Chapter 8

Public-Key Encryption

Contents in Brief

8.1 Introduction

This chapter considers various techniques for public-key encryption, also referred to as *asymmetric encryption*. As introduced previously (§1.8.1), in public-key encryption systems each entity A has a *public key* e and a corresponding *private key* d. In secure systems, the task of computing d given e is computationally infeasible. The public key defines an *encryption transformation* E_e, while the private key defines the associated *decryption transformation* D_d. Any entity B wishing to send a message m to A obtains an authentic copy of A's public key e, uses the encryption transformation to obtain the ciphertext $c = E_e(m)$, and transmits c to A. To decrypt c, A applies the decryption transformation to obtain the original message $m = D_d(c)$.

The public key need not be kept secret, and, in fact, may be widely available – only its authenticity is required to guarantee that A is indeed the only party who knows the corresponding private key. A primary advantage of such systems is that providing authentic public keys is generally easier than distributing secret keys securely, as required in symmetric-key systems.

The main objective of public-key encryption is to provide *privacy* or *confidentiality*. Since A's encryption transformation is public knowledge, public-key encryption alone does not provide *data origin authentication* (Definition 9.76) or *data integrity* (Definition 9.75). Such assurances must be provided through use of additional techniques (see §9.6), including message authentication codes and digital signatures.

Public-key encryption schemes are typically substantially slower than symmetric-key encryption algorithms such as DES (§7.4). For this reason, public-key encryption is most commonly used in practice for the transport of keys subsequently used for bulk data encryption by symmetric algorithms and other applications including data integrity and authentication, and for encrypting small data items such as credit card numbers and PINs.

Public-key decryption may also provide authentication guarantees in entity authentication and authenticated key establishment protocols.

Chapter outline

The remainder of the chapter is organized as follows. §8.1.1 provides introductory material. The RSA public-key encryption scheme is presented in §8.2; related security and implementation issues are also discussed. Rabin's public-key encryption scheme, which is provably as secure as factoring, is the topic of §8.3. §8.4 considers the ElGamal encryption scheme; related security and implementation issues are also discussed. The McEliece public-key encryption scheme, based on error-correcting codes, is examined in §8.5. Although known to be insecure, the Merkle-Hellman knapsack public-key encryption scheme is presented in §8.6 for historical reasons – it was the first concrete realization of a public-key encryption scheme. Chor-Rivest encryption is also presented (§8.6.2) as an example of an as-yet unbroken public-key encryption scheme based on the subset sum (knapsack) problem. §8.7 introduces the notion of probabilistic public-key encryption, designed to meet especially stringent security requirements. §8.8 concludes with Chapter notes and references.

The number-theoretic computational problems which form the security basis for the public-key encryption schemes discussed in this chapter are listed in Table 8.1.

public-key encryption scheme	computational problem
RSA	integer factorization problem (§3.2) RSA problem (§3.3)
Rabin	integer factorization problem (§3.2) square roots modulo composite n (§3.5.2)
ElGamal	discrete logarithm problem (§3.6) Diffie-Hellman problem (§3.7)
generalized ElGamal	generalized discrete logarithm problem (§3.6) generalized Diffie-Hellman problem (§3.7)
McEliece	linear code decoding problem
Merkle-Hellman knapsack	subset sum problem (§3.10)
Chor-Rivest knapsack	subset sum problem (§3.10)
Goldwasser-Micali probabilistic	quadratic residuosity problem (§3.4)
Blum-Goldwasser probabilistic	integer factorization problem (§3.2) Rabin problem (§3.9.3)

Table 8.1: *Public-key encryption schemes discussed in this chapter, and the related computational problems upon which their security is based.*

8.1.1 Basic principles

Objectives of adversary

The primary objective of an adversary who wishes to "attack" a public-key encryption scheme is to systematically recover plaintext from ciphertext intended for some other entity A. If this is achieved, the encryption scheme is informally said to have been *broken*. A more ambitious objective is *key recovery* – to recover A's private key. If this is achieved, the encryption scheme is informally said to have been *completely broken* since the adversary then has the ability to decrypt *all* ciphertext sent to A.

Types of attacks

Since the encryption transformations are public knowledge, a passive adversary can always mount a *chosen-plaintext attack* on a public-key encryption scheme (cf. §1.13.1). A stronger attack is a *chosen-ciphertext attack* where an adversary selects ciphertext of its choice, and then obtains by some means (from the victim A) the corresponding plaintext (cf. §1.13.1). Two kinds of these attacks are usually distinguished.

1. In an *indifferent* chosen-ciphertext attack, the adversary is provided with decryptions of any ciphertexts of its choice, but these ciphertexts must be chosen prior to receiving the (target) ciphertext c it actually wishes to decrypt.
2. In an *adaptive* chosen-ciphertext attack, the adversary may use (or have access to) A's decryption machine (but not the private key itself) even after seeing the target ciphertext c. The adversary may request decryptions of ciphertext which may be related to both the target ciphertext, and to the decryptions obtained from previous queries; a restriction is that it may not request the decryption of the target c itself.

Chosen-ciphertext attacks are of concern if the environment in which the public-key encryption scheme is to be used is subject to such an attack being mounted; if not, the existence of a chosen-ciphertext attack is typically viewed as a *certificational* weakness against a particular scheme, although apparently not directly exploitable.

Distributing public keys

The public-key encryption schemes described in this chapter assume that there is a means for the sender of a message to obtain an *authentic* copy of the intended receiver's public key. In the absence of such a means, the encryption scheme is susceptible to an *impersonation* attack, as outlined in §1.8.2. There are many techniques in practice by which authentic public keys can be distributed, including exchanging keys over a trusted channel, using a trusted public file, using an on-line trusted server, and using an off-line server and certificates. These and related methods are discussed in §13.4.

Message blocking

Some of the public-key encryption schemes described in this chapter assume that the message to be encrypted is, at most, some fixed size (bitlength). Plaintext messages longer than this maximum must be broken into *blocks*, each of the appropriate size. Specific techniques for breaking up a message into blocks are not discussed in this book. The component blocks can then be encrypted independently (cf. ECB mode in §7.2.2(i)). To provide protection against manipulation (e.g., re-ordering) of the blocks, the *Cipher Block Chaining* (CBC) mode may be used (cf. §7.2.2(ii) and Example 9.84). Since the CFB and OFB modes (cf. §7.2.2(iii) and §7.2.2(iv)) employ only single-block encryption (and not decryption) for both message encryption and decryption, they cannot be used with public-key encryption schemes.

8.2 RSA public-key encryption

The RSA cryptosystem, named after its inventors R. Rivest, A. Shamir, and L. Adleman, is the most widely used public-key cryptosystem. It may be used to provide both secrecy and digital signatures and its security is based on the intractability of the integer factorization problem (§3.2). This section describes the RSA encryption scheme, its security, and some implementation issues; the RSA signature scheme is covered in §11.3.1.

8.2.1 Description

8.1 Algorithm Key generation for RSA public-key encryption

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

1. Generate two large random (and distinct) primes p and q, each roughly the same size.
2. Compute $n = pq$ and $\phi = (p-1)(q-1)$. (See Note 8.5.)
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.

8.2 Definition The integers e and d in RSA key generation are called the *encryption exponent* and the *decryption exponent*, respectively, while n is called the *modulus*.

8.3 Algorithm RSA public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key (n, e).
   (b) Represent the message as an integer m in the interval $[0, n-1]$.
   (c) Compute $c = m^e \bmod n$ (e.g., using Algorithm 2.143).
   (d) Send the ciphertext c to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Use the private key d to recover $m = c^d \bmod n$.

Proof that decryption works. Since $ed \equiv 1 \pmod{\phi}$, there exists an integer k such that $ed = 1 + k\phi$. Now, if $\gcd(m, p) = 1$ then by Fermat's theorem (Fact 2.127),

$$m^{p-1} \equiv 1 \pmod{p}.$$

Raising both sides of this congruence to the power $k(q-1)$ and then multiplying both sides by m yields

$$m^{1+k(p-1)(q-1)} \equiv m \pmod{p}.$$

On the other hand, if $\gcd(m, p) = p$, then this last congruence is again valid since each side is congruent to 0 modulo p. Hence, in all cases

$$m^{ed} \equiv m \pmod{p}.$$

By the same argument,

$$m^{ed} \equiv m \pmod{q}.$$

Finally, since p and q are distinct primes, it follows that

$$m^{ed} \equiv m \pmod{n},$$

and, hence,

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

8.4 Example (*RSA encryption with artificially small parameters*)
Key generation. Entity A chooses the primes $p = 2357$, $q = 2551$, and computes $n = pq = 6012707$ and $\phi = (p-1)(q-1) = 6007800$. A chooses $e = 3674911$ and, using the extended Euclidean algorithm, finds $d = 422191$ such that $ed \equiv 1 \pmod{\phi}$. A's public key is the pair $(n = 6012707, e = 3674911)$, while A's private key is $d = 422191$.
Encryption. To encrypt a message $m = 5234673$, B uses an algorithm for modular exponentiation (e.g., Algorithm 2.143) to compute

$$c = m^e \bmod n = 5234673^{3674911} \bmod 6012707 = 3650502,$$

and sends this to A.
Decryption. To decrypt c, A computes

$$c^d \bmod n = 3650502^{422191} \bmod 6012707 = 5234673. \qquad \square$$

8.5 Note (*universal exponent*) The number $\lambda = \operatorname{lcm}(p-1, q-1)$, sometimes called the *universal exponent* of n, may be used instead of $\phi = (p-1)(q-1)$ in RSA key generation (Algorithm 8.1). Observe that λ is a proper divisor of ϕ. Using λ can result in a smaller decryption exponent d, which may result in faster decryption (cf. Note 8.9). However, if p and q are chosen at random, then $\gcd(p-1, q-1)$ is expected to be small, and consequently ϕ and λ will be roughly of the same size.

8.2.2 Security of RSA

This subsection discusses various security issues related to RSA encryption. Various attacks which have been studied in the literature are presented, as well as appropriate measures to counteract these threats.

(i) Relation to factoring

The task faced by a passive adversary is that of recovering plaintext m from the corresponding ciphertext c, given the public information (n, e) of the intended receiver A. This is called the *RSA problem* (RSAP), which was introduced in §3.3. There is no efficient algorithm known for this problem.

One possible approach which an adversary could employ to solving the RSA problem is to first factor n, and then compute ϕ and d just as A did in Algorithm 8.1. Once d is obtained, the adversary can decrypt any ciphertext intended for A.

On the other hand, if an adversary could somehow compute d, then it could subsequently factor n efficiently as follows. First note that since $ed \equiv 1 \pmod{\phi}$, there is an integer k such that $ed - 1 = k\phi$. Hence, by Fact 2.126(i), $a^{ed-1} \equiv 1 \pmod{n}$ for all $a \in \mathbb{Z}_n^*$. Let $ed - 1 = 2^s t$, where t is an odd integer. Then it can be shown that there exists an $i \in [1, s]$ such that $a^{2^{i-1}t} \not\equiv \pm 1 \pmod{n}$ and $a^{2^i t} \equiv 1 \pmod{n}$ for at least half of all $a \in \mathbb{Z}_n^*$; if a and i are such integers then $\gcd(a^{2^{i-1}t} - 1, n)$ is a non-trivial factor of n. Thus the adversary simply needs to repeatedly select random $a \in \mathbb{Z}_n^*$ and check if an $i \in [1, s]$ satisfying the above property exists; the expected number of trials before a non-trivial factor of n is obtained is 2. This discussion establishes the following.

8.6 Fact The problem of computing the RSA decryption exponent d from the public key (n, e), and the problem of factoring n, are computationally equivalent.

When generating RSA keys, it is imperative that the primes p and q be selected in such a way that factoring $n = pq$ is computationally infeasible; see Note 8.8 for more details.

(ii) Small encryption exponent e

In order to improve the efficiency of encryption, it is desirable to select a small encryption exponent e (see Note 8.9) such as $e = 3$. A group of entities may all have the same encryption exponent e, however, each entity in the group must have its own distinct modulus (cf. §8.2.2(vi)). If an entity A wishes to send the same message m to three entities whose public moduli are n_1, n_2, n_3, and whose encryption exponents are $e = 3$, then A would send $c_i = m^3 \bmod n_i$, for $i = 1, 2, 3$. Since these moduli are most likely pairwise relatively prime, an eavesdropper observing c_1, c_2, c_3 can use Gauss's algorithm (Algorithm 2.121) to find a solution x, $0 \le x < n_1n_2n_3$, to the three congruences

$$\begin{cases} x \equiv c_1 \pmod{n_1} \\ x \equiv c_2 \pmod{n_2} \\ x \equiv c_3 \pmod{n_3}. \end{cases}$$

Since $m^3 < n_1n_2n_3$, by the Chinese remainder theorem (Fact 2.120), it must be the case that $x = m^3$. Hence, by computing the integer cube root of x, the eavesdropper can recover the plaintext m.

Thus a small encryption exponent such as $e = 3$ should not be used if the same message, or even the same message with known variations, is sent to many entities. Alternatively, to prevent against such an attack, a pseudorandomly generated bitstring of appropriate length (taking into account Coppersmith's attacks mentioned on pages 313–314) should be appended to the plaintext message prior to encryption; the pseudorandom bitstring should be independently generated for each encryption. This process is sometimes referred to as *salting* the message.

Small encryption exponents are also a problem for small messages m, because if $m < n^{1/e}$, then m can be recovered from the ciphertext $c = m^e \bmod n$ simply by computing the integer e^{th} root of c; salting plaintext messages also circumvents this problem.

(iii) Forward search attack

If the message space is small or predictable, an adversary can decrypt a ciphertext c by simply encrypting all possible plaintext messages until c is obtained. Salting the message as described above is one simple method of preventing such an attack.

