Chapter 11

Digital Signatures

Contents in Brief

11.1 Introduction

This chapter considers techniques designed to provide the digital counterpart to a handwritten signature. A *digital signature* of a message is a number dependent on some secret known only to the signer, and, additionally, on the content of the message being signed. Signatures must be verifiable; if a dispute arises as to whether a party signed a document (caused by either a lying signer trying to *repudiate* a signature it did create, or a fraudulent claimant), an unbiased third party should be able to resolve the matter equitably, without requiring access to the signer's secret information (private key).

Digital signatures have many applications in information security, including authentication, data integrity, and non-repudiation. One of the most significant applications of digital signatures is the certification of public keys in large networks. Certification is a means for a trusted third party (TTP) to bind the identity of a user to a public key, so that at some later time, other entities can authenticate a public key without assistance from a trusted third party.

The concept and utility of a digital signature was recognized several years before any practical realization was available. The first method discovered was the RSA signature scheme, which remains today one of the most practical and versatile techniques available. Subsequent research has resulted in many alternative digital signature techniques. Some offer significant advantages in terms of functionality and implementation. This chapter is an account of many of the results obtained to date, with emphasis placed on those developments which are practical.

Chapter outline

§11.2 provides terminology used throughout the chapter, and describes a framework for digital signatures that permits a useful classification of the various schemes. It is more abstract than succeeding sections. §11.3 provides an indepth discussion of the RSA signature scheme, as well as closely related techniques. Standards which have been adopted to implement RSA and related signature schemes are also considered here. §11.4 looks at methods which arise from identification protocols described in Chapter 10. Techniques based on the intractability of the discrete logarithm problem, such as the Digital Signature Algorithm (DSA) and ElGamal schemes, are the topic of §11.5. One-time signature schemes, many of which arise from symmetric-key cryptography, are considered in §11.6. §11.7 describes arbitrated digital signatures and the ESIGN signature scheme. Variations on the basic concept of digital signatures, including blind, undeniable, and fail-stop signatures, are discussed in §11.8. Further notes, including subtle points on schemes documented in the chapter and variants (e.g., designated confirmer signatures, convertible undeniable signatures, group signatures, and electronic cash) may be found in §11.9.

11.2 A framework for digital signature mechanisms

§1.6 provides a brief introduction to the basic ideas behind digital signatures, and §1.8.3 shows how these signatures can be realized through reversible public-key encryption techniques. This section describes two general models for digital signature schemes. A complete understanding of the material in this section is not necessary in order to follow subsequent sections; the reader unfamiliar with some of the more concrete methods such as RSA (§11.3) and ElGamal (§11.5) is well advised not to spend an undue amount of time. The idea of a redundancy function is necessary in order to understand the algorithms which give digital signatures with message recovery. The notation provided in Table 11.1 will be used throughout the chapter.

11.2.1 Basic definitions

1. A *digital signature* is a data string which associates a message (in digital form) with some originating entity.
2. A *digital signature generation algorithm* (or *signature generation algorithm*) is a method for producing a digital signature.
3. A *digital signature verification algorithm* (or *verification algorithm*) is a method for verifying that a digital signature is authentic (i.e., was indeed created by the specified entity).
4. A *digital signature scheme* (or *mechanism*) consists of a signature generation algorithm and an associated verification algorithm.
5. A *digital signature signing process* (or *procedure*) consists of a (mathematical) digital signature generation algorithm, along with a method for formatting data into messages which can be signed.
6. A *digital signature verification process* (or *procedure*) consists of a verification algorithm, along with a method for recovering data from the message.[1]

[1] Often little distinction is made between the terms scheme and process, and they are used interchangeably.

This chapter is, for the most part, concerned simply with digital signature schemes. In order to use a digital signature scheme in practice, it is necessary to have a digital signature process. Several processes related to various schemes have emerged as commercially relevant standards; two such processes, namely ISO/IEC 9796 and PKCS #1, are described in §11.3.5 and §11.3.6, respectively. Notation used in the remainder of this chapter is provided in Table 11.1. The sets and functions listed in Table 11.1 are all publicly known.

Notation	Meaning
$\mathcal{M}$	a set of elements called the *message space*.
$\mathcal{M}_\mathcal{S}$	a set of elements called the *signing space*.
$\mathcal{S}$	a set of elements called the *signature space*.
R	a $1-1$ mapping from $\mathcal{M}$ to $\mathcal{M}_\mathcal{S}$ called the *redundancy function*.
$\mathcal{M}_R$	the image of R (i.e., $\mathcal{M}_R = \mathrm{Im}(R)$).
R^{-1}	the inverse of R (i.e., $R^{-1}\colon \mathcal{M}_R \longrightarrow \mathcal{M}$).
$\mathcal{R}$	a set of elements called the *indexing set for signing*.
h	a one-way function with domain $\mathcal{M}$.
$\mathcal{M}_h$	the image of h (i.e., $h\colon \mathcal{M} \longrightarrow \mathcal{M}_h$); $\mathcal{M}_h \subseteq \mathcal{M}_\mathcal{S}$ called the *hash value space*.

Table 11.1: *Notation for digital signature mechanisms.*

11.1 Note (*comments on Table 11.1*)

(i) (*messages*) $\mathcal{M}$ is the set of elements to which a signer can affix a digital signature.
(ii) (*signing space*) $\mathcal{M}_\mathcal{S}$ is the set of elements to which the signature transformations (to be described in §11.2.2 and §11.2.3) are applied. The signature transformations are not applied directly to the set $\mathcal{M}$.
(iii) (*signature space*) $\mathcal{S}$ is the set of elements associated to messages in $\mathcal{M}$. These elements are used to bind the signer to the message.
(iv) (*indexing set*) $\mathcal{R}$ is used to identify specific signing transformations.

A classification of digital signature schemes

§11.2.2 and §11.2.3 describe two general classes of digital signature schemes, which can be briefly summarized as follows:

1. Digital signature schemes with appendix require the original message as input to the verification algorithm. (See Definition 11.3.)
2. Digital signature schemes with message recovery do not require the original message as input to the verification algorithm. In this case, the original message is recovered from the signature itself. (See Definition 11.7.)

These classes can be further subdivided according to whether or not $|\mathcal{R}| = 1$, as noted in Definition 11.2.

11.2 Definition A digital signature scheme (with either message recovery or appendix) is said to be a *randomized digital signature scheme* if $|\mathcal{R}| > 1$; otherwise, the digital signature scheme is said to be *deterministic*.

Figure 11.1 illustrates this classification. Deterministic digital signature mechanisms can be further subdivided into *one-time signature schemes* (§11.6) and *multiple-use schemes*.

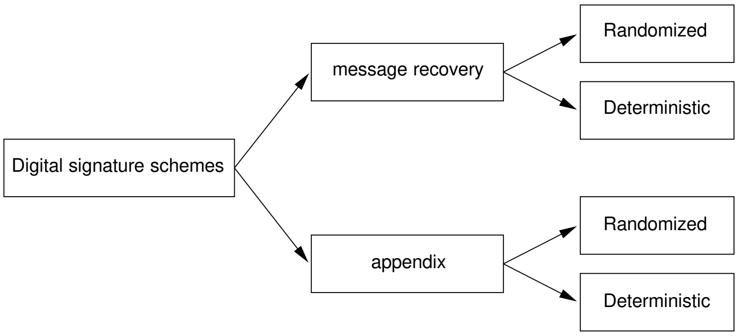


Figure 11.1: *A taxonomy of digital signature schemes.*

11.2.2 Digital signature schemes with appendix

Digital signature schemes with appendix, as discussed in this section, are the most commonly used in practice. They rely on cryptographic hash functions rather than customized redundancy functions, and are less prone to existential forgery attacks (§11.2.4).

11.3 Definition Digital signature schemes which require the message as input to the verification algorithm are called *digital signature schemes with appendix*.

Examples of mechanisms providing digital signatures with appendix are the DSA (§11.5.1), ElGamal (§11.5.2), and Schnorr (§11.5.3) signature schemes. Notation for the following discussion is given in Table 11.1.

11.4 Algorithm Key generation for digital signature schemes with appendix

SUMMARY: each entity creates a private key for signing messages, and a corresponding public key to be used by other entities for verifying signatures.

1. Each entity A should select a private key which defines a set $\mathcal{S}_A = \{S_{A,k}\colon k \in \mathcal{R}\}$ of transformations. Each $S_{A,k}$ is a 1-1 mapping from $\mathcal{M}_h$ to $\mathcal{S}$ and is called a *signing transformation*.
2. $\mathcal{S}_A$ defines a corresponding mapping V_A from $\mathcal{M}_h \times \mathcal{S}$ to {true, false} such that
$$V_A(\widetilde{m}, s^*) = \begin{cases} \text{true}, & \text{if } S_{A,k}(\widetilde{m}) = s^*, \\ \text{false}, & \text{otherwise,} \end{cases}$$
for all $\widetilde{m} \in \mathcal{M}_h$, $s^* \in \mathcal{S}$; here, $\widetilde{m} = h(m)$ for $m \in \mathcal{M}$. V_A is called a *verification transformation* and is constructed such that it may be computed without knowledge of the signer's private key.
3. A's public key is V_A; A's private key is the set $\mathcal{S}_A$.

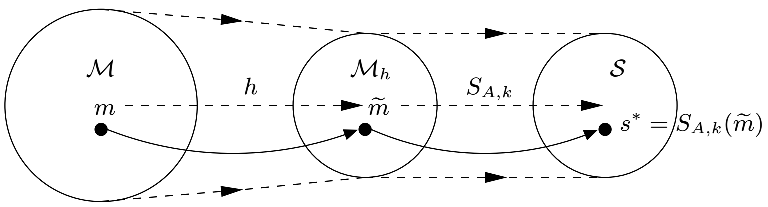
11.5 Algorithm Signature generation and verification (digital signature schemes with appendix)

SUMMARY: entity A produces a signature $s \in \mathcal{S}$ for a message $m \in \mathcal{M}$, which can later be verified by any entity B.

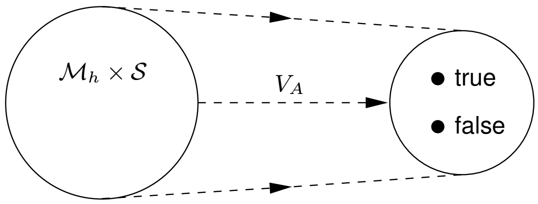
1. *Signature generation.* Entity A should do the following:
   (a) Select an element $k \in \mathcal{R}$.
   (b) Compute $\widetilde{m} = h(m)$ and $s^* = S_{A,k}(\widetilde{m})$.
   (c) A's signature for m is s^*. Both m and s^* are made available to entities which may wish to verify the signature.
2. *Verification.* Entity B should do the following:
   (a) Obtain A's authentic public key V_A.
   (b) Compute $\widetilde{m} = h(m)$ and $u = V_A(\widetilde{m}, s^*)$.
   (c) Accept the signature if and only if $u =$ true.

Figure 11.2 provides a schematic overview of a digital signature scheme with appendix. The following properties are required of the signing and verification transformations:

(i) for each $k \in \mathcal{R}$, $S_{A,k}$ should be efficient to compute;
(ii) V_A should be efficient to compute; and
(iii) it should be computationally infeasible for an entity other than A to find an $m \in \mathcal{M}$ and an $s^* \in \mathcal{S}$ such that $V_A(\widetilde{m}, s^*) =$ true, where $\widetilde{m} = h(m)$.



(a) The signing process



(b) The verification process

Figure 11.2: *Overview of a digital signature scheme with appendix.*

11.6 Note (*use of hash functions*) Most digital signature schemes with message recovery (§11.2.3) are applied to messages of a fixed length, while digital signatures with appendix are applied to messages of arbitrary length. The one-way function h in Algorithm 11.5 is typically selected to be a collision-free hash function (see Definition 9.3). An alternative to hashing is to break the message into blocks of a fixed length which can be individually signed using a signature scheme with message recovery. Since signature generation is relatively slow for many schemes, and since reordering of multiple signed blocks presents a security risk, the preferred method is to hash.

11.2.3 Digital signature schemes with message recovery

The digital signature schemes described in this section have the feature that the message signed can be recovered from the signature itself. In practice, this feature is of use for short messages (see §11.3.3(viii)).

11.7 Definition A *digital signature scheme with message recovery* is a digital signature scheme for which a priori knowledge of the message is not required for the verification algorithm.

Examples of mechanisms providing digital signatures with message recovery are RSA (§11.3.1), Rabin (§11.3.4), and Nyberg-Rueppel (§11.5.4) public-key signature schemes.

11.8 Algorithm Key generation for digital signature schemes with message recovery

SUMMARY: each entity creates a private key to be used for signing messages, and a corresponding public key to be used by other entities for verifying signatures.

1. Each entity A should select a set $\mathcal{S}_A = \{S_{A,k}\colon k \in \mathcal{R}\}$ of transformations. Each $S_{A,k}$ is a 1-1 mapping from $\mathcal{M}_S$ to $\mathcal{S}$ and is called a *signing transformation*.
2. $\mathcal{S}_A$ defines a corresponding mapping V_A with the property that $V_A \circ S_{A,k}$ is the identity map on $\mathcal{M}_S$ for all $k \in \mathcal{R}$. V_A is called a *verification transformation* and is constructed such that it may be computed without knowledge of the signer's private key.
3. A's public key is V_A; A's private key is the set $\mathcal{S}_A$.

11.9 Algorithm Signature generation and verification for schemes with message recovery

SUMMARY: entity A produces a signature $s \in \mathcal{S}$ for a message $m \in \mathcal{M}$, which can later be verified by any entity B. The message m is recovered from s.

1. *Signature generation.* Entity A should do the following:
   (a) Select an element $k \in \mathcal{R}$.
   (b) Compute $\widetilde{m} = R(m)$ and $s^* = S_{A,k}(\widetilde{m})$. ($R$ is a redundancy function; see Table 11.1 and Note 11.10.)
   (c) A's signature is s^*; this is made available to entities which may wish to verify the signature and recover m from it.
2. *Verification.* Entity B should do the following:
   (a) Obtain A's authentic public key V_A.
   (b) Compute $\widetilde{m} = V_A(s^*)$.
   (c) Verify that $\widetilde{m} \in \mathcal{M}_R$. (If $\widetilde{m} \notin \mathcal{M}_R$, then reject the signature.)
   (d) Recover m from $\widetilde{m}$ by computing $R^{-1}(\widetilde{m})$.

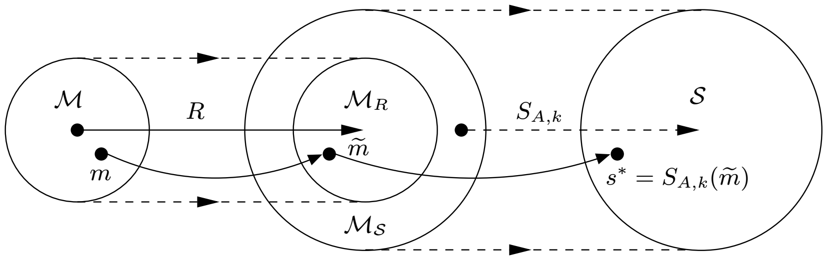


Figure 11.3: *Overview of a digital signature scheme with message recovery.*

Figure 11.3 provides a schematic overview of a digital signature scheme with message recovery. The following properties are required of the signing and verification transformations:

(i) for each $k \in \mathcal{R}$, $S_{A,k}$ should be efficient to compute;
(ii) V_A should be efficient to compute; and
(iii) it should be computationally infeasible for an entity other than A to find any $s^* \in \mathcal{S}$ such that $V_A(s^*) \in \mathcal{M}_R$.

11.10 Note (*redundancy function*) The redundancy function R and its inverse R^{-1} are publicly known. Selecting an appropriate R is critical to the security of the system. To illustrate this point, suppose that $\mathcal{M}_R = \mathcal{M}_\mathcal{S}$. Suppose R and $S_{A,k}$ are bijections from $\mathcal{M}$ to $\mathcal{M}_R$ and $\mathcal{M}_\mathcal{S}$ to $\mathcal{S}$, respectively. This implies that $\mathcal{M}$ and $\mathcal{S}$ have the same number of elements. Then for any $s^* \in \mathcal{S}, V_A(s^*) \in \mathcal{M}_R$, and it is trivial to find messages m and corresponding signatures s^* which will be accepted by the verification algorithm (step 2 of Algorithm 11.9) as follows.

1. Select random $k \in \mathcal{R}$ and random $s^* \in \mathcal{S}$.
2. Compute $\widetilde{m} = V_A(s^*)$.
3. Compute $m = R^{-1}(\widetilde{m})$.

The element s^* is a valid signature for the message m and was created without knowledge of the set of signing transformations $\mathcal{S}_A$.

11.11 Example (*redundancy function*) Suppose $\mathcal{M} = \{m\colon m \in \{0,1\}^n\}$ for some fixed positive integer n and $\mathcal{M}_\mathcal{S} = \{t\colon t \in \{0,1\}^{2n}\}$. Define $R\colon \mathcal{M} \longrightarrow \mathcal{M}_\mathcal{S}$ by $R(m) = m\|m$, where $\|$ denotes concatenation; that is, $\mathcal{M}_R = \{m\|m\colon m \in \mathcal{M}\} \subseteq \mathcal{M}_\mathcal{S}$. For large values of n, the quantity $|\mathcal{M}_R|/|\mathcal{M}_\mathcal{S}| = (\frac{1}{2})^n$ is a negligibly small fraction. This redundancy function is suitable provided that no judicious choice of s^* on the part of an adversary will have a non-negligible probability of yielding $V_A(s^*) \in \mathcal{M}_R$. □

11.12 Remark (*selecting a redundancy function*) Even though the redundancy function R is public knowledge and R^{-1} is easy to compute, selection of R is critical and should not be made independently of the choice of the signing transformations in $\mathcal{S}_A$. Example 11.21 provides a specific example of a redundancy function which compromises the security of the signature scheme. An example of a redundancy function which has been accepted as an international standard is given in §11.3.5. This redundancy function is not appropriate for all digital signature schemes with message recovery, but does apply to the RSA (§11.3.1) and Rabin (§11.3.4) digital signature schemes.

11.13 Remark (*a particular class of message recovery schemes*) §1.8.3 describes a class of digital signature schemes with message recovery which arise from reversible public-key encryption methods. Examples include the RSA (§8.2) and Rabin (§8.3) encryption schemes. The corresponding signature mechanisms are discussed in §11.3.1 and §11.3.4, respectively.

11.14 Note (*signatures with appendix from schemes providing message recovery*) Any digital signature scheme with message recovery can be turned into a digital signature scheme with appendix by simply hashing the message and then signing the hash value. The message is now required as input to the verification algorithm. A schematic for this situation can be derived from Figure 11.3 and is illustrated in Figure 11.4. The redundancy function R is no longer critical to the security of the signature scheme, and can be any $1-1$ function from $\mathcal{M}_h$ to $\mathcal{M}_{\mathcal{S}}$.

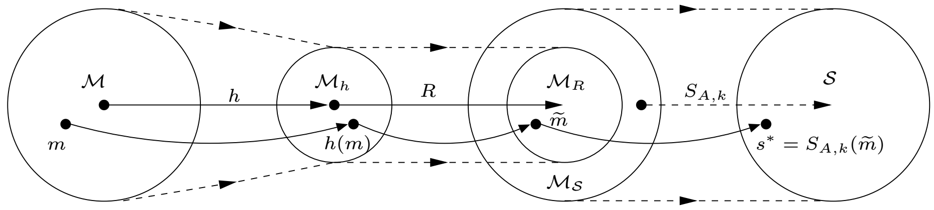


Figure 11.4: *Signature scheme with appendix obtained from one providing message recovery.*

11.2.4 Types of attacks on signature schemes

The goal of an adversary is to *forge* signatures; that is, produce signatures which will be accepted as those of some other entity. The following provides a set of criteria for what it means to break a signature scheme.

1. *total break*. An adversary is either able to compute the private key information of the signer, or finds an efficient signing algorithm functionally equivalent to the valid signing algorithm. (For example, see §11.3.2(i).)
2. *selective forgery*. An adversary is able to create a valid signature for a particular message or class of messages chosen a priori. Creating the signature does not directly involve the legitimate signer. (See Example 11.21.)
3. *existential forgery*. An adversary is able to forge a signature for at least one message. The adversary has little or no control over the message whose signature is obtained, and the legitimate signer may be involved in the deception (for example, see Note 11.66(iii)).

There are two basic attacks against public-key digital signature schemes.

1. *key-only attacks*. In these attacks, an adversary knows only the signer's public key.
2. *message attacks*. Here an adversary is able to examine signatures corresponding either to known or chosen messages. Message attacks can be further subdivided into three classes:
   (a) *known-message attack*. An adversary has signatures for a set of messages which are known to the adversary but not chosen by him.

(b) *chosen-message attack.* An adversary obtains valid signatures from a chosen list of messages before attempting to break the signature scheme. This attack is *non-adaptive* in the sense that messages are chosen before any signatures are seen. Chosen-message attacks against signature schemes are analogous to chosen-ciphertext attacks against public-key encryption schemes (see §1.13.1).

(c) *adaptive chosen-message attack.* An adversary is allowed to use the signer as an oracle; the adversary may request signatures of messages which depend on the signer's public key and he may request signatures of messages which depend on previously obtained signatures or messages.

11.15 Note (*adaptive chosen-message attack*) In principle, an adaptive chosen-message attack is the most difficult type of attack to prevent. It is conceivable that given enough messages and corresponding signatures, an adversary could deduce a pattern and then forge a signature of its choice. While an adaptive chosen-message attack may be infeasible to mount in practice, a well-designed signature scheme should nonetheless be designed to protect against the possibility.

11.16 Note (*security considerations*) The level of security required in a digital signature scheme may vary according to the application. For example, in situations where an adversary is only capable of mounting a key-only attack, it may suffice to design the scheme to prevent the adversary from being successful at selective forgery. In situations where the adversary is capable of a message attack, it is likely necessary to guard against the possibility of existential forgery.

11.17 Note (*hash functions and digital signature processes*) When a hash function h is used in a digital signature scheme (as is often the case), h should be a fixed part of the signature process so that an adversary is unable to take a valid signature, replace h with a weak hash function, and then mount a selective forgery attack.

11.3 RSA and related signature schemes

This section describes the RSA signature scheme and other closely related methods. The security of the schemes presented here relies to a large degree on the intractability of the integer factorization problem (see §3.2). The schemes presented include both digital signatures with message recovery and appendix (see Note 11.14).

11.3.1 The RSA signature scheme

The message space and ciphertext space for the RSA public-key encryption scheme (§8.2) are both $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ where $n = pq$ is the product of two randomly chosen distinct prime numbers. Since the encryption transformation is a bijection, digital signatures can be created by reversing the roles of encryption and decryption. The RSA signature scheme is a deterministic digital signature scheme which provides message recovery (see Definition 11.7). The signing space $\mathcal{M}_\mathcal{S}$ and signature space $\mathcal{S}$ are both $\mathbb{Z}_n$ (see Table 11.1 for notation). A redundancy function $R\colon \mathcal{M} \longrightarrow \mathbb{Z}_n$ is chosen and is public knowledge.

11.18 Algorithm Key generation for the RSA signature scheme

SUMMARY: each entity creates an RSA public key and a corresponding private key.
Each entity A should do the following:

1. Generate two large distinct random primes p and q, each roughly the same size (see §11.3.2).
2. Compute $n = pq$ and $\phi = (p-1)(q-1)$.
3. Select a random integer e, $1 < e < \phi$, such that $\gcd(e, \phi) = 1$.
4. Use the extended Euclidean algorithm (Algorithm 2.107) to compute the unique integer d, $1 < d < \phi$, such that $ed \equiv 1 \pmod{\phi}$.
5. A's public key is (n, e); A's private key is d.

11.19 Algorithm RSA signature generation and verification

SUMMARY: entity A signs a message $m \in \mathcal{M}$. Any entity B can verify A's signature and recover the message m from the signature.

1. *Signature generation.* Entity A should do the following:
   (a) Compute $\widetilde{m} = R(m)$, an integer in the range $[0, n-1]$.
   (b) Compute $s = \widetilde{m}^d \bmod n$.
   (c) A's signature for m is s.
2. *Verification.* To verify A's signature s and recover the message m, B should:
   (a) Obtain A's authentic public key (n, e).
   (b) Compute $\widetilde{m} = s^e \bmod n$.
   (c) Verify that $\widetilde{m} \in \mathcal{M}_R$; if not, reject the signature.
   (d) Recover $m = R^{-1}(\widetilde{m})$.

Proof that signature verification works. If s is a signature for a message m, then $s \equiv \widetilde{m}^d \bmod n$ where $\widetilde{m} = R(m)$. Since $ed \equiv 1 \pmod{\phi}$, $s^e \equiv \widetilde{m}^{ed} \equiv \widetilde{m} \pmod{n}$. Finally, $R^{-1}(\widetilde{m}) = R^{-1}(R(m)) = m$.

