2

Security Mechanisms

This chapter deals with security mechanisms that can be used to realize information security services. It first explains which cryptographic systems or cryptosystems are suitable for implementation and then describes the most widely used ones in detail.

2.1 Data Integrity Mechanisms

One way to protect data integrity is to use an encryption mechanism (e.g., DES in CBC mode, see Section 2.2). In this way both data integrity and data confidentiality are ensured. Unfortunately, encryption alone is not secure enough because of the possibility of *bit flipping* attacks [1]. If no authentication is provided, an attacker can flip bits in the ciphertext (i.e., exchange "0" for "1" or vice versa) without being detected. If the encrypted plaintext is not a human-readable message but a string automatically processed by a running program, the result from decryption of the altered ciphertext can potentially be interpreted in such a way as to cause serious damage to the program or the receiving host. The protection is either to add some authentication information to the plaintext before encryption or, if only integrity protection is required, to send the original message together with the ciphertext.

Another way to ensure integrity is to use a digital signature mechanism (see Section 2.3). Digital signatures provide not only data integrity but also

nonrepudiation. If only data integrity is desired, without confidentiality or nonrepudiation, it can be achieved by applying a message authentication code (MAC) based on a *cryptographic hash function* to the data to be protected (see Section 2.1.2). In general, cryptographic hash functions are very fast—far faster than encryption mechanisms.

2.1.1 Cryptographic Hash Functions

If a cryptographic hash function is applied to an input value of any length (up to a maximum possible length, for example 2^{64} for SHA-1), the resulting output value will always be of a constant length (for example, 160 bit for SHA-1). This fixed-length output is referred to as the *message digest* or *checksum,* or *hashsum*. Since the set of all possible inputs is much larger than the set of all possible outputs, many different input values will be mapped to the same output value. However, it should be rendered computationally expensive to find different inputs that are mapped to the same output. In other words, the function must be made easy to compute in one direction (i.e., h: *input* → *output*), but not in the opposite direction. For this reason, cryptographic hash functions are often referred to as the *one-way (hash) functions*. Strictly speaking, a cryptographic hash function $y = h(x)$ must satisfy the following conditions:

It is computationally infeasible to find

(a) x such that $h(x) = y$, for any given y

(b) $y \neq x$ such that $h(x) = h(y)$, for any given x

(c) (x,y) such that $h(x) = h(y)$

In general, there are two serious types of attacks against cryptographic hash functions. The first consists in finding a message *M' yielding the same hashsum as the original message M*. Such an attack can be very dangerous where a digital signature is generated from the shorter hashsum instead of from the longer message. This is usually done as a matter of convenience, for generating a signature is a time- and resource-consuming task. As an example, suppose that *A* edited a message *M* and signed the hashsum $h(M)$, *M* being a bank order to transfer 100 euros to *B*'s account. If condition (b) were not satisfied, *B* could easily find another message *M' so that* $h(M) = h(M')$, in which 10,000 euros instead of 100 euros would be transferred. If condition (a) were satisfied, however, this type of attack would be extremely time consuming even for short hashsums.

The second type of attack is much more serious. This is when *B* tries to find two messages, *M* and *M'*, that yield the same hashsum but have completely different meanings. Suppose *B* wants *A* to transfer 10,000 euros to *B*'s account. *B* knows that *A* would never agree to transfer more than 100 euros, so it is necessary somehow for *B* to obtain *A*'s signature on the home-banking order. Note that in this case *B* has much more freedom, since there are many different ways to say that *A* wants to give *B* 100 euros, or 10,000 euros. Therefore the probability of finding two suitable messages is significantly higher than in the first attack, in which one of the messages is given. Actually, the probability is quite surprisingly higher, which is often referred to as the *birthday paradox*.

2.1.1.1 Birthday Paradox

The birthday paradox can be explained in terms of a hash function with people as inputs and birthdays as outputs—thus, *h*(*person*) = *birthday*.

There are over five billion people on our planet, and only 366 different birthdays. The first type of attack goes as follows: Given a particular person *A*, how many randomly chosen people must be asked for their birthdays until there is a probability higher than 50% that one of them has the same birthday as *A*? The answer is 183. The second type of attack (birthday attack) needs the smallest group of randomly chosen people for which there is a probability higher than 50% that at least two people in the group have the same birthday. This group needs only 23 people.

In terms of cryptographic hash functions, the first attack would require hundreds of thousands of years of computing time, while the second attack would be a matter of hours, at least for short (less than 100-bit) hashsums. For this reason it is of crucial importance to use a cryptographic hash function that not only satisfies the conditions (a) - (b), but also produces outputs that are long enough to make the birthday attack infeasible with current technology.

The most popular cryptographic hash function family is the MD (message digest) family developed by R. Rivest. MD5, which is specified in a Request for Comments (RFC) document issued by the Internet Engineering Task Force [2],[1] is the latest member of the family. Since it has a 128-bit output, it is potentially vulnerable to a birthday attack and therefore not considered secure enough for the latest technology (it also has some structural problems).

1. http://www.ieft.org

SHA-1 (Secure Hash Standard) is a much better choice since it produces a 160-bit output [3]. It is based on principles similar to those used by R. Rivest when designing MD4 and MD5. The input message can be up to 2^{64} bits long. It is divided into 512-bit blocks that are sequentially processed in such a way that the hashsum depends on all input blocks. A block consists of 16 *words*. Words are basic processing units on which the following operations are performed:

- Bitwise logical "and," "inclusive-or," "exclusive-or," and "complement";
- Addition modulo 2^{32};
- Circular left shift.

SHA-1 additionally uses some carefully chosen constants. The computation requires two *buffers* with five 32-bit words each, and a sequence of eighty 32-bit words. The standard describes two methods of computation, one of which requires less memory than the other, but longer execution time. Implementers can make use of these possibilities to trade off memory against execution time.

2.1.2 Message Authentication Code

Cryptographic hash functions can be used to implement a *data authentication* mechanism. Data authentication is a combination of authentication and data integrity. The so-called MAC is computed in the following way:

$$\text{MAC}(message) = f(Secret\ Key, message)$$

in which $f()$ is a function based on a specific combination of the cryptographic hash functions. If a sender and a receiver both know the secret key, the receiver can check the sender authenticity and the message integrity by applying the combination of known cryptographic hash functions to the secret key and the message. The first proposal for MAC computation was simply to apply a cryptographic hash function $h()$ to the concatenation of the secret key and the message, that is, to compute $h(Secret\ Key, message)$ or $h(message, Secret\ Key)$. Unfortunately, that approach proved to be insecure [4].[2] A combined approach was to prefix and suffix two different secret keys

2. See CRYPTO/EUROCRYPT papers at http://www.cryptography.com/resources/papers/index.htm

and compute *h*(*Secret Key 1*, *message*, *Secret Key 2*). This approach is much more secure, but there is an attack, although impractical, that makes it possible to find the secret keys. The best approach so far is to apply an iterated hash function [4], for example h[*secret key*, h(*secret key*, *message*)], and use some padding. This approach was chosen as mandatory to implement for many Internet security protocols [5], such as IPsec and SSL/TLS.

2.2 Encryption Mechanisms

A data confidentiality service can be implemented with encryption mechanisms. A cryptographic system, or *cryptosystem*, is a single parameter family $\{E_K\}_{K \in \mathcal{K}}$ of invertible transformations

$$E_K : \mathcal{M} \rightarrow \mathcal{C}$$

from a space M of *plaintext* (or unencrypted) messages to a space C of *ciphertext* (or encrypted) messages. The cryptographic key K is selected from a finite set $\mathcal{K}$ called the *keyspace*. Basically, there are two types of cryptosystems, namely *symmetric* or *secret key* systems, and *asymmetric* or *public key* systems. The inverse transformation $(E_K)^{-1}$ is denoted by D_K. E_K is referred to as encryption and D_K as decryption.