(iv) Small decryption exponent d

As was the case with the encryption exponent e, it may seem desirable to select a small decryption exponent d in order to improve the efficiency of decryption.[1] However, if $\gcd(p-1, q-1)$ is small, as is typically the case, and if d has up to approximately one-quarter as many bits as the modulus n, then there is an efficient algorithm (referenced on page 313) for computing d from the public information (n, e). This algorithm cannot be extended to the case where d is approximately the same size as n. Hence, to avoid this attack, the decryption exponent d should be roughly the same size as n.

(v) Multiplicative properties

Let m_1 and m_2 be two plaintext messages, and let c_1 and c_2 be their respective RSA encryptions. Observe that

$$(m_1m_2)^e \equiv m_1^e m_2^e \equiv c_1c_2 \pmod{n}.$$

[1]In this case, one would select d first and then compute e in Algorithm 8.1, rather than vice-versa.

In other words, the ciphertext corresponding to the plaintext $m = m_1m_2 \bmod n$ is $c = c_1c_2 \bmod n$; this is sometimes referred to as the *homomorphic property* of RSA. This observation leads to the following *adaptive chosen-ciphertext attack* on RSA encryption.

Suppose that an active adversary wishes to decrypt a particular ciphertext $c = m^e \bmod n$ intended for A. Suppose also that A will decrypt arbitrary ciphertext for the adversary, other than c itself. The adversary can conceal c by selecting a random integer $x \in \mathbb{Z}_n^*$ and computing $\overline{c} = cx^e \bmod n$. Upon presentation of $\overline{c}$, A will compute for the adversary $\overline{m} = (\overline{c})^d \bmod n$. Since

$$\overline{m} \equiv (\overline{c})^d \equiv c^d(x^e)^d \equiv mx \pmod{n},$$

the adversary can then compute $m = \overline{m}x^{-1} \bmod n$.

This adaptive chosen-ciphertext attack should be circumvented in practice by imposing some structural constraints on plaintext messages. If a ciphertext c is decrypted to a message not possessing this structure, then c is rejected by the decryptor as being fraudulent. Now, if a plaintext message m has this (carefully chosen) structure, then with high probability $mx \bmod n$ will not for $x \in \mathbb{Z}_n^*$. Thus the adaptive chosen-ciphertext attack described in the previous paragraph will fail because A will not decrypt $\overline{c}$ for the adversary. Note 8.63 provides a powerful technique for guarding against adaptive chosen-ciphertext and other kinds of attacks.

(vi) Common modulus attack

The following discussion demonstrates why it is imperative for each entity to choose its own RSA modulus n.

It is sometimes suggested that a central trusted authority should select a single RSA modulus n, and then distribute a distinct encryption/decryption exponent pair (e_i, d_i) to each entity in a network. However, as shown in (i) above, knowledge of any (e_i, d_i) pair allows for the factorization of the modulus n, and hence any entity could subsequently determine the decryption exponents of all other entities in the network. Also, if a single message were encrypted and sent to two or more entities in the network, then there is a technique by which an eavesdropper (any entity not in the network) could recover the message with high probability using only publicly available information.

(vii) Cycling attacks

Let $c = m^e \bmod n$ be a ciphertext. Let k be a positive integer such that $c^{e^k} \equiv c \pmod{n}$; since encryption is a permutation on the message space $\{0, 1, \ldots, n-1\}$ such an integer k must exist. For the same reason it must be the case that $c^{e^{k-1}} \equiv m \pmod{n}$. This observation leads to the following *cycling attack* on RSA encryption. An adversary computes $c^e \bmod n$, $c^{e^2} \bmod n$, $c^{e^3} \bmod n, \ldots$ until c is obtained for the first time. If $c^{e^k} \bmod n = c$, then the previous number in the cycle, namely $c^{e^{k-1}} \bmod n$, is equal to the plaintext m.

A *generalized cycling attack* is to find the smallest positive integer u such that $f = \gcd(c^{e^u} - c, n) > 1$. If

$$c^{e^u} \equiv c \pmod{p} \text{ and } c^{e^u} \not\equiv c \pmod{q} \tag{8.1}$$

then $f = p$. Similarly, if

$$c^{e^u} \not\equiv c \pmod{p} \text{ and } c^{e^u} \equiv c \pmod{q} \tag{8.2}$$

then $f = q$. In either case, n has been factored, and the adversary can recover d and then m. On the other hand, if both

$$c^{e^u} \equiv c \pmod{p} \text{ and } c^{e^u} \equiv c \pmod{q}, \tag{8.3}$$

then $f = n$ and $c^{e^u} \equiv c \pmod{n}$. In fact, u must be the smallest positive integer k for which $c^{e^k} \equiv c \pmod{n}$. In this case, the basic cycling attack has succeeded and so $m = c^{e^{u-1}} \bmod n$ can be computed efficiently. Since (8.3) is expected to occur much less frequently than (8.1) or (8.2), the generalized cycling attack usually terminates before the cycling attack does. For this reason, the generalized cycling attack can be viewed as being essentially an algorithm for factoring n.

Since factoring n is assumed to be intractable, these cycling attacks do not pose a threat to the security of RSA encryption.

(viii) Message concealing

A plaintext message m, $0 \le m \le n-1$, in the RSA public-key encryption scheme is said to be *unconcealed* if it encrypts to itself; that is, $m^e \equiv m \pmod{n}$. There are always some messages which are unconcealed (for example $m = 0$, $m = 1$, and $m = n-1$). In fact, the number of unconcealed messages is exactly

$$[1 + \gcd(e-1, p-1)] \cdot [1 + \gcd(e-1, q-1)].$$

Since $e-1$, $p-1$ and $q-1$ are all even, the number of unconcealed messages is always at least 9. If p and q are random primes, and if e is chosen at random (or if e is chosen to be a small number such as $e = 3$ or $e = 2^{16} + 1 = 65537$), then the proportion of messages which are unconcealed by RSA encryption will, in general, be negligibly small, and hence unconcealed messages do not pose a threat to the security of RSA encryption in practice.

8.2.3 RSA encryption in practice

There are numerous ways of speeding up RSA encryption and decryption in software and hardware implementations. Some of these techniques are covered in Chapter 14, including fast modular multiplication (§14.3), fast modular exponentiation (§14.6), and the use of the Chinese remainder theorem for faster decryption (Note 14.75). Even with these improvements, RSA encryption/decryption is substantially slower than the commonly used symmetric-key encryption algorithms such as DES (Chapter 7). In practice, RSA encryption is most commonly used for the transport of symmetric-key encryption algorithm keys and for the encryption of small data items.

The RSA cryptosystem has been patented in the U.S. and Canada. Several standards organizations have written, or are in the process of writing, standards that address the use of the RSA cryptosystem for encryption, digital signatures, and key establishment. For discussion of patent and standards issues related to RSA, see Chapter 15.

8.7 Note (*recommended size of modulus*) Given the latest progress in algorithms for factoring integers (§3.2), a 512-bit modulus n provides only marginal security from concerted attack. As of 1996, in order to foil the powerful quadratic sieve (§3.2.6) and number field sieve (§3.2.7) factoring algorithms, a modulus n of at least 768 bits is recommended. For long-term security, 1024-bit or larger moduli should be used.

8.8 Note (*selecting primes*)

(i) As mentioned in §8.2.2(i), the primes p and q should be selected so that factoring $n = pq$ is computationally infeasible. The major restriction on p and q in order to avoid the elliptic curve factoring algorithm (§3.2.4) is that p and q should be about the same bitlength, and sufficiently large. For example, if a 1024-bit modulus n is to be used, then each of p and q should be about 512 bits in length.

(ii) Another restriction on the primes p and q is that the difference $p - q$ should not be too small. If $p - q$ is small, then $p \approx q$ and hence $p \approx \sqrt{n}$. Thus, n could be factored efficiently simply by trial division by all odd integers close to $\sqrt{n}$. If p and q are chosen at random, then $p - q$ will be appropriately large with overwhelming probability.

(iii) In addition to these restrictions, many authors have recommended that p and q be strong primes. A prime p is said to be a *strong prime* (cf. Definition 4.52) if the following three conditions are satisfied:

(a) $p - 1$ has a large prime factor, denoted r;
(b) $p + 1$ has a large prime factor; and
(c) $r - 1$ has a large prime factor.

An algorithm for generating strong primes is presented in §4.4.2. The reason for condition (a) is to foil Pollard's $p-1$ factoring algorithm (§3.2.3) which is efficient only if n has a prime factor p such that $p - 1$ is smooth. Condition (b) foils the $p + 1$ factoring algorithm mentioned on page 125 in §3.12, which is efficient only if n has a prime factor p such that $p + 1$ is smooth. Finally, condition (c) ensures that the cycling attacks described in §8.2.2(vii) will fail.

If the prime p is randomly chosen and is sufficiently large, then both $p-1$ and $p+1$ can be expected to have large prime factors. In any case, while strong primes protect against the $p-1$ and $p+1$ factoring algorithms, they do not protect against their generalization, the elliptic curve factoring algorithm (§3.2.4). The latter is successful in factoring n if a randomly chosen number of the same size as p (more precisely, this number is the order of a randomly selected elliptic curve defined over $\mathbb{Z}_p$) has only small prime factors. Additionally, it has been shown that the chances of a cycling attack succeeding are negligible if p and q are randomly chosen (cf. §8.2.2(vii)). Thus, strong primes offer little protection beyond that offered by random primes. Given the current state of knowledge of factoring algorithms, there is no compelling reason for requiring the use of strong primes in RSA key generation. On the other hand, they are no less secure than random primes, and require only minimal additional running time to compute; thus there is little real additional cost in using them.

8.9 Note (*small encryption exponents*)

(i) If the encryption exponent e is chosen at random, then RSA encryption using the repeated square-and-multiply algorithm (Algorithm 2.143) takes k modular squarings and an expected $k/2$ (less with optimizations) modular multiplications, where k is the bitlength of the modulus n. Encryption can be sped up by selecting e to be small and/or by selecting e with a small number of 1's in its binary representation.

(ii) The encryption exponent $e = 3$ is commonly used in practice; in this case, it is necessary that neither $p-1$ nor $q-1$ be divisible by 3. This results in a very fast encryption operation since encryption only requires 1 modular multiplication and 1 modular squaring. Another encryption exponent used in practice is $e = 2^{16} + 1 = 65537$. This number has only two 1's in its binary representation, and so encryption using the repeated square-and-multiply algorithm requires only 16 modular squarings and 1 modular multiplication. The encryption exponent $e = 2^{16} + 1$ has the advantage over $e = 3$ in that it resists the kind of attack discussed in §8.2.2(ii), since it is unlikely the same message will be sent to $2^{16}+1$ recipients. But see also Coppersmith's attacks mentioned on pages 313–314.

8.3 Rabin public-key encryption

A desirable property of any encryption scheme is a proof that breaking it is as difficult as solving a computational problem that is widely believed to be difficult, such as integer factorization or the discrete logarithm problem. While it is widely believed that breaking the RSA encryption scheme is as difficult as factoring the modulus n, no such equivalence has been proven. The Rabin public-key encryption scheme was the first example of a *provably secure* public-key encryption scheme – the problem faced by a passive adversary of recovering plaintext from some given ciphertext is computationally equivalent to factoring.

8.10 Algorithm Key generation for Rabin public-key encryption

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

1. Generate two large random (and distinct) 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).

8.11 Algorithm Rabin public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key n.
   (b) Represent the message as an integer m in the range $\{0, 1, \ldots, n-1\}$.
   (c) Compute $c = m^2 \bmod n$.
   (d) Send the ciphertext c to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Use Algorithm 3.44 to find the four square roots m_1, m_2, m_3, and m_4 of c modulo n.[2] (See also Note 8.12.)
   (b) The message sent was either m_1, m_2, m_3, or m_4. A somehow (cf. Note 8.14) decides which of these is m.

8.12 Note (*finding square roots of c modulo $n = pq$ when $p \equiv q \equiv 3 \pmod 4$*) If p and q are both chosen to be $\equiv 3 \pmod 4$, then Algorithm 3.44 for computing the four square roots of c modulo n simplifies as follows:

1. Use the extended Euclidean algorithm (Algorithm 2.107) to find integers a and b satisfying $ap + bq = 1$. Note that a and b can be computed once and for all during the key generation stage (Algorithm 8.10).
2. Compute $r = c^{(p+1)/4} \bmod p$.
3. Compute $s = c^{(q+1)/4} \bmod q$.
4. Compute $x = (aps + bqr) \bmod n$.
5. Compute $y = (aps - bqr) \bmod n$.
6. The four square roots of c modulo n are x, $-x \bmod n$, y, and $-y \bmod n$.

[2] In the very unlikely case that $\gcd(m, n) \neq 1$, the ciphertext c does not have four distinct square roots modulo n, but rather only one or two.

8.13 Note (*security of Rabin public-key encryption*)

(i) The task faced by a passive adversary is to recover plaintext m from the corresponding ciphertext c. This is precisely the SQROOT problem of §3.5.2. Recall (Fact 3.46) that the problems of factoring n and computing square roots modulo n are computationally equivalent. Hence, assuming that factoring n is computationally intractable, the Rabin public-key encryption scheme is *provably secure* against a passive adversary.

