Chapter 13

Digital signatures

In this chapter and the next we develop the concept of a digital signature. Although there are some parallels between physical world signatures and digital signatures, the two are quite different. We motivate digital signatures with three examples.

Software distribution. Suppose a software company, SoftAreUs, releases a software update for its product. Customers download the software update file F by some means, say from a public distribution site or from a peer to peer network. Before installing F on their machine, customers want to verify that F really is from SoftAreUs. To facilitate this, SoftAreUs appends a short tag to F, called a signature. Only SoftAreUs can generate a signature on F, but anyone in the world can verify it. Note that there are no secrecy issues here — the file F is available in the clear to everyone. MACs, unfortunately, are of no use in these settings since SoftAreUs does not maintain a shared secret key with each of its customers. We need a new cryptographic mechanism called a digital signature. In more detail, the signing process works as follows:

- First, SoftAreUs generates a secret signing key sk along with some corresponding public key denoted pk. SoftAreUs keeps the secret key sk to itself. The public key pk is hard-coded into all copies of the software sold by SoftAreUs and is used to verify signatures issued using sk.
- To sign a software update file F SoftAreUs runs a signing algorithm S giving it sk and F as input. The algorithm outputs a short signature σ. SoftAreUs then ships the pair (F, σ) to all its customers.
- A customer Bob, given (F, σ), checks validity of this message-signature pair using a signature verification algorithm V that takes (pk, F, σ) as input. The algorithm outputs either accept or reject depending on whether the signature is valid or not. Note that Bob already has pk since it came pre-installed with the initial software package.

This mechanism is widely used in practice in a variety of software update systems. We emphasize that a digital signature σ is a function of the data being signed. This is very different from signatures in the physical world where the signature is always the same no matter what document is being signed.

Authenticated email. As a second motivating example suppose Bob receives an email claiming to be from his friend Alice. Bob wants to verify that the email really is from Alice. A MAC would do the job, but requires that Alice and Bob have a shared secret key. What if they never met before and do not share a secret key? Digital signatures provide a simple solution. First, Alice generates a public/secret key pair (pk, sk). For now, assume Alice places pk in a public read-only directory. We will get rid of this directory in just a minute.

When sending an email m to Bob, Alice generates a signature σ on m derived using her secret key. She then sends (m, σ) to Bob. Bob receives (m, σ) and verifies that m is from Alice in two steps. First, Bob retrieves Alice's public key pk. Second, Bob runs the signature verification algorithm on the triple (pk, m, σ). If the algorithm outputs accept then Bob is assured that the message came from Alice. More precisely, Bob is assured that the message was sent by someone who knows Alice's secret key. Normally this would only be Alice, but if Alice's key is stolen then the message could have come from the thief.

Non-repudiation. An interesting property of the authenticated email system above is that Bob now has evidence that the message m is from Alice. He could show the pair (m, σ) to a judge who could also verify Alice's signature. Thus, for example, if m says that Alice agrees to sell her car to Bob then Alice is (in some sense) now committed to this transaction. Bob can use Alice's signature as proof that Alice agreed to sell her car to Bob — the signature binds Alice to the message m. This property provided by digital signatures is often called non-repudiation.

Unfortunately, things are not quite that simple. Alice can repudiate the signature by claiming that the public key pk is not hers and therefore the signature was not issued by her. Or she could claim that her secret key sk was stolen and the signature was issued by the thief. After all, computers are compromised and keys are stolen all the time. Even worse, Alice could deliberately leak her secret key right after generating it there by invalidating all her signatures. The judge at this point has no idea who to believe.

The preceding paragraph may explain why digital signatures are not often used for legal purposes. Digital signatures are just a cryptographic tool used for authenticating data. They are a useful building block for higher level systems such as key-exchange protocols, but have little to do with the legal system. Several legislative efforts in the U.S. and Europe attempt to clarify the process of digitally signing a document. In the U.S., for example, electronically signing a document does not require a cryptographic digital signature. We discuss the legal aspects of digital signatures in Section 13.7.