2.2.1 Symmetric Mechanisms

In a symmetric cryptosystem, the encryption and decryption transformations are identical or easily derived from each other. If the message to be encrypted (plaintext) is denoted by M, the encrypted message (cyphertext) by C, and the cryptographic key by K, the symmetric encryption E and decryption D can be defined as follows:

$$E_K(M) = C$$

$$D_K(C) = M$$

In a symmetric cryptosystem the same key is used for both encryption and decryption. This key is called the *secret key* since it must remain secret to everybody except the message sender(s) and the message receiver(s). Obviously, it is necessary that the receiver obtain not only the encrypted message, but also the corresponding key. The encrypted message may be sent over an

insecure communication channel—after all, that is why it needs to be encrypted. The key, however, must not be sent over the same channel, and this leads to a serious problem of symmetric cryptosystems: *key management*. The secret key must either be sent over a separate, secure channel (e.g., a sealed envelope), or it must be sent encrypted. For the encryption of symmetric keys in transfer, a public key mechanism can be used (see Section 2.2.2).

2.2.1.1 One-Time Pad

Encryption techniques are much older than computers. In fact, one of the earliest known encryption techniques was used by the Roman dictator Julius Caesar (100–44 B.C.). In the Caesar Cipher, each plaintext character of the Latin alphabet is replaced by the character three positions to the right of it ("A" is replaced by "D," "B" by "E," etc.). The one-time pad is also a classic technique. Invented by Gilbert Vernam in 1917 and improved by Major Joseph Mauborgne, it was originally used for spy messages.

The one-time pad is very important for cryptography because it is the only *perfect* encryption scheme known. In other words, the ciphertext yields absolutely no information about the plaintext except its length [6]. The definition of perfect secrecy given by C. E. Shannon in 1943 is actually younger than the one-time pad. It turns out that perfect secrecy requires that

- The encryption key be *at least as long* as the message to be encrypted;
- Each key be used only *once*.

This is exactly the case with the one-time pad. Unfortunately, it makes key management extremely difficult, since new keys must be exchanged each time.

The one-time pad key is a large, nonrepeating set of truly random key letters. The encryption is the addition modulo 26 of one plaintext character and one one-time pad key character. Plaintext characters are mapped to numbers corresponding to their positions in the English alphabet. The one-time pad is a symmetric mechanism, since the same key is used for both encryption and decryption. For example,

Plaintext:	M	E	S	S	A	G	E
Key:	T	B	F	R	G	F	A
Ciphertext:	G	G	Y	K	H	M	F

because

M+T mod 26 = 13+20 mod 26 = 7 = G

E+B mod 26 = 5+2 mod 26 = 7 = G

S+F mod 26 = 19+6 mod 26 = 25 = Y

and so on.

Decryption works the other way around, that is by subtracting the letters of the ciphertext and the letters of the key modulo 26:

G–T mod 26 = 7–20 mod 26 = –13 mod 26 = 13 = M

G–B mod 26 = 7–2 mod 26 = 5 = E

Y–F mod 26 = 25–6 mod 26 = 19 = S

and so on.

2.2.1.2 Data Encryption Standard

The Data Encryption Standard (DES) was developed in the United States by IBM and NIST (the National Institute of Standards and Technology[3]) in 1976. DES is standardized as the Data Encryption Algorithm (DEA) by ANSI (the American National Standards Institute[4]) [7], and as DEA-1 by ISO[5] [8]. Its main advantage, apart from not yet being broken by cryptoanalysts despite its age, is that it can be easily and efficiently implemented in hardware. More information on the background of DES can be found in [6].

DES is a block cipher since it encrypts data in 64-bit blocks. If data is longer, it must be divided into 64-bit blocks. It may happen that the last part of some data is shorter than 64 bits. In such a case it is usual to fill the remaining part of the block with zeros (*padding*). The result of DES encryption is also a 64-bit block. The key has 56 bits and 8 parity bits. The same algorithm is used for both encryption and decryption, but with reverse key ordering.

DES Techniques

The main cryptographic techniques applied in DES are *confusion* and *diffusion*. Both techniques were known long before DES, but in DES they were

3. http://www.csrc.nist.gov

4. http://www.ansi.org

5. http://www.iso.ch

combined for the first time in such a way as to result in an encryption algorithm that has withstood all cryptoanalysts' attacks for twenty-four years now.

The purpose of *confusion* is to obscure the relationship between the plaintext and the ciphertext. Substitution is an example of a confusion technique. However, if one encrypts an English text simply by substituting, for example, letter K for letter A, then someone analyzing the ciphertext can easily conclude that K stands for A by comparing the relative frequency of K in the ciphertext with the well-known relative letter frequencies for English. There are better substitution techniques that can change the probabilities to some extent, but in general, substitution alone is not sufficiently secure.

In DES, substitution is done not with letters, but with bit strings. DES has eight different substitution tables called *S-boxes*. Each S-box uses a 6-bit input and a 4-bit output. An S-box is a table with 4 rows (0–3) and 16 (0–15) columns. Each entry in the table is a 4-bit binary number. For example, the S-box No.1 is shown in Table 2.1.

The substitution is defined as follows: To determine the row in an S-box, take the first and the last bit of the input. The middle four bits yield the column. The output (substitution result) is the entry at the intersection of the row and the column. For example:

S-box No. 1

Input: 110011;

The first and the last bit are 11 $\Rightarrow$ row 3;

The middle four bits are 1001 $\Rightarrow$ column 9;

Output: the number in row 3, column 9 is $11_{10} = 1011_2$.

Table 2.1
DES S-Box No. 1

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-boxes are crucial for DES security, although substitution is generally a weak technique. The S-boxes are nonlinear and therefore very difficult to analyze. It was not until 1992 that the design criteria for the S-boxes were even published. Actually, it is possible to find better S-boxes than the DES S-boxes, but it is not an easy task.

Diffusion dissipates the redundancy of the plaintext by spreading it out over the ciphertext. An example of a diffusion technique is permutation. A very simple permutation of the word MESSAGE is SMEEGAS. In this example, the key is 2317654, meaning: Move the first letter to the second position, move the second letter to the third position, etc. In DES there are several permutations. The *initial permutation,* for example, begins as follows:

58, 50, 42, 34, 26, 18, 10, 2, 60, 52…,

meaning:

move bit 58 of the plaintext to bit position 1,

move bit 50 of the plaintext to bit position 2,

and so on.

Another type of permutation used in DES is the *expansion permutation,* which, as the name says, yields a longer output than the input. In this way the dependency of the output bits on the input bits can occur at an earlier stage in the DES computation. A small change in either the plaintext or the key produces a significant change in the ciphertext, which is referred to as the *avalanche effect*. Without this effect it would be easy to observe the propagation of changes from the plaintext to the ciphertext, which would make cryptoanalysis easier.

DES Rounds

DES has sixteen *rounds*. A simplified DES computation is shown in Figure 2.1. In each round, a 48-bit subkey computed by the *compression permutation* is XORed (i.e., added modulo 2) to the right half of the data expanded to 48 bits by the expansion permutation. The result is fed into the S-boxes. The result of the S-box substitution is permuted once more (*P-box permutation*). Before the first round the data is permuted with the initial permutation. After the last round, the intermediate result is permuted for the last time. This *final permutation* is the inverse of the initial permutation. These two permutations do not affect DES's security, however.

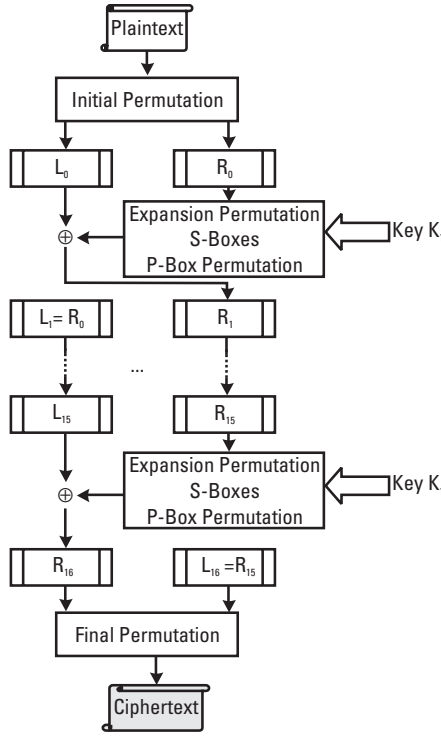

Figure 2.1 DES.