11.20 Example (*RSA signature generation with artificially small parameters*)
Key generation. Entity A selects primes $p = 7927$, $q = 6997$, and computes $n = pq = 55465219$ and $\phi = 7926 \times 6996 = 55450296$. A chooses $e = 5$ and solves $ed = 5d \equiv 1 \pmod{55450296}$, yielding $d = 44360237$. A's public key is $(n = 55465219, e = 5)$; A's private key is $d = 44360237$.
Signature generation. For the sake of simplicity (but see §11.3.3(ii)), assume that $\mathcal{M} = \mathbb{Z}_n$ and that the redundancy function $R\colon \mathcal{M} \longrightarrow \mathbb{Z}_n$ is the identity map $R(m) = m$ for all $m \in \mathcal{M}$. To sign a message $m = 31229978$, A computes $\widetilde{m} = R(m) = 31229978$, and computes the signature $s = \widetilde{m}^d \bmod n = 31229978^{44360237} \bmod 55465219 = 30729435$.
Signature verification. B computes $\widetilde{m} = s^e \bmod n = 30729435^5 \bmod 55465219 = 31229978$. Finally, B accepts the signature since $\widetilde{m}$ has the required redundancy (i.e., $\widetilde{m} \in \mathcal{M}_R$), and recovers $m = R^{-1}(\widetilde{m}) = 31229978$. □

11.3.2 Possible attacks on RSA signatures

(i) Integer factorization

If an adversary is able to factor the public modulus n of some entity A, then the adversary can compute ϕ and then, using the extended Euclidean algorithm (Algorithm 2.107), deduce the private key d from ϕ and the public exponent e by solving $ed \equiv 1 \pmod{\phi}$. This constitutes a total break of the system. To guard against this, A must select p and q so that factoring n is a computationally infeasible task. For further information, see §8.2.2(i) and Note 8.8.

(ii) Multiplicative property of RSA

The RSA signature scheme (as well as the encryption method, cf. §8.2.2(v)) has the following multiplicative property, sometimes referred to as the *homomorphic property*. If $s_1 = m_1^d \bmod n$ and $s_2 = m_2^d \bmod n$ are signatures on messages m_1 and m_2, respectively (or more properly on messages with redundancy added), then $s = s_1 s_2 \bmod n$ has the property that $s = (m_1 m_2)^d \bmod n$. If $m = m_1 m_2$ has the proper redundancy (i.e., $m \in \mathcal{M}_R$), then s will be a valid signature for it. Hence, it is important that the redundancy function R is not multiplicative, i.e., for essentially all pairs $a, b \in \mathcal{M}$, $R(a \cdot b) \neq R(a)R(b)$. As Example 11.21 shows, this condition on R is necessary but not sufficient for security.

11.21 Example (*insecure redundancy function*) Let n be an RSA modulus and d the private key. Let $k = \lceil \lg n \rceil$ be the bitlength of n, and let t be a fixed positive integer such that $t < k/2$. Let $w = 2^t$ and let messages be integers m in the interval $[1, n2^{-t} - 1]$. The redundancy function R is taken to be $R(m) = m2^t$ (the least significant t bits of the binary representation of $R(m)$ are 0's). For most choices of n, R will not have the multiplicative property. The general existential forgery attack described in Note 11.10 would have a probability of success of $(\frac{1}{2})^t$. But for this redundancy function, a selective forgery attack (which is more serious) is possible, as is now explained.

Suppose that an adversary wishes to forge a signature on a message m. The adversary knows n but not d. The adversary can mount the following chosen-message attack to obtain the signature on m. Apply the extended Euclidean algorithm (Algorithm 2.107) to n and $\widetilde{m} = R(m) = m2^t = mw$. At each stage of the extended Euclidean algorithm, integers x, y, and r are computed such that $xn + y\widetilde{m} = r$. It can be shown that at some stage there exists a y and r such that $|y| < n/w$ and $r < n/w$, provided $w \leq \sqrt{n}$. If $y > 0$, form integers $m_2 = rw$ and $m_3 = yw$. If $y < 0$, form integers $m_2 = rw$ and $m_3 = -yw$. In either case, m_2 and m_3 have the required redundancy. If signatures $s_2 = m_2^d \bmod n$ and $s_3 = m_3^d \bmod n$ are obtained from the legitimate signer, then the adversary can compute a signature for m as follows:

- if $y > 0$, compute $\frac{s_2}{s_3} = \frac{m_2^d}{m_3^d} = (\frac{rw}{yw})^d = (\frac{r}{y})^d = \widetilde{m}^d \bmod n$;
- if $y < 0$, compute $\frac{s_2}{-s_3} = \frac{m_2^d}{(-m_3)^d} = (\frac{rw}{yw})^d = (\frac{r}{y})^d = \widetilde{m}^d \bmod n$.

In either case, the adversary has a signed message of its choice with the required redundancy. This attack is an example of a chosen-message attack providing selective forgery. It emphasizes the requirement for judicious choice of the redundancy function R. □

11.3.3 RSA signatures in practice

(i) Reblocking problem

One suggested use of RSA is to sign a message and then encrypt the resulting signature. One must be concerned about the relative sizes of the moduli involved when implementing this procedure. Suppose that A wishes to sign and then encrypt a message for B. Suppose that (n_A, e_A) and (n_B, e_B) are A's and B's public keys, respectively. If $n_A > n_B$, then there is a chance that the message cannot be recovered by B, as illustrated in Example 11.22.

11.22 Example (*reblocking problem*) Let $n_A = 8387 \times 7499 = 62894113$, $e_A = 5$, and $d_A = 37726937$; and $n_B = 55465219$, $e_B = 5$, $d_B = 44360237$. Notice that $n_A > n_B$. Suppose $m = 1368797$ is a message with redundancy to be signed under A's private key and then encrypted using B's public key. A computes the following:

1. $s = m^{d_A} \bmod n_A = 1368797^{37726937} \bmod 62894113 = 59847900$.
2. $c = s^{e_B} \bmod n_B = 59847900^5 \bmod 55465219 = 38842235$.

To recover the message and verify the signature, B computes the following:

1. $\widehat{s} = c^{d_B} \bmod n_B = 38842235^{44360237} \bmod 55465219 = 4382681$.
2. $\widehat{m} = \widehat{s}^{e_A} \bmod n_A = 4382681^5 \bmod 62894113 = 54383568$.

Observe that $m \neq \widehat{m}$. The reason for this is that s is larger than the modulus n_B. Here, the probability of this problem occurring is $(n_A - n_B)/n_A \approx 0.12$. □

There are various ways to overcome the reblocking problem.

1. *reordering*. The problem of incorrect decryption will never occur if the operation using the smaller modulus is performed first. That is, if $n_A > n_B$, then entity A should first encrypt the message using B's public key, and then sign the resulting ciphertext using A's private key. The preferred order of operations, however, is always to sign the message first and then encrypt the signature; for if A encrypts first and then signs, an adversary could remove the signature and replace it with its own signature. Even though the adversary will not know what is being signed, there may be situations where this is advantageous to the adversary. Thus, reordering is not a prudent solution.
2. *two moduli per entity*. Have each entity generate separate moduli for encrypting and for signing. If each user's signing modulus is smaller than all of the possible encrypting moduli, then incorrect decryption never occurs. This can be guaranteed by requiring encrypting moduli to be $(t+1)$-bit numbers and signing moduli t-bit numbers.
3. *prescribing the form of the modulus*. In this method, one selects the primes p and q so that the modulus n has a special form: the highest-order bit is a 1 and the k following bits are all 0's. A t-bit modulus n of this form can be found as follows. For n to have the required form, $2^{t-1} \leq n < 2^{t-1} + 2^{t-k-1}$. Select a random $\lceil t/2 \rceil$-bit prime p, and search for a prime q in the interval between $\lceil 2^{t-1}/p \rceil$ and $\lfloor (2^{t-1} + 2^{t-k-1})/p \rfloor$; then $n = pq$ is a modulus of the required type (see Example 11.23). This choice for the modulus n does not completely prevent the incorrect decryption problem, but it can reduce the probability of its occurrence to a negligibly small number. Suppose that n_A is such a modulus and $s = m^{d_A} \bmod n_A$ is a signature on m. Suppose further that s has a 1 in one of the high-order $k+1$ bit positions, other than the highest. Then s, since it is smaller than n_A, must have a 0 in the highest-order bit position and so is necessarily smaller than any other modulus of a similar form. The probability that s does not have any 1's in the high-order $k+1$ bit positions, other than the highest, is less than $(\frac{1}{2})^k$, which is negligibly small if k is selected to be around 100.

11.23 Example (*prescribing the form of the modulus*) Suppose one wants to construct a 12-bit modulus n such that the high order bit is a 1 and the next $k = 3$ bits are 0's. Begin by selecting a 6-bit prime $p = 37$. Select a prime q in the interval between $\lceil 2^{11}/p \rceil = 56$ and $\lfloor (2^{11} + 2^8)/p \rfloor = 62$. The possibilities for q are 59 and 61. If $q = 59$ is selected, then $n = 37 \times 59 = 2183$, having binary representation 100010000111. If $q = 61$ is selected, then $n = 37 \times 61 = 2257$, having binary representation 100011010001. □

(ii) Redundancy functions

In order to avoid an existential forgery attack (see §11.2.4) on the RSA signature scheme, a suitable redundancy function R is required. §11.3.5 describes one such function which has been accepted as an international standard. Judicious choice of a redundancy function is crucial to the security of the system (see §11.3.2(ii)).

(iii) The RSA digital signature scheme with appendix

Note 11.14 describes how any digital signature scheme with message recovery can be modified to give a digital signature scheme with appendix. For example, if MD5 (Algorithm 9.51) is used to hash messages of arbitrary bitlengths to bitstrings of length 128, then Algorithm 11.9 could be used to sign these hash values. If n is a k-bit RSA modulus, then a suitable redundancy function R is required to assign 128-bit integers to k-bit integers. §11.3.6 describes a method for doing this which is often used in practice.

(iv) Performance characteristics of signature generation and verification

Let $n = pq$ be a $2k$-bit RSA modulus where p and q are each k-bit primes. Computing a signature $s = m^d \bmod n$ for a message m requires $O(k^3)$ bit operations (regarding modular multiplication, see §14.3; and for modular exponentiation, §14.6). Since the signer typically knows p and q, she can compute $s_1 = m^d \bmod p$, $s_2 = m^d \bmod q$, and determine s by using the Chinese remainder theorem (see Note 14.75). Although the complexity of this procedure remains $O(k^3)$, it is considerably more efficient in some situations.

Verification of signatures is significantly faster than signing if the public exponent is chosen to be a small number. If this is done, verification requires $O(k^2)$ bit operations. Suggested values for e in practice are 3 or $2^{16} + 1$;[2] of course, p and q must be chosen so that $\gcd(e, (p-1)(q-1)) = 1$.

The RSA signature scheme is thus ideally suited to situations where signature verification is the predominant operation being performed. For example, when a trusted third party creates a public-key certificate for an entity A, this requires only one signature generation, and this signature may be verified many times by various other entities (see §13.4.2).

(v) Parameter selection

As of 1996, a minimum of 768 bits is recommended for RSA signature moduli. A modulus of at least 1024 bits is recommended for signatures which require much longer lifetimes or which are critical to the overall security of a large network. It is prudent to remain aware of progress in integer factorization, and to be prepared to adjust parameters accordingly.

No weaknesses in the RSA signature scheme have been reported when the public exponent e is chosen to be a small number such as 3 or $2^{16}+1$. It is not recommended to restrict the size of the private exponent d in order to improve the efficiency of signature generation (cf. §8.2.2(iv)).

(vi) Bandwidth efficiency

Bandwidth efficiency for digital signatures with message recovery refers to the ratio of the logarithm (base 2) of the size of the signing space $\mathcal{M}_S$ to the logarithm (base 2) of the size of $\mathcal{M}_R$, the image space of the redundancy function. Hence, the bandwidth efficiency is determined by the redundancy R. For RSA (and the Rabin digital signature scheme, §11.3.4), the redundancy function specified by ISO/IEC 9796 (§11.3.5) takes k-bit messages and encodes them to $2k$-bit elements in $\mathcal{M}_S$ from which a $2k$-bit signature is formed. The bandwidth

[2]The choice of $e = 2^{16} + 1$ is based on the fact that e is a prime number, and $\widetilde{m}^e \bmod n$ can be computed with only 16 modular squarings and one modular multiplication (see §14.6.1).

efficiency in this case is $\frac{1}{2}$. For example, with a modulus of size 1024 bits, the maximum size of a message which can be signed is 512 bits.

(vii) System-wide parameters

Each entity must have a distinct RSA modulus; it is insecure to use a system-wide modulus (see §8.2.2(vi)). The public exponent e can be a system-wide parameter, and is in many applications (see Note 8.9(ii)).

(viii) Short vs. long messages

Suppose n is a $2k$-bit RSA modulus which is used in Algorithm 11.19 to sign k-bit messages (i.e., the bandwidth efficiency is $\frac{1}{2}$). Suppose entity A wishes to sign a kt-bit message m. One approach is to partition m into k-bit blocks such that $m = m_1||m_2||\cdots||m_t$ and sign each block individually (but see Note 11.6 regarding why this is not recommended). The bandwidth requirement for this is $2kt$ bits. Alternatively, A could hash message m to a bitstring of length $l \le k$ and sign the hash value. The bandwidth requirement for this signature is $kt+2k$, where the term kt comes from sending the message m. Since $kt+2k \le 2kt$ whenever $t \ge 2$, it follows that the most bandwidth efficient method is to use RSA digital signatures with appendix. For a message of size at most k-bits, RSA with message recovery is preferred.

11.3.4 The Rabin public-key signature scheme

The Rabin public-key signature scheme is similar to RSA (Algorithm 11.19), but it uses an even public exponent e. [3] For the sake of simplicity, it will be assumed that $e = 2$. The signing space $\mathcal{M}_S$ is Q_n (the set of quadratic residues modulo n — see Definition 2.134) and signatures are square roots of these. A redundancy function R from the message space $\mathcal{M}$ to $\mathcal{M}_S$ is selected and is public knowledge.

Algorithm 11.25 describes the basic version of the Rabin public-key signature scheme. A more detailed version (and one more useful in practice) is presented in Algorithm 11.30.

11.24 Algorithm Key generation for the Rabin public-key signature scheme

SUMMARY: each entity creates a public key and corresponding private key.
Each entity A should do the following:

1. Generate two large distinct random primes p and q, each roughly the same size.
2. Compute $n = pq$.
3. A's public key is n; A's private key is (p, q).

11.25 Algorithm Rabin signature generation and verification

SUMMARY: entity A signs a message $m \in \mathcal{M}$. Any entity B can verify A's signature and recover the message m from the signature.

1. *Signature generation.* Entity A should do the following:
   (a) Compute $\widetilde{m} = R(m)$.
   (b) Compute a square root s of $\widetilde{m} \bmod n$ (using Algorithm 3.44).
   (c) A's signature for m is s.

[3] Since p and q are distinct primes in an RSA modulus, $\phi = (p-1)(q-1)$ is even. In RSA, the public exponent e must satisfy $\gcd(e, \phi) = 1$ and so must be odd.

2. *Verification.* To verify A's signature s and recover the message m, B should:
   (a) Obtain A's authentic public key n.
   (b) Compute $\widetilde{m} = s^2 \bmod n$.
   (c) Verify that $\widetilde{m} \in \mathcal{M}_R$; if not, reject the signature.
   (d) Recover $m = R^{-1}(\widetilde{m})$.

11.26 Example (*Rabin signature generation with artificially small parameters*)
Key generation. Entity A selects primes $p = 7$, $q = 11$, and computes $n = 77$. A's public key is $n = 77$; A's private key is $(p = 7, q = 11)$. The signing space is $\mathcal{M}_{\mathcal{S}} = Q_{77} = \{1, 4, 9, 15, 16, 23, 25, 36, 37, 53, 58, 60, 64, 67, 71\}$. For the sake of simplicity (but see Note 11.27), take $\mathcal{M} = \mathcal{M}_{\mathcal{S}}$ and the redundancy function R to be the identity map (i.e., $\widetilde{m} = R(m) = m$).
Signature generation. To sign a message $m = 23$, A computes $R(m) = \widetilde{m} = 23$, and then finds a square root of $\widetilde{m}$ modulo 77. If s denotes such a square root, then $s \equiv \pm 3 \pmod 7$ and $s \equiv \pm 1 \pmod{11}$, implying $s = 10, 32, 45$, or 67. The signature for m is chosen to be $s = 45$. (The signature could be any one of the four square roots.)
Signature verification. B computes $\widetilde{m} = s^2 \bmod 77 = 23$. Since $\widetilde{m} = 23 \in \mathcal{M}_R$, B accepts the signature and recovers $m = R^{-1}(\widetilde{m}) = 23$. □

11.27 Note (*redundancy*)

(i) As with the RSA signature scheme (Example 11.21), an appropriate choice of a redundancy function R is crucial to the security of the Rabin signature scheme. For example, suppose that $\mathcal{M} = \mathcal{M}_{\mathcal{S}} = Q_n$ and $R(m) = m$ for all $m \in \mathcal{M}$. If an adversary selects any integer $s \in \mathbb{Z}_n^*$ and squares it to get $\widetilde{m} = s^2 \bmod n$, then s is a valid signature for $\widetilde{m}$ and is obtained without knowledge of the private key. (Here, the adversary has little control over what the message will be.) In this situation, existential forgery is trivial.

(ii) In most practical applications of digital signature schemes with message recovery, the message space $\mathcal{M}$ consists of bitstrings of some fixed length. For the Rabin scheme, determining a redundancy function R is a challenging task. For example, if a message m is a bitstring, R might assign it to the integer whose binary representation is the message. There is, however, no guarantee that the resulting integer is a quadratic residue modulo n, and so computing a square root might be impossible. One might try to append a small number of random bits to m and apply R again in the hope that $R(m) \in Q_n$. On average, two such attempts would suffice, but a deterministic method would be preferable.

Modified-Rabin signature scheme

To overcome the problem discussed in Note 11.27(ii), a modified version of the basic Rabin signature scheme is provided. The technique presented is similar to that used in the ISO/IEC 9796 digital signature standard (§11.3.5). It provides a deterministic method for associating messages with elements in the signing space $\mathcal{M}_{\mathcal{S}}$, such that computing a square root (or something close to it) is always possible. An understanding of this method will facilitate the reading of §11.3.5.

11.28 Fact Let p and q be distinct primes each congruent to 3 modulo 4, and let $n = pq$.

(i) If $\gcd(x, n) = 1$, then $x^{(p-1)(q-1)/2} \equiv 1 \pmod n$.
(ii) If $x \in Q_n$, then $x^{(n-p-q+5)/8} \bmod n$ is a square root of x modulo n.

(iii) Let x be an integer having Jacobi symbol $\left(\frac{x}{n}\right) = 1$, and let $d = (n - p - q + 5)/8$. Then

$$x^{2d} \bmod n = \begin{cases} x, & \text{if } x \in Q_n, \\ n - x, & \text{if } x \notin Q_n. \end{cases}$$

(iv) If $p \not\equiv q \pmod 8$, then $\left(\frac{2}{n}\right) = -1$. Hence, multiplication of any integer x by 2 or $2^{-1} \bmod n$ reverses the Jacobi symbol of x. (Integers of the form $n = pq$ where $p \equiv q \equiv 3 \pmod 4$ and $p \not\equiv q \pmod 8$ are sometimes called *Williams integers*.)

Algorithm 11.30 is a modified version of the Rabin digital signature scheme. Messages to be signed are from $\mathcal{M}_\mathcal{S} = \{m \in \mathbb{Z}_n \colon m \equiv 6 \pmod{16}\}$. Notation is given in Table 11.2. In practice, the redundancy function R should be more complex to prevent existential forgery (see §11.3.5 for an example).

Symbol	Term	Description
$\mathcal{M}$	message space	$\{m \in \mathbb{Z}_n \colon m \le \lfloor (n-6)/16 \rfloor\}$
$\mathcal{M}_\mathcal{S}$	signing space	$\{m \in \mathbb{Z}_n \colon m \equiv 6 \pmod{16}\}$
$\mathcal{S}$	signature space	$\{s \in \mathbb{Z}_n \colon (s^2 \bmod n) \in \mathcal{M}_\mathcal{S}\}$
R	redundancy function	$R(m) = 16m + 6$ for all $m \in \mathcal{M}$
$\mathcal{M}_R$	image of R	$\{m \in \mathbb{Z}_n \colon m \equiv 6 \pmod{16}\}$

Table 11.2: *Definition of sets and functions for Algorithm 11.30.*

11.29 Algorithm Key generation for the modified-Rabin signature scheme

SUMMARY: each entity creates a public key and corresponding private key.
Each entity A should do the following:

1. Select random primes $p \equiv 3 \pmod 8$, $q \equiv 7 \pmod 8$ and compute $n = pq$.
2. A's public key is n; A's private key is $d = (n - p - q + 5)/8$.

11.30 Algorithm Modified-Rabin public-key signature generation and verification

SUMMARY: entity A signs a message $m \in \mathcal{M}$. Any entity B can verify A's signature and recover the message m from the signature.

1. *Signature generation.* Entity A should do the following:
   (a) Compute $\widetilde{m} = R(m) = 16m + 6$.
   (b) Compute the Jacobi symbol $J = \left(\frac{\widetilde{m}}{n}\right)$ (using Algorithm 2.149).
   (c) If $J = 1$ then compute $s = \widetilde{m}^d \bmod n$.
   (d) If $J = -1$ then compute $s = (\widetilde{m}/2)^d \bmod n$. [4]
   (e) A's signature for m is s.
2. *Verification.* To verify A's signature s and recover the message m, B should:
   (a) Obtain A's authentic public key n.
   (b) Compute $m' = s^2 \bmod n$. (Note the original message m itself is not required.)
   (c) If $m' \equiv 6 \pmod 8$, take $\widetilde{m} = m'$.
   (d) If $m' \equiv 3 \pmod 8$, take $\widetilde{m} = 2m'$.

[4] If $J \ne 1$ or -1 then $J = 0$, implying $\gcd(\widetilde{m}, n) \ne 1$. This leads to a factorization of n. In practice, the probability that this will ever occur is negligible.

(e) If $m' \equiv 7 \pmod 8$, take $\widetilde{m} = n - m'$.
(f) If $m' \equiv 2 \pmod 8$, take $\widetilde{m} = 2(n - m')$.
(g) Verify that $\widetilde{m} \in \mathcal{M}_R$ (see Table 11.2); if not, reject the signature.
(h) Recover $m = R^{-1}(\widetilde{m}) = (\widetilde{m} - 6)/16$.

Proof that signature verification works. The signature generation phase signs either $v = \widetilde{m}$ or $v = \widetilde{m}/2$ depending upon which has Jacobi symbol 1. By Fact 11.28(iv), exactly one of $\widetilde{m}$, $\widetilde{m}/2$ has Jacobi symbol 1. The value v that is signed is such that $v \equiv 3$ or $6 \pmod 8$. By Fact 11.28(iii), $s^2 \bmod n = v$ or $n - v$ depending on whether or not $v \in Q_n$. Since $n \equiv 5 \pmod 8$, these cases can be uniquely distinguished.

11.31 Example (*modified-Rabin signature scheme with artificially small parameters*)
Key generation. A chooses $p = 19$, $q = 31$, and computes $n = pq = 589$ and $d = (n - p - q + 5)/8 = 68$. A's public key is $n = 589$, while A's private key is $d = 68$. The signing space $\mathcal{M}_S$ is given in the following table, along with the Jacobi symbol of each element.

m	6	22	54	70	86	102	118	134	150	166
$\left(\frac{m}{589}\right)$	-1	1	-1	-1	1	1	1	1	-1	1
m	182	198	214	230	246	262	278	294	326	358
$\left(\frac{m}{589}\right)$	-1	1	1	1	1	-1	1	-1	-1	-1
m	374	390	406	422	438	454	470	486	502	518
$\left(\frac{m}{589}\right)$	-1	-1	-1	1	1	1	-1	-1	1	-1
m	534	550	566	582						
$\left(\frac{m}{589}\right)$	-1	1	-1	1						

Signature generation. To sign a message $m = 12$, A computes $\widetilde{m} = R(12) = 198$, $\left(\frac{\widetilde{m}}{n}\right) = \left(\frac{198}{589}\right) = 1$, and $s = 198^{68} \bmod 589 = 102$. A's signature for $m = 12$ is $s = 102$.
Signature verification. B computes $m' = s^2 \bmod n = 102^2 \bmod 589 = 391$. Since $m' \equiv 7 \pmod 8$, B takes $\widetilde{m} = n - m' = 589 - 391 = 198$. Finally, B computes $m = R^{-1}(\widetilde{m}) = (198 - 6)/16 = 12$, and accepts the signature. □

11.32 Note (*security of modified-Rabin signature scheme*)

(i) When using Algorithm 11.30, one should never sign a value v having Jacobi symbol -1, since this leads to a factorization of n. To see this, observe that $y = v^{2d} = s^2$ must have Jacobi symbol 1; but $y^2 \equiv (v^2)^{2d} \equiv v^2 \pmod n$ by Fact 11.28(iii). Therefore, $(v-y)(v+y) \equiv 0 \pmod n$. Since v and y have opposite Jacobi symbols, $v \not\equiv y \pmod n$ and thus $\gcd(v - y, n) = p$ or q.

(ii) Existential forgery is easily accomplished for the modified-Rabin scheme as it was for the original Rabin scheme (see Note 11.27(i)). One only needs to find an s, $1 \le s \le n - 1$, such that either s^2 or $n - s^2$ or $2s^2$ or $2(n - s^2) \bmod n$ is congruent to 6 modulo 16. In any of these cases, s is a valid signature for $m' = s^2 \bmod n$.

11.33 Note (*performance characteristics of the Rabin signature scheme*) Algorithm 11.25 requires a redundancy function from $\mathcal{M}$ to $\mathcal{M}_S = Q_n$ which typically involves computing a Jacobi symbol (Algorithm 2.149). Signature generation then involves computing at least one Jacobi symbol (see Note 11.27) and a square root modulo n. The square root computation is comparable to an exponentiation modulo n (see Algorithm 3.44). Since computing the Jacobi symbol is equivalent to a small number of modular multiplications, Rabin signature generation is not significantly more computationally intensive than an RSA signature generation with the same modulus size. Signature verification is very fast if $e = 2$; it requires only one modular multiplication. Squaring can be performed slightly more efficiently than a general modular multiplication (see Note 14.18). This, too, compares favorably with RSA signature verification even when the RSA public exponent is $e = 3$. The modified Rabin scheme (Algorithm 11.30) specifies the message space and redundancy function. Signature generation requires the evaluation of a Jacobi symbol and one modular exponentiation.

11.34 Note (*bandwidth efficiency*) The Rabin digital signature scheme is similar to the RSA scheme with respect to bandwidth efficiency (see §11.3.3(vi)).

11.3.5 ISO/IEC 9796 formatting

ISO/IEC 9796 was published in 1991 by the International Standards Organization as the first international standard for digital signatures. It specifies a digital signature process which uses a digital signature mechanism providing message recovery.

The main features of ISO/IEC 9796 are: (i) it is based on public-key cryptography; (ii) the particular signature algorithm is not specified but it must map k bits to k bits; (iii) it is used to sign messages of limited length and does not require a cryptographic hash function; (iv) it provides message recovery (see Note 11.14); and (v) it specifies the message padding, where required. Examples of mechanisms suitable for the standard are RSA (Algorithm 11.19) and modified-Rabin (Algorithm 11.30). The specific methods used for padding, redundancy, and truncation in ISO/IEC 9796 prevent various means to forge signatures. Table 11.3 provides notation for this subsection.

Symbol	Meaning
k	the bitlength of the signature.
d	the bitlength of the message m to be signed; it is required that $d \leq 8\lfloor(k+3)/16\rfloor$.
z	the number of bytes in the padded message; $z = \lceil d/8 \rceil$.
r	one more than the number of padding bits; $r = 8z - d + 1$.
t	the least integer such that a string of $2t$ bytes includes at least $k-1$ bits; $t = \lceil (k-1)/16 \rceil$.

Table 11.3: *ISO/IEC 9796 notation.*

11.35 Example (*sample parameter values for ISO/IEC 9796*) The following table lists sample values of parameters in the signing process for a 150-bit message and a 1024-bit signature.

Parameter	k (bits)	d (bits)	z (bytes)	r (bits)	t (bytes)
Value	1024	150	19	3	64

□

(i) Signature process for ISO/IEC 9796

The signature process consists of 5 steps as per Figure 11.5(a).

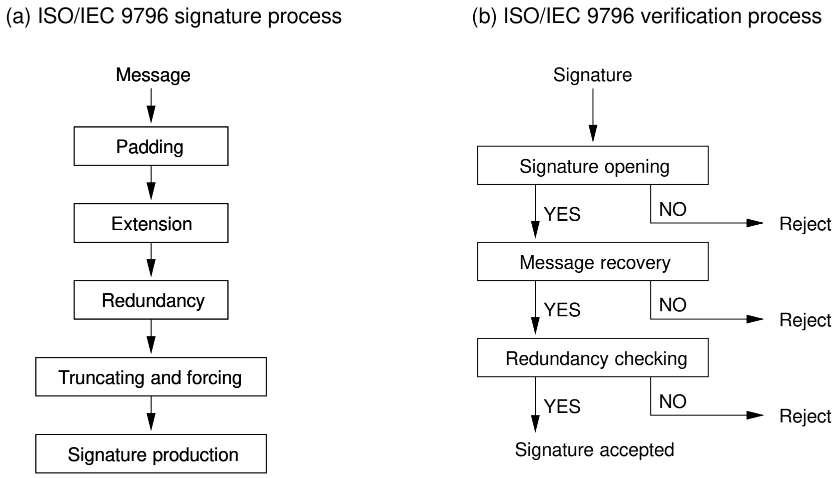


Figure 11.5: *Signature and verification processes for ISO/IEC 9796.*

1. *padding*. If m is the message, form the padded message $MP = 0^{r-1} \| m$ where $1 \leq r \leq 8$, such that the number of bits in MP is a multiple of 8. The number of bytes in MP is z: $MP = m_z \| m_{z-1} \| \cdots \| m_2 \| m_1$ where each m_i is a byte.
2. *message extension*. The extended message, denoted ME, is obtained from MP by repeated concatenation on the left of MP with itself until t bytes are in the string: $ME = ME_t \| ME_{t-1} \| \cdots \| ME_2 \| ME_1$ (each ME_i is a byte). If t is not a multiple of z, then the last bytes to be concatenated are a partial set of bytes from MP, where these bytes are consecutive bytes of MP from the right. More precisely, $ME_{i+1} = m_{(i \bmod z)+1}$ for $0 \leq i \leq t-1$.
3. *message redundancy*. Redundancy is added to ME to get the byte string $MR = MR_{2t} \| MR_{2t-1} \| \cdots \| MR_2 \| MR_1$ as follows. MR is obtained by interleaving the t bytes of ME with t redundant bytes and then adjusting byte MR_{2z} of the resulting string. More precisely, $MR_{2i-1} = ME_i$ and $MR_{2i} = S(ME_i)$ for $1 \leq i \leq t$, where $S(u)$ is called the *shadow function* of the byte u, and is defined as follows. If $u = u_2 \| u_1$ where u_1 and u_2 are nibbles (strings of bitlength 4), then $S(u) = \pi(u_2) \| \pi(u_1)$ where π is the permutation

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & A & B & C & D & E & F \\ E & 3 & 5 & 8 & 9 & 4 & 2 & F & 0 & D & B & 6 & 7 & A & C & 1 \end{pmatrix}.$$

   (For brevity, π is written with nibbles represented by hexadecimal characters.) Finally, MR is obtained by replacing MR_{2z} with $r \oplus MR_{2z}$.[5]
4. *truncation and forcing*. Form the k-bit intermediate integer IR from MR as follows:
   (a) to the least significant $k-1$ bits of MR, append on the left a single bit 1;
   (b) modify the least significant byte $u_2 \| u_1$ of the result, replacing it by $u_1 \| 0110$. (This is done to ensure that $IR \equiv 6 \pmod{16}$.)

[5]The purpose of MR_{2z} is to permit the verifier of a signature to recover the length d of the message. Since $d = 8z - r + 1$, it suffices to know z and r. These values can be deduced from MR.

5. *signature production*. A signature mechanism is used which maps k-bit integers to k-bit integers (and allows message recovery). IR is signed using this mechanism; let s denote the resulting signature.

11.36 Note (*RSA, Rabin*) ISO/IEC 9796 was intended for use with the RSA (Algorithm 11.19)[6] and Rabin (Algorithm 11.25)[7] digital signature mechanisms. For these particular schemes, signature production is stated more explicitly. Let e be the public exponent for the RSA or Rabin algorithms, n the modulus, and d the private exponent. First form the representative element RR which is: (i) IR if e is odd, or if e is even and the Jacobi symbol of IR (treated as an integer) with respect to the modulus n is 1; (ii) $IR/2$ if e is even and the Jacobi symbol of IR with respect to n is -1. The signature for m is $s = (RR)^d \bmod n$. ISO/IEC 9796 specifies that the signature s should be the lesser of $(RR)^d \bmod n$ and $n-((RR)^d \bmod n)$.

(ii) Verification process for ISO/IEC 9796

The verification process for an ISO/IEC 9796 digital signature can be separated into three stages, as per Figure 11.5(b).

1. *signature opening*. Let s be the signature. Then the following steps are performed.
   (a) Apply the public verification transformation to s to recover an integer IR'.
   (b) Reject the signature if IR' is not a string of k bits with the most significant bit being a 1, or if the least significant nibble does not have value 0110.
2. *message recovery*. A string MR' of $2t$ bytes is constructed from IR' by performing the following steps.
   (a) Let X be the least significant $k-1$ bits of IR'.
   (b) If $u_4\|u_3\|u_2\|0110$ are the four least significant nibbles of X, replace the least significant byte of X by $\pi^{-1}(u_4)\|u_2$.
   (c) MR' is obtained by padding X with between 0 and 15 zero bits so that the resulting string has $2t$ bytes.

   The values z and r are computed as follows.
   (a) From the $2t$ bytes of MR', compute the t sums $MR'_{2i} \oplus S(MR'_{2i-1})$, $1 \le i \le t$. If all sums are 0, reject the signature.
   (b) Let z be the smallest value of i for which $MR'_{2i} \oplus S(MR'_{2i-1}) \neq 0$.
   (c) Let r be the least significant nibble of the sum found in step (b). Reject the signature if the hexadecimal value of r is not between 1 and 8.

   From MR', the z-byte string MP' is constructed as follows.
   (a) $MP'_i = MR'_{2i-1}$ for $1 \le i \le z$.
   (b) Reject the signature if the $r-1$ most significant bits of MP' are not all 0's.
   (c) Let M' be the $8z-r+1$ least significant bits of MP'.
3. *redundancy checking*. The signature s is verified as follows.
   (a) From M' construct a string MR'' by applying the message padding, message extension, and message redundancy steps of the signing process.
   (b) Accept the signature if and only if the $k-1$ least significant bits of MR'' are equal to the $k-1$ least significant bits of MR'.

[6]Since steps 1 through 4 of the signature process describe the redundancy function R, $\widetilde{m}$ in step 1a of Algorithm 11.19 is taken to be IR.
[7]$\widetilde{m}$ is taken to be IR in step 1 of Algorithm 11.25.

11.3.6 PKCS #1 formatting

Public-key cryptography standards (PKCS) are a suite of specifications which include techniques for RSA encryption and signatures (see §15.3.6). This subsection describes the digital signature process specified in PKCS #1 ("RSA Encryption Standard").

The digital signature mechanism in PKCS #1 does not use the message recovery feature of the RSA signature scheme. It requires a hashing function (either MD2, or MD5 — see Algorithm 9.51) and, therefore, is a digital signature scheme with appendix. Table 11.4 lists notation used in this subsection. Capital letters refer to octet strings. If X is an octet string, then X_i is octet i counting from the left.

Symbol	Meaning	Symbol	Meaning
k	the length of n in octets ($k \geq 11$)	EB	encryption block
n	the modulus, $2^{8(k-1)} \leq n < 2^{8k}$	ED	encrypted data
p, q	the prime factors of n	octet	a bitstring of length 8
e	the public exponent	ab	hexadecimal octet value
d	the private exponent	BT	block type
M	message	PS	padding string
MD	message digest	S	signature
MD′	comparative message digest	$\|X\|$	length of X in octets

Table 11.4: *PKCS #1 notation.*

(i) PKCS #1 data formatting

The data is an octet string D, where $\|\mathrm{D}\| \leq k-11$. BT is a single octet whose hexadecimal representation is either 00 or 01. PS is an octet string with $\|\mathrm{PS}\| = k-3-\|\mathrm{D}\|$. If BT $=$ 00, then all octets in PS are 00; if BT $=$ 01, then all octets in PS are ff. The formatted data block (called the *encryption block*) is EB $=$ 00$\|$BT$\|$PS$\|$00$\|$D.

11.37 Note (*data formatting rationale*)

(i) The leading 00 block ensures that the octet string EB, when interpreted as an integer, is less than the modulus n.

(ii) If the block type is BT $=$ 00, then either D must begin with a non-zero octet or its length must be known, in order to permit unambiguous parsing of EB.

(iii) If BT $=$ 01, then unambiguous parsing is always possible.

(iv) For the reason given in (iii), and to thwart certain potential attacks on the signature mechanism, BT $=$ 01 is recommended.

11.38 Example (*PKCS #1 data formatting for particular values*) Suppose that n is a 1024-bit modulus (so $k = 128$). If $\|\mathrm{D}\| = 20$ octets, then $\|\mathrm{PS}\| = 105$ octets, and $\|\mathrm{EB}\| = 128$ octets. □

(ii) Signature process for PKCS #1

The signature process involves the steps as per Figure 11.6(a).

The input to the signature process is the message M, and the signer's private exponent d and modulus n.

1. *message hashing.* Hash the message M using the selected message-digest algorithm to get the octet string MD.

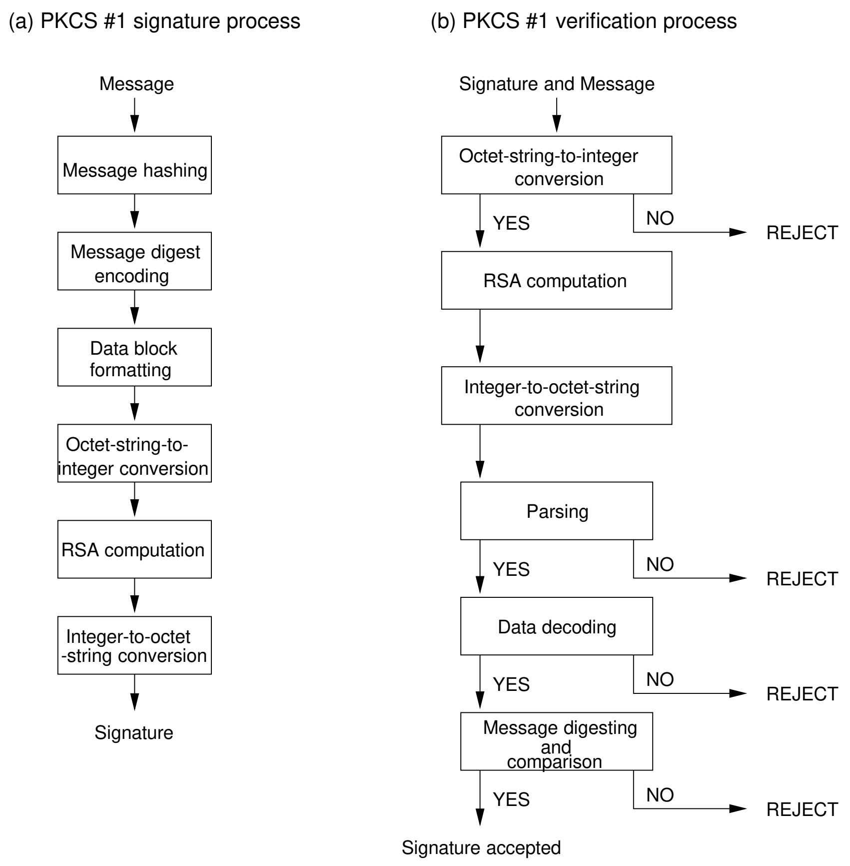


Figure 11.6: *Signature and verification processes for PKCS #1.*

2. *message digest encoding*. MD and the hash algorithm identifier are combined into an ASN.1 (*abstract syntax notation*) value and then BER-encoded (*basic encoding rules*) to give an octet data string D.
3. *data block formatting*. With data string input D, use the data formatting from §11.3.6(i) to form octet string EB.
4. *octet-string-to-integer conversion*. Let the octets of EB be $\mathrm{EB}_1\|\mathrm{EB}_2\|\cdots\|\mathrm{EB}_k$. Define $\widetilde{\mathrm{EB}}_i$ to be the integer whose binary representation is the octet EB_i (least significant bit is on the right). The integer representing EB is $m=\sum_{i=1}^{k}2^{8(k-i)}\widetilde{\mathrm{EB}}_i$. [8]
5. *RSA computation*. Compute $s=m^d \bmod n$.
6. *integer-to-octet-string conversion*. Convert s to an octet string $\mathrm{ED}=\mathrm{ED}_1\|\mathrm{ED}_2\|\cdots\|\mathrm{ED}_k$, where the octets ED_i satisfy $s=\sum_{i=1}^{k}2^{8(k-i)}\widetilde{\mathrm{ED}}_i$. The signature is S = ED.

(iii) Verification process for PKCS #1

The verification process involves the steps as per Figure 11.6(b). The input to the verification process is the message M, the signature S, the public exponent e, and modulus n.

1. *octet-string-to-integer conversion*.
   (a) Reject S if the bitlength of S is not a multiple of 8.

[8] Since $\mathrm{EB}_1=00$ and $n\geq 2^{8(k-1)}$, then $0\leq m<n$.

   (b) Convert S to an integer s as in step 4 of the signature process.
   (c) Reject the signature if $s > n$.
2. *RSA computation*. Compute $m = s^e \bmod n$.
3. *integer-to-octet-string conversion.* Convert m to an octet string EB of length k octets as in step 6 of the signature process.
4. *parsing*. Parse EB into a block type BT, a padding string PS, and the data D.
   (a) Reject the signature if EB cannot be parsed unambiguously.
   (b) Reject the signature if BT is not one of 00 or 01.
   (c) Reject the signature if PS consists of < 8 octets or is inconsistent with BT.
5. *data decoding.*
   (a) BER-decode D to get a message digest MD and a hash algorithm identifier.
   (b) Reject the signature if the hashing algorithm identifier does not identify one of MD2 or MD5.
6. *message digesting and comparison*.
   (a) Hash the message M with the selected message-digest algorithm to get MD$'$.
   (b) Accept the signature S on M if and only if MD$'$ = MD.

11.4 Fiat-Shamir signature schemes

As described in Note 10.30, any identification scheme involving a witness-challenge response sequence can be converted to a signature scheme by replacing the random challenge of the verifier with a one-way hash function. This section describes two signature mechanisms which arise in this way. The basis for this methodology is the Fiat-Shamir identification protocol (Protocol 10.24).

11.4.1 Feige-Fiat-Shamir signature scheme

The Feige-Fiat-Shamir signature scheme is a modification of an earlier signature scheme of Fiat and Shamir, and requires a one-way hash function $h\colon \{0,1\}^* \longrightarrow \{0,1\}^k$ for some fixed positive integer k. Here $\{0,1\}^k$ denotes the set of bitstrings of bitlength k, and $\{0,1\}^*$ denotes the set of all bitstrings (of arbitrary bitlengths). The method provides a digital signature with appendix, and is a randomized mechanism.

11.39 Algorithm Key generation for the Feige-Fiat-Shamir signature scheme

SUMMARY: each entity creates a public key and corresponding private key.
Each entity A should do the following:

1. Generate random distinct secret primes p, q and form $n = pq$.
2. Select a positive integer k and distinct random integers $s_1, s_2, \ldots, s_k \in \mathbb{Z}_n^*$.
3. Compute $v_j = s_j^{-2} \bmod n$, $1 \le j \le k$.
4. A's public key is the k-tuple $(v_1, v_2, \ldots, v_k)$ and the modulus n; A's private key is the k-tuple $(s_1, s_2, \ldots, s_k)$.

11.40 Algorithm Feige-Fiat-Shamir signature generation and verification

SUMMARY: entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key.

1. *Signature generation.* Entity A should do the following:
   (a) Select a random integer r, $1 \le r \le n-1$.
   (b) Compute $u = r^2 \bmod n$.
   (c) Compute $e = (e_1, e_2, \ldots, e_k) = h(m\|u)$; each $e_i \in \{0,1\}$.
   (d) Compute $s = r \cdot \prod_{j=1}^{k} s_j^{e_j} \bmod n$.
   (e) A's signature for m is (e, s).
2. *Verification.* To verify A's signature (e, s) on m, B should do the following:
   (a) Obtain A's authentic public key $(v_1, v_2, \ldots, v_k)$ and n.
   (b) Compute $w = s^2 \cdot \prod_{j=1}^{k} v_j^{e_j} \bmod n$.
   (c) Compute $e' = h(m\|w)$.
   (d) Accept the signature if and only if $e = e'$.

Proof that signature verification works.

$$w \equiv s^2 \cdot \prod_{j=1}^{k} v_j^{e_j} \equiv r^2 \cdot \prod_{j=1}^{k} s_j^{2e_j} \prod_{j=1}^{k} v_j^{e_j} \equiv r^2 \cdot \prod_{j=1}^{k} (s_j^2 v_j)^{e_j} \equiv r^2 \equiv u \pmod{n}.$$

Hence, $w = u$ and therefore $e = e'$.

11.41 Example (*Feige-Fiat-Shamir signature generation with artificially small parameters*)
Key generation. Entity A generates primes $p = 3571$, $q = 4523$, and computes $n = pq = 16151633$. The following table displays the selection of s_j (A's private key) and integers v_j (A's public key) along with intermediate values s_j^{-1}.

j	1	2	3	4	5
s_j	42	73	85	101	150
$s_j^{-1} \bmod n$	4999315	885021	6270634	13113207	11090788
$v_j = s_j^{-2} \bmod n$	503594	4879739	7104483	1409171	6965302

Signature generation. Suppose $h\colon \{0,1\}^* \longrightarrow \{0,1\}^5$ is a hash function. A selects a random integer $r = 23181$ and computes $u = r^2 \bmod n = 4354872$. To sign message m, A evaluates $e = h(m\|u) = 10110$ (the hash value has been contrived for this example). A forms $s = rs_1s_3s_4 \bmod n = (23181)(42)(85)(101) \bmod n = 7978909$; the signature for m is $(e = 10110, s = 7978909)$.
Signature verification. B computes $s^2 \bmod n = 2926875$ and $v_1v_3v_4 \bmod n = (503594)(7104483)(1409171) \bmod n = 15668174$. B then computes $w = s^2v_1v_3v_4 \bmod n = 4354872$. Since $w = u$, it follows that $e' = h(m\|w) = h(m\|u) = e$ and, hence, B accepts the signature. □

11.42 Note (*security of Feige-Fiat-Shamir signature scheme*)

(i) Unlike the RSA signature scheme (Algorithm 11.19), all entities may use the same modulus n (cf. §8.2.2(vi)). In this scenario, a trusted third party (TTP) would need to generate the primes p and q and also public and private keys for each entity.

(ii) The security of the Feige-Fiat-Shamir scheme is based on the intractability of computing square roots modulo n (see §3.5.2). It has been proven to be secure against an adaptive chosen-message attack, provided that factoring is intractable, h is a random function, and the s_i's are distinct.

11.43 Note (*parameter selection and key storage requirements*) If n is a t-bit integer, the private key constructed in Algorithm 11.39 is kt bits in size. This may be reduced by selecting the random values s_j, $1 \le j \le k$, as numbers of bitlength $t' < t$; t', however, should not be chosen so small that guessing the s_j is feasible. The public key is $(k+1)t$ bits in size. For example, if $t = 768$ and $k = 128$, then the private key requires 98304 bits and the public key requires 99072 bits.

11.44 Note (*identity-based Feige-Fiat-Shamir signatures*) Suppose a TTP constructs primes p and q and modulus n; the modulus is common to all entities in the system. Algorithm 11.39 can be modified so that the scheme is identity-based. Entity A's bitstring I_A contains information which identifies A. The TTP computes $v_j = f(I_A \| j)$, $1 \le j \le k$, where f is a one-way hash function from $\{0,1\}^*$ to Q_n and j is represented in binary, and computes a square root s_j of v_j^{-1} modulo n, $1 \le j \le k$. A's public key is simply the identity information I_A, while A's private key (transported securely and secretly by the TTP to A) is the k-tuple $(s_1, s_2, \ldots, s_k)$. The functions h, f, and the modulus n are system-wide quantities.

This procedure has the advantage that the public key generated in Algorithm 11.39 might be generated from a smaller quantity I_A, potentially reducing the storage and transmission cost. It has the disadvantages that the private keys of entities are known to the TTP, and the modulus n is system-wide, making it a more attractive target.

11.45 Note (*small prime variation of Feige-Fiat-Shamir signatures*) This improvement aims to reduce the size of the public key and increase the efficiency of signature verification. Unlike the modification described in Note 11.44, each entity A generates its own modulus n_A and a set of k small primes $v_1, v_2, \ldots, v_k \in Q_n$ (each prime will require around 2 bytes to represent). Entity A selects one of the square roots s_j of v_j^{-1} modulo n for each j, $1 \le j \le k$; these form the private key. The public key consists of n_A and the values $v_1, v_2, \ldots, v_k$. Verification of signatures proceeds more efficiently since computations are done with much smaller numbers.

11.46 Note (*performance characteristics of Feige-Fiat-Shamir signatures*) With the RSA scheme and a modulus of length $t = 768$, signature generation using naive techniques requires, on average, 1152 modular multiplications (more precisely, 768 squarings and 384 multiplications). Signature generation for the Feige-Fiat-Shamir scheme (Algorithm 11.40) requires, on average, $k/2$ modular multiplications. To sign a message with this scheme, a modulus of length $t = 768$ and $k = 128$ requires, on average, 64 modular multiplications, or less than 6% of the work required by a naive implementation of RSA. Signature verification requires only one modular multiplication for RSA if the public exponent is $e = 3$, and 64 modular multiplications, on average, for Feige-Fiat-Shamir. For applications where signature generation must be performed quickly and key space storage is not limited, the Feige-Fiat-Shamir scheme (or DSA-like schemes — see §11.5) may be preferable to RSA.

11.4.2 GQ signature scheme

The Guillou-Quisquater (GQ) identification protocol (§10.4.3) can be turned into a digital signature mechanism (Algorithm 11.48) if the challenge is replaced with a one-way hash function. Let $h\colon \{0,1\}^* \longrightarrow \mathbb{Z}_n$ be a hash function where n is a positive integer.

