Chapter 10
Digital Signatures

Digital signatures are one of the most important cryptographic tools they and are widely used today. Applications for digital signatures range from digital certificates for secure e-commerce to legal signing of contracts to secure software updates. Together with key establishment over insecure channels, they form the most important instance for public-key cryptography.

Digital signatures share some functionality with handwritten signatures. In particular, they provide a method to assure that a message is authentic to one user, i.e., it in fact originates from the person who claims to have generated the message. However, they actually provide much more functionality, as we'll learn in this chapter.

In this chapter you will learn:

- The principle of digital signatures
- Security services, that is, the specific objectives that can be achieved by a security system
- The RSA digital signature scheme
- The Elgamal digital signature scheme and two extensions of it, the digital signature algorithm (DSA) and the elliptic curve digital signature algorithm (ECDSA)

10.1 Introduction

In this section, we first provide a motivating example why digital signatures are needed and why they must be based on asymmetric cryptography. We then develop the principles of digital signatures. Actual signature algorithms are introduced in subsequent sections.

10.1.1 Odd Colors for Cars, or: Why Symmetric Cryptography Is Not Sufficient

The crypto schemes that we have encountered so far had two main goals: either to encrypt data (e.g., with AES, 3DES or RSA encryption) or to establish a shared key (e.g., with the Diffie–Hellman or elliptic curve key exchange). One might be tempted to think that we are now in a position to satisfy any security needs that arise in practice. However, there are many other security needs besides encryption and key exchange, which are in fact termed security services; these are discussed in detail in Sect. 10.1.3. We now discuss a setting in which symmetric cryptography fails to provide a desirable security function.

Assume we have two communicating parties, Alice and Bob, who share a secret key. Furthermore, the secret key is used for encryption with a block cipher. When Alice receives and decrypts a message which makes semantic sense, e.g., the decrypted message is an actual (English) text, she can in many cases conclude that the message was in fact generated by a person with whom he shares the secret key[1]. If only Alice and Bob know the key, they can be reasonably sure that an attacking third party has not changed the message in transit. So far we've always assumed that the bad guy is an external party that we often named Oscar. However, in practice it is often the case that Alice and Bob do want to communicate securely with each other, but at the same time they might be interested in cheating each other. It turns out that symmetric-key schemes do not protect the two parties *against each other*. Consider the following scenario:

Suppose that Alice owns a dealership for new cars where you can select and order cars online. We assume that Bob, the customer, and Alice, the dealer, have established a shared secret k_{AB}, e.g., by using the Diffie–Hellman key exchange. Bob now specifies the car that he likes, which includes a color choice of pink for the interior and an external color of orange — choices most people would not make. He sends the order form AES-encrypted to Alice. She decrypts the order and is happy to have sold another model for $25,000. Upon delivery of the car three weeks later, Bob has second thoughts about his choice, in part because his spouse is threatening

[1] One has to be a bit careful with such a conclusion, though. For instance, if Alice and Bob use a stream cipher an attacker can flip individual bits of the ciphertext, which results in bit flips in the received plaintext. Depending on the application, the attacker might be able to manipulate the message in a way that is semantically still correct. However, using block ciphers, especially in a chaining mode, makes it quite likely that ciphertext manipulations can be detected after decryption.

him with divorce after *seeing* the car. Unfortunately for Bob (and his family), Alice has a "no return" policy. Given that she is an experienced car dealer, she knows too well that it will not be easy to sell a pink and orange car, and she is thus set on not making any exceptions. Since Bob now claims that he never ordered the car, she has no other choice but to sue him. In front of the judge, Alice's lawyer presents Bob's digital car order together with the encrypted version of it. Obviously, the lawyer argues, Bob must have generated the order since he is in possession of k_{AB} with which the ciphertext was generated. However, if Bob's lawyer is worth his money, he will patiently explain to the judge that the car dealer, Alice, also knows k_{AB} and that Alice has, in fact, a high incentive to generate faked car orders. The judge, it turns out, has no way of knowing whether the plaintext–ciphertext pair was generated by Bob or Alice! Given the laws in most countries, Bob probably gets away with his dishonesty.