Like many other symmetric block ciphers, DES is also a Feistel network [6]. The name comes from Horst Feistel, who first proposed such a network in the early 1970s. In a Feistel network the plaintext is divided into two halves for the first round of computation, which is repeated a number of times (i.e., in the subsequent rounds). Generally, the output of the ith round is determined from the output of the previous round in the following way:

$$L_i = R_{i-1}$$

$$R_i = L_i \oplus f(R_{i-1}, K_i)$$

where $f()$ represents the round function, and K_i the key for the ith round.

Triple DES

Since DES is, in contrast to the one-time pad, not perfectly secure and thus vulnerable to a brute-force attack (the trying of all possible keys), key length

plays a significant security role. Nowadays it is not recommended to use DES with a 56-bit key. The algorithm itself does not allow varying key lengths, but it can be applied more than once with *different keys,* which effectively means using a longer key. This is possible because DES is not an algebraic group, as was proven by Campbell and Wiener in 1992. If one takes a 64-bit input and applies all possible DES keys to encrypt it, there will be $2^{56} < 10^{17}$ different 64-bit outputs. However, there are $2^{64}! > 10^{10^{20}}$ possible 64-bit outputs [9]. In other words, most of the outputs are "unused" by one DES key. This effectively means that for the given keys K_1 and K_2 there is usually no key K_3 such that $E_{K_2}(E_{K_1}(M)) = E_{K_3}(M)$. One can therefore conclude that multiple DES encryption is stronger than single DES encryption. Surprisingly, however, double DES is not much stronger than single DES because of the *meet-in-the-middle attack* [9]. Triple DES was finally adopted as a stronger variant of DES, even if only two different keys, K_1 and K_2, are used:

$$C = E_{K_1}\left(D_{K_2}\left(E_{K_1}(M)\right)\right)$$

D instead of E in the middle of the expression is introduced for compatibility with single DES. In other words, if triple DES encryption is defined as a new function with two parameters $E3(K_1, K_2)$, then $E3(K_1, K_1)$ represents single DES encryption.

DES Modes

DES, like all other block ciphers, can be applied in several different *modes*, for example

- Electronic codebook (ECB) mode;
- Cipher-block chaining (CBC) mode;
- Cipher feedback (CFB) mode;
- Output feedback (OFB) mode;
- Counter mode.

ECB is the fastest and easiest mode. In this mode each plaintext block is encrypted independently from other blocks. It is, however, the least secure mode, since identical plaintext blocks result in identical ciphertext blocks such that block redundancies in the plaintext can easily be detected. CBC

solves this problem by introducing *feedback*. Each plaintext block P_i is "chained" to the encryption result C_{i-1} of the previous plaintext block P_{i-1}:

$$\text{Encryption: } C_i = E_K(P_i \oplus C_{i-1})$$
$$\text{Decryption: } P_i = C_{i-1} \oplus D_K(C_i)$$

The first plaintext block is chained to an *initialization vector* (IV) known to both the sender and the receiver (i.e., $C_1 = E_K(P_1 \oplus IV)$). Sometimes it is necessary to encrypt data units smaller than the block size, for example, if there is no time to wait for enough data to fill a block. In such cases CFB is used, which also adds feedback and requires an IV. With OFB, most of the encryption process can occur off-line, before the plaintext message even exists. With both CFB and OFB, a block cipher is actually used as a *stream* cipher. Unlike block ciphers, stream ciphers convert plaintext to ciphertext one bit or byte at a time.

If it is necessary to encrypt data units smaller than the block size, block ciphers can also be applied in counter mode. In counter mode, sequence numbers or pseudorandom sequences are used as the input to the encryption algorithm.

DES Today

The fastest DES chips today achieve an encryption speed of approximately 1 Gbps with a 56-bit key. The fastest software solutions are much slower, about 10 Mbps.

The latest record in cracking DES (as of September 1999), set by the Electronic Frontier Foundation's "Deep Crack" is 22 hours and 15 minutes [10]. It involved about 100,000 PCs on the Internet. It was performed as a "known ciphertext attack" based on a challenge from the RSA Laboratories.[6] The task was to find a 56-bit DES key for a given plaintext and a given ciphertext.

2.2.1.3 Other Symmetric Encryption Algorithms

IDEA (International Data Encryption Algorithm), proposed in 1992, was the "European answer to DES" and to the United States export restrictions on cryptographic algorithms. IDEA is a block cipher that encrypts a 64-bit plaintext block with a 128-bit key. It applies the same basic cryptographic techniques as DES (confusion and diffusion), but is twice as fast. Its "disadvantages" are that it has not been cryptoanalyzed as long as DES, and that it

6. http://www.rsasecurity.com/rslabs/

is patented and must be licensed for commercial use. The patent holder is the Swiss company ASCOM.[7]

RC (Rivest Cipher) is a family of symmetric algorithms. RC2 is a variable-key-size 64-bit block cipher that was designed as a possible replacement for DES. RC2 and RC4 with a 40-bit key were used in the Netscape implementation of SSL (Secure Sockets Layer) since they were the first cryptographic algorithms allowed for export from the United States. However, in 1995 Doligez successfully cracked RC4 (a stream cipher) with a 40-bit key in less than 32 hours by a brute-force attack.[8] RC5 is a block cipher with a variable block size, key size, and number of rounds. The latest algorithm in the series is RC6, an improved version of RC5, which was submitted by RSA Laboratories, Inc. as a candidate for the Advanced Encryption Standard in April 1998.

2.2.1.4 Advanced Encryption Standard

The designation Advanced Encryption Standard (AES,[9] will replace DES. RC6, MARS, Rijndael, Serpent, and Twofish are the five finalist AES candidate algorithms that are currently (as of November 1999) being analyzed by the global cryptographic community.

RC6[10] by Rivest et al. is a parameterized family of encryption algorithms. As DES, it is based on a Feistel network. The parameters are word size, number of rounds, and key length. The version submitted as an AES candidate operates with 32-bit words and has 20 rounds. Software implementations in ANSI C on a 200 MHz Pentium achieve a rate of about 45 Mbps. Hardware implementation estimates are about 1.3 Gbps.

MARS is a block cipher supporting 128-bit blocks and variable key size developed at IBM Research.[11] It is also a Feistel network, but offers better security than triple DES. Hardware implementations are approximately 10 times faster than software implementations in C, which achieve about 65 Mbps on a 200 MHz Pentium-Pro.

7. http://www.ascom.ch/infosec/idea/licensing.html
8. http://www.pauillac.inria.fr/~doligez/ssl/
9. http://www.csrc.nist.gov/encryption/aes/aes_home.htm
10. http://www.rsa.com/rsalabs.aes/rc6vll.pdf
11. http://www.research.ibm.com/security/mars.html

Rijndael, a block cipher by Joan Daemen and Vincent Rijmen[12] has a variable block length and key length. Currently (as of November 1999) it is specified how to use keys with a length of 128, 192, or 256 bits to encrypt blocks with a length of 128, 192 or 256 bits. Rijndael is not a Feistel network, but defines a round as a composition of three distinct invertible uniform transformations, called "layers." A C implementation with a 128-bit key and 128-bit block has a rate of about 30 to 70 Mbps on a 200 MHz Pentium. In dedicated hardware, rates of 1 Gbps and higher could be achieved.

Serpent is a 128-bit block cipher designed by Ross Anderson, Eli Biham, and Lars Knudsen.[13] The currently fastest C version runs at about 26 Mbps on a 200 MHz Pentium, which is comparable to DES, but the designers believe it to be more secure than triple DES. Serpent's structure is very similar to DES. It has 32 rounds and uses stronger S-boxes.