11.47 Algorithm Key generation for the GQ signature scheme

SUMMARY: each entity creates a public key (n, e, J_A) and corresponding private key a. Entity A should do the following:

1. Select random distinct secret primes p, q and form $n = pq$.
2. Select an integer $e \in \{1, 2, \ldots, n-1\}$ such that $\gcd(e, (p-1)(q-1)) = 1$. (See Note 11.50 for guidance on selecting e.)
3. Select an integer J_A, $1 < J_A < n$, which serves as an identifier for A and such that $\gcd(J_A, n) = 1$. (The binary representation of J_A could be used to convey information about A such as name, address, driver's license number, etc.)
4. Determine an integer $a \in \mathbb{Z}_n$ such that $J_A a^e \equiv 1 \pmod n$ as follows:
   - 4.1 Compute $J_A^{-1} \bmod n$.
   - 4.2 Compute $d_1 = e^{-1} \bmod (p-1)$ and $d_2 = e^{-1} \bmod (q-1)$.
   - 4.3 Compute $a_1 = (J_A^{-1})^{d_1} \bmod p$ and $a_2 = (J_A^{-1})^{d_2} \bmod q$.
   - 4.4 Find a solution a to the simultaneous congruences $a \equiv a_1 \pmod p$, $a \equiv a_2 \pmod q$.
5. A's public key is (n, e, J_A); A's private key is a.

11.48 Algorithm GQ signature generation and verification

SUMMARY: entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key.

1. *Signature generation.* Entity A should do the following:
   - (a) Select a random integer k and compute $r = k^e \bmod n$.
   - (b) Compute $l = h(m \| r)$.
   - (c) Compute $s = ka^l \bmod n$.
   - (d) A's signature for m is the pair (s, l).
2. *Verification.* To verify A's signature (s, l) on m, B should do the following:
   - (a) Obtain A's authentic public key (n, e, J_A).
   - (b) Compute $u = s^e {J_A}^l \bmod n$ and $l' = h(m \| u)$.
   - (c) Accept the signature if and only if $l = l'$.

Proof that signature verification works. Note that $u \equiv s^e {J_A}^l \equiv (ka^l)^e {J_A}^l \equiv k^e (a^e J_A)^l \equiv k^e \equiv r \pmod n$. Hence, $u = r$ and therefore $l = l'$.

11.49 Example (*GQ signature generation with artificially small parameters*)
Key generation. Entity A chooses primes $p = 20849$, $q = 27457$, and computes $n = pq = 572450993$. A selects an integer $e = 47$, an identifier $J_A = 1091522$, and solves the congruence $J_A a^e \equiv 1 \pmod n$ to get $a = 214611724$. A's public key is ($n = 572450993$, $e = 47$, $J_A = 1091522$), while A's private key is $a = 214611724$.
Signature generation. To sign the message $m = 1101110001$, A selects a random integer

$k = 42134$ and computes $r = k^e \bmod n = 297543350$. A then computes $l = h(m\|r) = 2713833$ (the hash value has been contrived for this example) and $s = ka^l \bmod n = (42134)214611724^{2713833} \bmod n = 252000854$. A's signature for m is the pair ($s = 252000854, l = 2713833$).

Signature verification. B computes $s^e \bmod n = 252000854^{47} \bmod n = 398641962$, ${J_A}^l \bmod n = 1091522^{2713833} \bmod n = 110523867$, and finally $u = s^e {J_A}^l \bmod n = 297543350$. Since $u = r$, $l' = h(m\|u) = h(m\|r) = l$, and so B accepts the signature. □

11.50 Note (*security of GQ signature scheme*) In Algorithm 11.47, e must be sufficiently large to exclude the possibility of forgery based on the birthday paradox (see §2.1.5). The potential attack proceeds along the following lines. The adversary selects a message m and computes $l = h(m\|{J_A}^t)$ for sufficiently many values of t until $l \equiv t \pmod e$; this is expected to occur within $O(\sqrt{e})$ trials. Having determined such a pair (l, t), the adversary determines an integer x such that $t = xe + l$ and computes $s = {J_A}^x \bmod n$. Observe that $s^e {J_A}^l \equiv ({J_A}^x)^e {J_A}^l \equiv {J_A}^{xe+l} \equiv {J_A}^t \pmod n$, and, hence, $h(m\|{J_A}^t) = l$. Thus, (s, l) is a valid (forged) signature for message m.

11.51 Note (*parameter selection*) Current methods (as of 1996) for integer factorization suggest that a modulus n of size at least 768 bits is prudent. Note 11.50 suggests that e should be at least 128 bits in size. Typical values for the outputs of secure hash functions are 128 or 160 bits. With a 768-bit modulus and a 128-bit e, the public key for the GQ scheme is $896 + u$ bits in size, where u is the number of bits needed to represent J_A. The private key a is 768 bits in size.

11.52 Note (*performance characteristics of GQ signatures*) Signature generation for GQ (Algorithm 11.48) requires two modular exponentiations and one modular multiplication. Using a 768-bit modulus n, a 128-bit value e, and a hash function with a 128-bit output l, signature generation (using naive techniques for exponentiation) requires on average 384 modular multiplications (128 squarings and 64 multiplications for each of e and l). Signature verification requires a similar amount of work. Compare this with RSA (naively 1152 modular multiplications) and Feige-Fiat-Shamir (64 modular multiplications) for signature generation (see Note 11.46). GQ is computationally more intensive than Feige-Fiat-Shamir but requires significantly smaller key storage space (see Note 11.51).

11.53 Note (*message recovery variant of GQ signatures*) Algorithm 11.48 can be modified as follows to provide message recovery. Let the signing space be $\mathcal{M}_{\mathcal{S}} = \mathbb{Z}_n$, and let $m \in \mathcal{M}_{\mathcal{S}}$. In signature generation, select a random k such that $\gcd(k, n) = 1$ and compute $r = k^e \bmod n$ and $l = mr \bmod n$. The signature is $s = ka^l \bmod n$. Verification gives $s^e {J_A}^l \equiv k^e a^{el} {J_A}^l \equiv k^e \equiv r \pmod n$. Message m is recovered from $lr^{-1} \bmod n$. As for all digital signature schemes with message recovery, a suitable redundancy function R is required to guard against existential forgery.

11.5 The DSA and related signature schemes

This section presents the Digital Signature Algorithm (DSA) and several related signature schemes. Most of these are presented over $\mathbb{Z}_p^*$ for some large prime p, but all of these mechanisms can be generalized to any finite cyclic group; this is illustrated explicitly for the El-

Gamal signature scheme in §11.5.2. All of the methods discussed in this section are randomized digital signature schemes (see Definition 11.2). All give digital signatures with appendix and can be modified to provide digital signatures with message recovery (see Note 11.14). A necessary condition for the security of all of the signature schemes described in this section is that computing logarithms in $\mathbb{Z}_p^*$ be computationally infeasible. This condition, however, is not necessarily sufficient for the security of these schemes; analogously, it remains unproven that RSA signatures are secure even if factoring integers is hard.

11.5.1 The Digital Signature Algorithm (DSA)

In August of 1991, the U.S. National Institute of Standards and Technology (NIST) proposed a digital signature algorithm (DSA). The DSA has become a U.S. Federal Information Processing Standard (FIPS 186) called the *Digital Signature Standard* (DSS), and is the first digital signature scheme recognized by any government. The algorithm is a variant of the ElGamal scheme (§11.5.2), and is a digital signature scheme with appendix.

The signature mechanism requires a hash function $h\colon \{0,1\}^* \longrightarrow \mathbb{Z}_q$ for some integer q. The DSS explicitly requires use of the Secure Hash Algorithm (SHA-1), given by Algorithm 9.53.

11.54 Algorithm Key generation for the DSA

SUMMARY: each entity creates a public key and corresponding private key.
Each entity A should do the following:

1. Select a prime number q such that $2^{159} < q < 2^{160}$.
2. Choose t so that $0 \leq t \leq 8$, and select a prime number p where $2^{511+64t} < p < 2^{512+64t}$, with the property that q divides $(p-1)$.
3. (Select a generator α of the unique cyclic group of order q in $\mathbb{Z}_p^*$.)
   - 3.1 Select an element $g \in \mathbb{Z}_p^*$ and compute $\alpha = g^{(p-1)/q} \bmod p$.
   - 3.2 If $\alpha = 1$ then go to step 3.1.
4. Select a random integer a such that $1 \leq a \leq q-1$.
5. Compute $y = \alpha^a \bmod p$.
6. A's public key is (p, q, α, y); A's private key is a.

11.55 Note (*generation of DSA primes p and q*) In Algorithm 11.54 one must select the prime q first and then try to find a prime p such that q divides $(p-1)$. The algorithm recommended by the DSS for accomplishing this is Algorithm 4.56.

11.56 Algorithm DSA signature generation and verification

SUMMARY: entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key.

1. *Signature generation.* Entity A should do the following:
   - (a) Select a random secret integer k, $0 < k < q$.
   - (b) Compute $r = (\alpha^k \bmod p) \bmod q$ (e.g., using Algorithm 2.143).
   - (c) Compute $k^{-1} \bmod q$ (e.g., using Algorithm 2.142).
   - (d) Compute $s = k^{-1}\{h(m) + ar\} \bmod q$.
   - (e) A's signature for m is the pair (r, s).

2. *Verification.* To verify A's signature (r, s) on m, B should do the following:
   (a) Obtain A's authentic public key (p, q, α, y).
   (b) Verify that $0 < r < q$ and $0 < s < q$; if not, then reject the signature.
   (c) Compute $w = s^{-1} \bmod q$ and $h(m)$.
   (d) Compute $u_1 = w \cdot h(m) \bmod q$ and $u_2 = rw \bmod q$.
   (e) Compute $v = (\alpha^{u_1} y^{u_2} \bmod p) \bmod q$.
   (f) Accept the signature if and only if $v = r$.

Proof that signature verification works. If (r, s) is a legitimate signature of entity A on message m, then $h(m) \equiv -ar + ks \pmod q$ must hold. Multiplying both sides of this congruence by w and rearranging gives $w \cdot h(m) + arw \equiv k \pmod q$. But this is simply $u_1 + au_2 \equiv k \pmod q$. Raising α to both sides of this equation yields $(\alpha^{u_1} y^{u_2} \bmod p) \bmod q = (\alpha^k \bmod p) \bmod q$. Hence, $v = r$, as required.

11.57 Example (*DSA signature generation with artificially small parameters*)
Key generation. A selects primes $p = 124540019$ and $q = 17389$ such that q divides $(p - 1)$; here, $(p - 1)/q = 7162$. A selects a random element $g = 110217528 \in \mathbb{Z}_p^*$ and computes $\alpha = g^{7162} \bmod p = 10083255$. Since $\alpha \neq 1$, α is a generator for the unique cyclic subgroup of order q in $\mathbb{Z}_p^*$. A next selects a random integer $a = 12496$ satisfying $1 \leq a \leq q - 1$, and computes $y = \alpha^a \bmod p = 10083255^{12496} \bmod 124540019 = 119946265$. A's public key is $(p = 124540019, q = 17389, \alpha = 10083255, y = 119946265)$, while A's private key is $a = 12496$.
Signature generation. To sign m, A selects a random integer $k = 9557$, and computes $r = (\alpha^k \bmod p) \bmod q = (10083255^{9557} \bmod 124540019) \bmod 17389 = 27039929 \bmod 17389 = 34$. A then computes $k^{-1} \bmod q = 7631$, $h(m) = 5246$ (the hash value has been contrived for this example), and finally $s = (7631)\{5246 + (12496)(34)\} \bmod q = 13049$. The signature for m is the pair $(r = 34, s = 13049)$.
Signature verification. B computes $w = s^{-1} \bmod q = 1799$, $u_1 = w \cdot h(m) \bmod q = (5246)(1799) \bmod 17389 = 12716$, and $u_2 = rw \bmod q = (34)(1799) \bmod 17389 = 8999$. B then computes $v = (\alpha^{u_1} y^{u_2} \bmod p) \bmod q = (10083255^{12716} \cdot 119946265^{8999} \bmod 124540019) \bmod 17389 = 27039929 \bmod 17389 = 34$. Since $v = r$, B accepts the signature. □

11.58 Note (*security of DSA*) The security of the DSA relies on two distinct but related discrete logarithm problems. One is the logarithm problem in $\mathbb{Z}_p^*$ where the powerful index-calculus methods apply; the other is the logarithm problem in the cyclic subgroup of order q, where the best current methods run in "square-root" time. For further discussion, see §3.6.6. Since the DSA is a special case of ElGamal signatures (§11.5.2) with respect to the equation for s, security considerations for the latter are pertinent here (see Note 11.66).

11.59 Note (*recommended parameter sizes*) The size of q is fixed by Algorithm 11.54 (as per FIPS 186) at 160 bits, while the size of p can be any multiple of 64 between 512 and 1024 bits inclusive. A 512-bit prime p provides marginal security against a concerted attack. As of 1996, a modulus of at least 768 bits is recommended. FIPS 186 does not permit primes p larger than 1024 bits.

11.60 Note (*performance characteristics of the DSA*) For concreteness, suppose p is a 768-bit integer. Signature generation requires one modular exponentiation, taking on average (using naive techniques for exponentiation) 240 modular multiplications, one modular inverse with a 160-bit modulus, two 160-bit modular multiplications, and one addition. The 160-bit operations are relatively minor compared to the exponentiation. The DSA has the advantage that the exponentiation can be precomputed and need not be done at the time of signature generation. By comparison, no precomputation is possible with the RSA signature scheme. The major portion of the work for signature verification is two exponentiations modulo p, each to 160-bit exponents. On average, these each require 240 modular multiplications or 480 in total. Some savings can be realized by doing the two exponentiations simultaneously (cf. Note 14.91); the cost, on average, is then 280 modular multiplications.

11.61 Note (*system-wide parameters*) It is not necessary for each entity to select its own primes p and q. The DSS permits p, q, and α to be system-wide parameters. This does, however, present a more attractive target for an adversary.

11.62 Note (*probability of failure*) Verification requires the computation of $s^{-1} \bmod q$. If $s = 0$, then s^{-1} does not exist. To avoid this situation, the signer may check that $s \neq 0$; but if s is assumed to be a random element in $\mathbb{Z}_q$, then the probability that $s = 0$ is $(\frac{1}{2})^{160}$. In practice, this is extremely unlikely ever to occur. The signer may also check that $r \neq 0$. If the signer detects that either $r = 0$ or $s = 0$, a new value of k should be generated.

11.5.2 The ElGamal signature scheme

The ElGamal signature scheme is a randomized signature mechanism. It generates digital signatures with appendix on binary messages of arbitrary length, and requires a hash function $h\colon \{0,1\}^* \longrightarrow \mathbb{Z}_p$ where p is a large prime number. The DSA (§11.5.1) is a variant of the ElGamal signature mechanism.

11.63 Algorithm Key generation for the ElGamal signature scheme

SUMMARY: each entity creates a public key and corresponding private key.
Each entity A should do the following:

1. Generate a large random prime p and a generator α of the multiplicative group $\mathbb{Z}_p^*$ (using Algorithm 4.84).
2. Select a random integer a, $1 \leq a \leq p-2$.
3. Compute $y = \alpha^a \bmod p$ (e.g., using Algorithm 2.143).
4. A's public key is (p, α, y); A's private key is a.

11.64 Algorithm ElGamal signature generation and verification

SUMMARY: entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key.

1. *Signature generation.* Entity A should do the following:
   (a) Select a random secret integer k, $1 \leq k \leq p-2$, with $\gcd(k, p-1) = 1$.
   (b) Compute $r = \alpha^k \bmod p$ (e.g., using Algorithm 2.143).
   (c) Compute $k^{-1} \bmod (p-1)$ (e.g., using Algorithm 2.142).
   (d) Compute $s = k^{-1}\{h(m) - ar\} \bmod (p-1)$.
   (e) A's signature for m is the pair (r, s).
2. *Verification.* To verify A's signature (r, s) on m, B should do the following:

(a) Obtain A's authentic public key (p, α, y).
(b) Verify that $1 \leq r \leq p-1$; if not, then reject the signature.
(c) Compute $v_1 = y^r r^s \bmod p$.
(d) Compute $h(m)$ and $v_2 = \alpha^{h(m)} \bmod p$.
(e) Accept the signature if and only if $v_1 = v_2$.

Proof that signature verification works. If the signature was generated by A, then $s \equiv k^{-1} \{h(m) - ar\} \pmod{p-1}$. Multiplying both sides by k gives $ks \equiv h(m) - ar \pmod{p-1}$, and rearranging yields $h(m) \equiv ar + ks \pmod{p-1}$. This implies $\alpha^{h(m)} \equiv \alpha^{ar+ks} \equiv (\alpha^a)^r r^s \pmod{p}$. Thus, $v_1 = v_2$, as required.

11.65 Example (*ElGamal signature generation with artificially small parameters*)
Key generation. A selects the prime $p = 2357$ and a generator $\alpha = 2$ of $\mathbb{Z}_{2357}^*$. A chooses the private key $a = 1751$ and computes $y = \alpha^a \bmod p = 2^{1751} \bmod 2357 = 1185$. A's public key is ($p = 2357$, $\alpha = 2$, $y = 1185$).
Signature generation. For simplicity, messages will be integers from $\mathbb{Z}_p$ and $h(m) = m$ (i.e., for this example only, take h to be the identity function). To sign the message $m = 1463$, A selects a random integer $k = 1529$, computes $r = \alpha^k \bmod p = 2^{1529} \bmod 2357 = 1490$, and $k^{-1} \bmod (p-1) = 245$. Finally, A computes $s = 245\{1463 - 1751(1490)\} \bmod 2356 = 1777$. A's signature for $m = 1463$ is the pair ($r = 1490, s = 1777$).
Signature verification. B computes $v_1 = 1185^{1490} \cdot 1490^{1777} \bmod 2357 = 1072$, $h(m) = 1463$, and $v_2 = 2^{1463} \bmod 2357 = 1072$. B accepts the signature since $v_1 = v_2$. □

11.66 Note (*security of ElGamal signatures*)

(i) An adversary might attempt to forge A's signature (per Algorithm 11.64) on m by selecting a random integer k and computing $r = \alpha^k \bmod p$. The adversary must then determine $s = k^{-1}\{h(m) - ar\} \bmod (p-1)$. If the discrete logarithm problem is computationally infeasible, the adversary can do no better than to choose an s at random; the success probability is only $\frac{1}{p}$, which is negligible for large p.
(ii) A different k must be selected for each message signed; otherwise, the private key can be determined with high probability as follows. Suppose $s_1 = k^{-1}\{h(m_1) - ar\} \bmod (p-1)$ and $s_2 = k^{-1}\{h(m_2) - ar\} \bmod (p-1)$. Then $(s_1 - s_2)k \equiv (h(m_1) - h(m_2)) \pmod{p-1}$. If $s_1 - s_2 \not\equiv 0 \pmod{p-1}$, then $k = (s_1 - s_2)^{-1}(h(m_1) - h(m_2)) \bmod (p-1)$. Once k is known, a is easily found.
(iii) If no hash function h is used, the signing equation is $s = k^{-1}\{m - ar\} \bmod (p-1)$. It is then easy for an adversary to mount an existential forgery attack as follows. Select any pair of integers (u, v) with $\gcd(v, p-1) = 1$. Compute $r = \alpha^u y^v \bmod p = \alpha^{u+av} \bmod p$ and $s = -rv^{-1} \bmod (p-1)$. The pair (r, s) is a valid signature for the message $m = su \bmod (p-1)$, since $(\alpha^m \alpha^{-ar})^{s^{-1}} = \alpha^u y^v = r$.
(iv) Step 2b in Algorithm 11.64 requires the verifier to check that $0 < r < p$. If this check is not done, then an adversary can sign messages of its choice provided it has one valid signature created by entity A, as follows. Suppose that (r, s) is a signature for message m produced by A. The adversary selects a message m' of its choice and computes $h(m')$ and $u = h(m') \cdot [h(m)]^{-1} \bmod (p-1)$ (assuming $[h(m)]^{-1} \bmod (p-1)$ exists). It then computes $s' = su \bmod (p-1)$ and r' such that $r' \equiv ru \pmod{p-1}$ and $r' \equiv r \pmod{p}$. The latter is always possible by the Chinese Remainder Theorem (Fact 2.120). The pair (r', s') is a signature for message m' which would be accepted by the verification algorithm (Algorithm 11.64) if step 2b were ignored.

11.67 Note (*security based on parameter selection*)

(i) (*index-calculus attack*) The prime p should be sufficiently large to prevent efficient use of the index-calculus methods (§3.6.5).

(ii) (*Pohlig-Hellman attack*) $p-1$ should be divisible by a prime number q sufficiently large to prevent a Pohlig-Hellman discrete logarithm attack (§3.6.4).

(iii) (*weak generators*) Suppose that $p \equiv 1 \pmod 4$ and the generator α satisfies the following conditions:

(a) α divides $(p-1)$; and

(b) computing logarithms in the subgroup S of order α in $\mathbb{Z}_p^*$ can be efficiently done (for example, if a Pohlig-Hellman attack (§3.6.4) can be mounted in S).

It is then possible for an adversary to construct signatures (without knowledge of A's private key) which will be accepted by the verification algorithm (step 2 of Algorithm 11.64). To see this, suppose that $p-1 = \alpha q$. To sign a message m the adversary does the following:

(a) Compute $t = (p-3)/2$ and set $r = q$.

(b) Determine z such that $\alpha^{qz} \equiv y^q \pmod p$ where y is A's public key. (This is possible since α^q and y^q are elements of S and α^q is a generator of S.)

(c) Compute $s = t \cdot \{h(m) - qz\} \bmod (p-1)$.

(d) (r, s) is a signature on m which will be accepted by step 2 of Algorithm 11.64.

This attack works because the verification equation $r^s y^r \equiv \alpha^{h(m)} \pmod p$ is satisfied. To see this, first observe that $\alpha q \equiv -1 \pmod p$, $\alpha \equiv -q^{-1} \pmod p$, and that $q^{(p-1)/2} \equiv -1 \pmod p$. (The latter congruence follows from the fact that α is a generator of $\mathbb{Z}_p^*$ and $q \equiv -\alpha^{-1} \pmod p$.) From these, one deduces that $q^t = q^{(p-1)/2} q^{-1} \equiv -q^{-1} \equiv \alpha \pmod p$. Now $r^s y^r = (q^t)^{[h(m)-qz]} y^q \equiv \alpha^{h(m)} \alpha^{-qz} y^q \equiv \alpha^{h(m)} y^{-q} y^q = \alpha^{h(m)} \pmod p$. Notice in the case where $\alpha = 2$ is a generator that the conditions specified in (iii) above are trivially satisfied.

The attack can be avoided if α is selected as a generator for a subgroup of $\mathbb{Z}_p^*$ of prime order rather than a generator for $\mathbb{Z}_p^*$ itself.

11.68 Note (*performance characteristics of ElGamal signatures*)

(i) Signature generation by Algorithm 11.64 is relatively fast, requiring one modular exponentiation ($\alpha^k \bmod p$), the extended Euclidean algorithm (for computing $k^{-1} \bmod (p-1)$), and two modular multiplications. (Modular subtraction is negligible when compared with modular multiplication.) The exponentiation and application of the extended Euclidean algorithm can be done off-line, in which case signature generation (in instances where precomputation is possible) requires only two (on-line) modular multiplications.

(ii) Signature verification is more costly, requiring three exponentiations. Each exponentiation (using naive techniques) requires $\frac{3}{2}\lceil \lg p \rceil$ modular multiplications, on average, for a total cost of $\frac{9}{2}\lceil \lg p \rceil$ multiplications. The computing costs can be reduced by modifying the verification slightly. Compute $v_1 = \alpha^{-h(m)} y^r r^s \bmod p$, and accept the signature as valid if and only if $v_1 = 1$. Now, v_1 can be computed more efficiently by doing the three exponentiations simultaneously (see Note 14.91); the total cost is now about $\frac{15}{8}\lceil \lg p \rceil$ modular multiplications, almost 2.5 times as cost efficient as before.

(iii) Signature verification calculations are all performed modulo p, while signature generation calculations are done modulo p and modulo $(p-1)$.

11.69 Note (*recommended parameter sizes*) Given the latest progress on the discrete logarithm problem in $\mathbb{Z}_p^*$ (§3.6), a 512-bit modulus p provides only marginal security from concerted attack. As of 1996, a modulus p of at least 768 bits is recommended. For long-term security, 1024-bit or larger moduli should be used.

11.70 Note (*system-wide parameters*) All entities may elect to use the same prime number p and generator α, in which case p and α are not required to be part of the public key (cf. Note 11.61).

(i) Variations of the ElGamal scheme

Many variations of the basic ElGamal signature scheme (Algorithm 11.64) have been proposed. Most of these alter what is commonly referred to as the *signing equation* (given in step 1d of Algorithm 11.64). After suitable rearrangement, this signing equation can be written as $u = av + kw \bmod (p-1)$ where $u = h(m)$, $v = r$, and $w = s$ (i.e., $h(m) = ar + ks \bmod (p-1)$). Other signing equations can be obtained by permitting u, v, and w to take on the values s, r, and $h(m)$ in different orders. Table 11.5 lists the 6 possibilities.

	u	v	w	Signing equation	Verification
1	$h(m)$	r	s	$h(m) = ar + ks$	$\alpha^{h(m)} = (\alpha^a)^r r^s$
2	$h(m)$	s	r	$h(m) = as + kr$	$\alpha^{h(m)} = (\alpha^a)^s r^r$
3	s	r	$h(m)$	$s = ar + kh(m)$	$\alpha^s = (\alpha^a)^r r^{h(m)}$
4	s	$h(m)$	r	$s = ah(m) + kr$	$\alpha^s = (\alpha^a)^{h(m)} r^r$
5	r	s	$h(m)$	$r = as + kh(m)$	$\alpha^r = (\alpha^a)^s r^{h(m)}$
6	r	$h(m)$	s	$r = ah(m) + ks$	$\alpha^r = (\alpha^a)^{h(m)} r^s$

Table 11.5: *Variations of the ElGamal signing equation. Signing equations are computed modulo* $(p-1)$*; verification is done modulo p.*

11.71 Note (*comparing variants of the ElGamal signature scheme*)

(i) Some of the signing equations listed in Table 11.5 are more efficient to compute than the original ElGamal equation in Algorithm 11.64. For example, equations (3) and (4) of Table 11.5 do not require the computation of an inverse to determine the signature s. Equations (2) and (5) require the signer to compute $a^{-1} \bmod (p-1)$, but this fixed quantity need only be computed once.

(ii) Verification equations (2) and (4) involve the expression r^r. Part of the security of signature schemes based on these signing equations is the intractability of finding solutions to an expression of the form $x^x \equiv c \pmod p$ for fixed c. This problem appears to be intractable for large values of p, but has not received the same attention as the discrete logarithm problem.

(ii) The generalized ElGamal signature scheme

The ElGamal digital signature scheme, originally described in the setting of the multiplicative group $\mathbb{Z}_p^*$, can be generalized in a straightforward manner to work in any finite abelian group G. The introductory remarks for §8.4.2 are pertinent to the algorithm presented in this section. Algorithm 11.73 requires a cryptographic hash function $h\colon \{0,1\}^* \longrightarrow \mathbb{Z}_n$

where n is the number of elements in G. It is assumed that each element $r \in G$ can be represented in binary so that $h(r)$ is defined.[9]

11.72 Algorithm Key generation for the generalized ElGamal signature scheme

SUMMARY: each entity selects a finite group G; generator of G; public and private keys. Each entity A should do the following:

1. Select an appropriate cyclic group G of order n, with generator α. (Assume that G is written multiplicatively.)
2. Select a random secret integer a, $1 \le a \le n-1$. Compute the group element $y = \alpha^a$.
3. A's public key is (α, y), together with a description of how to multiply elements in G; A's private key is a.

11.73 Algorithm Generalized ElGamal signature generation and verification

SUMMARY: entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key.

1. *Signature generation.* Entity A should do the following:
   (a) Select a random secret integer k, $1 \le k \le n-1$, with $\gcd(k, n) = 1$.
   (b) Compute the group element $r = \alpha^k$.
   (c) Compute $k^{-1} \bmod n$.
   (d) Compute $h(m)$ and $h(r)$.
   (e) Compute $s = k^{-1}\{h(m) - ah(r)\} \bmod n$.
   (f) A's signature for m is the pair (r, s).
2. *Verification.* To verify A's signature (r, s) on m, B should do the following:
   (a) Obtain A's authentic public key (α, y).
   (b) Compute $h(m)$ and $h(r)$.
   (c) Compute $v_1 = y^{h(r)} \cdot r^s$.
   (d) Compute $v_2 = \alpha^{h(m)}$.
   (e) Accept the signature if and only if $v_1 = v_2$.

11.74 Example (*generalized ElGamal signatures with artificially small parameters*)
Key generation. Consider the finite field $\mathbb{F}_{2^5}$ constructed from the irreducible polynomial $f(x) = x^5 + x^2 + 1$ over $\mathbb{F}_2$. (See Example 2.231 for examples of arithmetic in the field $\mathbb{F}_{2^4}$.) The elements of this field are the 31 binary 5-tuples displayed in Table 11.6, along with 00000. The element $\alpha = (00010)$ is a generator for $G = \mathbb{F}^*_{2^5}$, the multiplicative cyclic group of the field. The order of this group G is $n = 31$. Let $h\colon \{0,1\}^* \longrightarrow \mathbb{Z}_{31}$ be a hash function. Entity A selects the private key $a = 19$ and computes $y = \alpha^a = (00010)^{19} = (00110)$. A's public key is $(\alpha = (00010), y = (00110))$.
Signature generation. To sign the message $m = 10110101$, A selects a random integer $k = 24$, and computes $r = \alpha^{24} = (11110)$ and $k^{-1} \bmod 31 = 22$. A then computes $h(m) = 16$ and $h(r) = 7$ (the hash values have been contrived for this example) and $s = 22 \cdot \{16 - (19)(7)\} \bmod 31 = 30$. A's signature for message m is $(r = (11110), s = 30)$.
Signature verification. B computes $h(m) = 16$, $h(r) = 7$, $v_1 = y^{h(r)} r^s = (00110)^7 \cdot (11110)^{30} = (11011)$, and $v_2 = \alpha^{h(m)} = \alpha^{16} = (11011)$. B accepts the signature since $v_1 = v_2$. □

[9]More precisely, one would define a function $f\colon G \longrightarrow \{0,1\}^*$ and write $h(f(r))$ instead of $h(r)$.

i	α^i	i	α^i	i	α^i	i	α^i
0	00001	8	01101	16	11011	24	11110
1	00010	9	11010	17	10011	25	11001
2	00100	10	10001	18	00011	26	10111
3	01000	11	00111	19	00110	27	01011
4	10000	12	01110	20	01100	28	10110
5	00101	13	11100	21	11000	29	01001
6	01010	14	11101	22	10101	30	10010
7	10100	15	11111	23	01111		

Table 11.6: *The elements of* $\mathbb{F}_{2^5}$ *as powers of a generator* α.

11.75 Note (*security of generalized ElGamal*) Much of the security of Algorithm 11.73 relies on the intractability of the discrete logarithm problem in the group G (see §3.6). Most of the security comments in Note 11.66 apply to the generalized ElGamal scheme.

11.76 Note (*signing and verification operations*) Signature generation requires computations in the group G (i.e., $r = \alpha^k$) and computations in $\mathbb{Z}_n$. Signature verification only requires computations in the group G.

11.77 Note (*generalized ElGamal using elliptic curves*) One of the most promising implementations of Algorithm 11.73 is the case where the finite abelian group G is constructed from the set of points on an elliptic curve over a finite field $\mathbb{F}_q$. The discrete logarithm problem in groups of this type appears to be more difficult than the discrete logarithm problem in the multiplicative group of a finite field $\mathbb{F}_q$. This implies that q can be chosen smaller than for corresponding implementations in groups such as $G = \mathbb{F}_q^*$.

11.5.3 The Schnorr signature scheme

Another well-known variant of the ElGamal scheme (Algorithm 11.64) is the Schnorr signature scheme. As with the DSA (Algorithm 11.56), this technique employs a subgroup of order q in $\mathbb{Z}_p^*$, where p is some large prime number. The method also requires a hash function $h\colon \{0,1\}^* \longrightarrow \mathbb{Z}_q$. Key generation for the Schnorr signature scheme is the same as DSA key generation (Algorithm 11.54), except that there are no constraints on the sizes of p and q.

11.78 Algorithm Schnorr signature generation and verification

SUMMARY: entity A signs a binary message m of arbitrary length. Any entity B can verify this signature by using A's public key.

1. *Signature generation.* Entity A should do the following:
   (a) Select a random secret integer k, $1 \le k \le q-1$.
   (b) Compute $r = \alpha^k \bmod p$, $e = h(m\|r)$, and $s = ae + k \bmod q$.
   (c) A's signature for m is the pair (s, e).

2. *Verification.* To verify A's signature (s, e) on m, B should do the following:
   (a) Obtain A's authentic public key (p, q, α, y).
   (b) Compute $v = \alpha^s y^{-e} \bmod p$ and $e' = h(m\|v)$.
   (c) Accept the signature if and only if $e' = e$.

Proof that signature verification works. If the signature was created by A, then $v \equiv \alpha^s y^{-e} \equiv \alpha^s \alpha^{-ae} \equiv \alpha^k \equiv r \pmod p$. Hence, $h(m\|v) = h(m\|r)$ and $e' = e$.

11.79 Example (*Schnorr's signature scheme with artificially small parameters*)
Key generation. A selects primes $p = 129841$ and $q = 541$; here, $(p-1)/q = 240$. A then selects a random integer $g = 26346 \in \mathbb{Z}_p^*$ and computes $\alpha = 26346^{240} \bmod p = 26$. Since $\alpha \neq 1$, α generates the unique cyclic subgroup of order 541 in $\mathbb{Z}_p^*$. A then selects the private key $a = 423$ and computes $y = 26^{423} \bmod p = 115917$. A's public key is $(p = 129841, q = 541, \alpha = 26, y = 115917)$.
Signature generation. To sign the message $m = 11101101$, A selects a random number $k = 327$ such that $1 \leq k \leq 540$, and computes $r = 26^{327} \bmod p = 49375$ and $e = h(m\|r) = 155$ (the hash value has been contrived for this example). Finally, A computes $s = 423 \cdot 155 + 327 \bmod 541 = 431$. The signature for m is $(s = 431, e = 155)$.
Signature verification. B computes $v = 26^{431} \cdot 115917^{-155} \bmod p = 49375$ and $e' = h(m\|v) = 155$. B accepts the signature since $e = e'$. □

11.80 Note (*performance characteristics of the Schnorr scheme*) Signature generation in Algorithm 11.78 requires one exponentiation modulo p and one multiplication modulo q. The exponentiation modulo p could be done off-line. Depending on the hash algorithm used, the time to compute $h(m\|r)$ should be relatively small. Verification requires two exponentiations modulo p. These two exponentiations can be computed by Algorithm 14.88 at a cost of about 1.17 exponentiations. Using the subgroup of order q does not significantly enhance computational efficiency over the ElGamal scheme of Algorithm 11.64, but does provide smaller signatures (for the same level of security) than those generated by the ElGamal method.

11.5.4 The ElGamal signature scheme with message recovery

The ElGamal scheme and its variants (§11.5.2) discussed so far are all randomized digital signature schemes with appendix (i.e., the message is required as input to the verification algorithm). In contrast, the signature mechanism of Algorithm 11.81 has the feature that the message can be recovered from the signature itself. Hence, this ElGamal variant provides a randomized digital signature with message recovery.

For this scheme, the signing space is $\mathcal{M}_\mathcal{S} = \mathbb{Z}_p^*$, p a prime, and the signature space is $\mathcal{S} = \mathbb{Z}_p \times \mathbb{Z}_q$, q a prime, where q divides $(p-1)$. Let R be a redundancy function from the set of messages $\mathcal{M}$ to $\mathcal{M}_\mathcal{S}$ (see Table 11.1). Key generation for Algorithm 11.81 is the same as DSA key generation (Algorithm 11.54), except that there are no constraints on the sizes of p and q.

11.81 Algorithm Nyberg-Rueppel signature generation and verification

SUMMARY: entity A signs a message $m \in \mathcal{M}$. Any entity B can verify A's signature and recover the message m from the signature.

1. *Signature generation.* Entity A should do the following:
   (a) Compute $\widetilde{m} = R(m)$.
   (b) Select a random secret integer k, $1 \le k \le q-1$, and compute $r = \alpha^{-k} \bmod p$.
   (c) Compute $e = \widetilde{m}r \bmod p$.
   (d) Compute $s = ae + k \bmod q$.
   (e) A's signature for m is the pair (e, s).
2. *Verification.* To verify A's signature (e, s) on m, B should do the following:
   (a) Obtain A's authentic public key (p, q, α, y).
   (b) Verify that $0 < e < p$; if not, reject the signature.
   (c) Verify that $0 \le s < q$; if not, reject the signature.
   (d) Compute $v = \alpha^s y^{-e} \bmod p$ and $\widetilde{m} = ve \bmod p$.
   (e) Verify that $\widetilde{m} \in \mathcal{M}_R$; if $\widetilde{m} \notin \mathcal{M}_R$ then reject the signature.
   (f) Recover $m = R^{-1}(\widetilde{m})$.

Proof that signature verification works. If A created the signature, then $v \equiv \alpha^s y^{-e} \equiv \alpha^{s-ae} \equiv \alpha^k \pmod p$. Thus $ve \equiv \alpha^k \widetilde{m} \alpha^{-k} \equiv \widetilde{m} \pmod p$, as required.

11.82 Example (*Nyberg-Rueppel signature generation with artificially small parameters*)
Key generation. Entity A selects primes $p = 1256993$ and $q = 3571$, where q divides $(p-1)$; here, $(p-1)/q = 352$. A then selects a random number $g = 42077 \in \mathbb{Z}_p^*$ and computes $\alpha = 42077^{352} \bmod p = 441238$. Since $\alpha \neq 1$, α generates the unique cyclic subgroup of order 3571 in $\mathbb{Z}_p^*$. Finally, A selects a random integer $a = 2774$ and computes $y = \alpha^a \bmod p = 1013657$. A's public key is $(p = 1256993, q = 3571, \alpha = 441238, y = 1013657)$, while A's private key is $a = 2774$.
Signature generation. To sign a message m, A computes $\widetilde{m} = R(m) = 1147892$ (the value $R(m)$ has been contrived for this example). A then randomly selects $k = 1001$, computes $r = \alpha^{-k} \bmod p = 441238^{-1001} \bmod p = 1188935$, $e = \widetilde{m}r \bmod p = 138207$, and $s = (2774)(138207) + 1001 \bmod q = 1088$. The signature for m is $(e = 138207, s = 1088)$.
Signature verification. B computes $v = 441238^{1088} \cdot 1013657^{-138207} \bmod 1256993 = 504308$, and $\widetilde{m} = v \cdot 138207 \bmod 1256993 = 1147892$. B verifies that $\widetilde{m} \in \mathcal{M}_R$ and recovers $m = R^{-1}(\widetilde{m})$. □

11.83 Note (*security of the Nyberg-Rueppel signature scheme*)

(i) Since Algorithm 11.81 is a variant of the basic ElGamal scheme (Algorithm 11.64), the security considerations of Note 11.66 apply. Like DSA (Algorithm 11.56), this ElGamal mechanism with message recovery relies on the difficulty of two related but distinct discrete logarithm problems (see Note 11.58).

(ii) Since Algorithm 11.81 provides message recovery, a suitable redundancy function R is required (see Note 11.10) to guard against existential forgery. As is the case with RSA, the multiplicative nature of this signature scheme must be carefully considered when choosing a redundancy function R. The following possible attack should be kept in mind. Suppose $m \in \mathcal{M}$, $\widetilde{m} = R(m)$, and (e, s) is a signature for m. Then $e = \widetilde{m}\alpha^{-k} \bmod p$ for some integer k and $s = ae + k \bmod q$. Let $\widetilde{m}^* = \widetilde{m}\alpha^l \bmod p$ for some integer l. If $s^* = s + l \bmod q$ and $\widetilde{m}^* \in \mathcal{M}_R$, then (e, s^*)

is a valid signature for $m^* = R^{-1}(\widetilde{m}^*)$. To see this, consider the verification algorithm (step 2 of Algorithm 11.81). $v \equiv \alpha^{s^*} y^{-e} \equiv \alpha^{s+l}\alpha^{-ae} \equiv \alpha^{k+l} \pmod p$, and $ve \equiv \alpha^{k+l}\widetilde{m}\alpha^{-k} \equiv \widetilde{m}\alpha^l \equiv \widetilde{m}^* \pmod p$. Since $\widetilde{m}^* \in \mathcal{M}_R$, the forged signature (e, s^*) will be accepted as a valid signature for m^*.

(iii) The verification that $0 < e < p$ given in step 2b of Algorithm 11.81 is crucial. Suppose (e, s) is A's signature for the message m. Then $e = \widetilde{m}r \bmod p$ and $s = ae + k \bmod q$. An adversary can use this signature to compute a signature on a message m^* of its choice. It determines an e^* such that $e^* \equiv \widetilde{m}^* r \pmod p$ and $e^* \equiv e \pmod q$. (This is possible by the Chinese Remainder Theorem (Fact 2.120).) The pair (e^*, s) will pass the verification algorithm provided that $0 < e^* < p$ is not checked.

11.84 Note (*a generalization of ElGamal signatures with message recovery*) The expression $e = \widetilde{m}r \bmod p$ in step 1c of Algorithm 11.81 provides a relatively simple way to encrypt $\widetilde{m}$ with key r and could be generalized to any symmetric-key algorithm. Let $E = \{E_r : r \in \mathbb{Z}_p\}$ be a set of encryption transformations where each E_r is indexed by an element $r \in \mathbb{Z}_p^*$ and is a bijection from $\mathcal{M}_S = \mathbb{Z}_p^*$ to $\mathbb{Z}_p^*$. For any $m \in \mathcal{M}$, select a random integer k, $1 \le k \le q-1$, compute $r = \alpha^k \bmod p$, $e = E_r(\widetilde{m})$, and $s = ae + k \bmod q$. The pair (e, s) is a signature for m. The fundamental signature equation $s = ae + k \bmod q$ is a means to bind entity A's private key and the message m to a symmetric key which can then be used to recover the message by any other entity at some later time.

11.6 One-time digital signatures

One-time digital signature schemes are digital signature mechanisms which can be used to sign, at most, one message; otherwise, signatures can be forged. A new public key is required for each message that is signed. The public information necessary to verify one-time signatures is often referred to as *validation parameters*. When one-time signatures are combined with techniques for authenticating validation parameters, multiple signatures are possible (see §11.6.3 for a description of authentication trees).

Most, but not all, one-time digital signature schemes have the advantage that signature generation and verification are very efficient. One-time digital signature schemes are useful in applications such as chipcards, where low computational complexity is required.

11.6.1 The Rabin one-time signature scheme

Rabin's one-time signature scheme was one of the first proposals for a digital signature of any kind. It permits the signing of a single message. The verification of a signature requires interaction between the signer and verifier. Unlike other digital signature schemes, verification can be done only once. While not practical, it is presented here for historical reasons. Notation used in this section is given in Table 11.7.

Symbol	Meaning
M_0	$0^l =$ the all 0's string of bitlength l.
$M_0(i)$	$0^{l-e} \| b_{e-1} \cdots b_1 b_0$ where $b_{e-1} \cdots b_1 b_0$ is the binary representation of i.
$\mathcal{K}$	a set of l-bit strings.
E	a set of encryption transformations indexed by a key space $\mathcal{K}$.
E_t	an encryption transformation belonging to E with $t \in \mathcal{K}$. Each E_t maps l-bit strings to l-bit strings.
h	a publicly-known one-way hash function from $\{0,1\}^*$ to $\{0,1\}^l$.
n	a fixed positive integer which serves as a security parameter.

Table 11.7: *Notation for the Rabin one-time signature scheme.*

11.85 Algorithm Key generation for the Rabin one-time signature scheme

SUMMARY: each entity A selects a symmetric-key encryption scheme E, generates $2n$ random bitstrings, and creates a set of validation parameters.
Each entity A should do the following:

1. Select a symmetric-key encryption scheme E (e.g., DES).
2. Generate $2n$ random secret strings $k_1, k_2, \ldots, k_{2n} \in \mathcal{K}$, each of bitlength l.
3. Compute $y_i = E_{k_i}(M_0(i))$, $1 \le i \le 2n$.
4. A's public key is $(y_1, y_2, \ldots, y_{2n})$; A's private key is $(k_1, k_2, \ldots, k_{2n})$.

11.86 Algorithm Rabin one-time signature generation and verification

SUMMARY: entity A signs a binary message m of arbitrary length. Signature verification is interactive with A.

1. *Signature generation.* Entity A should do the following:
   (a) Compute $h(m)$.
   (b) Compute $s_i = E_{k_i}(h(m))$, $1 \le i \le 2n$.
   (c) A's signature for m is $(s_1, s_2, \ldots, s_{2n})$.
2. *Verification.* To verify A's signature $(s_1, s_2, \ldots, s_{2n})$ on m, B should:
   (a) Obtain A's authentic public key $(y_1, y_2, \ldots, y_{2n})$.
   (b) Compute $h(m)$.
   (c) Select n distinct random numbers r_j, $1 \le r_j \le 2n$, for $1 \le j \le n$.
   (d) Request from A the keys k_{r_j}, $1 \le j \le n$.
   (e) Verify the authenticity of the received keys by computing $z_j = E_{k_{r_j}}(M_0(r_j))$ and checking that $z_j = y_{r_j}$, for each $1 \le j \le n$.
   (f) Verify that $s_{r_j} = E_{k_{r_j}}(h(m))$, $1 \le j \le n$.

11.87 Note (*key sizes for Rabin's one-time signatures*) Since E_t outputs l bits (see Table 11.7), the public and private keys in Algorithm 11.86 each consist of $2nl$ bits. For $n = 80$ and $l = 64$, the keys are each 1280 bytes long.

11.88 Note (*resolution of disputes*) To resolve potential disputes between the signer A and the verifier B using Algorithm 11.86, the following procedure is followed:

1. B provides a trusted third party (TTP) with m and the signature $(s_1, s_2, \ldots, s_{2n})$.

2. The TTP obtains $k_1, k_2, \ldots, k_{2n}$ from A.
3. The TTP verifies the authenticity of the private key by computing $z_i = E_{k_i}(M_0(i))$ and checking that $y_i = z_i$, $1 \le i \le 2n$. If this fails, the TTP rules in favor of B (i.e., the signature is deemed to be valid).
4. The TTP computes $u_i = E_{k_i}(h(m))$, $1 \le i \le 2n$. If $u_i = s_i$ for at most n values of i, $1 \le i \le 2n$, the signature is declared a forgery and the TTP rules in favor of A (who denies having created the signature). If $n+1$ or more values of i give $u_i = s_i$, the signature is deemed valid and the TTP rules in favor of B.

11.89 Note (*rationale for dispute resolution protocol*) The rationale for adjudicating disputes in Rabin's one-time signature scheme, as outlined in Note 11.88, is as follows. If B has attempted to forge A's signature on a new message m', B either needs to determine at least one more key k' so that at least $n+1$ values of i give $u_i = s_i$, or determine m' such that $h(m) = h(m')$. This should be infeasible if the symmetric-key algorithm and hash function are chosen appropriately. If A attempts to create a signature which it can later disavow, A must ensure that $u_i = s_i$ for precisely n values of i and hope that B chooses these n values in step 2c of the verification procedure, the probability of which is only $1/\binom{2n}{n}$.

11.90 Note (*one-timeness of Algorithm 11.86*) A can sign at most one message with a given private key in Rabin's one-time scheme; otherwise, A will (with high probability) reveal $n+1$ or more of the private key values and enable B (and perhaps collaborators) to forge signatures on new messages (see Note 11.89). A signature can only be verified once without revealing (with high probability) more than n of the $2n$ private values.

11.6.2 The Merkle one-time signature scheme

Merkle's one-time digital signature scheme (Algorithm 11.92) differs substantially from that of Rabin (Algorithm 11.86) in that signature verification is not interactive with the signer. A TTP or some other trusted means is required to authenticate the validation parameters constructed in Algorithm 11.91.

11.91 Algorithm Key generation for the Merkle one-time signature scheme

SUMMARY: to sign n-bit messages, A generates $t = n + \lfloor \lg n \rfloor + 1$ validation parameters.
Each entity A should do the following:

1. Select $t = n + \lfloor \lg n \rfloor + 1$ random secret strings $k_1, k_2, \ldots, k_t$ each of bitlength l.
2. Compute $v_i = h(k_i)$, $1 \le i \le t$. Here, h is a preimage-resistant hash function $h\colon \{0,1\}^* \longrightarrow \{0,1\}^l$ (see §9.2.2).
3. A's public key is $(v_1, v_2, \ldots, v_t)$; A's private key is $(k_1, k_2, \ldots, k_t)$.

To sign an n-bit message m, a bitstring $w = m \| c$ is formed where c is the binary representation for the number of 0's in m. c is assumed to be a bitstring of bitlength $\lfloor \lg n \rfloor + 1$ with high-order bits padded with 0's, if necessary. Hence, w is a bitstring of bitlength $t = n + \lfloor \lg n \rfloor + 1$.

11.92 Algorithm Merkle one-time signature generation and verification

SUMMARY: entity A signs a binary message m of bitlength n. Any entity B can verify this signature by using A's public key.

1. *Signature generation.* Entity A should do the following:
   (a) Compute c, the binary representation for the number of 0's in m.
   (b) Form $w = m\|c = (a_1a_2\cdots a_t)$.
   (c) Determine the coordinate positions $i_1 < i_2 < \cdots < i_u$ in w such that $a_{i_j} = 1$, $1 \le j \le u$.
   (d) Let $s_j = k_{i_j}$, $1 \le j \le u$.
   (e) A's signature for m is $(s_1, s_2, \ldots, s_u)$.
2. *Verification.* To verify A's signature $(s_1, s_2, \ldots, s_u)$ on m, B should:
   (a) Obtain A's authentic public key $(v_1, v_2, \ldots, v_t)$.
   (b) Compute c, the binary representation for the number of 0's in m.
   (c) Form $w = m\|c = (a_1a_2\cdots a_t)$.
   (d) Determine the coordinate positions $i_1 < i_2 < \cdots < i_u$ in w such that $a_{i_j} = 1$, $1 \le j \le u$.
   (e) Accept the signature if and only if $v_{i_j} = h(s_j)$ for all $1 \le j \le u$.

11.93 Note (*security of Merkle's one-time signature scheme*) Let m be a message, $w = m\|c$ the bitstring formed in step 1b of Algorithm 11.92, and $(s_1, s_2, \ldots, s_u)$ a signature for m. If h is a preimage-resistant hash function, the following argument shows that no signature for a message $m' \ne m$ can be forged. Let $w' = m'\|c'$ where c' is the $(\lfloor \lg n \rfloor + 1)$-bit string which is the binary representation for the number of 0's in m'. Since an adversary has access to only that portion of the signer's private key which consists of $(s_1, s_2, \ldots, s_u)$, the set of coordinate positions in m' having a 1 must be a subset of the coordinate positions in m having a 1 (otherwise, m' will have a 1 in some position where m has a 0 and the adversary will require an element of the private key not revealed by the signer). But this means that m' has more 0's than m and that $c' > c$ (when considered as integers). In this case, c' will have a 1 in some position where c has a 0. The adversary would require a private key element, corresponding to this position, which was not revealed by the signer.