(ii) While provably secure against a passive adversary, the Rabin public-key encryption scheme succumbs to a chosen-ciphertext attack (but see Note 8.14(ii)). Such an attack can be mounted as follows. The adversary selects a random integer $m \in \mathbb{Z}_n^*$ and computes $c = m^2 \bmod n$. The adversary then presents c to A's decryption machine, which decrypts c and returns some plaintext y. Since A does not know m, and m is randomly chosen, the plaintext y is not necessarily the same as m. With probability $\frac{1}{2}$, $y \not\equiv \pm m \bmod n$, in which case $\gcd(m - y, n)$ is one of the prime factors of n. If $y \equiv \pm m \bmod n$, then the attack is repeated with a new m.[3]

(iii) The Rabin public-key encryption scheme is susceptible to attacks similar to those on RSA described in §8.2.2(ii), §8.2.2(iii), and §8.2.2(v). As is the case with RSA, attacks (ii) and (iii) can be circumvented by salting the plaintext message, while attack (v) can be avoided by adding appropriate redundancy prior to encryption.

8.14 Note (*use of redundancy*)

(i) A drawback of Rabin's public-key scheme is that the receiver is faced with the task of selecting the correct plaintext from among four possibilities. This ambiguity in decryption can easily be overcome in practice by adding prespecified redundancy to the original plaintext prior to encryption. (For example, the last 64 bits of the message may be replicated.) Then, with high probability, exactly one of the four square roots m_1, m_2, m_3, m_4 of a legitimate ciphertext c will possess this redundancy, and the receiver will select this as the intended plaintext. If none of the square roots of c possesses this redundancy, then the receiver should reject c as fraudulent.

(ii) If redundancy is used as above, Rabin's scheme is no longer susceptible to the chosen-ciphertext attack of Note 8.13(ii). If an adversary selects a message m having the required redundancy and gives $c = m^2 \bmod n$ to A's decryption machine, with very high probability the machine will return the plaintext m itself to the adversary (since the other three square roots of c will most likely not contain the required redundancy), providing no new information. On the other hand, if the adversary selects a message m which does not contain the required redundancy, then with high probability none of the four square roots of $c = m^2 \bmod n$ will possess the required redundancy. In this case, the decryption machine will fail to decrypt c and thus will not provide a response to the adversary. Note that the proof of equivalence of breaking the modified scheme by a passive adversary to factoring is no longer valid. However, if the natural assumption is made that Rabin decryption is composed of two processes, the first which finds the four square roots of $c \bmod n$, and the second which selects the distinguished square root as the plaintext, then the proof of equivalence holds. Hence, Rabin public-key encryption, suitably modified by adding redundancy, is of great practical interest.

[3] This chosen-ciphertext attack is an execution of the constructive proof of the equivalence of factoring n and the SQROOT problem (Fact 3.46), where A's decryption machine is used instead of the hypothetical polynomial-time algorithm for solving the SQROOT problem in the proof.

8.15 Example (*Rabin public-key encryption with artificially small parameters*)
Key generation. Entity A chooses the primes $p = 277$, $q = 331$, and computes $n = pq = 91687$. A's public key is $n = 91687$, while A's private key is $(p = 277, q = 331)$.
Encryption. Suppose that the last six bits of original messages are required to be replicated prior to encryption (cf. Note 8.14(i)). In order to encrypt the 10-bit message $\overline{m} = 1001111001$, B replicates the last six bits of $\overline{m}$ to obtain the 16-bit message $m = 1001111001111001$, which in decimal notation is $m = 40569$. B then computes

$$c = m^2 \bmod n = 40569^2 \bmod 91687 = 62111$$

and sends this to A.
Decryption. To decrypt c, A uses Algorithm 3.44 and her knowledge of the factors of n to compute the four square roots of $c \bmod n$:

$$m_1 = 69654, \quad m_2 = 22033, \quad m_3 = 40569, \quad m_4 = 51118,$$

which in binary are

$$\begin{array}{ll} m_1 = 10001000000010110, & m_2 = 101011000010001, \\ m_3 = 1001111001111001, & m_4 = 1100011110101110. \end{array}$$

Since only m_3 has the required redundancy, A decrypts c to m_3 and recovers the original message $\overline{m} = 1001111001$. □

8.16 Note (*efficiency*) Rabin encryption is an extremely fast operation as it only involves a single modular squaring. By comparison, RSA encryption with $e = 3$ takes one modular multiplication and one modular squaring. Rabin decryption is slower than encryption, but comparable in speed to RSA decryption.

8.4 ElGamal public-key encryption

The ElGamal public-key encryption scheme can be viewed as Diffie-Hellman key agreement (§12.6.1) in key transfer mode (cf. Note 8.23(i)). Its security is based on the intractability of the discrete logarithm problem (see §3.6) and the Diffie-Hellman problem (§3.7). The basic ElGamal and generalized ElGamal encryption schemes are described in this section.

8.4.1 Basic ElGamal encryption

8.17 Algorithm Key generation for ElGamal public-key encryption

SUMMARY: each entity creates a public key and a 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 $\mathbb{Z}_p^*$ of the integers modulo p (using Algorithm 4.84).
2. Select a random integer a, $1 \le a \le p-2$, and compute $\alpha^a \bmod p$ (using Algorithm 2.143).
3. A's public key is (p, α, α^a); A's private key is a.

8.18 Algorithm ElGamal public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key (p, α, α^a).
   (b) Represent the message as an integer m in the range $\{0, 1, \ldots, p-1\}$.
   (c) Select a random integer k, $1 \leq k \leq p-2$.
   (d) Compute $\gamma = \alpha^k \bmod p$ and $\delta = m \cdot (\alpha^a)^k \bmod p$.
   (e) Send the ciphertext $c = (\gamma, \delta)$ to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Use the private key a to compute $\gamma^{p-1-a} \bmod p$ (note: $\gamma^{p-1-a} = \gamma^{-a} = \alpha^{-ak}$).
   (b) Recover m by computing $(\gamma^{-a}) \cdot \delta \bmod p$.

Proof that decryption works. The decryption of Algorithm 8.18 allows recovery of original plaintext because

$$\gamma^{-a} \cdot \delta \equiv \alpha^{-ak} m \alpha^{ak} \equiv m \pmod{p}.$$

8.19 Example (*ElGamal encryption with artificially small parameters*)
Key generation. Entity A selects the prime $p = 2357$ and a generator $\alpha = 2$ of $\mathbb{Z}^*_{2357}$. A chooses the private key $a = 1751$ and computes

$$\alpha^a \bmod p = 2^{1751} \bmod 2357 = 1185.$$

A's public key is $(p = 2357, \alpha = 2, \alpha^a = 1185)$.
Encryption. To encrypt a message $m = 2035$, B selects a random integer $k = 1520$ and computes

$$\gamma = 2^{1520} \bmod 2357 = 1430$$

and

$$\delta = 2035 \cdot 1185^{1520} \bmod 2357 = 697.$$

B sends $\gamma = 1430$ and $\delta = 697$ to A.
Decryption. To decrypt, A computes

$$\gamma^{p-1-a} = 1430^{605} \bmod 2357 = 872,$$

and recovers m by computing

$$m = 872 \cdot 697 \bmod 2357 = 2035. \qquad \square$$

8.20 Note (*common system-wide parameters*) All entities may elect to use the same prime p and generator α, in which case p and α need not be published as part of the public key. This results in public keys of smaller sizes. An additional advantage of having a fixed base α is that exponentiation can then be expedited via precomputations using the techniques described in §14.6.3. A potential disadvantage of common system-wide parameters is that larger moduli p may be warranted (cf. Note 8.24).

8.21 Note (*efficiency of ElGamal encryption*)

(i) The encryption process requires two modular exponentiations, namely $\alpha^k \bmod p$ and $(\alpha^a)^k \bmod p$. These exponentiations can be sped up by selecting random exponents k having some additional structure, for example, having low Hamming weights. Care must be taken that the possible number of exponents is large enough to preclude a search via a baby-step giant-step algorithm (cf. Note 3.59).

(ii) A disadvantage of ElGamal encryption is that there is *message expansion* by a factor of 2. That is, the ciphertext is twice as long as the corresponding plaintext.

8.22 Remark (*randomized encryption*) ElGamal encryption is one of many encryption schemes which utilizes randomization in the encryption process. Others include McEliece encryption (§8.5), and Goldwasser-Micali (§8.7.1), and Blum-Goldwasser (§8.7.2) probabilistic encryption. Deterministic encryption schemes such as RSA may also employ randomization in order to circumvent some attacks (e.g., see §8.2.2(ii) and §8.2.2(iii)). The fundamental idea behind randomized encryption (see Definition 7.3) techniques is to use randomization to increase the cryptographic security of an encryption process through one or more of the following methods:

(i) increasing the effective size of the plaintext message space;

(ii) precluding or decreasing the effectiveness of chosen-plaintext attacks by virtue of a one-to-many mapping of plaintext to ciphertext; and

(iii) precluding or decreasing the effectiveness of statistical attacks by leveling the a priori probability distribution of inputs.

8.23 Note (*security of ElGamal encryption*)

(i) The problem of breaking the ElGamal encryption scheme, i.e., recovering m given p, α, α^a, γ, and δ, is equivalent to solving the Diffie-Hellman problem (see §3.7). In fact, the ElGamal encryption scheme can be viewed as simply comprising a Diffie-Hellman key exchange to determine a session key α^{ak}, and then encrypting the message by multiplication with that session key. For this reason, the security of the ElGamal encryption scheme is said to be *based* on the discrete logarithm problem in $\mathbb{Z}_p^*$, although such an equivalence has not been proven.

(ii) It is critical that different random integers k be used to encrypt different messages. Suppose the same k is used to encrypt two messages m_1 and m_2 and the resulting ciphertext pairs are (γ_1, δ_1) and (γ_2, δ_2). Then $\delta_1/\delta_2 = m_1/m_2$, and m_2 could be easily computed if m_1 were known.

8.24 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. For common system-wide parameters (cf. Note 8.20) even larger key sizes may be warranted. This is because the dominant stage in the index-calculus algorithm (§3.6.5) for discrete logarithms in $\mathbb{Z}_p^*$ is the precomputation of a database of factor base logarithms, following which individual logarithms can be computed relatively quickly. Thus computing the database of logarithms for one particular modulus p will compromise the secrecy of all private keys derived using p.

8.4.2 Generalized ElGamal encryption

The ElGamal encryption scheme is typically described in the setting of the multiplicative group $\mathbb{Z}_p^*$, but can be easily generalized to work in any finite cyclic group G.

As with ElGamal encryption, the security of the generalized ElGamal encryption scheme is *based* on the intractability of the discrete logarithm problem in the group G. The group G should be carefully chosen to satisfy the following two conditions:

1. for *efficiency*, the group operation in G should be relatively easy to apply; and
2. for *security*, the discrete logarithm problem in G should be computationally infeasible.

The following is a list of groups that appear to meet these two criteria, of which the first three have received the most attention.

1. The multiplicative group $\mathbb{Z}_p^*$ of the integers modulo a prime p.
2. The multiplicative group $\mathbb{F}_{2^m}^*$ of the finite field $\mathbb{F}_{2^m}$ of characteristic two.
3. The group of points on an elliptic curve over a finite field.
4. The multiplicative group $\mathbb{F}_q^*$ of the finite field $\mathbb{F}_q$, where $q = p^m$, p a prime.
5. The group of units $\mathbb{Z}_n^*$, where n is a composite integer.
6. The jacobian of a hyperelliptic curve defined over a finite field.
7. The class group of an imaginary quadratic number field.

8.25 Algorithm Key generation for generalized ElGamal public-key encryption

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

1. Select an appropriate cyclic group G of order n, with generator α. (It is assumed here that G is written multiplicatively.)
2. Select a random integer a, $1 \leq a \leq n-1$, and compute the group element α^a.
3. A's public key is (α, α^a), together with a description of how to multiply elements in G; A's private key is a.

8.26 Algorithm Generalized ElGamal public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key (α, α^a).
   (b) Represent the message as an element m of the group G.
   (c) Select a random integer k, $1 \leq k \leq n-1$.
   (d) Compute $\gamma = \alpha^k$ and $\delta = m \cdot (\alpha^a)^k$.
   (e) Send the ciphertext $c = (\gamma, \delta)$ to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Use the private key a to compute γ^a and then compute γ^{-a}.
   (b) Recover m by computing $(\gamma^{-a}) \cdot \delta$.

8.27 Note (*common system-wide parameters*) All entities may elect to use the same cyclic group G and generator α, in which case α and the description of multiplication in G need not be published as part of the public key (cf. Note 8.20).

8.28 Example (*ElGamal encryption using the multiplicative group of $\mathbb{F}_{2^m}$, with artificially small parameters*)

Key generation. Entity A selects the group G to be the multiplicative group of the finite field $\mathbb{F}_{2^4}$, whose elements are represented by the polynomials over $\mathbb{F}_2$ of degree less than 4, and where multiplication is performed modulo the irreducible polynomial $f(x) = x^4 + x + 1$ (cf. Example 2.231). For convenience, a field element $a_3x^3 + a_2x^2 + a_1x + a_0$ is represented by the binary string $(a_3a_2a_1a_0)$. The group G has order $n = 15$ and a generator is $\alpha = (0010)$.

A chooses the private key $a = 7$ and computes $\alpha^a = \alpha^7 = (1011)$. A's public key is $\alpha^a = (1011)$ (together with $\alpha = (0010)$ and the polynomial $f(x)$ which defines the multiplication in G, if these parameters are not common to all entities).

Encryption. To encrypt a message $m = (1100)$, B selects a random integer $k = 11$ and computes $\gamma = \alpha^{11} = (1110)$, $(\alpha^a)^{11} = (0100)$, and $\delta = m \cdot (\alpha^a)^{11} = (0101)$. B sends $\gamma = (1110)$ and $\delta = (0101)$ to A.

Decryption. To decrypt, A computes $\gamma^a = (0100)$, $(\gamma^a)^{-1} = (1101)$ and finally recovers m by computing $m = (\gamma^{-a}) \cdot \delta = (1100)$. □

8.5 McEliece public-key encryption

The McEliece public-key encryption scheme is based on error-correcting codes. The idea behind this scheme is to first select a particular code for which an efficient decoding algorithm is known, and then to disguise the code as a general linear code (see Note 12.36). Since the problem of decoding an arbitrary linear code is **NP**-hard (Definition 2.73), a description of the original code can serve as the private key, while a description of the transformed code serves as the public key.

The McEliece encryption scheme (when used with Goppa codes) has resisted cryptanalysis to date. It is also notable as being the first public-key encryption scheme to use randomization in the encryption process. Although very efficient, the McEliece encryption scheme has received little attention in practice because of the very large public keys (see Remark 8.33).

8.29 Algorithm Key generation for McEliece public-key encryption

SUMMARY: each entity creates a public key and a corresponding private key.

1. Integers k, n, and t are fixed as common system parameters.
2. Each entity A should perform steps 3 – 7.
3. Choose a $k \times n$ generator matrix G for a binary (n, k)-linear code which can correct t errors, and for which an efficient decoding algorithm is known. (See Note 12.36.)
4. Select a random $k \times k$ binary non-singular matrix S.
5. Select a random $n \times n$ permutation matrix P.
6. Compute the $k \times n$ matrix $\widehat{G} = SGP$.
7. A's public key is $(\widehat{G}, t)$; A's private key is (S, G, P).

8.30 Algorithm McEliece public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key $(\widehat{G}, t)$.
   (b) Represent the message as a binary string m of length k.
   (c) Choose a random binary error vector z of length n having at most t 1's.
   (d) Compute the binary vector $c = m\widehat{G} + z$.
   (e) Send the ciphertext c to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Compute $\widehat{c} = cP^{-1}$, where P^{-1} is the inverse of the matrix P.
   (b) Use the decoding algorithm for the code generated by G to decode $\widehat{c}$ to $\widehat{m}$.
   (c) Compute $m = \widehat{m}S^{-1}$.

Proof that decryption works. Since

$$\widehat{c} \;=\; cP^{-1} \;=\; (m\widehat{G} + z)P^{-1} \;=\; (mSGP + z)P^{-1} \;=\; (mS)G + zP^{-1},$$

and zP^{-1} is a vector with at most t 1's, the decoding algorithm for the code generated by G corrects $\widehat{c}$ to $\widehat{m} = mS$. Finally, $\widehat{m}S^{-1} = m$, and, hence, decryption works.

A special type of error-correcting code, called a *Goppa code*, may be used in step 3 of the key generation. For each irreducible polynomial $g(x)$ of degree t over $\mathbb{F}_{2^m}$, there exists a binary Goppa code of length $n = 2^m$ and dimension $k \geq n - mt$ capable of correcting any pattern of t or fewer errors. Furthermore, efficient decoding algorithms are known for such codes.

8.31 Note (*security of McEliece encryption*) There are two basic kinds of attacks known.

(i) From the public information, an adversary may try to compute the key G or a key G' for a Goppa code equivalent to the one with generator matrix G. There is no efficient method known for accomplishing this.

(ii) An adversary may try to recover the plaintext m directly given some ciphertext c. The adversary picks k columns at random from $\widehat{G}$. If $\widehat{G}_k$, c_k and z_k denote the restriction of $\widehat{G}$, c and z, respectively, to these k columns, then $(c_k + z_k) = m\widehat{G}_k$. If $z_k = 0$ and if $\widehat{G}_k$ is non-singular, then m can be recovered by solving the system of equations $c_k = m\widehat{G}_k$. Since the probability that $z_k = 0$, i.e., the selected k bits were not in error, is only $\binom{n-t}{k}/\binom{n}{k}$, the probability of this attack succeeding is negligibly small.

8.32 Note (*recommended parameter sizes*) The original parameters suggested by McEliece were $n = 1024$, $t = 50$, and $k \geq 524$. Based on the security analysis (Note 8.31), an optimum choice of parameters for the Goppa code which maximizes the adversary's work factor appears to be $n = 1024$, $t = 38$, and $k \geq 644$.

8.33 Remark (*McEliece encryption in practice*) Although the encryption and decryption operations are relatively fast, the McEliece scheme suffers from the drawback that the public key is very large. A (less significant) drawback is that there is message expansion by a factor of n/k. For the recommended parameters $n = 1024$, $t = 38$, $k \geq 644$, the public key is about 2^{19} bits in size, while the message expansion factor is about 1.6. For these reasons, the scheme receives little attention in practice.

8.6 Knapsack public-key encryption

Knapsack public-key encryption schemes are based on the subset sum problem, which is **NP**-complete (see §2.3.3 and §3.10). The basic idea is to select an instance of the subset sum problem that is easy to solve, and then to disguise it as an instance of the general subset sum problem which is hopefully difficult to solve. The original knapsack set can serve as the private key, while the transformed knapsack set serves as the public key.

The Merkle-Hellman knapsack encryption scheme (§8.6.1) is important for historical reasons, as it was the first concrete realization of a public-key encryption scheme. Many variations have subsequently been proposed but most, including the original, have been demonstrated to be insecure (see Note 8.40), a notable exception being the Chor-Rivest knapsack scheme (§8.6.2).

8.6.1 Merkle-Hellman knapsack encryption

The Merkle-Hellman knapsack encryption scheme attempts to disguise an easily solved instance of the subset sum problem, called a *superincreasing subset sum problem*, by modular multiplication and a permutation. It is however not recommended for use (see Note 8.40).

8.34 Definition A *superincreasing sequence* is a sequence $(b_1, b_2, \ldots, b_n)$ of positive integers with the property that $b_i > \sum_{j=1}^{i-1} b_j$ for each i, $2 \leq i \leq n$.

Algorithm 8.35 efficiently solves the subset sum problem for superincreasing sequences.

8.35 Algorithm Solving a superincreasing subset sum problem

INPUT: a superincreasing sequence $(b_1, b_2, \ldots, b_n)$ and an integer s which is the sum of a subset of the b_i.
OUTPUT: $(x_1, x_2, \ldots, x_n)$ where $x_i \in \{0, 1\}$, such that $\sum_{i=1}^{n} x_i b_i = s$.

1. $i \leftarrow n$.
2. While $i \geq 1$ do the following:
   2.1 If $s \geq b_i$ then $x_i \leftarrow 1$ and $s \leftarrow s - b_i$. Otherwise $x_i \leftarrow 0$.
   2.2 $i \leftarrow i - 1$.
3. Return($(x_1, x_2, \ldots, x_n)$).

8.36 Algorithm Key generation for basic Merkle-Hellman knapsack encryption

SUMMARY: each entity creates a public key and a corresponding private key.

1. An integer n is fixed as a common system parameter.
2. Each entity A should perform steps 3 – 7.
3. Choose a superincreasing sequence $(b_1, b_2, \ldots, b_n)$ and modulus M such that $M > b_1 + b_2 + \cdots + b_n$.
4. Select a random integer W, $1 \leq W \leq M - 1$, such that $\gcd(W, M) = 1$.
5. Select a random permutation π of the integers $\{1, 2, \ldots, n\}$.
6. Compute $a_i = W b_{\pi(i)} \bmod M$ for $i = 1, 2, \ldots, n$.
7. A's public key is $(a_1, a_2, \ldots, a_n)$; A's private key is $(\pi, M, W, (b_1, b_2, \ldots, b_n))$.

8.37 Algorithm Basic Merkle-Hellman knapsack public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key $(a_1, a_2, \ldots, a_n)$.
   (b) Represent the message m as a binary string of length n, $m = m_1 m_2 \cdots m_n$.
   (c) Compute the integer $c = m_1 a_1 + m_2 a_2 + \cdots + m_n a_n$.
   (d) Send the ciphertext c to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Compute $d = W^{-1} c \bmod M$.
   (b) By solving a superincreasing subset sum problem (Algorithm 8.35), find integers $r_1, r_2, \ldots, r_n$, $r_i \in \{0,1\}$, such that $d = r_1 b_1 + r_2 b_2 + \cdots + r_n b_n$.
   (c) The message bits are $m_i = r_{\pi(i)}$, $i = 1, 2, \ldots, n$.

Proof that decryption works. The decryption of Algorithm 8.37 allows recovery of original plaintext because

$$d \equiv W^{-1} c \equiv W^{-1} \sum_{i=1}^{n} m_i a_i \equiv \sum_{i=1}^{n} m_i b_{\pi(i)} \pmod{M}.$$

Since $0 \le d < M$, $d = \sum_{i=1}^{n} m_i b_{\pi(i)} \bmod M$, and hence the solution of the superincreasing subset sum problem in step (b) of the decryption gives the message bits, after application of the permutation π.

8.38 Example (*basic Merkle-Hellman knapsack encryption with artificially small parameters*)
Key generation. Let $n = 6$. Entity A chooses the superincreasing sequence $(12, 17, 33, 74, 157, 316)$, $M = 737$, $W = 635$, and the permutation π of $\{1,2,3,4,5,6\}$ defined by $\pi(1) = 3$, $\pi(2) = 6$, $\pi(3) = 1$, $\pi(4) = 2$, $\pi(5) = 5$, and $\pi(6) = 4$. A's public key is the knapsack set $(319, 196, 250, 477, 200, 559)$, while A's private key is $(\pi, M, W, (12, 17, 33, 74, 157, 316))$.
Encryption. To encrypt the message $m = 101101$, B computes

$$c = 319 + 250 + 477 + 559 = 1605$$

and sends this to A.
Decryption. To decrypt, A computes $d = W^{-1} c \bmod M = 136$, and solves the superincreasing subset sum problem

$$136 = 12r_1 + 17r_2 + 33r_3 + 74r_4 + 157r_5 + 316r_6$$

to get $136 = 12 + 17 + 33 + 74$. Hence, $r_1 = 1$, $r_2 = 1$, $r_3 = 1$, $r_4 = 1$, $r_5 = 0$, $r_6 = 0$, and application of the permutation π yields the message bits $m_1 = r_3 = 1$, $m_2 = r_6 = 0$, $m_3 = r_1 = 1$, $m_4 = r_2 = 1$, $m_5 = r_5 = 0$, $m_6 = r_4 = 1$. □

Multiple-iterated Merkle-Hellman knapsack encryption

One variation of the basic Merkle-Hellman scheme involves disguising the easy superincreasing sequence by a series of modular multiplications. The key generation for this variation is as follows.

8.39 Algorithm Key generation for multiple-iterated Merkle-Hellman knapsack encryption

SUMMARY: each entity creates a public key and a corresponding private key.

1. Integers n and t are fixed as common system parameters.
2. Each entity A should perform steps 3 – 6.
3. Choose a superincreasing sequence $(a_1^{(0)}, a_2^{(0)}, \dots, a_n^{(0)})$.
4. For j from 1 to t do the following:
   4.1 Choose a modulus M_j with $M_j > a_1^{(j-1)} + a_2^{(j-1)} + \cdots + a_n^{(j-1)}$.
   4.2 Select a random integer W_j, $1 \le W_j \le M_j - 1$, such that $\gcd(W_j, M_j) = 1$.
   4.3 Compute $a_i^{(j)} = a_i^{(j-1)} W_j \bmod M_j$ for $i = 1, 2, \dots, n$.
5. Select a random permutation π of the integers $\{1, 2, \dots, n\}$.
6. A's public key is $(a_1, a_2, \dots, a_n)$, where $a_i = a_{\pi(i)}^{(t)}$ for $i = 1, 2, \dots, n$; A's private key is $(\pi, M_1, \dots, M_t, W_1, \dots, W_t, a_1^{(0)}, a_2^{(0)}, \dots, a_n^{(0)})$.

Encryption is performed in the same way as in the basic Merkle-Hellman scheme (Algorithm 8.37). Decryption is performed by successively computing $d_j = W_j^{-1} d_{j+1} \bmod M_j$ for $j = t, t-1, \dots, 1$, where $d_{t+1} = c$. Finally, the superincreasing subset sum problem $d_1 = r_1 a_1^{(0)} + r_2 a_2^{(0)} + \cdots + r_n a_n^{(0)}$ is solved for r_i, and the message bits are recovered after application of the permutation π.

8.40 Note (*insecurity of Merkle-Hellman knapsack encryption*)

(i) A polynomial-time algorithm for breaking the basic Merkle-Hellman scheme is known. Given the public knapsack set, this algorithm finds a pair of integers U', M' such that U'/M' is close to U/M (where W and M are part of the private key, and $U = W^{-1} \bmod M$) and such that the integers $b_i' = U' a_i \bmod M$, $1 \le i \le n$, form a superincreasing sequence. This sequence can then be used by an adversary in place of $(b_1, b_2, \dots, b_n)$ to decrypt messages.

(ii) The most powerful general attack known on knapsack encryption schemes is the technique discussed in §3.10.2 which reduces the subset sum problem to the problem of finding a short vector in a lattice. It is typically successful if the density (see Definition 3.104) of the knapsack set is less than 0.9408. This is significant because the density of a Merkle-Hellman knapsack set must be less than 1, since otherwise there will in general be many subsets of the knapsack set with the same sum, in which case some ciphertexts will not be uniquely decipherable. Moreover, since each iteration in the multiple-iterated scheme lowers the density, this attack will succeed if the knapsack set has been iterated a sufficient number of times.

Similar techniques have since been used to break most knapsacks schemes that have been proposed, including the multiple-iterated Merkle-Hellman scheme. The most prominent knapsack scheme that has resisted such attacks to date is the Chor-Rivest scheme (but see Note 8.44).

8.6.2 Chor-Rivest knapsack encryption

The Chor-Rivest scheme is the only known knapsack public-key encryption scheme that does not use some form of modular multiplication to disguise an easy subset sum problem.