Twofish is a 128-bit block cipher (a 16-round Feistel network) proposed by Schneier[14] that accepts a variable-length key up to 256 bits. For a 256-bit key, the throughput achieved on a 200 MHz Pentium is about 45 Mbps for C implementations. The hardware performance is up to about 1.2 Gbps with a 150 MHz clock.

2.2.2 Public Key Mechanisms

The problem of key management in symmetric cryptosystems was successfully solved by the introduction of public key cryptosystems. These are often explained with the mailbox analogy as illustrated in Figure 2.2. The mailbox represents the *public key*, since anyone can throw a letter into it. However, only the mailbox owner has the mailbox key—the *private key*—with which she can open the mailbox and take out the letter.

In a public key cryptosystem, the encryption and decryption keys differ in such a way that it is not computationally feasible to derive one key from the other. One key is referred to as the *private key* and must be kept secret. Another key is referred to as the *public key* and should be made public, which eliminates the necessity of transmitting it in a secure way. The public key encryption transformation E_{PuK} and decryption transformation D_{PrK} are denoted as

12. http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

13. http://www.cl.cam.ac.uk/ftp/users/rja14/serpent.pdf

14. http://www.counterpane.com/twofish/pdf

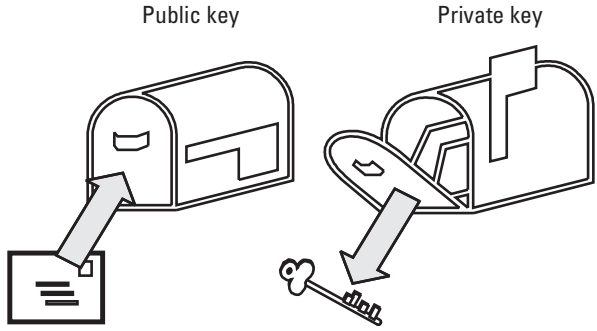

Figure 2.2 Mailbox as an analogy to a public key cryptosystem.

$$E_{PuK}(M) = C$$

$$D_{PrK}(C) = D_{PrK}\left(E_{PuK}(M)\right) = M$$

The encryption transformation E is uniquely determined through the public key *PuK*, so it is usual to write $E_{OwnerID}$ (ID stands for "identity"). The same applies to the decryption transformation, which is usually written $D_{OwnerID}$.

The pioneers of public key cryptography are W. Diffie and M. E. Hellman [11], who invented one of the first two public key cryptosystems (the second, by Merkle and Hellman, was based on the knapsack problem, but it was cracked a long time ago).

2.2.2.1 RSA

RSA is the most famous and widely used public key system. It was invented in 1978 by R. Rivest, A. Shamir, and L. Adleman [12], whose family names' initials form the name of the algorithm. The difficulty of breaking RSA is based on the factoring problem. However, it has never been mathematically proven that it is equally difficult to factor a large composite number as to break RSA.

In RSA, the large composite number is referred to as the modulus $n = pq$, *p and q* being large primes. Public key or public exponent e can be chosen as a prime number relatively prime to $(p-1)(q-1)$. Private key or private exponent d is then chosen to satisfy the following congruence:

$$ed \equiv 1 \bmod \phi(n) \qquad \text{(Eq. 2.1)}$$

To understand the congruence, we must first review some simple rules from modular arithmetic and *number theory* in general. Modular arithmetic operates with *residues* (represented by *r*):

$$a \bmod n = r \Rightarrow a = qn + r, 0 \le r < n \qquad \text{(Eq. 2.2)}$$

For example, 35 mod 4 = 3 since 35 = 8*4 + 3. All possible residues modulo 4 are {0,1,2,3}.

Like the nonmodular arithmetic everyone is familiar with, modular arithmetic is commutative, associative, and distributive with respect to addition and multiplication, that is,

$$(a + b) \bmod n = (a \bmod n) + (b \bmod n) = (b + a) \bmod n;$$
$$(ab) \bmod n = (a \bmod n)(b \bmod n) = (ba) \bmod n;$$
$$[(a + b) + c] \bmod n = (a \bmod n) + (b \bmod n) + (c \bmod n) =$$
$$[a + (b + c)] \bmod n;$$
$$[(ab)c] \bmod n = (a \bmod n)(b \bmod n)(c \bmod n) = [a(bc)] \bmod n;$$
$$[(a + b)c] \bmod n =$$
$$[(a \bmod n) + (b \bmod n)](c \bmod n) = (ac) \bmod n + (bc) \bmod n$$

Two integers *a* and *b* can be *congruent* ("≡") modulo *n*, that is,

$$a \equiv b \bmod n \Rightarrow n | (a - b)$$

"*a* | *b*" means "*a* divides *b*," or "*b* is a multiple of *a*" (for example, 2 divides 8). In other words, if two integers *a* and *b* have equal residues modulo *n*, they are also congruent modulo *n*:

$$(a \bmod n) = (b \bmod n) \Rightarrow a \equiv b \bmod n \qquad \text{(Eq. 2.3)}$$

For example, 35 and 59 are congruent modulo 4 since 35 mod 4 = 59 mod 4 = 3.

To determine the private RSA exponent *d*, one must compute the modular inverse of the public exponent *e*. To find the modular inverse means finding *x* such that

$$ax \bmod n = 1$$

However, if a and n are not relatively prime, there is no solution (gcd stands for "greatest common divisor"):

$$2x \bmod 14 = 1 \quad \text{no solution for } x \text{ since } \gcd(2,14) \neq 1$$

To compute the modular inverse, the number of positive integers less than the modulus and relatively prime to the modulus is needed. This number is usually referred to as *Euler's Totient Function* f(n). For p prime, $\phi(p) = p - 1$. For the RSA modulus $n = pq$,

$$\phi(n) = (p-1)(q-1)$$

Given $\phi(n)$, the inverse modulo n of any number relatively prime to n can be computed in the following way:

$$ax \bmod n = 1 \Rightarrow x = a^{\phi(n)-1} \bmod n, \quad \text{in which } \gcd(a,n) = 1$$

$$a^{\phi(n)} \bmod n = 1, \text{ if } \gcd(a,n) = 1 \qquad \text{(Eq. 2.4)}$$

For example, one can compute x from $5x \bmod 6 = 1$ in the following way:

$$\phi(n = 6 = 2 \times 3) = (2-1)(3-1) = 2$$
$$x = 5^{2-1} \bmod 6 = 5(5 \times 5 \bmod 6 = 1)$$

This result comes from Euler's generalization of Fermat's Little Theorem (FLT). FLT gives the formula for computing inverses modulo a prime:

$$ax \bmod p = 1 \Rightarrow x = a^{p-2} \bmod n \quad \text{in which } p \text{ prime and } \gcd(a,p) = 1$$

$$a^{p-1} \bmod p = 1 \quad \text{if } p \text{ prime and } \gcd(a,p) = 1$$

To compute d in RSA, one must first find the inverse modulo $\phi(n)$. RSA encryption and decryption are defined as

$$\text{encryption } C = M^e \bmod n$$

$$\text{Decryption } M = C^d \bmod n = M^{ed} \bmod n = M$$

M is the message to be encrypted (plaintext) and C is ciphertext. If the decryption equation is divided by M, the result is

$$M^{ed} \bmod n = M \text{ / divide by } M$$

$$M^{ed-1} \bmod n = 1$$

Comparing this equation with the formula for computing the modular inverse from Euler's generalization of FLT (2.4) shows that $(ed - 1)$ must be a multiple of $\phi(n)$, or, in other words, that $\phi(n) \mid (ed - 1)$. As we already know from (2.2), this condition can be expressed as

$$ed \equiv 1 \bmod \phi(n)$$

which is the RSA congruence from the beginning of this section (2.1).