This might sound like a rather specific and somewhat artificially constructed scenario, but in fact it is not. There are many, many situations where it is important to prove to a neutral third party, i.e., a person acting as a judge, that one of two (or more) parties generated a message. By *proving* we mean that the judge can conclude without doubt who has generated the message, even if all parties are potentially dishonest. Why can't we use some (complicated) symmetric-key scheme to achieve this goal? The high-level explanation is simple: Exactly because we have a symmetric set-up, Alice and Bob have the same knowledge (namely of keys) and thus the same capabilities. Everything that Alice can do can be done by Bob, too. Thus, a neutral third party cannot distinguish whether a certain cryptographic operation was performed by Alice or by Bob or by both. Generally speaking, the solution to this problem lies in public-key cryptography. The asymmetric set-up that is inherent in public-key algorithms might potentially enable a judge to distinguish between actions that only one person can perform (namely the person in possession of the private key), and those that can be done by both (namely computations involving the public key). It turns out that digital signatures are public-key algorithms which have the properties that are needed to resolve a situation of cheating participants. In the e-commerce car scenario above, Bob would have been required to digitally sign his order using his private key.

10.1.2 Principles of Digital Signatures

The property of proving that a certain person generated a message is obviously also very important outside the digital domain. In the real, "analog" world, this is achieved by handwritten signatures on paper. For instance, if we sign a contract or sign a check, the receiver can prove to a judge that we actually signed the message. (Of course, one can try to forge signatures, but there are legal and social barriers that prevent most people from even attempting to do so.) As with conventional handwritten signatures, only the person who creates a digital message must be capable of generating a valid signature. In order to achieve this with cryptographic primi-

tives, we have to apply public-key cryptography. The basic idea is that the person who signs the message uses a private key, and the receiving party uses the matching public key. The principle of a digital signature scheme is shown in Fig. 10.1.

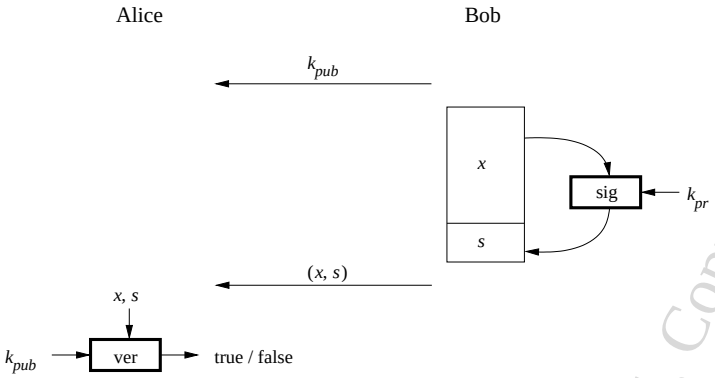

Fig. 10.1 Principle of digital signatures which involves signing and verifying a message

The process starts with Bob signing the message x. The signature algorithm is a function of Bob's private key, k_{pr}. Hence, assuming he in fact keeps his private key private, only Bob can sign a message x on his behalf. In order to relate a signature to the message, x is also an input to the signature algorithm. After signing the message, the signature s is appended to the message x and the pair (x, s) is sent to Alice. It is important to note that a digital signature by itself is of no use unless it is accompanied by the message. A digital signature without the message is the equivalent of a handwritten signature on a strip of paper without the contract or a check that is supposed to be signed.

The digital signature itself is merely a (large) integer value, for instance, a string of 2048 bits. The signature is only useful to Alice if she has means to *verify* whether the signature is valid or not. For this, a verification function is needed which takes both x and the signature s as inputs. In order to link the signature to Bob, the function also requires his public key. Even though the verification function has long inputs, its only output is the binary statement "true" or "false". If x was actually signed with the private key that belongs to the public verification key, the output is true, otherwise it is false.

From these general observations we can easily develop a generic digital signature protocol: