Chapter 13

Digital Signature Schemes

13.1 Digital Signatures – An Overview

In the previous chapter we explored how public-key encryption can be used to achieve *secrecy* in the public-key setting. *Integrity* (or *authenticity*) in the public-key setting is provided using *digital signature schemes.* These can be viewed as the public-key analogue of message authentication codes although, as we will see, there are several important differences between these primitives.

Signature schemes allow a *signer* S who has established a public key pk to "sign" a message using the associated private key sk in such a way that anyone who knows pk (and knows that this public key was established by S) can *verify* that the message originated from S and was not modified in transit. (Note that, in contrast to public-key encryption, in the context of digital signatures the owner of the public key acts as the *sender.*) As a prototypical application, consider a software company that wants to disseminate software updates in an authenticated manner; that is, when the company releases an update it should be possible for any of its clients to verify that the update is authentic, and a malicious third party should never be able to fool a client into accepting an update that was not actually released by the company. To do this, the company can generate a public key pk along with a private key sk, and then distribute pk in some reliable manner to its clients while keeping sk secret. (As in the case of public-key encryption, we assume that this initial distribution of the public key is carried out correctly so that all clients have a correct copy of pk. In the current example, pk could be bundled with the original software purchased by a client.) When releasing a software update m, the company computes a digital signature σ on m using its private key sk, and sends (m, σ) to every client. Each client can verify the authenticity of m by checking that σ is a correct signature on m with respect to the public key pk.

A malicious party might try to issue a fraudulent update by sending (m', σ') to a client, where m' represents an update that was never released by the company. This m' might be a modified version of some previous update, or it might be completely new and unrelated to any prior updates. If the signature scheme is "secure" (in a sense we will define more carefully soon), however, then when the client attempts to verify σ' it will find that this is an *invalid* signature on m' with respect to pk, and will therefore *reject* the signature. The client will reject even if m' is modified only slightly from a genuine update m.

The above is not just a theoretical application of digital signatures, but one that is in widespread use today for distributing software updates.

Comparison to Message Authentication Codes

Both message authentication codes and digital signature schemes are used to ensure the integrity of transmitted messages. Although the discussion in Chapter 11 comparing the public-key and private-key settings focused mainly on encryption, that discussion applies also to message integrity. Using digital signatures rather than message authentication codes simplifies key distribution and management, especially when a sender needs to communicate with multiple receivers as in the software-update example above. By using a digital signature scheme the sender avoids having to establish a distinct secret key with each potential receiver, and avoids having to compute a separate MAC tag with respect to each such key. Instead, the sender need only compute a single signature that can be verified by all recipients.

A *qualitative* advantage that digital signatures have as compared to message authentication codes is that signatures are *publicly verifiable*. This means that if a receiver verifies that a signature on a given message is legitimate, then all other parties who receive this signed message will also verify it as legitimate. This feature is not achieved by message authentication codes if the signer shares a separate key with each receiver: in such a setting a malicious sender might compute a correct MAC tag with respect to the key it shares with receiver A but an incorrect MAC tag with respect to the key it shares with a different user B. In this case, A knows that he received an authentic message from the sender but has no guarantee that B will agree.

Public verifiability implies that signatures are *transferable*: a signature σ on a message m by a signer S can be shown to a third party, who can then verify herself that σ is a legitimate signature on m with respect to S's public key (here, we assume this third party also knows S's public key). By making a copy of the signature, this third party can then show the signature to another party and convince *them* that S authenticated m, and so on. Public verifiability and transferability are essential for the application of digital signatures to certificates and public-key infrastructures, as we will discuss in Section 13.6.

Digital signature schemes also provide the very important property of *non-repudiation*. This means that once S signs a message he cannot later deny having done so (assuming the public key of S is widely publicized and distributed). This aspect of digital signatures is crucial for legal applications where a recipient may need to prove to a third party (say, a judge) that a signer did indeed "certify" a particular message (e.g., a contract): assuming S's public key is known to the judge, or is otherwise publicly available, a valid signature on a message serves as convincing evidence that S indeed signed that message. Message authentication codes simply cannot provide non-repudiation. To see this, say users S and R share a key k_{SR}, and S

sends a message m to R along with a (valid) MAC tag t computed using this key. Since the judge does *not* know k_{SR} (indeed, this key is kept secret by S and R), there is no way for the judge to determine whether t is valid or not. If R were to reveal the key k_{SR} to the judge, there would be no way for the judge to know whether this is the "actual" key that S and R shared, or whether it is some "fake" key manufactured by R. Finally, even if we assume the judge can somehow obtain the actual key k_{SR} shared by the parties, there is no way for the judge to distinguish whether S generated t or whether R did—this is because message authentication codes are a *symmetric*-key primitive; anything S can do, R can do also.