There is one more confusing aspect to examine. That is, (2.4) requires that M and n be relatively prime. How can that be guaranteed? It can happen that a message does not satisfy this condition (i.e., that either $\gcd(M,n) = p$ or $\gcd(M,n) = q$). Luckily, the RSA formula holds even in such cases. The proof for $\gcd(M,n) = p$ is as follows: Let $M = cp$. It holds that $M^{\phi(q)} \bmod q = 1$ since $\gcd(M,q) = 1$ (see FLT):

$$M^{\phi(q)} \bmod q = 1 \text{ /}^{\phi(p)}$$

$$\left[M^{\phi(q)}\right]^{\phi(q)} \bmod q = M^{\phi(n)} \bmod q = 1 \Rightarrow$$

$$M^{\phi(n)} = 1 + kq \quad \text{/multiply by } M = cp$$

$$M^{\phi(n)+1} = M + kcpq = M + kcn$$

$$M^{\phi(n)+1} \equiv M \bmod n$$

$$M^{\phi(n)} \equiv 1 \bmod n$$

Since $\phi(n) \mid (ed - 1)$, the following holds true:

$$M^{ed-1} \equiv 1 \bmod n$$

$$M^{ed-1} \bmod n = 1 \quad /\text{multiply by } M$$

$$M^{ed} \bmod n = M$$

and this is the RSA decryption.

Primality Test

For RSA it is of crucial importance that p and q, the factors of the modulus n, be large primes. How can one find a large prime? It is not just a random number, although when generating an RSA modulus one should try to pick two large primes as randomly as possible. A simple primality test is based on the following theorem: If there exist solutions to $\left(x^2 \equiv 1 \bmod p\right)$ other than ± 1, then p is not a prime. The test then goes thus:

If $p > 2$ prime, then $\left(x^2 \equiv 1 \bmod p\right)$ has only two solutions, $\left(x_1 \equiv 1 \bmod p\right)$ and $\left(x_2 \equiv -1 \bmod p\right)$.

The proof of the theorem is very simple. It is necessary to find solutions for

$$x^2 - 1 \equiv 0 \bmod p$$

$$(x+1)(x-1) \equiv 0 \bmod p$$

p can divide $(x+1)$ or $(x-1)$ or both. If p divides both, then it holds that

$$x + 1 = kp$$

$$x - 1 = jp$$

If these two equations are subtracted, it can be concluded that p equals 2:

$$2 = (k - j)p \Rightarrow p = 2$$

This is a contradiction, since p must be greater than 2. Now assume that p divides $(x+1)$. In this case it holds that

$$x - 1 = kp \Rightarrow x \equiv 1 \bmod p$$

which is the first possible solution if p is a prime. Similarly, if p divides $(x-1)$, it also holds that

$$x - 1 = jp \Rightarrow x \equiv -1 \bmod p$$

which is the second possible solution for p prime.

This theorem is used in Lehmann's primality test, but because the probability of success in one pass is not higher than 50%, the Rabin-Miller test is usually preferred in practice (see [6]).

RSA Today

In hardware, RSA is about a thousand times slower than DES: the RSA hardware encryption speed with a 512-bit key is about 1 Mbps. In software, DES is about a hundred times faster than RSA: the RSA software encryption speed is about 10 Kbps. According to *Moore's law*, computing power doubles approximately every 18 months, and computing costs fall to 1/10 after five years. Since RSA and DES are, unlike the one-time pad, not perfectly secure, it is necessary to use longer keys as encryption technology improves. This poses a major problem if RSA or any other nonperfect cryptosystem is used for digital signatures (see Section 2.3) of legal documents. Let us suppose somebody digitally signs a will today with a 512-bit RSA key and dies in 2020. In twenty years it will probably be quite cheap to break a 512-bit RSA key, and that might prove an irresistible temptation for less preferred heirs.

Security of RSA depends on the difficulty of factoring the modulus n. In August 1999, a team of scientists of the National Research Institute for Mathematics and Computer Science in the Netherlands, led by Herman te Riele, succeeded in factoring a 512-bit number [13]. About 300 fast workstations and PCs had spent about 35 years of computing time to find the prime factors. They were running in parallel, mostly overnight and on weekends, so the whole task was accomplished in about seven months. In practical terms, this means that the key size of 512 bits is no longer safe against even a moderately powerful attacker. Some 25 years ago it was estimated that 50 billion years of computing time would be needed to factor a 512-bit number, so the Dutch result is a major scientific breakthrough.

The latest news about breaking RSA (as of September 1999) is that the famous Israeli cryptographer Adi Shamir has designed a factoring device named "TWINKLE" (The Weizmann INstitute Key Locating Engine) that can be used to break a 512-bit RSA key within a few days [14]. For this,

about 300 to 400 devices would be necessary, each costing about $5,000. Although the use of TWINKLE would be quite expensive (approximately $2 million), it is a very good reason to abandon the use of 512-bit RSA encryption in all existing applications immediately.

2.2.2.2 Elliptic Curves

Elliptic curves have been studied extensively for the past 150 years, but their application to cryptography was first proposed in 1985 by Neal Koblitz and Victor Miller, independently. Elliptic curves can be used to define public key cryptosystems that are close analogs of the existing schemes. However, only those elliptic curve cryptosystems whose security depends on the *elliptic curve discrete logarithm problem* are of special interest today, since the only available algorithms for solving these problems need exponential time. In other words, these methods become infeasible much faster than the methods for solving the integer factorization problem that RSA is based upon (such methods need subexponential time) [15]. This means that an elliptic curve cryptosystem requires much shorter keys than RSA to achieve the same level of security. For example, a 160-bit elliptic curve key is roughly as secure as a 1024-bit RSA key. This advantage is of crucial importance for devices with limited storage and processing capacity, such as smart cards.

Elliptic curve cryptosystems are far more complicated to explain than RSA. An excellent interactive Web tutorial on elliptic curves, which was used as one of the sources for the following explanation, is published by Certicom.[15]

Elliptic curve groups are *additive groups*; that is, their basic function is *addition*: the sum of two points on an elliptic curve must also be a point on the elliptic curve. The addition is defined *geometrically*. To illustrate how it works, we will consider here *elliptic curves over real numbers*.

The negative of a point $P = (x_P, y_P)$ is its reflection on the x-axis: $-P = (x_P, -y_P)$. To double a point P, that is, to add it to itself, one draws a *tangent line* to the curve at point P. If $y_P \neq 0$, then the tangent line intersects the elliptic curve at exactly one other point, $(-2P)$, which is reflected on the x-axis to $2P$ (see Figure 2.3). It holds that $P + (-P) = O$, the point at infinity.

By the same principle one can compute $2P$, $3P$, etc. In general, to add two distinct points P and $Q(P \neq -Q)$, one draws a line through them. The

15. http://www.certicom.com/ecc

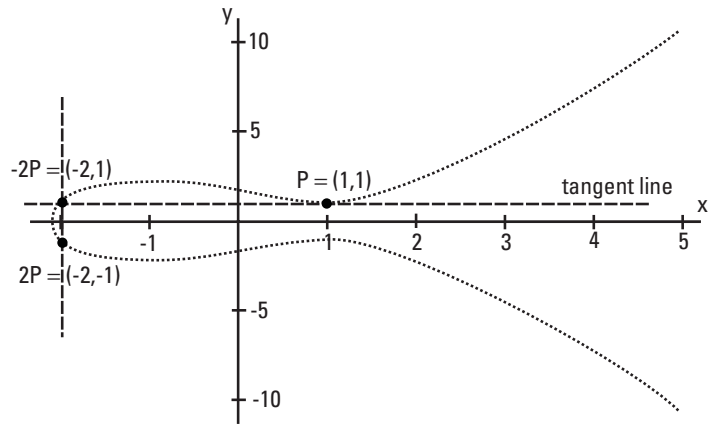

Figure 2.3 Elliptic curve $y^2=x^3-3x+3$.

line intersects the elliptic curve at one point, $-R$, which is reflected on the x-axis to the point $R = P + Q$.

Now the *elliptic curve discrete logarithm problem* can be defined: Given points P and Q in the group, find a number k such that $kP = Q$. The algorithms available for solving this problem need to be much longer than the algorithms for solving the standard discrete logarithm problem (see Section 2.3.2).