8.41 Algorithm Key generation for Chor-Rivest public-key encryption

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

1. Select a finite field $\mathbb{F}_q$ of characteristic p, where $q = p^h$, $p \geq h$, and for which the discrete logarithm problem is feasible (see Note 8.45(ii)).
2. Select a random monic irreducible polynomial $f(x)$ of degree h over $\mathbb{Z}_p$ (using Algorithm 4.70). The elements of $\mathbb{F}_q$ will be represented as polynomials in $\mathbb{Z}_p[x]$ of degree less than h, with multiplication performed modulo $f(x)$.
3. Select a random primitive element $g(x)$ of the field $\mathbb{F}_q$ (using Algorithm 4.80).
4. For each ground field element $i \in \mathbb{Z}_p$, find the discrete logarithm $a_i = \log_{g(x)}(x+i)$ of the field element $(x+i)$ to the base $g(x)$.
5. Select a random permutation π on the set of integers $\{0, 1, 2, \ldots, p-1\}$.
6. Select a random integer d, $0 \leq d \leq p^h - 2$.
7. Compute $c_i = (a_{\pi(i)} + d) \bmod (p^h - 1)$, $0 \leq i \leq p-1$.
8. A's public key is $((c_0, c_1, \ldots, c_{p-1}), p, h)$; A's private key is $(f(x), g(x), \pi, d)$.

8.42 Algorithm Chor-Rivest public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key $((c_0, c_1, \ldots, c_{p-1}), p, h)$.
   (b) Represent the message m as a binary string of length $\lfloor \lg \binom{p}{h} \rfloor$, where $\binom{p}{h}$ is a binomial coefficient (Definition 2.17).
   (c) Consider m as the binary representation of an integer. Transform this integer into a binary vector $M = (M_0, M_1, \ldots, M_{p-1})$ of length p having exactly h 1's as follows:
      i. Set $l \leftarrow h$.
      ii. For i from 1 to p do the following:
      If $m \geq \binom{p-i}{l}$ then set $M_{i-1} \leftarrow 1$, $m \leftarrow m - \binom{p-i}{l}$, $l \leftarrow l-1$. Otherwise, set $M_{i-1} \leftarrow 0$. (Note: $\binom{n}{0} = 1$ for $n \geq 0$; $\binom{0}{l} = 0$ for $l \geq 1$.)
   (d) Compute $c = \sum_{i=0}^{p-1} M_i c_i \bmod (p^h - 1)$.
   (e) Send the ciphertext c to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Compute $r = (c - hd) \bmod (p^h - 1)$.
   (b) Compute $u(x) = g(x)^r \bmod f(x)$ (using Algorithm 2.227).
   (c) Compute $s(x) = u(x) + f(x)$, a monic polynomial of degree h over $\mathbb{Z}_p$.
   (d) Factor $s(x)$ into linear factors over $\mathbb{Z}_p$: $s(x) = \prod_{j=1}^{h}(x + t_j)$, where $t_j \in \mathbb{Z}_p$ (cf. Note 8.45(iv)).
   (e) Compute a binary vector $M = (M_0, M_1, \ldots, M_{p-1})$ as follows. The components of M that are 1 have indices $\pi^{-1}(t_j)$, $1 \leq j \leq h$. The remaining components are 0.
   (f) The message m is recovered from M as follows:
      i. Set $m \leftarrow 0$, $l \leftarrow h$.
      ii. For i from 1 to p do the following:
      If $M_{i-1} = 1$ then set $m \leftarrow m + \binom{p-i}{l}$ and $l \leftarrow l-1$.

Proof that decryption works. Observe that

$$\begin{aligned}
u(x) &= g(x)^r \bmod f(x) \\
&\equiv g(x)^{c-hd} \equiv g(x)^{(\sum_{i=0}^{p-1} M_i c_i)-hd} \pmod{f(x)} \\
&\equiv g(x)^{(\sum_{i=0}^{p-1} M_i (a_{\pi(i)}+d))-hd} \equiv g(x)^{\sum_{i=0}^{p-1} M_i a_{\pi(i)}} \pmod{f(x)} \\
&\equiv \prod_{i=0}^{p-1} [g(x)^{a_{\pi(i)}}]^{M_i} \equiv \prod_{i=0}^{p-1} (x+\pi(i))^{M_i} \pmod{f(x)}.
\end{aligned}$$

Since $\prod_{i=0}^{p-1}(x+\pi(i))^{M_i}$ and $s(x)$ are monic polynomials of degree h and are congruent modulo $f(x)$, it must be the case that

$$s(x) = u(x) + f(x) = \prod_{i=0}^{p-1}(x+\pi(i))^{M_i}.$$

Hence, the h roots of $s(x)$ all lie in $\mathbb{Z}_p$, and applying π^{-1} to these roots gives the coordinates of M that are 1.

8.43 Example (*Chor-Rivest public-key encryption with artificially small parameters*)
Key generation. Entity A does the following:

1. Selects $p = 7$ and $h = 4$.
2. Selects the irreducible polynomial $f(x) = x^4 + 3x^3 + 5x^2 + 6x + 2$ of degree 4 over $\mathbb{Z}_7$. The elements of the finite field $\mathbb{F}_{7^4}$ are represented as polynomials in $\mathbb{Z}_7[x]$ of degree less than 4, with multiplication performed modulo $f(x)$.
3. Selects the random primitive element $g(x) = 3x^3 + 3x^2 + 6$.
4. Computes the following discrete logarithms:

$$\begin{aligned}
a_0 &= \log_{g(x)}(x) &&= 1028 \\
a_1 &= \log_{g(x)}(x+1) &&= 1935 \\
a_2 &= \log_{g(x)}(x+2) &&= 2054 \\
a_3 &= \log_{g(x)}(x+3) &&= 1008 \\
a_4 &= \log_{g(x)}(x+4) &&= 379 \\
a_5 &= \log_{g(x)}(x+5) &&= 1780 \\
a_6 &= \log_{g(x)}(x+6) &&= 223.
\end{aligned}$$

5. Selects the random permutation π on $\{0,1,2,3,4,5,6\}$ defined by $\pi(0) = 6$, $\pi(1) = 4$, $\pi(2) = 0$, $\pi(3) = 2$, $\pi(4) = 1$, $\pi(5) = 5$, $\pi(6) = 3$.
6. Selects the random integer $d = 1702$.
7. Computes

$$\begin{aligned}
c_0 &= (a_6 + d) \bmod 2400 = 1925 \\
c_1 &= (a_4 + d) \bmod 2400 = 2081 \\
c_2 &= (a_0 + d) \bmod 2400 = 330 \\
c_3 &= (a_2 + d) \bmod 2400 = 1356 \\
c_4 &= (a_1 + d) \bmod 2400 = 1237 \\
c_5 &= (a_5 + d) \bmod 2400 = 1082 \\
c_6 &= (a_3 + d) \bmod 2400 = 310.
\end{aligned}$$

8. A's public key is $((c_0, c_1, c_2, c_3, c_4, c_5, c_6), p = 7, h = 4)$, while A's private key is $(f(x), g(x), \pi, d)$.

Encryption. To encrypt a message $m = 22$ for A, B does the following:

(a) Obtains authentic A's public key.
(b) Represents m as a binary string of length 5: $m = 10110$. (Note that $\lfloor \lg \binom{7}{4} \rfloor = 5$.)
(c) Uses the method outlined in step 1(c) of Algorithm 8.42 to transform m to the binary vector $M = (1, 0, 1, 1, 0, 0, 1)$ of length 7.
(d) Computes $c = (c_0 + c_2 + c_3 + c_6) \bmod 2400 = 1521$.
(e) Sends $c = 1521$ to A.

Decryption. To decrypt the ciphertext $c = 1521$, A does the following:

(a) Computes $r = (c - hd) \bmod 2400 = 1913$.
(b) Computes $u(x) = g(x)^{1913} \bmod f(x) = x^3 + 3x^2 + 2x + 5$.
(c) Computes $s(x) = u(x) + f(x) = x^4 + 4x^3 + x^2 + x$.
(d) Factors $s(x) = x(x + 2)(x + 3)(x + 6)$ (so $t_1 = 0, t_2 = 2, t_3 = 3, t_4 = 6$).
(e) The components of M that are 1 have indices $\pi^{-1}(0) = 2$, $\pi^{-1}(2) = 3$, $\pi^{-1}(3) = 6$, and $\pi^{-1}(6) = 0$. Hence, $M = (1, 0, 1, 1, 0, 0, 1)$.
(f) Uses the method outlined in step 2(f) of Algorithm 8.42 to transform M to the integer $m = 22$, thus recovering the original plaintext. □

8.44 Note (*security of Chor-Rivest encryption*)

(i) When the parameters of the system are carefully chosen (see Note 8.45 and page 318), there is no feasible attack known on the Chor-Rivest encryption scheme. In particular, the density of the knapsack set $(c_0, c_1, \ldots, c_{p-1})$ is $p/\lg(\max c_i)$, which is large enough to thwart the low-density attacks on the general subset sum problem (§3.10.2).
(ii) It is known that the system is insecure if portions of the private key are revealed, for example, if $g(x)$ and d in some representation of $\mathbb{F}_q$ are known, or if $f(x)$ is known, or if π is known.

8.45 Note (*implementation*)

(i) Although the Chor-Rivest scheme has been described only for the case p a prime, it extends to the case where the base field $\mathbb{Z}_p$ is replaced by a field of prime power order.
(ii) In order to make the discrete logarithm problem feasible in step 1 of Algorithm 8.41, the parameters p and h may be chosen so that $q = p^h - 1$ has only small factors. In this case, the Pohlig-Hellman algorithm (§3.6.4) can be used to efficiently compute discrete logarithms in the finite field $\mathbb{F}_q$.
(iii) In practice, the recommended size of the parameters are $p \approx 200$ and $h \approx 25$. One particular choice of parameters originally suggested is $p = 197$ and $h = 24$; in this case, the largest prime factor of $197^{24} - 1$ is 10316017, and the density of the knapsack set is about 1.077. Other parameter sets originally suggested are $\{p = 211, h = 24\}$, $\{p = 3^5, h = 24\}$ (base field $\mathbb{F}_{3^5}$), and $\{p = 2^8, h = 25\}$ (base field $\mathbb{F}_{2^8}$).
(iv) Encryption is a very fast operation. Decryption is much slower, the bottleneck being the computation of $u(x)$ in step 2b. The roots of $s(x)$ in step 2d can be found simply by trying all possibilities in $\mathbb{Z}_p$.
(v) A major drawback of the Chor-Rivest scheme is that the public key is fairly large, namely, about $(ph \cdot \lg p)$ bits. For the parameters $p = 197$ and $h = 24$, this is about 36000 bits.

(vi) There is message expansion by a factor of $\lg p^h / \lg \binom{p}{h}$. For $p = 197$ and $h = 24$, this is 1.797.

8.7 Probabilistic public-key encryption

A minimal security requirement of an encryption scheme is that it must be difficult, in essentially all cases, for a passive adversary to recover plaintext from the corresponding ciphertext. However, in some situations, it may be desirable to impose more stringent security requirements.

The RSA, Rabin, and knapsack encryption schemes are *deterministic* in the sense that under a fixed public key, a particular plaintext m is always encrypted to the same ciphertext c. A deterministic scheme has some or all of the following drawbacks.

1. The scheme is not secure for all probability distributions of the message space. For example, in RSA the messages 0 and 1 always get encrypted to themselves, and hence are easy to detect.
2. It is sometimes easy to compute partial information about the plaintext from the ciphertext. For example, in RSA if $c = m^e \bmod n$ is the ciphertext corresponding to a plaintext m, then
$$\left(\frac{c}{n}\right) = \left(\frac{m^e}{n}\right) = \left(\frac{m}{n}\right)^e = \left(\frac{m}{n}\right)$$
since e is odd, and hence an adversary can easily gain one bit of information about m, namely the Jacobi symbol $\left(\frac{m}{n}\right)$.
3. It is easy to detect when the same message is sent twice.

Of course, any deterministic encryption scheme can be converted into a randomized scheme by requiring that a portion of each plaintext consist of a randomly generated bitstring of a pre-specified length l. If the parameter l is chosen to be sufficiently large for the purpose at hand, then, in practice, the attacks listed above are thwarted. However, the resulting randomized encryption scheme is generally not provably secure against the different kinds of attacks that one could conceive.

Probabilistic encryption utilizes randomness to attain a *provable* and very strong level of security. There are two strong notions of security that one can strive to achieve.

8.46 Definition A public-key encryption scheme is said to be *polynomially secure* if no passive adversary can, in expected polynomial time, select two plaintext messages m_1 and m_2 and then correctly distinguish between encryptions of m_1 and m_2 with probability significantly greater than $\frac{1}{2}$.

8.47 Definition A public-key encryption scheme is said to be *semantically secure* if, for all probability distributions over the message space, whatever a passive adversary can compute in expected polynomial time about the plaintext given the ciphertext, it can also compute in expected polynomial time without the ciphertext.

Intuitively, a public-key encryption scheme is semantically secure if the ciphertext does not leak any partial information whatsoever about the plaintext that can be computed in expected polynomial time.

8.48 Remark (*perfect secrecy vs. semantic security*) In Shannon's theory (see §1.13.3(i)), an encryption scheme has *perfect secrecy* if a passive adversary, even with infinite computational resources, can learn nothing about the plaintext from the ciphertext, except possibly its length. The limitation of this notion is that perfect secrecy cannot be achieved unless the key is at least as long as the message. By contrast, the notion of semantic security can be viewed as a polynomially bounded version of perfect secrecy — a passive adversary with polynomially bounded computational resources can learn nothing about the plaintext from the ciphertext. It is then conceivable that there exist semantically secure encryption schemes where the keys are much shorter that the messages.