As in the case of private-key vs. public-key encryption, message authentication codes have the advantage of being shorter and roughly 2–3 orders of magnitude more efficient to generate/verify than digital signatures. Thus, in situations where public verifiability, transferability, and/or non-repudiation are not needed, and the sender communicates primarily with a single recipient (with whom it is able to share a secret key), message authentication codes should be used.

Relation to Public-Key Encryption

Digital signatures are often mistakenly viewed as the "inverse" of public-key encryption, with the roles of the sender and receiver interchanged. Historically,[1] in fact, it has been suggested that digital signatures can be obtained by "reversing" public-key encryption, i.e., signing a message m by decrypting it (using the private key) to obtain σ, and verifying a signature σ by encrypting it (using the corresponding public key) and checking whether the result is m. The suggestion to construct signature schemes in this way is *completely unfounded*: in most cases, it is simply inapplicable, and even in cases where it can be applied it results in signature schemes that are not secure.

13.2 Definitions

Digital signatures are the public-key counterpart of message authentication codes, and their syntax and security guarantees are analogous. The algorithm that the sender applies to a message is here denoted Sign (rather than Mac), and the output of this algorithm is now called a *signature* (rather than a tag).

[1] This view no doubt arose because, as we will see in Section 13.4.1, plain RSA signatures are the reverse of plain RSA encryption. However, neither plain RSA signatures nor plain RSA encryption meet even minimal notions of security.

The algorithm that the receiver applies to a message and a signature in order to check validity is still denoted Vrfy.

DEFINITION 13.1 *A* (digital) signature scheme *consists of three probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *such that:*

1. *The* key-generation algorithm Gen *takes as input a security parameter* 1^n *and outputs a pair of keys* (pk, sk)*. These are called the* public key *and the* private key*, respectively. We assume that* pk *and* sk *each has length at least* n*, and that* n *can be determined from* pk *or* sk*.*

2. *The* signing algorithm Sign *takes as input a private key* sk *and a message* m *from some message space (that may depend on* pk*). It outputs a signature* σ*, and we write this as* $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$*.*

3. *The deterministic* verification algorithm Vrfy *takes as input a public key* pk*, a message* m*, and a signature* σ*. It outputs a bit* b*, with* $b = 1$ *meaning* valid *and* $b = 0$ *meaning* invalid*. We write this as* $b := \mathsf{Vrfy}_{pk}(m, \sigma)$*.*

It is required that except with negligible probability over (pk, sk) *output by* $\mathsf{Gen}(1^n)$*, it holds that* $\mathsf{Vrfy}_{pk}(m, \mathsf{Sign}_{sk}(m)) = 1$ *for every (legal) message* m*.*

If there is a function ℓ *such that for every* (pk, sk) *output by* $\mathsf{Gen}(1^n)$ *the message space is* $\{0,1\}^{\ell(n)}$*, then we say that* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is a* signature scheme for messages of length $\ell(n)$*.*

We call σ a *valid signature* on a message m (with respect to some public key pk understood from the context) if $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

A signature scheme is used in the following way. One party S, who acts as the *sender*, runs $\mathsf{Gen}(1^n)$ to obtain keys (pk, sk). The public key pk is then publicized as belonging to S; e.g., S can put the public key on its webpage or place it in some public directory. As in the case of public-key encryption, we assume that any other party is able to obtain a legitimate copy of S's public key (see discussion below). When S wants to authenticate a message m, it computes the signature $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$ and sends (m, σ). Upon receipt of (m, σ), a receiver who knows pk can verify the authenticity of m by checking whether $\mathsf{Vrfy}_{pk}(m, \sigma) \stackrel{?}{=} 1$. This establishes both that S sent m, and also that m was not modified in transit. As in the case of message authentication codes, however, it does not say anything about *when* m was sent, and replay attacks are still possible (see Section 4.2).