The slope s of the tangent line for an elliptic curve $F = -y^2 + x^3 + ax + b$ is computed as follows (∂ means derivation):

$$s = -(\partial F / \partial x) / (\partial F / \partial y) = (3x^2 + a) / 2y$$

For the point $P = (1,1)$ from Figure 2.3 the slope s is

$$s = (3x_p^2 - 3) / 2y_P = 0$$

It means that the tangent line at P is defined as $y = 1$. To find the coordinates of $Q = -2P$ one determines the point of intersection of the tangent line and the elliptic curve. We already know that $y_Q = 1$, so x_Q can be computed from the elliptic curve equation:

$$1 = x_Q^3 - 3x_Q + 3$$

$x_Q^3 - 3x_Q + 2 = 0 = (x_Q + 2)(x_Q - 1)^2 \Rightarrow x_Q = -2$ (since x_P already equals 1)

In general, the coordinates of $Q = 2P$ for an elliptic curve $y^2 = x^3 + ax + b$ can be computed as follows:

$$s = (3x_P^2 + a) / (2y_P)$$

$$x_Q = s^2 - 2x_P$$

$$y_Q = -y_P + s(x_P - x_Q)$$

For this type of elliptic curve it must hold that the discriminant of the cubic $x^3 + ax + b$ is not zero, that is, $4a^3 + 27b^2 \neq 0$. In other words, the cubic must not have multiple roots [16].

Galois Fields

Elliptic curves over real numbers are not suitable for cryptographic purposes. To define an elliptic curve cryptosystem, *elliptic curves over finite fields* are used. In particular, the *characteristic two finite fields* are of special interest since they lead to the most efficient implementations of elliptic curve arithmetic. Such a finite field is the *Galois Field* (GF) of a polynomial, GF(2^m).

GF is called *finite* because it has a finite number of elements (2^m elements). GF(2^m) can be defined by either *polynomial representation* or *optimal normal basis representation.* Here the polynomial representation is preferred for purposes of explanation. An element of GF(2^m) is a polynomial of the form

$$a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_2x^2 + a_1x + a_0, \text{ in which } a_i = 0 \text{ or } 1$$

The coefficients of the polynomial a_i are integers modulo 2 (i.e., they are always *reduced* modulo 2). The elements of GF(2^m) can be expressed as vectors of the form

$$(a_{m-1}, a_{m-2} \ldots, a_2, a_1, a_0)$$

Table 2.2
Elements of GF(2^4)

Polynomial	Vector
0	0000
1	0001
x	0010
$x+1$	0011
x^2	0100
x^2+1	0101
x^2+x	0110
...	...
x^3+x^2+x	1110
x^3+x^2+x+1	1111

To define GF(2^m) completely, one should reduce the polynomials as well. For this purpose, an *irreducible* polynomial $f(x)$ is needed, whose role is similar to that of a prime modulus in the standard discrete logarithm problem. Its degree is m, and it must not be factorable into polynomials of degree less than m, with coefficients 0 or 1. The leading coefficient must always equal 1:

$$x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \ldots + f_2x^2 + f_1x + f_0, \text{ in which } f_i = 0 \text{ or } 1$$

As an example, the elements of GF(2^4) are shown in Table 2.2.

Let the irreducible polynomial be $f(x) = x^4 + x + 1$. When two elements from GF(2^4) are added, the coefficients of the corresponding powers are added modulo 2:

$$(x^2+1)(x^3+x^2+x)$$

$$= (1 \bmod 2)x^3 + (2 \bmod 2)x^2 + (1 \bmod 2)x + (1 \bmod 1) = x^3 + x + 1$$

The same holds for subtraction. When two elements from GF(2^4) are multiplied, the result of the multiplication must also be an element of GF

$\left(2^4\right)$, since it is a field (and therefore an algebraic group as well). However, if the same elements as in the previous example are multiplied, the result will be

$$\left(x^2+1\right)\left(x^3+x^2+x\right)$$

$$=(1\bmod 2)x^5+(1\bmod 2)x^4+(2\bmod 2)x^3+(1\bmod 2)x^2+(1\bmod 2)x$$

$$=x^5+x^4+x^2+x$$

But this is not an element from GF$\left(2^4\right)$ since its grade is higher than $m-1$. Now the irreducible polynomial is needed to reduce the grade of the result:

$$\left(x^5+x^4+x^2+x\right)\bmod\left(x^4+x+1\right)=x^4$$

It works in the same way as for integers (2.2):

$$\left(x^5+x^4+x^2+x\right)=\left(x^4+x+1\right)x+x^4$$

For GF$\left(2^m\right)$ the elliptic curves of a different general form are used (the condition that there be no multiple roots is satisfied if $b\neq 0$):

$$y^2+xy=x^3+ax^2+b$$

The principle of how to compute $Q=2P$ is the same as with elliptic curves over real numbers. The slope of such an elliptic curve over GF(2^m) can be computed as follows:

$$s=-(\partial F/\partial x)/(\partial F/\partial y)=-\left(3x^2+2ax-y\right)/\left(-2y-x\right)=$$
$$-\left(x^2+2x^2+2ax-y\right)/\left(-2y-x\right)$$

Since all coefficients may be reduced modulo 2, it holds that

$$s=-\left(x^2-y\right)/(-x)=-\left(-x+y/x\right)=x-y/x$$

The negative sign may be ignored since (-1 mod 2) = (1 mod 2) = 1:

$$s = x + y / x$$

The negative of a point (x, y) is the point $(x, x + y)$. The coordinates of the point $Q = 2P$ can be computed as follows:

$$s = x_P + y_P / x_P$$

$$x_Q = s^2 + s + a$$

$$y_Q = x_P + (s + 1)x_Q$$

Elliptic Curve Security

For the security of an elliptic curve cryptosystem it is of crucial importance that the number of points on the curve (*the order of the curve*) have a large prime factor if the cryptosystem used is an analog of Diffie-Hellman (Section 3.1.2) or DSA (Section 2.3.3). There is a polynomial-time algorithm by Schoof [17] for counting the number of points on an elliptic curve.

Additionally, the order of the point P used in the elliptic curve discrete logarithm problem $kP = Q$ must be a large prime number. P has the role of the generator g in the finite-field Diffie-Hellman system (Section 3.1.1). The order of P is defined similarly to that described in the section on DSA: If n is the order of P, then n is the least element of the field (elliptic curve) such that $nP = O$ (the point at infinity). An elliptic curve with a point P whose order is a 160-bit prime offers approximately the same level of security as DSA with a 1024-bit modulus p and RSA with a 1024-bit modulus n.

The Elliptic Curve Digital Signature Algorithm (ECDSA) is being adopted as both an ANSI X9.62 standard and an IEEE P1363 standard [18][16]. ISO has standardized a certificate-based digital signature mechanism based on elliptic curves, and discrete logarithms in general [19]. Much more about elliptic curves can be found in [16] and [20].

2.3 Digital Signature Mechanisms

The purpose of digital signature mechanisms is to make it possible to sign digital documents. The digital signature cannot be a pure digital analogy to the hand-written signature, for then it could easily be copied and attached to any document. Also, signed documents could be changed after having been

16. http://www.certicom.com/ecc

digitally signed. As the RSA inventors realized, a digital signature must be *message*-dependent as well as *signer*-dependent [12].

Public key cryptosystems in which the result of first decrypting (by applying the private key) and then encrypting (by applying the public key) the message is the message itself, that is,

$$E_{PuK}\left(D_{PrK}\left(M\right)\right) = M$$

can be used as digital signature mechanisms. Since only the owner of the public key pair knows the private key, he is the only person that can produce a valid signature. On the other hand, anyone can verify the signature, since the public key is publicly available.

2.3.1 RSA Digital Signature

If RSA is used as a digital signature technique, generating a signature means computing S as follows:

$$S = D\left(h\left(M\right)\right) = h\left(M\right)^{d} \bmod n$$

in which $h()$ is a cryptographic hash function as explained in Section 2.1. The hash function output (hashsum) has a fixed length and is usually rather short compared to the whole message. The process of generating a signature is computationally expensive, so it is faster to decrypt the hashsum than the original message.

To verify a signature it is necessary to receive M, S, and the signer's public key (e,n) as well as information about which hash function and which signature algorithm were used to generate S. Then the verifier can compute the message hashsum $h(M)$ and compare it with the result of encrypting the signature S:

$$\text{Does}\left[E\left(S\right) = S^{e} \bmod n\right] = \left[h\left(M\right)\right] \text{ hold true?}$$

If yes, the signature is valid;

If not, the signature is not valid.

A signature is generated only once but usually verified more often. For this reason it is helpful for the verification process to be fast, and with RSA this can be achieved by choosing a small public exponent *e*. Like paper documents, digital documents should always bear a time stamp.

RSA (see Section 2.2.2.1) is the most frequently used digital signature mechanism. However, for political reasons, some countries, such as the United States, restrict the use of encryption. Until recently, it was not permitted in the United States to use an algorithm for digital signatures that could also be used for encryption. That is why the Digital Signature Algorithm was originally developed.

2.3.2 Digital Signature Algorithm

DSA and RSA are the two algorithms for digital signature generation and verification recommended by the Digital Signature Standard [21]. DSA belongs to a family of signature algorithms [22], together with ElGamal's signature algorithm and others, that are based on the *discrete logarithm problem*: For known *b*, *a*, and *p* prime, compute *x* such that

$$b = a^x \bmod p$$

What looks very simple is a hard problem for large primes. DSA requires the following public parameters:

- *p* large prime
- *q* large prime, $q|(p-1), 2^{159} < q < 2^{160}$
- *g*, generator modulo *p* of order *q*, i.e., $g = h^{(p-1)/q} \bmod p > 1 \left(1 < h < p-1\right)$

A signer's key consists of two numbers:

- *x* randomly generated integer, $0 < x < q$ (*private key*);
- $y = g^x \bmod p$ *(public key)*.

With *g*, it is possible to generate a set of integers $\{a_1, a_2, \ldots a_q\}$, $1<a_i<p-1$, $a_i \neq a_j$ if $i \neq j$, in the following way:

$$a_1 = g^1 \bmod p \neq 1$$

$$a_2 = g^2 \bmod p$$

…

$$a_{q-1} = g^{q-1} \bmod p$$

$$a_q = g^q \bmod p = h^{p-1} \bmod p = 1 \textit{ (see FLT in Section 2.2.2.1)}$$

Each time the exponent is a multiple of q, the result will be equal to 1. Therefore g is referred to as the generator of order q modulo p. Because g is used to generate one of the private DSA keys, it must be able to generate a large set of values; otherwise someone could easily guess the private key. Consequently, p must be large as well.

Each time a signature is generated, an additional parameter $k, 0 < k < q$, is randomly chosen. It must be kept secret. DSA and other similar digital signature algorithms that use a random number for signature generation have many opponents, since they can be used to pass information secretly to a chosen verifier (i.e., to establish a *subliminal channel* between the signer and the verifier). If the verifier knows the signer's private key, the subliminal channel can be established through the value of k [23]. For example, if a government digitally signs passports by such an algorithm, it can hide in the signature information about the passport owner that is normally restricted under data protection laws (e.g., criminal records).

The DSA signature of a message M is represented by a pair of numbers (r, s) computed in the following way:

$$r = \left(g^k \bmod p\right) \bmod q$$

$$s = \left[k^{-1}\left(h(M) + xr\right)\right] \bmod q$$

To verify the signature, the verifier computes

$$w = s^{-1} \bmod p$$

$$u_1 = h(M)w \bmod q$$

$$u_2 = rw \bmod q$$

$$v = \left(g^{u_1} y^{u_2} \bmod p\right) \bmod q$$

If $v = r$ the signature is valid. From signature generation it is known that $h(M) + xr = sk \bmod q$. Now it can be seen that v really must be equal to r:

$$v = \left(g^{u_1} y^{u_2} \bmod p\right) \bmod q = \left(g^{h(M)w \bmod q} g^{xrw \bmod q} \bmod p\right) \bmod q$$

$$= \left(g^{h(M)w \bmod q + xrw \bmod q} \bmod p\right) \bmod q = \left(g^{w(h(M)+xr) \bmod q} \bmod p\right) \bmod p$$

$$= \left(g^{s^{-1}sk \bmod q} \bmod p\right) \bmod q = \left(g^{k} \bmod p\right) \bmod q = r$$

2.3.3 Elliptic Curve Analog of DSA

ECDSA is being adopted as both an ANSI X9.62 standard and an IEEE P1363 standard. ECDSA is based on the elliptic curve discrete logarithm problem: Given points P and Q in the group, find a number k such that $kP = Q$ (see also Section 2.2.2.2).

ECDSA requires the following public parameters:

- q large prime, $q > 2^{160}$;
- E elliptic curve over a finite field GF(2^r) whose order is divisible by q;
- P fixed point on E of order q.

P has the role of the generator g in DSA but does not have to be a generator of the group of points on E [16].

A signer's key consists of two numbers, x and Q:

- x statistically unique and unpredictable generated integer, $0 < x < q$ (*private key*);
- $Q = xP$ (*public key*).

For each signature a unique and unpredictable integer k is chosen, $0 < x < q$. k must be chosen in such a way that the integer obtained as the binary representation of the x-coordinate of kP is not a multiple of q, that is

$x_P \bmod q \neq 0$. The ECDSA signature of a message M is represented by a pair of integers (r, s) computed in the following way:

$$r = x_P \bmod q$$

$$s = \left[k^{-1}\left(h(M) + xr\right)\right] \bmod q$$

If $s = 0$, the signature verification process has to be repeated by choosing a new k. To verify the signature the verifier computes

$$w = s^{-1} \bmod q$$

$$u_1 = h(M)w \bmod q$$

$$u_2 = rw \bmod q$$

$$u_1 P + u_2 Q = (x_0, y_0)$$

$$v = x_0 \bmod q$$

If $v = r$ the signature is valid.

2.3.4 Public Key Management

Public key distribution centers are usually called *certification authorities,* since their role is not only to make public keys broadly available but also to issue *certificates* that bind a public key to the name of a particular principal. Public key certificates are digitally signed by issuing of certification authority. Implementing a public key infrastructure that provides generation and verification of legally binding digital signatures is, both organizationally and technically, a very complex task. It is explained in Section 3.2.

2.4 Access Control Mechanisms

In order to access a protected resource in a system, a principal must first be successfully authenticated (i.e., prove his identity). In many systems this is not sufficient, however, because not all principals (or subjects) are granted the same *type* of access to all resources (or objects). Consequently, each

principal must be assigned implicit or explicit rights for accessing the object. In other words, the principal (or subject) must be *authorized* to access the object.

2.4.1 Identity-Based Access Control

Identity-based access control involves authorization criteria based on specific, individualized attributes. It is sometimes referred to as *discretionary access control* because authorization is performed at the discretion of the object owner. It is usually expressed in the form of an *access control matrix*.

The rows of the access control matrix represent subjects (users, processes), and the columns represent objects (files, programs, devices). The intersection of a row and a column contains the type of access right (e.g., read, write, delete, copy) of the subject to the corresponding object. In practice, the access matrix is implemented in one of the following two ways (see Figure 2.4):

- The row-wise implementation is referred to as a *capability list*, where for each subject there is a list of objects and the subject's access rights to each object.
- The column-wise implementation is referred to as an *access control list*, where for each object there is a list of subjects that have access to it and their access rights.