Although Definition 8.47 appears to be stronger than Definition 8.46, the next result asserts that they are, in fact, equivalent.

8.49 Fact A public-key encryption scheme is semantically secure if and only if it is polynomially secure.

8.7.1 Goldwasser-Micali probabilistic encryption

The Goldwasser-Micali scheme is a probabilistic public-key system which is semantically secure assuming the intractability of the quadratic residuosity problem (see §3.4).

8.50 Algorithm Key generation for Goldwasser-Micali probabilistic encryption

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

1. Select two large random (and distinct) primes p and q, each roughly the same size.
2. Compute $n = pq$.
3. Select a $y \in \mathbb{Z}_n$ such that y is a quadratic non-residue modulo n and the Jacobi symbol $\left(\frac{y}{n}\right) = 1$ (y is a pseudosquare modulo n); see Remark 8.54.
4. A's public key is (n, y); A's private key is the pair (p, q).

8.51 Algorithm Goldwasser-Micali probabilistic public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key (n, y).
   (b) Represent the message m as a binary string $m = m_1 m_2 \cdots m_t$ of length t.
   (c) For i from 1 to t do:
      i. Pick an $x \in \mathbb{Z}_n^*$ at random.
      ii. If $m_i = 1$ then set $c_i \leftarrow y x^2 \bmod n$; otherwise set $c_i \leftarrow x^2 \bmod n$.
   (d) Send the t-tuple $c = (c_1, c_2, \ldots, c_t)$ to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) For i from 1 to t do:
      i. Compute the Legendre symbol $e_i = \left(\frac{c_i}{p}\right)$ (using Algorithm 2.149).
      ii. If $e_i = 1$ then set $m_i \leftarrow 0$; otherwise set $m_i \leftarrow 1$.
   (b) The decrypted message is $m = m_1 m_2 \cdots m_t$.

Proof that decryption works. If a message bit m_i is 0, then $c_i = x^2 \bmod n$ is a quadratic residue modulo n. If a message bit m_i is 1, then since y is a pseudosquare modulo n, $c_i = yx^2 \bmod n$ is also a pseudosquare modulo n. By Fact 2.137, c_i is a quadratic residue modulo n if and only if c_i is a quadratic residue modulo p, or equivalently $\left(\frac{c_i}{p}\right) = 1$. Since A knows p, she can compute this Legendre symbol and hence recover the message bit m_i.

8.52 Note (*security of Goldwasser-Micali probabilistic encryption*) Since x is selected at random from $\mathbb{Z}_n^*$, $x^2 \bmod n$ is a random quadratic residue modulo n, and $yx^2 \bmod n$ is a random pseudosquare modulo n. Hence, an eavesdropper sees random quadratic residues and pseudosquares modulo n. Assuming that the quadratic residuosity problem is difficult, the eavesdropper can do no better that guess each message bit. More formally, if the quadratic residuosity problem is hard, then the Goldwasser-Micali probabilistic encryption scheme is semantically secure.

8.53 Note (*message expansion*) A major disadvantage of the Goldwasser-Micali scheme is the message expansion by a factor of $\lg n$ bits. Some message expansion is unavoidable in a probabilistic encryption scheme because there are many ciphertexts corresponding to each plaintext. Algorithm 8.56 is a major improvement of the Goldwasser-Micali scheme in that the plaintext is only expanded by a constant factor.

8.54 Remark (*finding pseudosquares*) A pseudosquare y modulo n can be found as follows. First find a quadratic non-residue a modulo p and a quadratic non-residue b modulo q (see Remark 2.151). Then use Gauss's algorithm (Algorithm 2.121) to compute the integer y, $0 \le y \le n-1$, satisfying the simultaneous congruences $y \equiv a \pmod{p}$, $y \equiv b \pmod{q}$. Since y ($\equiv a \pmod{p}$) is a quadratic non-residue modulo p, it is also a quadratic non-residue modulo n (Fact 2.137). Also, by the properties of the Legendre and Jacobi symbols (§2.4.5), $\left(\frac{y}{n}\right) = \left(\frac{y}{p}\right)\left(\frac{y}{q}\right) = (-1)(-1) = 1$. Hence, y is a pseudosquare modulo n.

8.7.2 Blum-Goldwasser probabilistic encryption

The Blum-Goldwasser probabilistic public-key encryption scheme is the most efficient probabilistic encryption scheme known and is comparable to the RSA encryption scheme, both in terms of speed and message expansion. It is semantically secure (Definition 8.47) assuming the intractability of the integer factorization problem. It is, however, vulnerable to a chosen-ciphertext attack (see Note 8.58(iii)). The scheme uses the Blum-Blum-Shub generator (§5.5.2) to generate a pseudorandom bit sequence which is then XORed with the plaintext. The resulting bit sequence, together with an encryption of the random seed used, is transmitted to the receiver who uses his trapdoor information to recover the seed and subsequently reconstruct the pseudorandom bit sequence and the plaintext.

8.55 Algorithm Key generation for Blum-Goldwasser probabilistic encryption

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

1. Select two large random (and distinct) primes p, q, each congruent to 3 modulo 4.
2. Compute $n = pq$.
3. Use the extended Euclidean algorithm (Algorithm 2.107) to compute integers a and b such that $ap + bq = 1$.
4. A's public key is n; A's private key is (p, q, a, b).

8.56 Algorithm Blum-Goldwasser probabilistic public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption.* B should do the following:
   (a) Obtain A's authentic public key n.
   (b) Let $k = \lfloor \lg n \rfloor$ and $h = \lfloor \lg k \rfloor$. Represent the message m as a string $m = m_1 m_2 \cdots m_t$ of length t, where each m_i is a binary string of length h.
   (c) Select as a seed x_0, a random quadratic residue modulo n. (This can be done by selecting a random integer $r \in \mathbb{Z}_n^*$ and setting $x_0 \leftarrow r^2 \bmod n$.)
   (d) For i from 1 to t do the following:
      i. Compute $x_i = x_{i-1}^2 \bmod n$.
      ii. Let p_i be the h least significant bits of x_i.
      iii. Compute $c_i = p_i \oplus m_i$.
   (e) Compute $x_{t+1} = x_t^2 \bmod n$.
   (f) Send the ciphertext $c = (c_1, c_2, \ldots, c_t, x_{t+1})$ to A.
2. *Decryption.* To recover plaintext m from c, A should do the following:
   (a) Compute $d_1 = ((p+1)/4)^{t+1} \bmod (p-1)$.
   (b) Compute $d_2 = ((q+1)/4)^{t+1} \bmod (q-1)$.
   (c) Compute $u = x_{t+1}^{d_1} \bmod p$.
   (d) Compute $v = x_{t+1}^{d_2} \bmod q$.
   (e) Compute $x_0 = vap + ubq \bmod n$.
   (f) For i from 1 to t do the following:
      i. Compute $x_i = x_{i-1}^2 \bmod n$.
      ii. Let p_i be the h least significant bits of x_i.
      iii. Compute $m_i = p_i \oplus c_i$.

Proof that decryption works. Since x_t is a quadratic residue modulo n, it is also a quadratic residue modulo p; hence, $x_t^{(p-1)/2} \equiv 1 \pmod p$. Observe that

$$x_{t+1}^{(p+1)/4} \equiv (x_t^2)^{(p+1)/4} \equiv x_t^{(p+1)/2} \equiv x_t^{(p-1)/2} x_t \equiv x_t \pmod p.$$

Similarly, $x_t^{(p+1)/4} \equiv x_{t-1} \pmod p$ and so

$$x_{t+1}^{((p+1)/4)^2} \equiv x_{t-1} \pmod p.$$

Repeating this argument yields

$$u \equiv x_{t+1}^{d_1} \equiv x_{t+1}^{((p+1)/4)^{t+1}} \equiv x_0 \pmod p.$$

Analogously,

$$v \equiv x_{t+1}^{d_2} \equiv x_0 \pmod q.$$

Finally, since $ap + bq = 1$, $vap + ubq \equiv x_0 \pmod p$ and $vap + ubq \equiv x_0 \pmod q$. Hence, $x_0 = vap + ubq \bmod n$, and A recovers the same random seed that B used in the encryption, and consequently also recovers the original plaintext.

8.57 Example (*Blum-Goldwasser probabilistic encryption with artificially small parameters*) *Key generation.* Entity A selects the primes $p = 499$, $q = 547$, each congruent to 3 modulo 4, and computes $n = pq = 272953$. Using the extended Euclidean algorithm, A computes

the integers $a = -57$, $b = 52$ satisfying $ap + bq = 1$. A's public key is $n = 272953$, while A's private key is (p, q, a, b).
Encryption. The parameters k and h have the values 18 and 4, respectively. B represents the message m as a string $m_1m_2m_3m_4m_5$ ($t = 5$) where $m_1 = 1001$, $m_2 = 1100$, $m_3 = 0001$, $m_4 = 0000$, $m_5 = 1100$. B then selects a random quadratic residue $x_0 = 159201$ ($= 399^2 \bmod n$), and computes:

i	$x_i = x_{i-1}^2 \bmod n$	p_i	$c_i = p_i \oplus m_i$
1	180539	1011	0010
2	193932	1100	0000
3	245613	1101	1100
4	130286	1110	1110
5	40632	1000	0100

and $x_6 = x_5^2 \bmod n = 139680$. B sends the ciphertext

$$c = (0010, 0000, 1100, 1110, 0100, 139680)$$

to A.
Decryption. To decrypt c, A computes

$$\begin{aligned} d_1 &= ((p+1)/4)^6 \bmod (p-1) = 463 \\ d_2 &= ((q+1)/4)^6 \bmod (q-1) = 337 \\ u &= x_6^{463} \bmod p \qquad = 20 \\ v &= x_6^{337} \bmod q \qquad = 24 \\ x_0 &= vap + ubq \bmod n \qquad = 159201. \end{aligned}$$

Finally, A uses x_0 to construct the x_i and p_i just as B did for encryption, and recovers the plaintext m_i by XORing the p_i with the ciphertext blocks c_i. □

8.58 Note (*security of Blum-Goldwasser probabilistic encryption*)

(i) Observe first that n is a Blum integer (Definition 2.156). An eavesdropper sees the quadratic residue x_{t+1}. Assuming that factoring n is difficult, the h least significant bits of the principal square root x_t of x_{t+1} modulo n are simultaneously secure (see Definition 3.82 and Fact 3.89). Thus the eavesdropper can do no better than to guess the pseudorandom bits p_i, $1 \le i \le t$. More formally, if the integer factorization problem is hard, then the Blum-Goldwasser probabilistic encryption scheme is semantically secure. Note, however, that for a modulus n of a fixed bitlength (e.g., 1024 bits), this statement is no longer true, and the scheme should only be considered computationally secure.

(ii) As of 1996, the modulus n should be at least 1024 bits in length if long-term security is desired (cf. Note 8.7). If n is a 1025-bit integer, then $k = 1024$ and $h = 10$.

(iii) As with the Rabin encryption scheme (Algorithm 8.11), the Blum-Goldwasser scheme is also vulnerable to a chosen-ciphertext attack that recovers the private key from the public key. It is for this reason that the Blum-Goldwasser scheme has not received much attention in practice.

8.59 Note (*efficiency of Blum-Goldwasser probabilistic encryption*)

(i) Unlike Goldwasser-Micali encryption, the ciphertext in Blum-Goldwasser encryption is only longer than the plaintext by a constant number of bits, namely $k + 1$ (the size in bits of the integer x_{t+1}).

(ii) The encryption process is quite efficient — it takes only 1 modular multiplication to encrypt h bits of plaintext. By comparison, the RSA encryption process (Algorithm 8.3) requires 1 modular exponentiation ($m^e \bmod n$) to encrypt k bits of plaintext. Assuming that the parameter e is randomly chosen and assuming that an (unoptimized) modular exponentiation takes $3k/2$ modular multiplications, this translates to an encryption rate for RSA of $2/3$ bits per modular multiplication. If one chooses a special value for e, such as $e = 3$ (see Note 8.9), then RSA encryption is faster than Blum-Goldwasser encryption.

(iii) Blum-Goldwasser decryption (step 2 of Algorithm 8.56) is also quite efficient, requiring 1 exponentiation modulo $p-1$ (step 2a), 1 exponentiation modulo $q-1$ (step 2b), 1 exponentiation modulo p (step 2c), 1 exponentiation modulo q (step 2d), and t multiplications modulo n (step 2f) to decrypt ht ciphertext bits. (The time to perform step 2e is negligible.) By comparison, RSA decryption (step 2 of Algorithm 8.3) requires 1 exponentiation modulo n (which can be accomplished by doing 1 exponentiation modulo p and 1 exponentiation modulo q) to decrypt k ciphertext bits. Thus, for short messages ($< k$ bits), Blum-Goldwasser decryption is slightly slower than RSA decryption, while for longer messages, Blum-Goldwasser is faster.

8.7.3 Plaintext-aware encryption

While semantic security (Definition 8.47) is a strong security requirement for public-key encryption schemes, there are other measures of security.

8.60 Definition A public-key encryption scheme is said to be *non-malleable* if given a ciphertext, it is computationally infeasible to generate a different ciphertext such that the respective plaintexts are related in a known manner.

8.61 Fact If a public-key encryption scheme is non-malleable, it is also semantically secure.

Another notion of security is that of being plaintext-aware. In Definition 8.62, *valid ciphertext* means those ciphertext which are the encryptions of legitimate plaintext messages (e.g. messages containing pre-specified forms of redundancy).

8.62 Definition A public-key encryption scheme is said to be *plaintext-aware* if it is computationally infeasible for an adversary to produce a valid ciphertext without knowledge of the corresponding plaintext.

In the "random oracle model", the property of being plaintext-aware is a strong one — coupled with semantic security, it can be shown to imply that the encryption scheme is non-malleable and also secure against adaptive chosen-ciphertext attacks. Note 8.63 gives one method of transforming any k-bit to k-bit trapdoor one-way permutation (such as RSA) into an encryption scheme that is plaintext-aware and semantically secure.

8.63 Note (*Bellare-Rogaway plaintext-aware encryption*) Let f be a k-bit to k-bit trapdoor one-way permutation (such as RSA). Let k_0 and k_1 be parameters such that 2^{k_0} and 2^{k_1} steps each represent infeasible amounts of work (e.g., $k_0 = k_1 = 128$). The length of the plaintext m is fixed to be $n = k - k_0 - k_1$ (e.g., for $k = 1024$, $n = 768$). Let $G : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{n+k_1}$ and $H : \{0,1\}^{n+k_1} \longrightarrow \{0,1\}^{k_0}$ be random functions. Then the encryption function, as depicted in Figure 8.1, is

$$E(m) = f(\{m0^{k_1} \oplus G(r)\} \parallel \{r \oplus H(m0^{k_1} \oplus G(r))\}),$$

where $m0^{k_1}$ denotes m concatenated with a string of 0's of bitlength k_1, r is a random binary string of bitlength k_0, and $\|$ denotes concatenation.

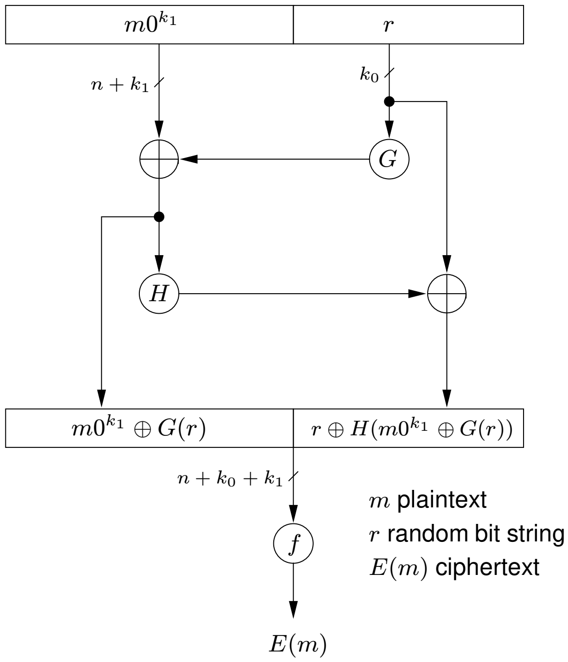

Figure 8.1: *Bellare-Rogaway plaintext-aware encryption scheme.*

Under the assumption that G and H are random functions, the encryption scheme E of Note 8.63 can be proven to be plaintext-aware and semantically secure. In practice, G and H can be derived from a cryptographic hash function such as the Secure Hash Algorithm (§9.4.2(iii)). In this case, the encryption scheme can no longer be proven to be plaintext-aware because the random function assumption is not true; however, such a scheme appears to provides greater security assurances than those designed using ad hoc techniques.

8.8 Notes and further references

§8.1

For an introduction to public-key cryptography and public-key encryption in particular, see §1.8. A particularly readable introduction is the survey by Diffie [343]. Historical notes on public-key cryptography are given in the notes to §1.8 on page 47. A comparison of the features of public-key and symmetric-key encryption is given in §1.8.4; see also §13.2.5.

Other recent proposals for public-key encryption schemes include those based on finite automata (Renji [1032]); hidden field equations (Patarin [965]); and isomorphism of polynomials (Patarin [965]).

§8.2

The RSA cryptosystem was invented in 1977 by Rivest, Shamir, and Adleman [1060]. Kaliski and Robshaw [655] provide an overview of the major attacks on RSA encryption and signatures, and the practical methods of counteracting these threats.

The computational equivalence of computing the decryption exponent d and factoring n (§8.2.2(i)) was shown by Rivest, Shamir and Adleman [1060], based on earlier work by Miller [876].

The attack on RSA with small encryption exponent (§8.2.2(ii)) is discussed by Håstad [544], who showed more generally that sending the encryptions of more than $e(e+1)/2$ *linearly related messages* (messages of the form $(a_i m + b_i)$, where the a_i and b_i are known) enables an eavesdropper to recover the messages provided that the moduli n_i satisfy $n_i > 2^{(e+1)(e+2)/4}(e+1)^{(e+1)}$. Håstad also showed that sending three linearly related messages using the Rabin public-key encryption scheme (Algorithm 8.11) is insecure.

The attack on RSA with small decryption exponent d (§8.2.2(iv)) is due to Wiener [1240]. Wiener showed that his attack can be avoided if the encryption exponent e is chosen to be at least 50% longer than the modulus n. In this case, d should be at least 160 bits in length to avoid the square-root discrete logarithm algorithms such as Pollard's rho algorithm (Algorithm 3.60) and the parallelized variant of van Oorschot and Wiener [1207].

The adaptive chosen-ciphertext attack on RSA encryption (§8.2.2(v)) is due to Davida [302]. See also the related discussion in Denning [327]. Desmedt and Odlyzko [341] described an indifferent chosen-ciphertext attack in which the adversary has to obtain the plaintext corresponding to about $L_n[\frac{1}{2}, \frac{1}{2}]$ carefully chosen-ciphertext, subsequent to which it can decrypt all further ciphertext in $L_n[\frac{1}{2}, \frac{1}{2}]$ time without having to use the authorized user's decryption machine.

The common modulus attacks on RSA (§8.2.2(vi)) are due to DeLaurentis [320] and Simmons [1137].

The cycling attack (§8.2.2(vii)) was proposed by Simmons and Norris [1151]. Shortly after, Rivest [1052] showed that the cycling attack is extremely unlikely to succeed if the primes p and q are chosen so that: (i) $p-1$ and $q-1$ have large prime factors p' and q', respectively; and (ii) $p'-1$ and $q'-1$ have large prime factors p'' and q'', respectively. Maurer [818] showed that condition (ii) is unnecessary. Williams and Schmid [1249] proposed the generalized cycling attack and showed that this attack is really a factoring algorithm. Rivest [1051] provided heuristic evidence that if the primes p and q are selected at random, each having the same bitlength, then the expected time before the generalized cycling attack succeeds is at least $p^{1/3}$.

The note on message concealing (§8.2.2(viii)) is due to Blakley and Borosh [150], who also extended this work to all composite integers n and determined the number of *deranging* exponents for a fixed n, i.e., exponents e for which the number of unconcealed messages is the minimum possible. For further work see Smith and Palmer [1158].

Suppose that two or more plaintext messages which have a (known) polynomial relationship (e.g. m_1 and m_2 might be *linearly related*: $m_1 = am_2 + b$) are encrypted with the same small encryption exponent (e.g. $e = 3$ or $e = 2^{16} + 1$). Coppersmith et al. [277] presented a new class of attacks on RSA which enable a passive adversary to recover such plaintext from the corresponding ciphertext. This attack is of practical significance because various cryptographic protocols have been proposed which require the encryption of polynomially related messages. Examples include the key distribution protocol of Tatebayashi, Matsuzaki, and Newman [1188], and the verifiable signature scheme of Franklin and Reiter [421]. Note that these attacks are different from those of §8.2.2(ii) and §8.2.2(vi) where the same plaintext is encrypted under different public keys.

Coppersmith [274] presented an efficient algorithm for finding a root of a polynomial of degree k over $\mathbb{Z}_n$, where n is an RSA-like modulus, provided that there there is a root smaller than $n^{1/k}$. The algorithm yielded the following two attacks on RSA with small encryption exponents. If $e = 3$ and if an adversary knows a ciphertext c and more than $2/3$ of the plaintext m corresponding to c, then the adversary can efficiently recover the rest of m. Suppose now that messages are padded with random bitstrings and encrypted with exponent $e = 3$. If an adversary knows two ciphertexts c_1 and c_2 which correspond to two encryptions of the same message m (with different padding), then the adversary can efficiently recovery m, provided that the padding is less than $1/9$ of the length of n. The latter attack suggests that caution must be exercised when using random padding in conjunction with a small encryption exponent.

Let $n = pq$ be a k-bit RSA modulus, where p and q are $k/2$-bit primes. Coppersmith [273] showed how n can be factored in polynomial time if the high order $k/4$ bits of p are known. This improves an algorithm of Rivest and Shamir [1058], which requires knowledge of the high order $k/3$ bits of p. For related theoretical work, see Maurer [814]. One implication of Coppersmith's result is that the method of Vanstone and Zuccherato [1214] for generating RSA moduli having a predetermined set of bits is insecure.

A trapdoor in the RSA cryptosystem was proposed by Anderson [26] whereby a hardware device generates the RSA modulus $n = pq$ in such a way that the hardware manufacturer can easily factor n, but factoring n remains difficult for all other parties. However, Kaliski [652] subsequently showed how to efficiently detect such trapdoors and, in some cases, to actually factor the modulus.

The arguments and recommendations about the use of strong primes in RSA key generation (Note 8.8) are taken from the detailed article by Rivest [1051].

Shamir [1117] proposed a variant of the RSA encryption scheme called *unbalanced RSA*, which makes it possible to enhance security by increasing the modulus size (e.g. from 500 bits to 5000 bits) without any deterioration in performance. In this variant, the public modulus n is the product of two primes p and q, where one prime (say q) is significantly larger in size than the other; plaintext messages m are in the interval $[0, p-1]$. For concreteness, consider the situation where p is a 500-bit prime, and q is a 4500-bit prime. Factoring such a 5000-bit modulus n is well beyond the reach of the special-purpose elliptic curve factoring algorithm of §3.2.4 (whose running time depends on the size of the smallest prime factor of n) and general-purpose factoring algorithms such as the number field sieve of §3.2.7. Shamir recommends that the encryption exponent e be in the interval $[20, 100]$, which makes the encryption time with a 5000-bit modulus comparable to the decryption time with a 500-bit modulus. Decryption of the ciphertext c ($= m^d \bmod n$) is accomplished by computing $m_1 = c^{d_1} \bmod p$, where $d_1 = d \bmod (p-1)$. Since $0 \le m < p$, m_1 is in fact equal to m. Decryption in unbalanced RSA thus only involves one exponentiation modulo a 500-bit prime, and takes the same time as decryption in ordinary RSA with a 500-bit modulus. This optimization does not apply to the RSA signature scheme (§11.3.1), since the verifier does not know the factor p of the public modulus n.

A *permutation polynomial* of $\mathbb{Z}_n$ is a polynomial $f(x) \in \mathbb{Z}_n[x]$ which induces a permutation of $\mathbb{Z}_n$ upon substitution of the elements of $\mathbb{Z}_n$; that is, $\{f(a) | a \in \mathbb{Z}_n\} = \mathbb{Z}_n$. In RSA encryption the permutation polynomial x^e of $\mathbb{Z}_n$ is used, where $\gcd(e, \phi) = 1$. Müller and Nöbauer [910] suggested replacing the polynomial x^e by the so-called *Dickson polynomials* to create a modified RSA encryption scheme called the *Dickson scheme*. The Dickson scheme was further studied by Müller and Nöbauer [909]. Other suitable classes of permutation polynomials were investigated by Lidl and Müller [763]. Smith and Lennon [1161] proposed an analogue of the RSA cryptosystem called LUC which is based on Lucas sequences. Due to the relationships between Dickson polynomials and the Lucas sequences, the LUC cryptosystem is closely related to the Dickson scheme. Bleichenbacher, Bosma, and Lenstra [154] presented a chosen-message attack on the LUC signature scheme, undermining the primary advantage claimed for LUC over RSA. Pinch [976, 977] extended the attacks on RSA with small encryption exponent (§8.2.2(ii)) and small decryption exponent (§8.2.2(iv)) to the LUC system.

An analogue of the RSA cryptosystem which uses special kinds of elliptic curves over $\mathbb{Z}_n$, where n is a composite integer, was proposed by Koyama et al. [708]. Demytko [321] presented an analogue where there is very little restriction on the types of elliptic curves that can be used. A new cryptosystem based on elliptic curves over $\mathbb{Z}_n$ in which the message is held in the exponent instead of the group element was proposed by Vanstone and Zuccherato [1213]. The security of all these schemes is based on the difficulty of factoring n. Kurosawa, Okada, and Tsujii [721] showed that the encryption schemes of Koyama et al. and Demytko are vulnerable to low exponent attacks (cf. §8.2.2(ii)); Pinch [977] demonstrated that the attack on RSA with small decryption exponent d (§8.2.2(iv)) also extends to these schemes. Kaliski [649] presented a chosen-ciphertext attack on the Demytko encryption scheme (and also a chosen-message attack on the corresponding signature scheme), and concluded that the present benefits of elliptic curve cryptosystems based on a composite modulus do not seem significant.

§8.3

The Rabin public-key encryption scheme (Algorithm 8.11) was proposed in 1979 by Rabin [1023]. In Rabin's paper, the encryption function was defined to be $E(m) = m(m + b) \bmod n$, where b and n comprise the public key. The security of this scheme is equivalent to the security of the scheme described in Algorithm 8.11 with encryption function $E(m) = m^2 \bmod n$. A related digital signature scheme is described in §11.3.4. Schwenk and Eisfeld [1104] consider public-key encryption and signature schemes whose security relies on the intractability of factoring polynomials over $\mathbb{Z}_n$.

Williams [1246] presented a public-key encryption scheme similar in spirit to Rabin's but using composite integers $n = pq$ with primes $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$. Williams' scheme also has the property that breaking it (that is, recovering plaintext from some given ciphertext) is equivalent to factoring n, but has the advantage over Rabin's scheme that there is an easy procedure for identifying the intended message from the four roots of a quadratic polynomial. The restrictions on the forms of the primes p and q were removed later by Williams [1248]. A simpler and more efficient scheme also having the properties of provable security and unique decryption was presented by Kurosawa, Ito, and Takeuchi [720]. As with Rabin, all these schemes are vulnerable to a chosen-ciphertext attack (but see Note 8.14).