The assumption that parties are able to obtain a legitimate copy of S's public key implies that S is able to transmit at least one message (namely, pk itself) in a reliable and authenticated manner. If S is able to transmit messages reliably, however, then why does it need a signature scheme at all? The answer is that reliable distribution of pk may be difficult and expensive, but using a signature scheme means that such distribution need only be carried out *once*, after which an unlimited number of messages can subsequently be sent reliably. Furthermore, as we will discuss in Section 13.6, signature schemes themselves are used to ensure the reliable distribution of *other* public keys. They thus serve as a central tool for setting up a "public-key infrastructure" to address the key-distribution problem.

Security of signature schemes. For a fixed public key pk generated by a signer S, a *forgery* is a message m along with a valid signature σ, where m was not previously signed by S. Security of a signature scheme means that an adversary should be unable to output a forgery even if it obtains signatures on many other messages of its choice. This is the direct analogue of the definition of security for message authentication codes, and we refer the reader to Section 4.2 for motivation and further discussion.

The formal definition of security is essentially the same as Definition 4.2, with the main difference being that here the adversary is given a public key. Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and consider the following experiment for an adversary $\mathcal{A}$ and parameter n:

The signature experiment $\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n)$:

1. $\mathsf{Gen}(1^n)$ *is run to obtain keys* (pk, sk).
2. *Adversary* $\mathcal{A}$ *is given* pk *and access to an oracle* $\mathsf{Sign}_{sk}(\cdot)$. *The adversary then outputs* (m, σ). *Let* $\mathcal{Q}$ *denote the set of all queries that* $\mathcal{A}$ *asked its oracle.*
3. $\mathcal{A}$ succeeds *if and only if (1)* $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$ *and (2)* $m \notin \mathcal{Q}$. *In this case the output of the experiment is defined to be* 1.

DEFINITION 13.2 *A signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* existentially unforgeable under an adaptive chosen-message attack, *or just* secure, *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$, *there is a negligible function* negl *such that:*

$$\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

Strong security can be defined analogously to Definition 4.3.

13.3 The Hash-and-Sign Paradigm

As in the case of public-key vs. private-key encryption, "native" signature schemes are orders of magnitude less efficient than message authentication codes. Fortunately, as with hybrid encryption (see Section 12.3), it is possible to obtain the functionality of digital signatures at the asymptotic cost of a private-key operation, at least for sufficiently long messages. This can be done using the *hash-and-sign* approach, discussed next.

The intuition behind the hash-and-sign approach is straightforward. Say we have a signature scheme for messages of length ℓ, and wish to sign a (longer) message $m \in \{0,1\}^*$. Rather than sign m itself, we can instead use a hash function H to *hash* the message to a fixed-length *digest* $H(m)$ of length ℓ, and then sign the resulting digest. This approach is exactly analogous to the hash-and-MAC approach discussed in Section 6.3.1.

CONSTRUCTION 13.3

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme for messages of length $\ell(n)$, and let $\Pi_H = (\mathsf{Gen}_H, H)$ be a hash function with output length $\ell(n)$. Construct signature scheme $\Pi' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ as follows:

- Gen': on input 1^n, run $\mathsf{Gen}(1^n)$ to obtain (pk, sk) and run $\mathsf{Gen}_H(1^n)$ to obtain s; the public key is $\langle pk, s\rangle$ and the private key is $\langle sk, s\rangle$.
- Sign': on input a private key $\langle sk, s\rangle$ and a message $m \in \{0,1\}^*$, output $\sigma \leftarrow \mathsf{Sign}_{sk}(H^s(m))$.
- Vrfy': on input a public key $\langle pk, s\rangle$, a message $m \in \{0,1\}^*$, and a signature σ, output 1 if and only if $\mathsf{Vrfy}_{pk}(H^s(m), \sigma) \stackrel{?}{=} 1$.

The hash-and-sign paradigm.

THEOREM 13.4 *If Π is a secure signature scheme for messages of length ℓ and Π_H is collision resistant, then Construction 13.3 is a secure signature scheme (for arbitrary-length messages).*

The proof of this theorem is almost identical to that of Theorem 6.6.

13.4 RSA-Based Signatures

We begin our consideration of concrete signature schemes with a discussion of schemes based on the RSA assumption.

13.4.1 Plain RSA Signatures

We first describe a simple, RSA-based signature scheme. Although the scheme is insecure, it serves as a useful starting point.