11.94 Note (*storage and computational requirements of Algorithm 11.92*)

(i) To sign an n-bit message m which has k ones requires $l \cdot (n + \lfloor \lg n \rfloor + 1)$ bits of storage for the validation parameters (public key), and $l \cdot (n + \lfloor \lg n \rfloor + 1)$ bits for the private key. The signature requires $l \cdot (k + k')$ bits of storage, where k' is the number of 1's in the binary representation of $n - k$. For example, if $n = 128$, $l = 64$, and $k = 72$, then the public and private keys each require 8704 bits (1088 bytes). The signature requires 4800 bits (600 bytes).

(ii) The private key can be made smaller by forming the k_i's from a single *seed* value. For example, if k^* is a bitstring of bitlength at least l, then form $k_i = h(k^*\|i)$, $1 \le i \le t$. Since only the seed k^* need be stored, the size of the private key is drastically reduced.

(iii) Signature generation is very fast, requiring no computation. Signature verification requires the evaluation of the hash function for fewer than $n + \lfloor \lg n \rfloor + 1$ values.

11.95 Note (*improving efficiency of Merkle's one-time scheme*) Algorithm 11.92 requires $l\cdot(n+\lfloor\lg n\rfloor+1)$ bits for each of the public and private keys. The public key must necessarily be this large because the signing algorithm considers individual bits of the message. The scheme can be made more efficient if the signing algorithm considers more than one bit at a time. Suppose entity A wishes to sign a kt-bit message m. Write $m = m_1\|m_2\|\cdots\|m_t$ where each m_i has bitlength k and each represents an integer between 0 and 2^k-1 inclusive. Define $U=\sum_{i=1}^{t}(2^k-m_i)\le t2^k$. U can be represented by $\lg U\le\lfloor\lg t\rfloor+1+k$ bits. If $r=\lceil(\lfloor\lg t\rfloor+1+k)/k\rceil$, then U can be written in binary as $U=u_1\|u_2\|\cdots\|u_r$, where each u_i has bitlength k. Form the bitstring $w=m_1\|m_2\|\cdots m_t\|u_1\|u_2\|\cdots\|u_r$. Generate $t+r$ random bitstrings $k_1,k_2,\ldots,k_{t+r}$ and compute $v_i=h^{2^k-1}(k_i)$, $1\le i\le t+r$. The private key for the modified scheme is $(k_1,k_2,\ldots,k_{t+r})$ and the public key is $(v_1,v_2,\ldots,v_{t+r})$. The signature for m is $(s_1,s_2,\ldots,s_{t+r})$ where $s_i=h^{m_i}(k_i)$, $1\le i\le t$, and $s_i=h^{u_i}(k_{t+i})$, $1\le i\le r$. Here, h^c denotes the c-fold composition of h with itself. As with the original scheme (Algorithm 11.92), the bits appended to the message act as a check-sum (see Note 11.93) as follows. Given an element $s_i=h^a(k_j)$, an adversary can easily compute $h^{a+\delta}(k_j)$ for $0\le\delta\le 2^k-a$, but is unable to compute $h^{a-\delta}$ for any $\delta>0$ if h is a one-way hash function. To forge a signature on a new message, an adversary can only reduce the value of the check-sum, which will make it impossible for him to compute the required hash values on the appended kr bits.

11.96 Example (*signing more than one bit at a time*) This example illustrates the modification of the Merkle scheme described in Note 11.95. Let $m=m_1\|m_2\|m_3\|m_4$ where $m_1=1011$, $m_2=0111$, $m_3=1010$, and $m_4=1101$. m_1, m_2, m_3, and m_4 are the binary representations of 11, 7, 10, and 13, respectively. $U=(16-m_1)+(16-m_2)+(16-m_3)+(16-m_4)=5+9+6+3=23$. In binary, $U=10111$. Form $w=m\|0001\,0111$. The signature is $(s_1,s_2,s_3,s_4,s_5,s_6)$ where $s_1=h^{11}(k_1)$, $s_2=h^7(k_2)$, $s_3=h^{10}(k_3)$, $s_4=h^{13}(k_4)$, $s_5=h^1(x_5)$, and $s_6=h^7(x_6)$. If an adversary tries to alter the message, he can only apply the function h to some s_i. This causes the sum of the exponents used (i.e., $\sum m_i$) to increase and, hence, $t2^d-\sum m_i$ to decrease. An adversary would be unable to modify the last two blocks since h^{-1} is required to decrease the sum. But, since h is preimage-resistant, h^{-1} cannot be computed by the adversary. □

11.6.3 Authentication trees and one-time signatures

§13.4.1 describes the basic structure of an authentication tree and relates how such a tree could be used, among other things, to authenticate a large number of public validation parameters for a one-time signature scheme. This section describes how an authentication tree can be used in conjunction with a one-time signature scheme to provide a scheme which allows multiple signatures. A small example will serve to illustrate how this is done.

11.97 Example (*an authentication tree for Merkle's one-time scheme*) Consider the one-time signature scheme of Algorithm 11.92 for signing n-bit messages. Let $h\colon\{0,1\}^*\longrightarrow\{0,1\}^l$ be a preimage-resistant hash function and $t=n+\lfloor\lg n\rfloor+1$. Figure 11.7 illustrates a 5-vertex binary tree created by an entity A in the course of signing five messages m_0,m_1,m_2,m_3,m_4. Each vertex in the tree is associated with one of the five messages. For the vertex associated with message m_i, A has selected $X_i=(x_{1i},x_{2i},\ldots,x_{ti})$, $U_i=(u_{1i},u_{2i},\ldots,u_{ti})$ and $W_i=(w_{1i},w_{2i},\ldots,w_{ti})$, $0\le i\le 4$, the elements of which are random bitstrings. From these lists, A has computed $Y_i=(h(x_{ji})\colon 1\le j\le t)$, $V_i=(h(u_{ji})\colon 1\le j\le t)$, and $Z_i=(h(w_{ji})\colon 1\le j\le t)$. Define $h(Y_i)=$

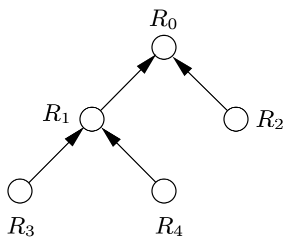


Figure 11.7: *An authentication tree for the Merkle one-time signature scheme (cf. Example 11.97).*

$h(h(x_{1i})\|h(x_{2i})\|\cdots\|h(x_{ti}))$ for $0 \le i \le 4$, and define $h(V_i)$ and $h(Z_i)$ analogously. Denote the Merkle one-time signature of m_i using private key X_i by $S_A(m_i, X_i)$, $0 \le i \le 4$. Y_i is the set of validation parameters for the signature $S_A(m_i, X_i)$. Finally, let $R_i = h(h(Y_i)\|h(V_i)\|h(Z_i))$, $0 \le i \le 4$. Table 11.8 summarizes the parameters associated with the vertex R_i. The sets U_i and W_i are used to sign the labels of the children

message	m_i
private parameters	X_i, U_i, W_i
public parameters	Y_i, V_i, Z_i
hash values	$h(Y_i), h(V_i), h(Z_i)$
R_i	$h(h(Y_i)\|h(V_i)\|h(Z_i))$
signature	$S_A(m_i, X_i)$
validation parameters	Y_i

Table 11.8: *Parameters and signature associated with vertex R_i, $0 \le i \le 4$ (cf. Figure 11.7).*

of vertex R_i. The signature on vertex R_0 is that of a trusted third party (TTP). Table 11.9 summarizes the parameters and signatures associated with each vertex label of the binary tree. To describe how the tree is used to verify signatures, consider message m_4 and signa-

Message	Vertex Label	Signature on Vertex Label	Authentication Parameters
m_0	R_0	Signature of TTP	—
m_1	R_1	$S_A(R_1, U_0)$	$V_0, h(Y_0), h(Z_0)$
m_2	R_2	$S_A(R_2, W_0)$	$Z_0, h(Y_0), h(V_0)$
m_3	R_3	$S_A(R_3, U_1)$	$V_1, h(Y_1), h(Z_1)$
m_4	R_4	$S_A(R_4, W_1)$	$Z_1, h(Y_1), h(V_1)$

Table 11.9: *Parameters and signatures associated with vertices of the binary tree (cf. Figure 11.7).*

ture $S_A(m_4, X_4)$. The signer A first provides the verifier B with the validation parameters Y_4. The verifier checks the Merkle one-time signature using step 2 of Algorithm 11.92. B must then be convinced that Y_4 is an authentic set of validation parameters created by A. To accomplish this, A provides B with a sequence of values enumerated in the steps below:

1. $h(V_4), h(Z_4)$; B computes $h(Y_4)$ and then $R_4 = h(h(Y_4)\|h(V_4)\|h(Z_4))$.
2. $S_A(R_4, W_1)$ and Z_1; B verifies the signature on R_4 using Algorithm 11.92.
3. $h(Y_1), h(V_1)$; B computes $h(Z_1)$ and then $R_1 = h(h(Y_1)\|h(V_1)\|h(Z_1))$.
4. $S_A(R_1, U_0)$ and V_0; B verifies the signature using Algorithm 11.92.

5. $h(Y_0)$, $h(Z_0)$; B computes $h(V_0)$ and then $R_0 = h(h(Y_0)\|h(V_0)\|h(Z_0))$.
6. the signature of the TTP for R_0; B verifies the TTP's signature using an algorithm appropriate to the signature mechanism for the TTP.

The binary tree on 5 vertices (Figure 11.7) could be extended indefinitely from any leaf as more signatures are created by A. The length of a longest authentication path (or equivalently, the depth of the tree) determines the maximum amount of information which A must provide B in order for B to verify the signature of a message associated with a vertex. □

11.6.4 The GMR one-time signature scheme

The Goldwasser, Micali, and Rivest (GMR) scheme (Algorithm 11.102) is a one-time signature scheme which requires a pair of claw-free permutations (see Definition 11.98). When combined with a tree authentication procedure, it provides a mechanism for signing more than one message. The GMR scheme is noteworthy as it was the first digital signature mechanism proven to be secure against an adaptive chosen-message attack. Although the GMR scheme is not practical, variations of it have been proposed which suggest that the concept is not purely of theoretical importance.

11.98 Definition Let $g_i\colon X \longrightarrow X$, $i = 0, 1$, be two permutations defined on a finite set X. g_0 and g_1 are said to be a *claw-free pair* of permutations if it is computationally infeasible to find $x, y \in X$ such that $g_0(x) = g_1(y)$. A triple (x, y, z) of elements from X with $g_0(x) = g_1(y) = z$ is called a *claw*. If both g_i, $i = 0, 1$, have the property that given additional information it is computationally feasible to determine g_0^{-1}, g_1^{-1}, respectively, the permutations are called a *trapdoor claw-free pair* of permutations.

In order for g_0, g_1 to be a claw-free pair, computing $g_i^{-1}(x)$, for both $i = 0$ and 1, must be computationally infeasible for essentially all $x \in X$. For, if g_1^{-1} (and similarly for g_0^{-1}) could be efficiently computed, one could select an $x \in X$, compute $g_0(x) = z$ and $g_1^{-1}(z) = y$, to obtain a claw (x, y, z).

11.99 Example (*trapdoor claw-free permutation pair*) Let $n = pq$ where $p \equiv 3 \pmod 4$ and $q \equiv 7 \pmod 8$. For this choice of p and q, $\left(\frac{-1}{n}\right) = 1$ but $-1 \notin Q_n$, and $\left(\frac{2}{n}\right) = -1$. Here, $\left(\frac{\cdot}{n}\right)$ denotes the Jacobi symbol (Definition 2.147). Define $D_n = \{x\colon \left(\frac{x}{n}\right) = 1 \text{ and } 0 < x < \frac{n}{2}\}$. Define $g_0\colon D_n \longrightarrow D_n$ and $g_1\colon D_n \longrightarrow D_n$ by

$$g_0(x) = \begin{cases} x^2 \bmod n, & \text{if } x^2 \bmod n < \frac{n}{2}, \\ -x^2 \bmod n, & \text{if } x^2 \bmod n > \frac{n}{2}, \end{cases}$$

$$g_1(x) = \begin{cases} 4x^2 \bmod n, & \text{if } 4x^2 \bmod n < \frac{n}{2}, \\ -4x^2 \bmod n, & \text{if } 4x^2 \bmod n > \frac{n}{2}. \end{cases}$$

If factoring n is intractable, then g_0, g_1 form a trapdoor claw-free pair of permutations; this can be seen as follows.

(i) (*g_0 and g_1 are permutations on D_n*) If $g_0(x) = g_0(y)$, then $x^2 \equiv y^2 \pmod n$ ($x^2 \equiv -y^2 \pmod n$ is not possible since $-1 \notin Q_n$), whence $x \equiv \pm y \pmod n$. Since $0 < x, y < n/2$, then $x = y$, and hence g_0 is a permutation on D_n. A similar argument shows that g_1 is a permutation on D_n.

(ii) (*g_0 and g_1 are claw-free*) Suppose that there is an efficient method for finding $x, y \in D_n$ such that $g_0(x) = g_1(y)$. Then $x^2 \equiv 4y^2 \pmod n$ ($x^2 \equiv -4y^2 \pmod n$ is impossible since $-1 \notin Q_n$), whence $(x-2y)(x+2y) \equiv 0 \pmod{n}$. Since $\left(\frac{x}{n}\right) = 1$ and $\left(\frac{\pm 2y}{n}\right) = -1$, $x \not\equiv \pm 2y \pmod{n}$ and, hence, $\gcd(x-2y, n)$ yields a non-trivial factor of n. This contradicts the assumption that factoring n is intractable.

(iii) (g_0, g_1 *is a trapdoor claw-free pair*) Knowing the factorization of n permits one to compute g_0^{-1} and g_1^{-1}. Hence, g_0, g_1 is a trapdoor claw-free permutation pair. □

The following example illustrates the general construction given in Example 11.99.

11.100 Example (*pair of claw-free permutations for artificially small parameters*) Let $p = 11$, $q = 7$, and $n = pq = 77$. $D_{77} = \{x\colon \left(\frac{x}{n}\right) = 1 \text{ and } 0 < x < 38\} = \{1, 4, 6, 9, 10, 13, 15, 16, 17, 19, 23, 24, 25, 36, 37\}$. The following table describes g_0 and g_1.

x	1	4	6	9	10	13	15	16	17	19	23	24	25	36	37
$g_0(x)$	1	16	36	4	23	15	6	25	19	24	10	37	9	13	17
$g_1(x)$	4	13	10	16	15	17	24	23	1	19	37	6	36	25	9

Notice that g_0 and g_1 are permutations on D_{77}. □

11.101 Algorithm Key generation for the GMR one-time signature scheme

SUMMARY: each entity selects a pair of trapdoor claw-free permutations and a validation parameter.
Each entity A should do the following:

1. Select a pair g_0, g_1 of trapdoor claw-free permutations on some set X. (It is "trapdoor" in that A itself can compute g_0^{-1} and g_1^{-1}.)
2. Select a random element $r \in X$. (r is called a *validation parameter*.)
3. A's public key is (g_0, g_1, r); A's private key is (g_0^{-1}, g_1^{-1}).

In the following, the notation for the composition of functions g_0, g_1 usually denoted $g_0 \circ g_1$ (see Definition 1.33) is simplified to $g_0 g_1$. Also, $(g_0 g_1)(r)$ will be written as $g_0 g_1(r)$. The signing space $\mathcal{M}_S$ consists of binary strings which are prefix-free (see Note 11.103).

11.102 Algorithm GMR one-time signature generation and verification

SUMMARY: A signs a binary string $m = m_1 m_2 \cdots m_t$. B verifies using A's public key.

1. *Signature generation.* Entity A should do the following:
   (a) Compute $S_r(m) = \prod_{i=0}^{t-1} g_{m_{t-i}}^{-1}(r)$.
   (b) A's signature for m is $S_r(m)$.
2. *Verification.* To verify A's signature $S_r(m)$ on m, B should do the following:
   (a) Obtain A's authentic public key (g_0, g_1, r).
   (b) Compute $r' = \prod_{i=1}^{t} g_{m_i}(S_r(m))$.
   (c) Accept the signature if and only if $r' = r$.

Proof that signature verification works.

$$\begin{aligned} r' = \prod_{i=1}^{t} g_{m_i}(S_r(m)) &= \prod_{i=1}^{t} g_{m_i} \prod_{j=0}^{t-1} g_{m_{t-j}}^{-1}(r) \\ &= g_{m_1} \circ g_{m_2} \circ \cdots \circ g_{m_t} \circ g_{m_t}^{-1} \circ g_{m_{t-1}}^{-1} \circ \cdots \circ g_{m_1}^{-1}(r) = r. \end{aligned}$$

Thus $r' = r$, as required.

11.103 Note (*message encoding and security*) The set of messages which can be signed using Algorithm 11.102 must come from a set of binary strings which are *prefix-free*. (For example, 101 and 10111 cannot be in the same space since 101 is a prefix of 10111.) One method to accomplish this is to encode a binary string $b_1b_2\cdots b_l$ as $b_1b_1b_2b_2\cdots b_lb_l01$. To see why the prefix-free requirement is necessary, suppose $m = m_1m_2\cdots m_t$ is a message whose signature is $S_r(m) = \prod_{i=0}^{t-1} g_{m_{t-i}}^{-1}(r)$. If $m' = m_1m_2\cdots m_u$, $u < t$, then an adversary can easily find a valid signature for m' from $S_r(m)$ by computing

$$S_r(m') = \prod_{j=u+1}^{t} g_{m_j}(S_r(m)) = \prod_{i=0}^{u-1} g_{m_{u-i}}^{-1}(r).$$

11.104 Note (*one-timeness of Algorithm 11.102*) To see that the GMR signature scheme is a one-time scheme, suppose that two prefix-free messages $m = m_1m_2\cdots m_t$ and $m' = n_1n_2\cdots n_u$ are both signed with the same validation parameter r. Then $S_r(m) = \prod_{i=0}^{t-1} g_{m_{t-i}}^{-1}(r)$ and $S_r(m') = \prod_{i=0}^{u-1} g_{n_{u-i}}^{-1}(r)$. Therefore, $\prod_{i=1}^{t} g_{m_i}(S_r(m)) = r = \prod_{i=1}^{u} g_{n_i}(S_r(m'))$. Since the message space is prefix-free, there is a smallest index $h \geq 1$ for which $m_h \neq n_h$. Since each g_j is a bijection, it follows that

$$\prod_{i=h}^{t} g_{m_i}(S_r(m)) = \prod_{i=h}^{u} g_{n_i}(S_r(m'))$$

or

$$g_{m_h}\prod_{i=h+1}^{t} g_{m_i}(S_r(m)) = g_{n_h}\prod_{i=h+1}^{u} g_{n_i}(S_r(m')).$$

Taking $x = \prod_{i=h+1}^{t} g_{m_i}(S_r(m))$, and $y = \prod_{i=h+1}^{u} g_{n_i}(S_r(m'))$, the adversary has a claw $(x, y, g_{m_h}(x))$. This violates the basic premise that it is computationally infeasible to find a claw. It should be noted that this does not necessarily mean that a signature for a new message can be forged. In the particular case given in Example 11.99, finding a claw factors the modulus n and permits anyone to sign an unlimited number of new messages (i.e., a total break of the system is possible).

11.105 Example (*GMR with artificially small parameters.*)
Key generation. Let n, p, q, g_0, g_1 be those given in Example 11.100. A selects the validation parameter $r = 15 \in D_{77}$.
Signature generation. Let $m = 1011000011$ be the message to be signed. Then

$$S_r(m) = g_1^{-1} \circ g_1^{-1} \circ g_0^{-1} \circ g_0^{-1} \circ g_0^{-1} \circ g_0^{-1} \circ g_1^{-1} \circ g_1^{-1} \circ g_0^{-1} \circ g_1^{-1}(15) = 23.$$

A's signature for message m is 23.
Signature verification. To verify the signature, B computes

$$r' = g_1 \circ g_0 \circ g_1 \circ g_1 \circ g_0 \circ g_0 \circ g_0 \circ g_0 \circ g_1 \circ g_1(23) = 15.$$

Since $r = r'$, B accepts the signature. □

GMR scheme with authentication trees

In order to sign multiple messages using the GMR one-time signature scheme (Algorithm 11.102), authentication trees (see §13.4.1) are required. Although conceptually similar to the method described in §11.6.3, only the leaves are used to produce the signature. Before giving details, an overview and some additional notation are necessary.

11.106 Definition A *full binary tree* with k *levels* is a binary tree which has $2^{k+1}-1$ vertices and 2^k leaves. The leaves are said to be at level k of the tree.

Let T be a full binary tree with k levels. Select public parameters $Y_1, Y_2, \ldots, Y_n$ where $n = 2^k$. Form an authentication tree T^* from T with root label R (see below). R is certified by a TTP and placed in a publicly available file. T^* can now be used to authenticate any of the Y_i by providing the authentication path values associated with the authentication path for Y_i. Each Y_i can now be used as the public parameter r for the GMR scheme. The details for constructing the authentication tree T^* now follow.

The tree T^* is constructed recursively. For the root vertex, select a value r and two t-bit binary strings r_L and r_R. Sign the string $r_L \| r_R$ with the GMR scheme using the public value r. The label for the root consists of the values r, r_L, r_R, and $S_r(r_L \| r_R)$. To authenticate the children of the root vertex, select t-bit binary strings b_{0L}, b_{1L}, b_{0R}, and b_{1R}. The label for the left child of the root is the set of values r_L, b_{0L}, b_{1L}, $S_{r_L}(b_{0L} \| b_{1L})$ and the label for the right child is r_R, b_{0R}, b_{1R}, $S_{r_R}(b_{0R} \| b_{1R})$. Using the strings b_{0L}, b_{1L}, b_{0R}, and b_{1R} as public values for the signing mechanism, one can construct labels for the children of the children of the root. Continuing in this manner, each vertex of T^* can be labeled. The method is illustrated in Figure 11.8.

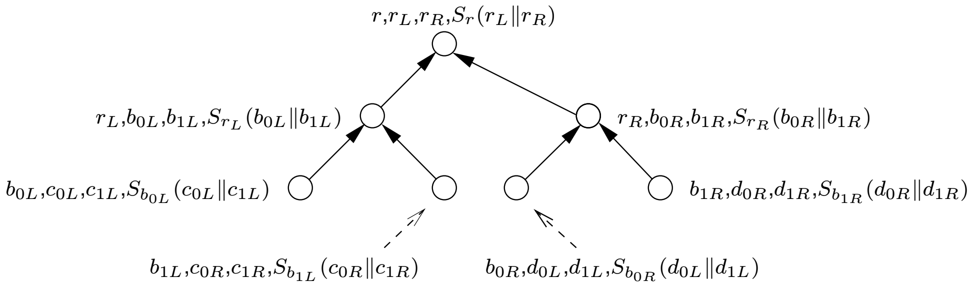


Figure 11.8: *A full binary authentication tree of level 2 for the GMR scheme.*

Each leaf of the authentication tree T^* can be used to sign a different binary message m. The signing procedure uses a pair of claw-free permutations g_0, g_1. If m is the binary message to be signed, and x is the public parameter in the label of a leaf which has not been used to sign any other message, then the signature for m consists of both $S_x(m)$ and the authentication path labels.

11.7 Other signature schemes

The signature schemes described in this section do not fall naturally into the general settings of §11.3 (RSA and related signature schemes), §11.4 (Fiat-Shamir signature schemes), §11.5 (DSA and related signature schemes), or §11.6 (one-time digital signatures).

11.7.1 Arbitrated digital signatures

11.107 Definition An *arbitrated digital signature scheme* is a digital signature mechanism requiring an unconditionally trusted third party (TTP) as part of the signature generation and verification.

Algorithm 11.109 requires a symmetric-key encryption algorithm $E = \{E_k : k \in \mathcal{K}\}$ where $\mathcal{K}$ is the key space. Assume that the inputs and outputs of each E_k are l-bit strings, and let $h : \{0,1\}^* \longrightarrow \{0,1\}^l$ be a one-way hash function. The TTP selects a key $k_T \in \mathcal{K}$ which it keeps secret. In order to verify a signature, an entity must share a symmetric key with the TTP.

11.108 Algorithm Key generation for arbitrated signatures

SUMMARY: each entity selects a key and transports it secretly with authenticity to the TTP. Each entity A should do the following:

1. Select a random secret key $k_A \in \mathcal{K}$.
2. Secretly and by some authentic means, make k_A available to the TTP.

11.109 Algorithm Signature generation and verification for arbitrated signatures

SUMMARY: entity A generates signatures using E_{k_A}. Any entity B can verify A's signature with the cooperation of the TTP.

1. *Signature generation.* To sign a message m, entity A should do the following:
   (a) A computes $H = h(m)$.
   (b) A encrypts H with E to get $u = E_{k_A}(H)$.
   (c) A sends u along with some identification string I_A to the TTP.
   (d) The TTP computes $E^{-1}_{k_A}(u)$ to get H.
   (e) The TTP computes $s = E_{k_T}(H||I_A)$ and sends s to A.
   (f) A's signature for m is s.
2. *Verification.* Any entity B can verify A's signature s on m by doing the following:
   (a) B computes $v = E_{k_B}(s)$.
   (b) B sends v and some identification string I_B to the TTP.
   (c) The TTP computes $E^{-1}_{k_B}(v)$ to get s.
   (d) The TTP computes $E^{-1}_{k_T}(s)$ to get $H\|I_A$.
   (e) The TTP computes $w = E_{k_B}(H\|I_A)$ and sends w to B.
   (f) B computes $E^{-1}_{k_B}(w)$ to get $H\|I_A$.
   (g) B computes $H' = h(m)$ from m.
   (h) B accepts the signature if and only if $H' = H$.