It is not the case that all public-key encryption schemes for which the decryption problem is provably as difficult as recovering the private key from the public key must succumb to a chosen-ciphertext attack. Goldwasser, Micali, and Rivest [484] were the first to observe this, and presented a digital signature scheme provably secure against an adaptive chosen-ciphertext attack (see §11.6.4). Naor and Yung [921] proposed the first concrete public-key encryption scheme that is semantically secure against *indifferent* chosen-ciphertext attack. The Naor-Yung scheme uses two independent keys of a probabilistic public-encryption scheme that is secure against a passive adversary (for example, the Goldwasser-Micali scheme of Algorithm 8.51) to encrypt the plaintext, and then both encryptions are sent along with a non-interactive zero-knowledge proof that the same message was encrypted with both keys. Following this work, Rackoff and Simon [1029] gave the first concrete construction for a public-key encryption scheme that is semantically secure against an *adaptive* chosen-ciphertext attack. Unfortunately, these schemes are all impractical because of the degree of message expansion.

Damgård [297] proposed simple and efficient methods for making public-key encryption schemes secure against indifferent chosen-ciphertext attacks. Zheng and Seberry [1269] noted that Damgård's schemes are insecure against an adaptive chosen-ciphertext attack, and proposed three practical schemes intended to resist such an attack. The Damgård and Zheng-Seberry schemes were not proven to achieve their claimed levels of security. Bellare and Rogaway [93] later proved that one of the Zheng-Seberry schemes is provably secure against adaptive chosen-ciphertext attacks for their *random oracle model*. Lim and Lee [766] proposed another method for making public-key schemes secure against adaptive chosen-ciphertext attacks; this scheme was broken by Frankel and Yung [419].

§8.4

The ElGamal cryptosystem was invented by ElGamal [368]. Haber and Lenstra (see Rueppel et al. [1083]) raised the possibility of a trapdoor in discrete logarithm cryptosystems whereby a modulus p is generated (e.g., by a hardware manufacturer) that is intentionally "weak"; cf. Note 4.58. Here, a "weak" prime p is one for which the discrete logarithm problem in $\mathbb{Z}_p^*$ is relatively easy. For example, $p-1$ may contain only small prime factors, in which case the Pohlig-Hellman algorithm (§3.6.4) would be especially effective. Another example is a prime p for which the number field sieve for discrete logarithms (page 128) is especially well-suited. However, Gordon [509] subsequently showed how such trapdoors can be easily detected. Gordon also showed that the probability of a randomly chosen prime possessing such a trapdoor is negligibly small.

Rivest and Sherman [1061] gave an overview and unified framework for randomized encryption, including comments on chosen-plaintext and chosen-ciphertext attacks.

Elliptic curves were first proposed for use in public-key cryptography by Koblitz [695] and Miller [878]. Recent work on the security and implementation of elliptic curve systems is reported by Menezes [840]. Menezes, Okamoto, and Vanstone [843] showed that if the elliptic curve belongs to a special family called *supersingular curves*, then the discrete logarithm problem in the elliptic curve group can be reduced in expected polynomial time to the discrete logarithm problem in a small extension of the underlying finite field. Hence, if a supersingular elliptic curve is desired in practice, then it should be carefully chosen.

A modification of ElGamal encryption employing the group of units $\mathbb{Z}_n^*$, where n is a composite integer, was proposed by McCurley [825]; the scheme has the property that breaking it is *provably* at least as difficult as factoring the modulus n (cf. Fact 3.80). If a cryptanalyst somehow learns the factors of n, then in order to recover plaintext from ciphertext it is still left with the task of solving the Diffie-Hellman problem (§3.7) modulo the factors of n.

Hyperelliptic curve cryptosystems were proposed by Koblitz [696] but little research has since been done regarding their security and practicality.

The possibility of using the class group of an imaginary quadratic number field in public-key cryptography was suggested by Buchmann and Williams [218], however, the attractiveness of this choice was greatly diminished after the invention of a subexponential-time algorithm for computing discrete logarithms in these groups by McCurley [826].

Smith and Skinner [1162] proposed analogues of the Diffie-Hellman key exchange (called LUCDIF) and ElGamal encryption and digital signature schemes (called LUCELG) which use Lucas sequences modulo a prime p instead of modular exponentiation. Shortly thereafter, Laih, Tu, and Tai [733] and Bleichenbacher, Bosma, and Lenstra [154] showed that the analogue of the discrete logarithm problem for Lucas functions polytime reduces to the discrete logarithm problem in the multiplicative group of the finite field $\mathbb{F}_{p^2}$. Since there are subexponential-time algorithms known for the discrete logarithm problem in these fields (cf. §3.6), LUCDIF and LUCELG appear not to offer any advantages over the original schemes.

§8.5

The McEliece encryption scheme (Algorithm 8.30) was introduced in 1978 by McEliece [828]. For information on Goppa codes and their decoding algorithms, see MacWilliams and Sloane [778]. The problem of decoding an arbitrary linear code was shown to be **NP**-hard by Berlekamp, McEliece, and van Tilborg [120]. The security of the McEliece scheme has been studied by Adams and Meijer [6], Lee and Brickell [742], van Tilburg [1212], Gibson [451], and by Chabaud [235]. Gibson showed that there are, in fact, many trapdoors to a given McEliece encryption transformation, any of which may be used for decryption; this is contrary to the results of Adams and Meijer. However, Gibson notes that there are probably sufficiently few trapdoors that finding one by brute force is computationally infeasible. The cryptanalytic attack reported by Korzhik and Turkin [707] has not been published in its entirety, and is not believed to be an effective attack.

The strength of the McEliece encryption scheme can be severely weakened if the Goppa code is replaced with another type of error-correcting code. For example, Gabidulin, Paramonov, and Tretjakov [435] proposed a modification which uses maximum-rank-distance (MRD) codes in place of Goppa codes. This scheme, and a modification of it by Gabidulin [434], were subsequently shown to be insecure by Gibson [452, 453].

§8.6

The basic and multiple-iterated Merkle-Hellman knapsack encryption schemes (§8.6.1) were introduced by Merkle and Hellman [857]. An elementary overview of knapsack systems is given by Odlyzko [941].

The first polynomial-time attack on the basic Merkle-Hellman scheme (cf. Note 8.40(i)) was devised by Shamir [1114] in 1982. The attack makes use of H. Lenstra's algorithm for integer programming which runs in polynomial time when the number of variables is fixed, but is inefficient in practice. Lagarias [723] improved the practicality of the attack by reducing the main portion of the procedure to a problem of finding an unusually good simultaneous diophantine approximation; the latter can be solved by the more efficient L^3-lattice basis reduction algorithm (§3.10.1). The first attack on the multiple-iterated Merkle-Hellman scheme was by Brickell [200]. For surveys of the cryptanalysis of knapsack schemes, see Brickell [201] and Brickell and Odlyzko [209]. Orton [960] proposed a modification to the multiple-iterated Merkle-Hellman scheme that permits a knapsack density approaching 1, thus avoiding currently known attacks. The high density also allows for a fast digital signature scheme.

Shamir [1109] proposed a fast signature scheme based on the knapsack problem, later broken by Odlyzko [939] using the L^3-lattice basis reduction algorithm.

The Merkle-Hellman knapsack scheme illustrates the limitations of using an **NP**-complete problem to design a secure public-key encryption scheme. Firstly, Brassard [190] showed that under reasonable assumptions, the problem faced by the cryptanalyst cannot be **NP**-hard unless **NP=co-NP**, which would be a very surprising result in computational complexity theory. Secondly, complexity theory is concerned primarily with *asymptotic* complexity of a problem. By contrast, in practice one works with a problem instance of a *fixed* size. Thirdly, **NP**-completeness is a measure of the *worst-case* complexity of a problem. By contrast, cryptographic security should depend on the *average-case* complexity of the problem (or even better, the problem should be intractable for *essentially all* instances), since the cryptanalyst's task should be hard for virtually all instances and not merely in the worst case. There are many **NP**-complete problems that are known to have polynomial-time average-case algorithms, for example, the graph coloring problem; see Wilf [1243]. Another interesting example is provided by Even and Yacobi [379] who describe a symmetric-key encryption scheme based on the subset sum problem for which breaking the scheme (under a chosen-plaintext attack) is an **NP**-hard problem, yet an algorithm exists which solves most instances in polynomial time.

The Chor-Rivest knapsack scheme (Algorithm 8.42) was proposed by Chor and Rivest [261]. Recently, Schnorr and Hörner [1100] introduced new algorithms for lattice basis reduction that are improvements on the L^3-lattice basis reduction algorithm (Algorithm 3.101), and used these to break the Chor-Rivest scheme with parameters $\{p = 103, h = 12\}$. Since the density of such knapsack sets is 1.271, the attack demonstrated that subset sum problems with density greater than 1 can be solved via lattice basis reduction. Schnorr and Hörner also reported some success solving Chor-Rivest subset sum problems with parameters $\{p = 151, h = 16\}$. It remains to be seen whether the techniques of Schnorr and Hörner can be successfully applied to the recommended parameter case $\{p = 197, h = 24\}$.

Depending on the choice of parameters, the computation of discrete logarithms in the Chor-Rivest key generation stage (step 4 of Algorithm 8.41) may be a formidable task. A modified version of the scheme which does not require the computation of discrete logarithms in a field was proposed by H. Lenstra [758]. This modified scheme is called the *powerline system* and is not a knapsack system. It was proven to be at least as secure as the original Chor-Rivest scheme, and is comparable in terms of encryption and decryption speeds.

Qu and Vanstone [1013] showed how the Merkle-Hellman knapsack schemes can be viewed as special cases of certain knapsack-like encryption schemes arising from subset factorizations of finite groups. They also proposed an efficient public-key encryption scheme based on subset factorizations of the additive group $\mathbb{Z}_n$ of integers modulo n. Blackburn, Murphy, and Stern [143] showed that a simplified variant which uses subset factorizations of the n-dimensional vector space $\mathbb{Z}_2^n$ over $\mathbb{Z}_2$ is insecure.

§8.7

The notion of probabilistic public-key encryption was conceived by Goldwasser and Micali [479], who also introduced the notions of polynomial and semantic security. The equivalence of these two notions (Fact 8.49) was proven by Goldwasser and Micali [479] and Micali, Rackoff, and Sloan [865]. Polynomial security was also studied by Yao [1258], who referred to it as *polynomial-time indistinguishability*.

The Goldwasser-Micali scheme (Algorithm 8.51) can be described in a general setting by using the notion of a trapdoor predicate. Briefly, a *trapdoor predicate* is a Boolean function $B : \{0,1\}^* \longrightarrow \{0,1\}$ such that given a bit v it is easy to choose an x at random satisfying $B(x) = v$. Moreover, given a bitstring x, computing $B(x)$ correctly with probability significantly greater than $\frac{1}{2}$ is difficult; however, if certain trapdoor information is known, then it is easy to compute $B(x)$. If entity A's public key is a trapdoor predicate B, then any other entity encrypts a message bit m_i by randomly selecting an x_i such that $B(x_i) = m_i$, and then sends x_i to A. Since A knows the trapdoor information, she can compute $B(x_i)$ to recover m_i, but an adversary can do no better than guess the value of m_i. Goldwasser and Micali [479] proved that if trapdoor predicates exist, then this probabilistic encryption scheme is polynomially secure. Goldreich and Levin [471] simplified the work of Yao [1258], and showed how any trapdoor length-preserving permutation f can be used to obtain a trapdoor predicate, which in turn can be used to construct a probabilistic public-key encryption scheme.

The Blum-Goldwasser scheme (Algorithm 8.56) was proposed by Blum and Goldwasser [164]. The version given here follows the presentation of Brassard [192]. Two probabilistic public-key encryption schemes, one whose breaking is equivalent to solving the RSA problem (§3.3), and the other whose breaking is equivalent to factoring integers, were proposed by Alexi et al. [23]. The scheme based on RSA is as follows. Let $h = \lfloor \lg \lg n \rfloor$, where (n, e) is entity A's RSA public key. To encrypt an h-bit message m for A, choose a random $y \in \mathbb{Z}_n^*$ such that the h least significant bits of y equal m, and compute the ciphertext $c = y^e \bmod n$. A can recover m by computing $y = c^d \bmod n$, and extracting the h least significant bits of y. While both the schemes proposed by Alexi et al. are more efficient than the Goldwasser-Micali scheme, they suffer from large message expansion and are consequently not as efficient as the Blum-Goldwasser scheme.

The idea of non-malleable cryptography (Definition 8.60) was introduced by Dolev, Dwork, and Naor [357], who also observed Fact 8.61. The paper gives the example of two contract bidders who encrypt their bids. It should not be possible for one bidder A to see the encrypted bid of the other bidder B and somehow be able to offer a bid that was slightly lower, even if A may not know what the resulting bid actually is at that time. Bellare and Rogaway [95] introduced the notion of plaintext-aware encryption (Definition 8.62). They presented the scheme described in Note 8.63, building upon earlier work of Johnson et al. [639]. Rigorous definitions and security proofs were provided, as well as a concrete instantiation of the plaintext-aware encryption scheme using RSA as the trapdoor permutation, and constructing the random functions G and H from the SHA-1 hash function (§9.4.2(iii)). Johnson and Matyas [640] presented some enhancements to the plaintext-aware encryption scheme. Bellare and Rogaway [93] presented various techniques for deriving appropriate random functions from standard cryptographic hash functions.