=} m$. Note that anyone with access to the public key can verify this.

In fact, this is exactly how "textbook" RSA signatures work: the public key is modulus $N$ and public exponent $e$ and the secret key is exponent $d$ such that $ed = 1 \bmod \varphi(N)$; signing a message $m \in \mathbb{Z}_N^*$ is done by computing $\sigma = m^d \bmod N$ (and verification is done in the obvious way).

Unfortunately, this approach is *not* secure! We now explore why. For one thing, note that no matter what trapdoor permutation is used, an adversary can always forge valid signatures on new messages by choosing arbitrary $\sigma \in \mathcal{D}_{\mathsf{k}}$, computing $m = f_{\mathsf{k}}(\sigma)$ (recall the adversary knows $\mathsf{k}$ since this is the public key), and claiming that $\sigma$ is a signature on $m$. Now, $m$ may not correspond to a meaningful message, but *we do not want to make any assumptions about what sorts of things a signature scheme will be used to sign.* So, a forgery on *any* message still "counts" as a forgery, and the scheme is not at all secure.

The situation is even worse when *specific* trapdoor permutations are used. For example, in the case of RSA we can note that an adversary who obtains signatures from the real signer can forge a signature on a new message *of his choice*. Thus, if an adversary wants to forge a signature on message $m$ the adversary can do the following: find $m_1, m_2 \in \mathbb{Z}_N^*$ such that $m_1 m_2 = m \bmod N$. Ask for signatures $\sigma_1$ and $\sigma_2$ on messages $m_1$ and $m_2$, respectively. The adversary outputs forgery $\sigma = \sigma_1 \sigma_2 \bmod N$ on the message $m$. To see that this is indeed a forgery, note that

$$\sigma^e = (\sigma_1 \sigma_2)^e = \sigma_1^e \sigma_2^e = m_1 m_2 = m \bmod N.$$

## 2 Construction of Secure 1-Time Signature Schemes

In fact, constructing secure signature schemes is very difficult; it is fair to say that no truly satisfactory *provably-secure* solution is yet known. As a start, we can consider relaxing the definition of security to see what can be achieved then. As in the case of message authentication, we consider the case when the adversary is limited to getting a single signature on a message of his choice before having to forge a new signature. Schemes which are secure against this form of attack are called *1-time signature schemes.*

The first provably-secure 1-time signature scheme is due to Lamport. This scheme can be based on any one-way function $f$, and works as follows:

- Assume we want to be able to sign messages of length $\ell$. To generate keys, choose $2\ell$ elements $x_{1,0}, x_{1,1}, \ldots, x_{\ell,0}, x_{\ell,1}$ uniformly at random from the domain of $f$. Then compute $y_{1,0} = f(x_{1,0}), y_{1,1} = f(x_{1,1}), \ldots, y_{\ell,0} = f(x_{\ell,0}), y_{\ell,1} = f(x_{\ell,1})$. The public key is $(y_{1,0}, y_{1,1}, \ldots, y_{\ell,0}, y_{\ell,1})$ and the secret key is $(x_{1,0}, x_{1,1}, \ldots, y_{\ell,0}, y_{\ell,1})$.

- To sign message $m = b_1 \cdots b_\ell$, simply let the signature be

$$(x_{1,b_1}, x_{2,b_2}, \ldots, x_{\ell,b_\ell}).$$

- To verify a signature $(x_1, \ldots, x_\ell)$ on message $m = b_1 \cdots b_\ell$, check whether

$$f(x_1) \stackrel{?}{=} y_{1,b_1}; f(x_2) \stackrel{?}{=} y_{2,b_2}; \cdots ; f(x_\ell) \stackrel{?}{=} y_{\ell,b_\ell}.$$

  (Only if all of these are true is the signature considered valid.)

We will see a proof of security for this scheme next time.