11.110 Note (*security of arbitrated signature scheme*) The security of Algorithm 11.109 is based on the symmetric-key encryption scheme chosen and the ability to distribute keys to participants in an authentic manner. §13.3 discusses techniques for distributing confidential keys.

11.111 Note (*performance characteristics of arbitrated signatures*) Since symmetric-key algorithms are typically much faster than public-key techniques, signature generation and verification by Algorithm 11.109 are (relatively) very efficient. A drawback is that interaction with the TTP is required, which places a much higher burden on the TTP and requires additional message exchanges between entities and the TTP.

11.7.2 ESIGN

ESIGN (an abbreviation for Efficient digital SIGNature) is another digital signature scheme whose security relies on the difficulty of factoring integers. It is a signature scheme with appendix and requires a one-way hash function $h\colon \{0,1\}^* \longrightarrow \mathbb{Z}_n$.

11.112 Algorithm Key generation for ESIGN

SUMMARY: each entity creates a public key and corresponding private key.
Each entity A should do the following:

1. Select random primes p and q such that $p \geq q$ and p, q are roughly of the same bitlength.
2. Compute $n = p^2q$.
3. Select a positive integer $k \geq 4$.
4. A's public key is (n, k); A's private key is (p, q).

11.113 Algorithm ESIGN signature generation and verification

SUMMARY: the signing algorithm computes an integer s such that $s^k \bmod n$ lies in a certain interval determined by the message. Verification demonstrates that $s^k \bmod n$ does indeed lie in the specified interval.

1. *Signature generation.* To sign a message m which is a bitstring of arbitrary length, entity A should do the following:
   (a) Compute $v = h(m)$.
   (b) Select a random secret integer x, $0 \leq x < pq$.
   (c) Compute $w = \lceil ((v - x^k) \bmod n)/(pq) \rceil$ and $y = w \cdot (kx^{k-1})^{-1} \bmod p$.
   (d) Compute $s = x + ypq \bmod n$.
   (e) A's signature for m is s.
2. *Verification.* To verify A's signature s on m, B should do the following:
   (a) Obtain A's authentic public key (n, k).
   (b) Compute $u = s^k \bmod n$ and $z = h(m)$.
   (c) If $z \leq u \leq z + 2^{\lceil \frac{2}{3} \lg n \rceil}$, accept the signature; else reject it.

Proof that signature verification works. Note that $s^k \equiv (x+ypq)^k \equiv \sum_{i=0}^{k} \binom{k}{i} x^{k-i}(ypq)^i \equiv x^k + kypqx^{k-1} \pmod{n}$. But $kx^{k-1}y \equiv w \pmod{p}$ and, thus, $kx^{k-1}y = w + lp$ for some $l \in \mathbb{Z}$. Hence, $s^k \equiv x^k + pq(w + lp) \equiv x^k + pqw \equiv x^k + pq \left\lceil \frac{(h(m)-x^k) \bmod n}{pq} \right\rceil \equiv x^k + pq \left(\frac{h(m)-x^k+jn+\epsilon}{pq} \right) \pmod{n}$, where $\epsilon = (x^k - h(m)) \bmod pq$. Therefore, $s^k \equiv x^k + h(m) - x^k + \epsilon \equiv h(m) + \epsilon \pmod{n}$. Since $0 \leq \epsilon < pq$, it follows that $h(m) \leq s^k \bmod n \leq h(m) + pq \leq h(m) + 2^{\lceil \frac{2}{3} \lg n \rceil}$, as required.

11.114 Example (*ESIGN for artificially small parameters*) In Algorithm 11.113, take messages to be integers m, $0 \leq m < n$, and the hash function h to be $h(m) = m$.

Key generation. A selects primes $p = 17389$ and $q = 15401$, $k = 4$, and computes $n = p^2q = 4656913120721$. A's public key is $(n = 4656913120721, k = 4)$; A's private key is $(p = 17389, q = 15401)$.

Signature generation. To sign the message $m = 3111527988477$, A computes $v = h(m) = 3111527988477$, and selects $x = 14222$ such that $0 \leq x < pq$. A then computes $w = \lceil((v - x^k) \bmod n)/(pq)\rceil = \lceil 2848181921806/267807989\rceil = \lceil 10635.16414\rceil = 10636$ and $y = w(kx^{k-1})^{-1} \bmod p = 10636(4 \times 14222^3)^{-1} \bmod 17389 = 9567$. Finally, A computes the signature $s = x + ypq \bmod n = 2562119044985$.

Signature verification. B obtains A's public key $(n = 4656913120721, k = 4)$, and computes $u = s^k \bmod n = 3111751837675$. Since $3111527988477 \leq 3111751837675 \leq 3111527988477 + 2^{29}$, B accepts the signature (here, $\lceil\frac{2}{3}\lg n\rceil = 29$). □

11.115 Note (*security of ESIGN*)

(i) The modulus $n = p^2q$ in Algorithm 11.113 differs from an RSA modulus by having a repeated factor of p. It is unknown whether or not moduli of this form are easier to factor than integers which are simply the product of two distinct primes.

(ii) Given a valid signature s for a message m, an adversary could forge a signature for a message m' if $h(m')$ is such that $h(m') \leq u \leq h(m') + 2^{\lceil\frac{2}{3}\lg n\rceil}$ (where $u = s^k \bmod n$). If an m' with this property is found, then s will be a signature for it. This will occur if $h(m)$ and $h(m')$ agree in the high-order $(\lg n)/3$ bits. Assuming that h behaves like a random function, one would expect to try $2^{(\lg n)/3}$ different values of m' before observing this.

(iii) Another possible approach to forging is to find a pair of messages m and m' such that $h(m)$ and $h(m')$ agree in the high-order $(\lg n)/3$ bits. By the birthday paradox (Fact 2.27(ii)), one can expect to find such a pair in $O(2^{(\lg n)/6})$ trials. If an adversary is able to get the legitimate signer to sign m, the same signature will be a signature for m'.

(iv) For the size of the integer n necessary to make the factorization of n infeasible, (ii) and (iii) above are extremely unlikely possibilities.

11.116 Note (*performance characteristics of ESIGN signatures*) Signature generation in Algorithm 11.113 is very efficient. For small values of k (e.g., $k = 4$), the most computationally intensive part is the modular inverse required in step 1c. Depending on the implementation, this corresponds to a small number of modular multiplications with modulus p. For $k = 4$ and a 768-bit modulus n, ESIGN signature generation may be between one and two orders of magnitude (10 to 100 times) faster than RSA signature generation with an equivalent modulus size. Signature verification is also very efficient and is comparable to RSA with a small public exponent.

11.8 Signatures with additional functionality

The mechanisms described in this section provide functionality beyond authentication and non-repudiation. In most instances, they combine a basic digital signature scheme (e.g., RSA) with a specific protocol to achieve additional features which the basic method does not provide.

11.8.1 Blind signature schemes

Rather than signature schemes as described in §11.2, *blind signature schemes* are two-party protocols between a *sender* A and a *signer* B. The basic idea is the following. A sends a piece of information to B which B signs and returns to A. From this signature, A can compute B's signature on an a priori message m of A's choice. At the completion of the protocol, B knows neither the message m nor the signature associated with it.

The purpose of a blind signature is to prevent the signer B from observing the message it signs and the signature; hence, it is later unable to associate the signed message with the sender A.

11.117 Example (*applications of blind signatures*) Blind signature schemes have applications where the sender A (the customer) does not want the signer B (the bank) to be capable of associating a postiori a message m and a signature $S_B(m)$ to a specific instance of the protocol. This may be important in electronic cash applications where a message m might represent a monetary value which A can spend. When m and $S_B(m)$ are presented to B for payment, B is unable to deduce which party was originally given the signed value. This allows A to remain anonymous so that spending patterns cannot be monitored. □

A blind signature protocol requires the following components:

1. A digital signature mechanism for signer B. $S_B(x)$ denotes the signature of B on x.
2. Functions f and g (known only to the sender) such that $g(S_B(f(m))) = S_B(m)$. f is called a *blinding function*, g an *unblinding function*, and $f(m)$ a *blinded message*.

Property 2 places many restrictions on the choice of S_B and g.

11.118 Example (*blinding function based on RSA*) Let $n = pq$ be the product of two large random primes. The signing algorithm S_B for entity B is the RSA signature scheme (Algorithm 11.19) with public key (n, e) and private key d. Let k be some fixed integer with $\gcd(n, k) = 1$. The blinding function $f\colon \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ is defined by $f(m) = m \cdot k^e \bmod n$ and the unblinding function $g\colon \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ by $g(m) = k^{-1}m \bmod n$. For this choice of f, g, and S_B, $g(S_B(f(m))) = g(S_B(mk^e \bmod n)) = g(m^d k \bmod n) = m^d \bmod n = S_B(m)$, as required by property 2. □

Protocol 11.119 presents a blind signature scheme which uses the digital signature mechanism and functions f and g described in Example 11.118.

11.119 Protocol Chaum's blind signature protocol

SUMMARY: sender A receives a signature of B on a blinded message. From this, A computes B's signature on a message m chosen a priori by A, $0 \le m \le n-1$. B has no knowledge of m nor the signature associated with m.

1. *Notation.* B's RSA public and private keys are (n, e) and d, respectively. k is a random secret integer chosen by A satisfying $0 \le k \le n-1$ and $\gcd(n, k) = 1$.
2. *Protocol actions.*
   (a) (*blinding*) A computes $m^* = mk^e \bmod n$ and sends this to B.
   (b) (*signing*) B computes $s^* = (m^*)^d \bmod n$ which it sends to A.
   (c) (*unblinding*) A computes $s = k^{-1}s^* \bmod n$, which is B's signature on m.

11.8.2 Undeniable signature schemes

Undeniable signature schemes are distinct from digital signatures in the sense of §11.2 in that the signature verification protocol requires the cooperation of the signer. The following example describes two scenarios where an undeniable signature could be applied.

11.120 Example (*scenarios for undeniable signatures*)

(i) Entity A (the customer) wishes to gain access to a secured area controlled by entity B (the bank). The secured area might, for example, be a safety-deposit box room. B requires A to sign a time and date document before access is granted. If A uses an undeniable signature, then B is unable to prove (at some later date) to anyone that A used the facility without A's direct involvement in the signature verification process.

(ii) Suppose some large corporation A creates a software package. A signs the package and sells it to entity B, who decides to make copies of this package and resell it to a third party C. C is unable to verify the authenticity of the software without the cooperation of A. Of course, this scenario does not prevent B from re-signing the package with its own signature but the marketing advantage associated with corporation A's name is lost to B. It will also be easier to trace the fraudulent activity of B. □

11.121 Algorithm Key generation for Algorithm 11.122

SUMMARY: each entity selects a private key and corresponding public key.
Each entity A should do the following:

1. Select a random prime $p = 2q + 1$ where q is also a prime.
2. (Select a generator α for the subgroup of order q in $\mathbb{Z}_p^*$.)
   2.1 Select a random element $\beta \in \mathbb{Z}_p^*$ and compute $\alpha = \beta^{(p-1)/q} \bmod p$.
   2.2 If $\alpha = 1$ then go to step 2.1.
3. Select a random integer $a \in \{1, 2, \ldots, q-1\}$ and compute $y = \alpha^a \bmod p$.
4. A's public key is (p, α, y); A's private key is a.

11.122 Algorithm Chaum-van Antwerpen undeniable signature scheme

SUMMARY: A signs a message m belonging to the subgroup of order q in $\mathbb{Z}_p^*$. Any entity B can verify this signature with the cooperation of A.

1. *Signature generation.* Entity A should do the following:
   (a) Compute $s = m^a \bmod p$.
   (b) A's signature on message m is s.
2. *Verification.* The protocol for B to verify A's signature s on m is the following:
   (a) B obtains A's authentic public key (p, α, y).
   (b) B selects random secret integers $x_1, x_2 \in \{1, 2, \ldots, q-1\}$.
   (c) B computes $z = s^{x_1} y^{x_2} \bmod p$ and sends z to A.
   (d) A computes $w = (z)^{a^{-1}} \bmod p$ (where $aa^{-1} \equiv 1 \pmod q$) and sends w to B.
   (e) B computes $w' = m^{x_1} \alpha^{x_2} \bmod p$ and accepts the signature if and only if $w = w'$.

Proof that signature verification works.

$$w \equiv (z)^{a^{-1}} \equiv (s^{x_1}y^{x_2})^{a^{-1}} \equiv (m^{ax_1}\alpha^{ax_2})^{a^{-1}} \equiv m^{x_1}\alpha^{x_2} \equiv w' \bmod p,$$

as required.

Fact 11.123 states that, with high probability, an adversary is unable to cause B to accept a fraudulent signature.

11.123 Fact (*detecting forgeries of undeniable signatures*) Suppose that s is a forgery of A's signature for a message m, i.e., $s \neq m^a \bmod p$. Then the probability of B accepting the signature in Algorithm 11.122 is only $1/q$; this probability is independent of the adversary's computational resources.

11.124 Note (*disavowing signatures*) The signer A could attempt to disavow a (valid) signature constructed by Algorithm 11.122 in one of three ways:

(i) refuse to participate in the verification protocol of Algorithm 11.122;
(ii) perform the verification protocol incorrectly; or
(iii) claim a signature a forgery even though the verification protocol is successful.

Disavowing a signature by following (i) would be considered as an obvious attempt at (wrongful) repudiation. (ii) and (iii) are more difficult to guard against, and require a disavowal protocol (Protocol 11.125).

Protocol 11.125 essentially applies the verification protocol of Algorithm 11.122 twice and then performs a check to verify that A has performed the protocol correctly.

11.125 Protocol Disavowal protocol for Chaum-van Antwerpen undeniable signature scheme

SUMMARY: this protocol determines whether the signer A is attempting to disavow a valid signature s using Algorithm 11.122, or whether the signature is a forgery.

1. B obtains A's authentic public key (p, α, y).
2. B selects random secret integers $x_1, x_2 \in \{1, 2, \dots, q-1\}$, and computes $z = s^{x_1}y^{x_2} \bmod p$, and sends z to A.
3. A computes $w = (z)^{a^{-1}} \bmod p$ (where $aa^{-1} \equiv 1 \pmod q$) and sends w to B.
4. If $w = m^{x_1}\alpha^{x_2} \bmod p$, B accepts the signature s and the protocol halts.
5. B selects random secret integers $x'_1, x'_2 \in \{1, 2, \dots, q-1\}$, and computes $z' = s^{x'_1}y^{x'_2} \bmod p$, and sends z' to A.
6. A computes $w' = (z')^{a^{-1}} \bmod p$ and sends w' to B.
7. If $w' = m^{x'_1}\alpha^{x'_2} \bmod p$, B accepts the signature s and the protocol halts.
8. B computes $c = (w\alpha^{-x_2})^{x'_1} \bmod p$ and $c' = (w'\alpha^{-x'_2})^{x_1} \bmod p$. If $c = c'$, then B concludes that s is a forgery; otherwise, B concludes that the signature is valid and A is attempting to disavow the signature s.

Fact 11.126 states that Protocol 11.125 achieves its desired objectives.

11.126 Fact Let m be a message and suppose that s is A's (purported) signature on m.

(i) If s is a forgery, i.e., $s \neq m^a \bmod p$, and if A and B follow Protocol 11.125 correctly, then $w = w'$ (and hence, B's conclusion that s is a forgery is correct).
(ii) Suppose that s is indeed A's signature for m, i.e., $s = m^a \bmod p$. Suppose that B follows Protocol 11.125 correctly, but that A does not. Then the probability that $w = w'$ (and hence A succeeds in disavowing the signature) is only $1/q$.

11.127 Note (*security of undeniable signatures*)

(i) The security of Algorithm 11.122 is dependent on the intractability of the discrete logarithm problem in the cyclic subgroup of order q in $\mathbb{Z}_p^*$ (see §3.6.6).

(ii) Suppose verifier B records the messages exchanged in step 2 of Algorithm 11.122, and also the random values x_1, x_2 used in the protocol. A third party C should never accept this transcript from B as a verification of signature s. To see why this is the case, it suffices to show how B could contrive a successful transcript of step 2 of Algorithm 11.122 without the signer A's participation. B chooses a message m, integers x_1, x_2 and l in the interval $[1, q-1]$, and computes $s = ((m^{x_1}\alpha^{x_2})^{l^{-1}}y^{-x_2})^{x_1^{-1}} \bmod p$. The protocol message from B to A would be $z = s^{x_1}y^{x_2} \bmod p$, and from A to B would be $w = z^l \bmod p$. Algorithm 11.122 will accept s as a valid signature of A for message m. This argument demonstrates that signatures can only be verified by interacting directly with the signer.

11.8.3 Fail-stop signature schemes

Fail-stop digital signatures are digital signatures which permit an entity A to prove that a signature purportedly (but not actually) signed by A is a forgery. This is done by showing that the underlying assumption on which the signature mechanism is based has been compromised. The ability to prove a forgery does not rely on any cryptographic assumption, but may fail with some small probability; this failure probability is independent of the computing power of the forger. Fail-stop signature schemes have the advantage that even if a very powerful adversary can forge a single signature, the forgery can be detected and the signing mechanism no longer used. Hence, the term *fail-then-stop* is also appropriate. A fail-stop signature scheme should have the following properties:

1. If a signer signs a message according to the mechanism, then a verifier upon checking the signature should accept it.
2. A forger cannot construct signatures that pass the verification algorithm without doing an exponential amount of work.
3. If a forger succeeds in constructing a signature which passes the verification test then, with high probability, the true signer can produce a proof of forgery.
4. A signer cannot construct signatures which are at some later time claimed to be forgeries.

Algorithm 11.130 is an example of a fail-stop mechanism. As described, it is a one-time signature scheme, but there are ways to generalize it to allow multiple signings; using authentication trees is one possibility (see §11.6.3). The proof-of-forgery algorithm is presented in Algorithm 11.134.

11.128 Algorithm Key generation for Algorithm 11.130

SUMMARY: key generation is divided between entity A and a trusted third party (TTP).

1. The TTP should do the following:
   (a) Select primes p and q such that q divides $(p-1)$ and the discrete logarithm problem in $\mathbb{Z}_q^*$ is intractable.
   (b) (Select a generator α for the cyclic subgroup G of $\mathbb{Z}_p^*$ having order q.)
      (i) Select a random element $g \in \mathbb{Z}_p^*$ and compute $\alpha = g^{(p-1)/q} \bmod p$.
      (ii) If $\alpha = 1$ then go to step (i).

(c) Select a random integer a, $1 \leq a \leq q-1$, and compute $\beta = \alpha^a \bmod p$. The integer a is kept secret by the TTP.
(d) Send (p, q, α, β) in the clear to entity A.

2. Entity A should do the following:
   (a) Select random secret integers x_1, x_2, y_1, y_2 in the interval $[0, q-1]$.
   (b) Compute $\beta_1 = \alpha^{x_1}\beta^{x_2}$ and $\beta_2 = \alpha^{y_1}\beta^{y_2} \bmod p$.
   (c) A's public key is $(\beta_1, \beta_2, p, q, \alpha, \beta)$; A's private key is the quadruple $\overline{x} = (x_1, x_2, y_1, y_2)$.

11.129 Note (*TTP's secret information*) Assuming that the discrete logarithm problem in the subgroup of order q in $\mathbb{Z}_p^*$ is intractable in Algorithm 11.128, the only entity which knows a, the discrete logarithm of β to the base α, is the TTP.

11.130 Algorithm Fail-stop signature scheme (van Heijst-Pedersen)

SUMMARY: this is a one-time digital signature scheme whose security is based on the discrete logarithm problem in the subgroup of order q in $\mathbb{Z}_p^*$.

1. *Signature generation.* To sign a message $m \in [0, q-1]$, A should do the following:
   (a) Compute $s_{1,m} = x_1 + my_1 \bmod q$ and $s_{2,m} = x_2 + my_2 \bmod q$.
   (b) A's signature for m is $(s_{1,m}, s_{2,m})$.
2. *Verification.* To verify A's signature $(s_{1,m}, s_{2,m})$ on m, B should do the following:
   (a) Obtain A's authentic public key $(\beta_1, \beta_2, p, q, \alpha, \beta)$.
   (b) Compute $v_1 = \beta_1\beta_2^m \bmod p$ and $v_2 = \alpha^{s_{1,m}}\beta^{s_{2,m}} \bmod p$.
   (c) Accept the signature if and only if $v_1 = v_2$.

Proof that signature verification works.

$$\begin{aligned} v_1 &\equiv \beta_1\beta_2^m \equiv (\alpha^{x_1}\beta^{x_2})(\alpha^{y_1}\beta^{y_2})^m \equiv \alpha^{x_1+my_1}\beta^{x_2+my_2} \\ &\equiv \alpha^{s_{1,m}}\beta^{s_{2,m}} \equiv v_2 \pmod{p}. \end{aligned}$$

Algorithm 11.130 is a one-time signature scheme since A's private key $\overline{x}$ can be computed if two messages are signed using $\overline{x}$. Before describing the algorithm for proof of forgery (Algorithm 11.134), a number of facts are needed. These are given in Fact 11.131 and illustrated in Example 11.132.

11.131 Fact (*number of distinct quadruples representing a public key and a signature*) Suppose that A's public key in Algorithm 11.130 is $(\beta_1, \beta_2, p, q, \alpha, \beta)$ and private key is the quadruple $\overline{x} = (x_1, x_2, y_1, y_2)$.

(i) There are exactly q^2 quadruples $\overline{x}' = (x_1', x_2', y_1', y_2')$ with $x_1', x_2', y_1', y_2' \in \mathbb{Z}_q$ which yield the same portion (β_1, β_2) of the public key.
(ii) Let T be the set of q^2 quadruples which yield the same portion of the public key (β_1, β_2). For each $m \in \mathbb{Z}_q$, there are exactly q quadruples in T which give the same signature $(s_{1,m}, s_{2,m})$ for m (where a signature is as described in Algorithm 11.130). Hence, the q^2 quadruples in T give exactly q different signatures for m.
(iii) Let $m' \in \mathbb{Z}_q$ be a message different from m. Then the q quadruples in T which yield A's signature $(s_{1,m}, s_{2,m})$ for m, yield q different signatures for m'.

11.132 Example (*illustration of Fact 11.131*) Let $p = 29$ and $q = 7$. $\alpha = 16$ is a generator of the subgroup of order q in $\mathbb{Z}_p^*$. Take $\beta = \alpha^5 \bmod 29 = 23$. Suppose A's private key is $\overline{x} = (2,3,5,2)$; A's public key is $\beta_1 = \alpha^2\beta^3 \bmod 29 = 7$, $\beta_2 = \alpha^5\beta^2 \bmod 29 = 16$. The following table lists the $q^2 = 49$ quadruples which give the same public key.

1603	2303	3003	4403	5103	6503	0203
1610	2310	3010	4410	5110	6510	0210
1624	2324	3024	4424	5124	6524	0224
1631	2331	3031	4431	5131	6531	0231
1645	2345	3045	4445	5145	6545	0245
1652	2352	3052	4452	5152	6552	0252
1666	2366	3066	4466	5166	6566	0266

If the 49 quadruples of this table are used to sign the message $m = 1$, exactly $q = 7$ signature pairs $(s_{1,m}, s_{2,m})$ arise. The next table lists the possibilities and those quadruples which generate each signature.

signature pair	$(2,6)$	$(3,3)$	$(4,0)$	$(5,4)$	$(6,1)$	$(0,5)$	$(1,2)$
quadruples	1610	1624	1631	1645	1652	1666	1603
	2303	2310	2324	2331	2345	2352	2366
	3066	3003	3010	3024	3031	3045	3052
	4452	4466	4403	4410	4424	4431	4445
	5145	5152	5166	5103	5110	5124	5131
	6531	6545	6552	6566	6503	6510	6524
	0224	0231	0245	0252	0266	0203	0210

The next table lists, for each message $m' \in \mathbb{Z}_7$, all signature pairs for the 7 quadruples which yield A's signature $(0,5)$ for $m = 1$.

	m'						
quadruple	0	1	2	3	4	5	6
1666	16	05	64	53	42	31	20
2352	23	05	50	32	14	66	41
3045	30	05	43	11	56	24	62
4431	44	05	36	60	21	52	13
5124	51	05	22	46	63	10	34
6510	65	05	15	25	35	45	55
0203	02	05	01	04	00	03	06

□

11.133 Note (*probability of successful forgery in Algorithm 11.130*) Suppose that an adversary (the forger) wishes to derive A's signature on some message m'. There are two possibilities to consider.

(i) The forger has access only to the signer's public key (i.e., the forger is not in possession of a message and valid signature). By Fact 11.131(ii), the probability that the signature created by the adversary is the same as A's signature for m' is only $q/q^2 = 1/q$; this probability is independent of the adversary's computational resources.

(ii) The forger has access to a message m and a signature $(s_{1,m}, s_{2,m})$ created by the signer. By Fact 11.131(iii), the probability that the signature created by the adversary is the same as A's signature for m' is only $1/q$; again, this probability is independent of the adversary's computational resources.

Suppose now that an adversary has forged A's signature on a message, and the signature passed the verification stage in Algorithm 11.130. The objective is that A should be able to prove that this signature is a forgery. The following algorithm shows how A can, with high probability, use the forged signature to derive the secret a. Since a was supposed to have been known only to the TTP (Note 11.129), it serves as proof of forgery.

11.134 Algorithm Proof-of-forgery algorithm for Algorithm 11.130

SUMMARY: to prove that a signature $s' = (s'_{1,m}, s'_{2,m})$ on a message m is a forgery, the signer derives the integer $a = \log_\alpha \beta$ which serves as proof of forgery.

The signer (entity A) should do the following:

1. Compute a signature pair $s = (s_{1,m}, s_{2,m})$ for message m using its private key $\overline{x}$ (see Algorithm 11.128).
2. If $s = s'$ return to step 1.
3. Compute $a = (s_{1,m} - s'_{1,m}) \cdot (s_{2,m} - s'_{2,m})^{-1} \bmod q$.

Proof that Algorithm 11.134 works. By Fact 11.131(iii), the probability that $s = s'$ in step 1 of Algorithm 11.134 is $1/q$. From the verification algorithm (Algorithm 11.130), $\alpha^{s_{1,m}}\beta^{s_{2,m}} \equiv \alpha^{s'_{1,m}}\beta^{s'_{2,m}} \pmod p$ or $\alpha^{s_{1,m}-s'_{1,m}} \equiv \alpha^{a(s'_{2,m}-s_{2,m})} \pmod p$ or $s_{1,m} - s'_{1,m} \equiv a(s'_{2,m} - s_{2,m}) \pmod q$. Hence, $a = (s_{1,m} - s'_{1,m}) \cdot (s_{2,m} - s'_{2,m})^{-1} \bmod q$.

11.135 Remark (*disavowing signatures*) In order for a signer to disavow a signature that it created with Algorithm 11.134, an efficient method for computing logarithms is required.

11.9 Notes and further references

§11.1

The concept of a digital signature was introduced in 1976 by Diffie and Hellman [344, 345]. Although the idea of a digital signature was clearly articulated, no practical realization emerged until the 1978 paper by Rivest, Shamir, and Adleman [1060]. Digital signatures appear to have been independently discovered by Merkle [849, 850] but not published until 1978. One of Merkle's contributions is discussed in §11.6.2. Other early research was due to Lamport [738], Rabin [1022, 1023], and Matyas [801].

A detailed survey on digital signatures is given by Mitchell, Piper, and Wild [882]. A thorough discussion of a selected subset of topics in the area is provided by Stinson [1178]. Other sources which provide a good overview are Meyer and Matyas [859], Goldwasser, Micali, and Rivest [484], Rivest [1054], and Schneier [1094].

§11.2

The original proposal for a digital signature scheme by Diffie and Hellman [344] considered only digital signatures with message recovery. The first discussion of digital signature schemes with appendix (although the term was not used per se) appears to be in the patent by Merkle and Hellman [553]. Davies and Price [308] and Denning [326] give brief introductions to digital signatures but restrict the discussion to digital signature schemes with message recovery and one-time digital signature schemes. Mitchell, Piper, and Wild [882] and Stinson [1178] give abstract definitions of digital signature schemes somewhat less general than those given in §11.2.

Excellent discussions on attacks against signature schemes are provided by Goldwasser, Micali, and Rivest [484] and Rivest [1054]. The former refers to the discovery of a functionally equivalent signing algorithm as *universal forgery*, and separates chosen-message attacks into *generic chosen-message attacks* and *directed chosen-message attacks*.

Many proposed digital signature schemes have been shown to be insecure. Among the most prominent of these are the Merkle-Hellman knapsack scheme proposed by Merkle and Hellman [857], shown to be totally breakable by Shamir [1114]; the Shamir fast signature scheme [1109], shown to be totally breakable by Odlyzko [939]; and the Ong-Schnorr-Shamir (OSS) scheme [958], shown to be totally breakable by Pollard (see Pollard and Schnorr [988]). Naccache [914] proposed a modification of the Ong-Schnorr-Shamir scheme to avoid the earlier attacks.

§11.3

The RSA signature scheme (Algorithm 11.19), discovered by Rivest, Shamir, and Adleman [1060], was the first practical signature scheme based on public-key techniques.

The multiplicative property of RSA (§11.3.2(ii)) was first exploited by Davida [302]. Denning [327] reports and expands on Davida's attack and credits Moore with a simplification. Gordon [515] uses the multiplicative property of RSA to show how to create public-key parameters and associated (forged) certificates if the signing authority does not take adequate precautions. The existential attack on RSA signatures having certain types of redundancy (Example 11.21) is due to de Jonge and Chaum [313]. Evertse and van Heijst [381] consider other types of attacks on RSA signatures which also rely on the multiplicative property.

The reblocking problem (§11.3.3(i)) is discussed by Davies and Price [308], who attribute the method of prescribing the form of the modulus to Guillou. An alternate way of constructing an (even) t-bit modulus $n = pq$ having a 1 in the high-order position followed by k 0's is the following. Construct an integer $u = 2^t + w2^{t/2}$ for some randomly selected $(t/2 - k)$-bit integer w. Select a $(t/2)$-bit prime p, and divide p into u to get a quotient q and a remainder r (i.e., $u = pq + r$). If q is a prime number, then $n = pq$ is an RSA modulus of the required type. For example, if $t = 14$ and $k = 3$, let $u = 2^{14} + w2^7$ where $w = 11$. If $p = 89$, then $q = 199$ and $n = pq = 17711$. The binary representation of n is 100010100101111.

The Rabin public-key signature scheme (Algorithm 11.25) is due to Rabin [1023]. Verification of signatures using the Rabin scheme is efficient since only one modular multiplication is required (cf. Note 11.33). Beller and Yacobi [101] take advantage of this aspect in their authenticated key transport protocol (see §12.5.3).

The modified-Rabin signature scheme (Algorithm 11.30) is derived from the RSA variant proposed by Williams [1246] (see also page 315). The purpose of the modification is to provide a deterministic procedure for signing. A similar methodology is incorporated in ISO/IEC 9796 (§11.3.5). The modified scheme can be generalized to other even public exponents besides $e = 2$. If $\gcd(e, (p-1)(q-1)/4) = 1$, then exponentiation by e is a permutation of Q_n.

ISO/IEC 9796 [596] became an international standard in October of 1991. This standard provides examples based on both the RSA and Rabin digital signature mechanisms. Although the standard permits the use of any digital signature scheme with message recovery which provides a t-bit signature for a $\lfloor \frac{t}{2} \rfloor$-bit message, the design was specifically tailored for the RSA and Rabin mechanisms. For design motivation, see Guillou et al. [525]. At the time of publication of ISO/IEC 9796, no other digital signature schemes providing message recovery were known, but since then several have been found; see Koyama et al. [708].

ISO/IEC 9796 is effective for signing messages which do not exceed a length determined by the signature process. Quisquater [1015] proposed a method for extending the utility of ISO/IEC 9796 to longer messages. Briefly, the modified scheme is as follows. Select a one-way hash function h which maps bitstrings of arbitrary length to k-bitstrings. If the signing capability of ISO/IEC 9796 is t bits and m is an n-bit message where $n > t$, then m is partitioned into two bitstrings m_c and m_s, where m_c is $(n-t+k)$ bits long. Compute $d = h(m)$ and form $m' = m_s \| d$; m' is a string of bitlength t. Sign m' using ISO/IEC 9796 to get J. The signature on message m is $m_c \| J$. This provides a randomized digital signature mechanism with message recovery, where the hash function provides the randomization.

§11.3.6 is from PKCS #1 [1072]. This document describes formatting for both encryption and digital signatures but only those details pertinent to digital signatures are mentioned here. The specification does not include message recovery as ISO/IEC 9796 does. It also does not specify the size of the primes, how they should be generated, nor the size of public and private keys. It is suggested that $e = 3$ or $e = 2^{16} + 1$ are widely used. The only attacks mentioned in PKCS #1 (which the formatting attempts to prevent) are those by den Boer and Bosselaers [324], and Desmedt and Odlyzko [341].

§11.4

The Feige-Fiat-Shamir digital signature scheme (Algorithm 11.40), proposed by Feige, Fiat, and Shamir [383], is a minor improvement of the Fiat-Shamir signature scheme [395], requiring less computation and providing a smaller signature. Fiat and Shamir [395] prove that their scheme is secure against existential forgery provided that factoring is intractable and that h is a truly random function. Feige, Fiat, and Shamir [383] prove that their modification has the same property.

Note 11.44 was suggested by Fiat and Shamir [395]. Note 11.45 is due to Micali and Shamir [868], who suggest that only the modulus n_A of entity A needs to be public if $v_1, v_2, \ldots, v_k$ are system-wide parameters. Since all entities have distinct moduli, it is unlikely that $v_j \in Q_n$, $1 \leq j \leq k$, for many different values of n. To overcome this problem, Micali and Shamir claim that some perturbation of k public values is possible to ensure that the resulting values are quadratic residues with respect to a particular modulus, but do not specify any method which provides the necessary perturbation.

The GQ signature scheme (Algorithm 11.48) is due to Guillou and Quisquater [524].

§11.5

The DSA (Algorithm 11.56) is due to Kravitz [711] and was proposed as a Federal Information Processing Standard in August of 1991 by the U.S. National Institute for Science and Technology. It became the Digital Signature Standard (DSS) in May 1994, as specified in FIPS 186 [406]. Smid and Branstad [1157] comment that the DSA was selected based on a number of important factors: the level of security provided, the applicability of patents, the ease of export from the U.S., the impact on national security and law enforcement, and the efficiency in a number of government and commercial applications. They provide a comparison of the computational efficiencies of DSA and RSA and address a number of negative responses received during the FIPS public comment period.

Naccache et al. [916] describe a number of techniques for improving the efficiency of the DSA. For example, the computation of $k^{-1} \bmod q$ in step 1c of Algorithm 11.56 can be replaced by the random generation of an integer b, the computation of $u = bk \bmod q$ and $s = b \cdot \{h(m) + ar\} \bmod q$. The signature is (r, s, u). The verifier computes $u^{-1} \bmod q$ and $u^{-1}s \bmod q = \widetilde{s}$. Verification of the signature $(r, \widetilde{s})$ now proceeds as in Algorithm 11.56. This variant might be useful for signature generation in chipcard applications where computing power is limited. Naccache et al. also propose the idea of *use and throw coupons* which eliminate the need to compute $r = (\alpha^k \bmod p) \bmod q$. Since this exponentiation is the most computationally intensive portion of DSA signature generation, use and throw coupons greatly improve efficiency. Coupons require storage, and only one signature can be created for each coupon. If storage is limited (as is often the case), only a fixed number of DSA signatures can be created with this method.

Béguin and Quisquater [82] show how to use an insecure server to aid in computations associated with DSA signature generation and verification. The method accelerates the computation of modular multiplication and exponentiation by using an untrusted auxiliary device to provide the majority of the computing. As such, it also applies to schemes other than DSA. Arazi [54] shows how to integrate a Diffie-Hellman key exchange into the DSA.

The ElGamal digital signature scheme (Algorithm 11.64) was proposed in 1984 by ElGamal [368]. ElGamal [368], Mitchell, Piper, and Wild [882], and Stinson [1178] comment further on its security.

Note 11.66(iv) is due to Bleichenbacher [153], as is Note 11.67(iii), which is a special case of the following more general result. Suppose p is a prime, α is a generator of $\mathbb{Z}_p^*$, and y is the public key of entity A for an instance of the ElGamal signature scheme. Suppose $p-1 = bq$ and logarithms in the subgroup of order b in $\mathbb{Z}_p^*$ can be efficiently computed. Finally, suppose that a generator $\beta = cq$ for some c, $0 < c < b$, and an integer t are known such that $\beta^t \equiv \alpha \pmod p$. For message m, the pair (r, s) with $r = \beta$ and $s = t \cdot \{h(m) - cqz\} \bmod (p-1)$ where z satisfies $\alpha^{qz} \equiv y^q \pmod p$ is a signature for message m which will be accepted by Algorithm 11.64. Bleichenbacher also describes how a trapdoor could be constructed for the ElGamal signature scheme when system-wide parameters p and α are selected by a fraudulent trusted third party.

Variations of the ElGamal signing equation described in §11.5.2 were proposed by ElGamal [366], Agnew, Mullin, and Vanstone [19], Kravitz [711], Schnorr [1098], and Yen and Laih [1259]. Nyberg and Rueppel [938] and, independently, Horster and Petersen [564], placed these variations in a much more general framework and compared their various properties.

ElGamal signatures based on elliptic curves over finite fields were first proposed by Koblitz [695] and independently by Miller [878] in 1985. A variation of the DSA based on elliptic curves and referred to as the ECDSA is currently being drafted for an IEEE standard.

The Schnorr signature scheme (Algorithm 11.78), due to Schnorr [1098], is derived from an identification protocol given in the same paper (see §10.4.4). Schnorr proposed a preprocessing method to improve the efficiency of the signature generation in Algorithm 11.78. Instead of generating a random integer k and computing $\alpha^k \bmod p$ for each signature, a small number of integers k_i and $\alpha^{k_i} \bmod p$, $1 \le i \le t$, are precomputed and stored, and subsequently combined and refreshed for each signature. De Rooij [315] showed that this preprocessing is insecure if t is small.

Brickell and McCurley [207] proposed a variant of the Schnorr scheme. Their method uses a prime p such that $p-1$ is hard to factor, a prime divisor q of $p-1$, and an element α of order q in $\mathbb{Z}_p^*$. The signing equation is $s = ae+k \bmod (p-1)$ as opposed to the Schnorr equation $s = ae+k \bmod q$. While computationally less efficient than Schnorr's, this variant has the advantage that its security is based on the difficulty of two hard problems: (i) computing logarithms in the cyclic subgroup of order q in $\mathbb{Z}_p^*$; and (ii) factoring $p-1$. If either of these problems is hard, then the problem of computing logarithms in $\mathbb{Z}_p^*$ is also hard.

Okamoto [949] describes a variant of the Schnorr scheme which he proves to be secure, provided that the discrete logarithm problem in $\mathbb{Z}_p^*$ is intractable and that correlation-free hash functions exist (no instance of a correlation-free hash function is yet known). Signature generation and verification are not significantly more computationally intensive than in the Schnorr scheme; however, the public key is larger.

The Nyberg-Rueppel scheme (Algorithm 11.81) is due to Nyberg and Rueppel [936]. For an extensive treatment including variants, see Nyberg and Rueppel [938]. They note that unlike RSA, this signature scheme cannot be used for encryption since the signing transformation S has a left inverse, namely, the verification transformation V, but S is not the left inverse of V; in other words, $V(S(m)) = m$ for all $m \in \mathbb{Z}_p$, but $S(V(m)) \neq m$ for most $m \in \mathbb{Z}_p$. The second paper also defines the notion of strong equivalence between signature schemes (two signature schemes are called *strongly equivalent* if the signature on a message m in one scheme can be transformed into the corresponding signature in the other scheme, without knowledge of the private key), and discusses how to modify DSA to provide message recovery.

Some digital signature schemes make it easy to conceal information in the signature which can only be recovered by entities privy to the concealment method. Information communicated this way is said to be *subliminal* and the conveying mechanism is called a *subliminal channel*. Among the papers on this subject are those of Simmons [1139, 1140, 1147, 1149]. Simmons [1139] shows that if a signature requires l_1 bits to convey and provides l_2 bits of security, then $l_1 - l_2$ bits are available for the subliminal channel. This does not imply that all $l_1 - l_2$ bits can, in fact, be used by the channel; this depends on the signature mechanism. If a large proportion of these bits are available, the subliminal channel is said to be *broadband*; otherwise, it is *narrowband*. Simmons [1149] points out that ElGamal-like signature schemes provide a broadband subliminal channel. For example, if the signing equation is $s = k^{-1} \cdot \{h(m) - ar\} \bmod (p-1)$ where a is the private key known to both the signer and the recipient of the signature, then k can be used to carry the subliminal message. This has the disadvantage that the signer must provide the recipient with the private key, allowing the recipient to sign messages that will be accepted as having originated with the signer. Simmons [1147] describes narrowband channels for the DSA.

§11.6

Rabin [1022] proposed the first one-time signature scheme (Algorithm 11.86) in 1978. Lamport [738] proposed a similar mechanism, popularized by Diffie and Hellman [347], which does not require interaction with the signer for verification. Diffie suggested the use of a one-way hash function to improve the efficiency of the method. For this reason, the mechanism is often referred to as the *Diffie-Lamport scheme*. Lamport [738] also describes a more efficient method for one-time digital signatures, which was rediscovered by Bos and Chaum [172]. Bos and Chaum provide more substantial modifications which lead to a scheme that can be proven to be existentially unforgeable under adaptive chosen-message attack, provided RSA is secure.

Merkle's one-time signature scheme (Algorithm 11.92) is due to Merkle [853]; see also §15.2.3(vi). The modification described in Note 11.95 is attributed by Merkle [853] to Winternitz. Bleichenbacher and Maurer [155] generalize the methods of Lamport, Merkle, and Winternitz through directed acyclic graphs and one-way functions.

Authentication trees were introduced by Merkle [850, 852, 853] at the time when public-key cryptography was in its infancy. Since public-key cryptography and, in particular, digital signatures had not yet been carefully scrutinized, it seemed prudent to devise alternate methods for providing authentication over insecure channels. Merkle [853] suggests that authentication trees provide as much versatility as public-key techniques and can be quite practical. An authentication tree, constructed by a single user to authenticate a large number of public values, requires the user to either regenerate the authentication path values at the time of use or to store all authentication paths and values in advance. Merkle [853] describes a method to minimize the storage requirements if public values are used in a prescribed order.

The GMR scheme (Algorithm 11.102) is due to Goldwasser, Micali, and Rivest [484], who introduced the notion of a claw-free pair of permutations, and described the construction of a claw-free pair of permutations (Example 11.99) based on the integer factorization problem. Combining the one-time signature scheme with tree authentication gives a digital signature mechanism which Goldwasser, Micali and Rivest prove existentially unforgeable under an adaptive chosen-message attack. In order to make their scheme more practical, the tree authentication structure is constructed in such a way that the system must retain some information about preceding signatures (i.e., *memory history* is required). Goldreich [465] suggested modifications to both the general scheme and the example based on integer factorization (Example 11.99), removing the memory constraint and, in the latter, improving the efficiency of the signing procedure. Bellare and Micali [92] generalized the GMR scheme by replacing the claw-free pair of permutations by any trapdoor one-way permutation (the latter requiring a weaker cryptographic assumption). Naor and Yung [920] further generalized the scheme by requiring only the existence of a one-way permutation. The most general result is due to Rompel [1068], who proved that digital signature schemes which are secure against an adaptive chosen-message attack exist if and only if one-way functions exist. Although attractive in theory (due to the fact that secure digital signatures can be reduced to the study of a single structure), none of these methods seem to provide techniques as efficient as RSA and other methods which, although their security has yet to be proven rigorously, have withstood all attacks to date.

On-line/off-line digital signatures (see also §15.2.3(ix)) were introduced by Even, Goldreich, and Micali [377, 378] as a means to speed up the signing process in applications where computing resources are limited and time to sign is critical (e.g., chipcard applications). The method uses both one-time digital signatures and digital signatures arising from public-key techniques (e.g., RSA, Rabin, DSA). The off-line portion of the signature generation is to create a set of validation parameters for a one-time signature scheme such as the Merkle scheme (Algorithm 11.92), and to hash this set and sign the resulting hash value using a public-key signature scheme. Since the public-key signature scheme is computationally more intensive, it is done off-line. The off-line computations are independent of the message to be signed. The on-line portion is to sign the message using the one-time signature scheme and the validation parameters which were constructed off-line; this part of the signature process is very efficient. Signatures are much longer than would be the case if only the public-key signature mechanism were used to sign the message directly and, consequently, bandwidth requirements are a disadvantage of this procedure.

§11.7

The arbitrated digital signature scheme of Algorithm 11.109 is from Davies and Price [308], based on work by Needham and Schroeder [923].

ESIGN (Algorithm 11.113; see also §15.2.2(i)), proposed by Okamoto and Shiraishi [953], was motivated by the signature mechanism OSS devised earlier by Ong, Schnorr, and Shamir [958]. The OSS scheme was shown to be insecure by Pollard in a private communication. Ong, Schnorr, and Shamir [958] modified their original scheme but this too was shown insecure by Estes et al. [374]. ESIGN bases its security on the integer factorization problem and the problem of solving polynomial inequalities. The original version [953] proposed $k = 2$ as the appropriate value for the public key. Brickell and DeLaurentis [202] demonstrated that this choice was insecure. Their attack also extends to the case $k = 3$; see Brickell and Odlyzko [209, p.516]. Okamoto [948] revised the method by requiring

$k \geq 4$. No weaknesses for these values of k have been reported in the literature. Fujioka, Okamoto, and Miyaguchi [428] describe an implementation of ESIGN which suggests that it is twenty times faster than RSA signatures with comparable key and signature lengths.

§11.8

Blind signatures (§11.8.1) were introduced by Chaum [242], who described the concept, desired properties, and a protocol for untraceable payments. The first concrete realization of the protocol (Protocol 11.119) was by Chaum [243]. Chaum and Pedersen [251] provide a digital signature scheme which is a variant of the ElGamal signature mechanism (§11.5.2), using a signing equation similar to the Schnorr scheme (§11.5.3), but computationally more intensive for both signing and verification. This signature technique is then used to provide a blind signature scheme.

The concept of a blind signature was extended by Chaum [245] to blinding for unanticipated signatures. Camenisch, Piveteau, and Stadler [228] describe a blind signature protocol based on the DSA (Algorithm 11.56) and one based on the Nyberg-Rueppel scheme (Algorithm 11.81). Horster, Petersen, and Michels [563] consider a number of variants of these protocols. Stadler, Piveteau, and Camenisch [1166] extend the idea of a blind signature to a *fair blind signature* where the signer in cooperation with a trusted third party can link the message and signature, and trace the sender.

Chaum, Fiat, and Naor [250] propose a scheme for *untraceable electronic cash*, which allows a participant A to receive an electronic cash token from a bank. A can subsequently spend the token at a shop B, which need not be on-line with the bank to accept and verify the authenticity of the token. When the token is cashed at the bank by B, the bank is unable to associate it with A. If, however, A attempts to spend the token twice (*double-spending*), A's identity is revealed. Okamoto [951] proposes a divisible electronic cash scheme. A *divisible electronic coin* is an element which has some monetary value associated with it, and which can be used to make electronic purchases many times, provided the total value of all transactions does not exceed the value of the coin.

Undeniable signatures (§11.8.2) were first introduced by Chaum and van Antwerpen [252], along with a disavowal protocol (Protocol 11.125). Chaum [246] shows how to modify the verification protocol for undeniable signatures (step 2 of Algorithm 11.122) to obtain a zero-knowledge verification.

One shortcoming of undeniable signature schemes is the possibility that the signer is unavailable or refuses to co-operate so that the signature cannot be verified by a recipient. Chaum [247] proposed the idea of a *designated confirmer signature* where the signer designates some entity as a confirmer of its signature. If the signer is unavailable or refuses to co-operate, the confirmer has the ability to interact with a recipient of a signature in order to verify it. The confirmer is unable to create signatures for the signer. Chaum [247] describes an example of designated confirmer signatures based on RSA encryption. Okamoto [950] provides a more indepth analysis of this technique and gives other realizations.

A *convertible undeniable digital signature*, introduced by Boyar et al. [181], is an undeniable signature (§11.8.2) with the property that the signer A can reveal a secret piece of information, causing all undeniable signatures signed by A to become ordinary digital signatures. These ordinary digital signatures can be verified by anyone using only the public key of A and requiring no interaction with A in the verification process; i.e., the signatures become *self-authenticating*. This secret information which is made available should not permit anyone to create new signatures which will be accepted as originating from A. As an application of this type of signature, consider the following scenario. Entity A signs all documents during her lifetime with convertible undeniable signatures. The secret piece of information needed to convert these signatures to self-authenticating signatures is placed in trust with her lawyer B. After the death of A, the lawyer can make the secret information public knowledge and all signatures can be verified. B does not have the ability to alter or create new signatures on behalf of A. Boyar et al. [181] give a realization of the concept of convertible undeniable signatures using ElGamal signatures (§11.5.2) and describe how one can reveal information selectively to convert some, but not all, previously created signatures to self-authenticating ones.

Chaum, van Heijst, and Pfitzmann [254] provide a method for constructing undeniable signatures which are unconditionally secure for the signer.

Fail-stop signatures were introduced by Waidner and Pfitzmann [1227] and formally defined by Pfitzmann and Waidner [971]. The first constructions for fail-stop signatures used claw-free pairs of permutations (Definition 11.98) and one-time signature methods (see Pfitzmann and Waidner [972]). More efficient techniques were provided by van Heijst and Pedersen [1201], whose construction is the basis for Algorithm 11.130; they describe three methods for extending the one-time nature of the scheme to multiple signings. Van Heijst, Pedersen, and Pfitzmann [1202] extended the idea of van Heijst and Pedersen to fail-stop signatures based on the integer factorization problem.

Damgård [298] proposed a signature scheme in which the signer can gradually and verifiably release the signature to a verifier.

Chaum and van Heijst [253] introduced the concept of a *group signature*. A group signature has the following properties: (i) only members of a predefined group can sign messages; (ii) anyone can verify the validity of a signature but no one is able to identify which member of the group signed; and (iii) in case of disputes, the signature can be opened (with or without the help of group members) to reveal the identity of the group member who signed it. Chen and Pedersen [255] extended this idea to provide group signatures with additional functionality.