The issue of "evidence" provided by a digital signature does not come up in the context of MACs since MACs are non-binding. To see why, suppose Alice and Bob share a secret key. Now Alice sends a message to Bob with an attached tag. Bob cannot use the tag to convince a judge that the message is from Alice since Bob could have just as easily generated the tag himself. Hence Alice can easily deny ever sending the message. The asymmetry of a digital signature makes it harder (though not impossible) for Alice to deny sending a signed message.

Certificates. As a third motivating example for digital signatures, we consider their most widely used application. In Chapter 11 and in the authenticated email system above, we assumed public keys are obtained from a read-only public directory. In practice, however, there is no public directory. Instead, Alice's public key pk_{Alice} is certified by some third party called a **certificate authority** or CA for short. We will see how this process works in more detail in Chapter 15. For now, we briefly explain how signatures are used in the certification process.

To generate a certified public key, Alice first generates a public/private key pair (pk, sk) for some public key encryption system as discussed in Chapter 11. Next, Alice sends her pk to the CA. The CA verifies that Alice is who she claims to be, and once the CA is convinced that it is speaking with Alice, the CA constructs a statement m saying "public key pk belongs to Alice." Finally, the CA signs the message m using its own secret key $\mathrm{sk}_{\mathrm{CA}}$ and sends the pair $c := (m, \sigma_{\mathrm{CA}})$ back to Alice. This pair c is called a **certificate** for pk. When Bob wants to obtain Alice's public key he downloads Alice's certificate from Alice's web site. Bob verifies the CA's signature in the certificate and if it verifies Bob has some confidence that pk is Alice's public key. The main purpose of the digital signature here is to prove to Bob that the statement m was issued by the CA. Of course, to verify the CA's signature Bob needs the CA's public key $\mathrm{pk}_{\mathrm{CA}}$. Typically, CA public keys come pre-installed with an operating system or a web browser. In other words, we simply assume that the CA's public key is already available on Bob's machine.

13.1 Definition of a digital signature

Now that we have an intuitive grasp of how digital signature systems work we can define them more precisely. Functionally, digital signatures are similar to MACs. The main difference is that in a MAC both the signing and verification algorithms use the same secret key. In a signature system the signing algorithm uses one key, sk, while the verification algorithm uses another, pk.

Definition 13.1. *A* ***signature*** *system $\mathcal{S} = (G, S, V)$ is a triple of efficient algorithms, G, S and V, where G is called a* ***key generation algorithm****, S is called a* ***signing algorithm****, and V is called a* ***verification algorithm****. Algorithm S is used to generate signatures and algorithm V is used to verify signatures.*

- *G is a probabilistic algorithm that takes no input. It outputs a pair $(\mathrm{pk}, \mathrm{sk})$ where sk is called a secret* ***signing key*** *and pk is called a public* ***verification key****.*
- *S is a probabilistic algorithm that is invoked as $\sigma \overset{R}{\leftarrow} E(\mathrm{sk}, m)$, where sk is a secret key (as output by G) and m is a message. The algorithm outputs a* ***signature*** *σ.*
- *V is a deterministic algorithm invoked as $V(\mathrm{pk}, m, \sigma)$. It outputs either* accept *or* reject*.*
- *We require that signatures generated by S are always accepts by V. That is, for $(\mathrm{pk}, \mathrm{sk}) \overset{R}{\leftarrow} G()$ the system (G, S, V) must satisfy the following* ***correctness property:***

$$\Pr[V(\mathrm{pk},\ m,\ S(\mathrm{sk},\ m)\ ) = \mathsf{accept}] = 1.$$

As usual, we say that messages lie in a finite ***message space*** *$\mathcal{M}$, and signatures lie in some finite* ***signature space*** *Σ. We say that $\mathcal{S} = (G, S, V)$ is defined over $(\mathcal{M}, \Sigma)$.*

13.1.1 Secure signatures

The definition of secure signatures is similar to the definition of secure MACs. We give the adversary the power to mount a **chosen message attack**, namely the attacker can request the signature on any message of his choice. Even with such power, the adversary should not be able to create an **existential forgery**, namely the attacker cannot output a valid message-signature pair (m, σ) for some new message m. Here "new" means a message that the adversary did not previously request a signature for.

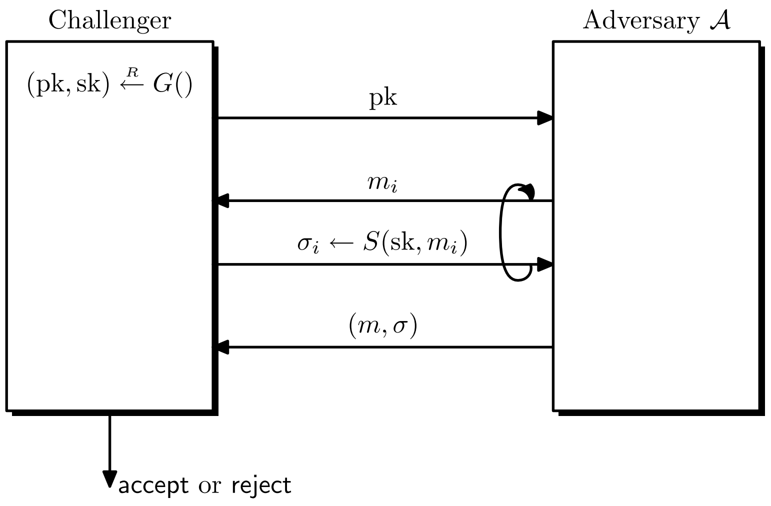
Challenger
Adversary $\mathcal{A}$
$(\mathrm{pk}, \mathrm{sk}) \xleftarrow{R} G()$
pk
m_i
$\sigma_i \leftarrow S(\mathrm{sk}, m_i)$
(m, σ)
accept or reject

Figure 13.1: Signature attack game (Attack Game 13.1)

More precisely, we define secure signatures using an attack game between a challenger and an adversary $\mathcal{A}$. The game is described below and in Fig. 13.1.

Attack Game 13.1 (Signature security). For a given signature system $\mathcal{S} = (G, S, V)$, defined over $(\mathcal{M}, \Sigma)$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger runs $(\mathrm{pk}, \mathrm{sk}) \xleftarrow{R} G()$ and sends pk to $\mathcal{A}$.
- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots$, the ith query consists of a message $m_i \in \mathcal{M}$. The challenger computes $\sigma_i \xleftarrow{R} S(\mathrm{sk}, m_i)$ and gives σ_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, \sigma) \in \mathcal{M} \times \Sigma$.

We say that the adversary wins the game if the following two conditions hold:

- $V(\mathrm{pk}, m, \sigma) = \textsf{accept}$, and
- m is new, namely $m \notin \{m_1, m_2, \ldots\}$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$, denoted $\mathrm{SIGAdv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. The queries issued by $\mathcal{A}$ during Attack Game 13.1 are called **signature queries**. Finally, we say that $\mathcal{A}$ is a q**-query adversary** if $\mathcal{A}$ issues at most q signature queries. $\square$

Definition 13.2. *We say that a signature system $\mathcal{S}$ is secure if for all efficient adversaries $\mathcal{A}$, the quantity* $\mathrm{SIGAdv}[\mathcal{A}, \mathcal{S}]$ *is negligible.*

In case the adversary wins Attack Game 13.1, the pair (m, σ) it outputs is called an **existential forgery**. Systems that satisfy Definition 13.2 are said to be **existentially unforgeable under a chosen message attack**.

Binding signatures. The security definition ensures that generating signatures without the secret key is difficult. The definition, however, does not require that the signer be bound to