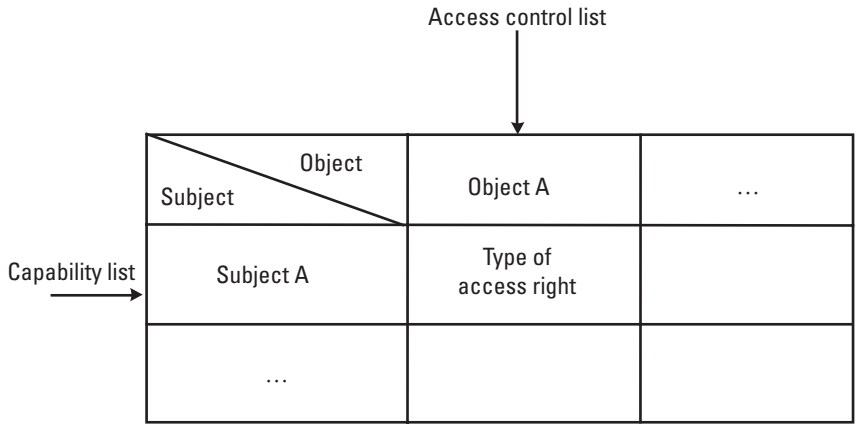

Figure 2.4 Access control matrix.

2.4.2 Rule-Based Access Control

In an information system with many security levels, it is not possible to enforce security with only an identity-based access control policy. Discretionary controls regulate the accessing of objects, but do not control what subjects might do with the information contained therein [24]. For this purpose *rule-based* access control policies can be used. These are based on a small number of general attributes or *sensitivity* classes that are universally enforced. Thus, all objects of the protected system must be marked with *security labels*. This type of access control is sometimes referred to as *mandatory access control* or *information flow control* [24].

One of the oldest rule-based access control models, the Bell-La Padula model [25], comes from the military world and is too restrictive for most commercial applications. There are other models that concentrate on integrity rather than confidentiality, such as the Chinese Wall model [26] or the Clark and Wilson model [27], and are more suitable for nonmilitary applications. As of this writing, rule-based access control is not widely deployed in practice.

2.5 Authentication Exchange Mechanisms

As shown in previous sections, symmetric or public key cryptosystems can be used to realize authentication mechanisms. This section explains an additional authentication technique, *zero-knowledge protocols*. A more complex authentication and key distribution system (Kerberos) will be explained in Part 2.

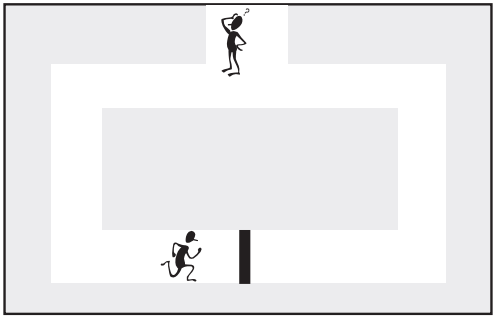

Figure 2.5 Zero-knowledge protocol with magic door.

2.5.1 Zero-Knowledge Protocols

As their name implies, zero-knowledge protocols allow a principal to prove knowledge of a secret without revealing anything about the secret.

A noncryptographic example will explain how it works. (The example is based on a description given in [28].) Suppose there is a building containing a passageway blocked by a magic door. Person *A* wants to prove to person *B* that she knows the secret that opens the magic door, but without actually telling *B* the secret (see Figure 2.5). They repeat the following protocol *n* times:

Step 1: *B* waits in front of the entrance to the building while *A* enters the building and goes to the right or the left (*A*'s random choice).

Step 2: *B* enters the building and calls to *A* to come out from the left side or from the right side (*B*'s random choice).

Step 3: If *A* comes out from whichever side *B* requested, the protocol run is considered successful. Otherwise it fails.

If all *n* protocol runs have been successful, the probability that *A* really knows the secret is $p = 1 - 0.5^n$.

2.5.2 Guillou-Quisquater

In 1988 Guillou and Quisquater developed a zero-knowledge protocol that can be used for authenticating smart cards [29]. Since smart cards have very limited processing power and memory, it is important to minimize the number of protocol runs required to give reasonable security that a card is authentic. In fact, Guillou-Quisquater's zero-knowledge protocol requires only one run and minimizes storage requirements. Zero-knowledge protocols are often referred to as *challenge-response* protocols: the verifier (e.g., bank terminal or automatic teller machine) sends a challenge to the card, whereupon the card computes a response and sends it back to the verifier.

In the Guillou-Quisquater protocol, the smart card (SC) has the following parameters:

- *J* credentials (public);
- *B* private key.

The integer *v* and the modulus *n* (a composite number with secret factors generated by and known to an authentication authority only) are public

parameters. The public exponent v can be, for example, $2^{17}+1$. B is chosen in such a way that

$$JB^v \equiv 1 \bmod n$$

The verifier V knows J, v and n (they are all public). The protocol goes as follows:

$SC \rightarrow V$: $T \equiv r^v \bmod n$ *witness*

$V \rightarrow SC$: d random *challenge*

$SC \rightarrow V$: $D \equiv rB^d \bmod n$ *response*

r is a random number, as well as $d\,(0 < d < v-1)$. Now V can compute T':

$$T' = D^v J^d \bmod n$$

and verify whether the following congruence holds true:

$$T \equiv T' \bmod n$$

If the card is authentic, it must hold true because

$$T' = D^v J^d \bmod n = \left(rB^d\right)^v J^d \bmod n = r^v B^{dv} J^d \bmod n$$

$$= r^v \left(JB^v\right)^d \bmod n = r^v \bmod n = T$$

since B is chosen in such a way that $JB^v \equiv 1 \bmod n$. If the protocol is successful, the probability of the card's being authentic is $1/v$. This zero-knowledge protocol was used as a base for a set of identity-based digital signature mechanisms standardized by ISO [30].

2.6 Traffic Padding Mechanisms

Traffic padding mechanisms protect against traffic analysis. It is sometimes possible for outsiders to draw conclusions based on the presence, absence,

amount, or frequency of data exchange. Valuable information may also be gleaned from a sudden change in the amount of data exchanged. Consider the example of a company that wants to start selling its shares on the stock market. It will most probably intensify communication with its bank during the preparation phase shortly before going public. Someone interested in buying up as many shares as possible may try watching the traffic between the company and the bank to learn when the shares will be available.

Traffic padding mechanisms keep traffic approximately constant, so that no one can gain information by observing it. Traffic padding can be achieved by generating random data and sending it encrypted over the network (see also Chapter 12).

2.7 Message Freshness

Message freshness is a mechanism that protects against replay attacks. One simple replay attack goes as follows: Suppose *A* uses homebanking software to send a digitally signed message to her bank with an order to transfer 1,000 euros to *B*'s account. *B* wants more, however, so he eavesdrops on *A*'s lines, copies *A*'s banking order and sends it 10 more times to *A*'s bank. Instead of 1,000 euros he gets 11,000 euros, and *A* is surprised to see her account emptied.

The protection against such attacks is to ensure that even messages with identical contents are different in transfer. *A* has two possibilities: she can

- Generate a random number (*nonce*), add it to the original message, and then sign it; or
- Add a time stamp (current date and time) to the original message, and then sign it.

In the first case, *A*'s bank has to store *A*'s messages for some period of time. Every time a message from *A* arrives, its nonce is compared to the previously stored nonces. If the nonce is not different from all the stored nonces, the message is not considered fresh and the order is rejected. But because the number of messages can grow very large, this solution has a scalability problem. Some homebanking programs use *transaction numbers* (TANs). Each accountholder obtains a set of TANs and uses one TAN per transaction. The TANs are stored in the bank database. As soon as a TAN is used, it can be deleted from that database.

When time stamps are used, *A*'s computer clock and *A*'s bank's computer clock must be synchronized. It is possible to allow a small tolerance interval, but that introduces additional insecurity because *B* can copy and resend *A*'s order very quickly. Another possibility is to use both a nonce and a time stamp, trading off scalability and security by limiting the number of stored nonces, and allowing some tolerance interval for time stamps.

2.8 Random Numbers

Cryptographic applications demand much better random generators than do other applications. Since it is very difficult to provide a real source of randomness, the random generators in widespread use are in fact *pseudorandom sequence generators*. Random generators are also subject to attack, so cryptographically secure pseudorandom sequences must be unpredictable in such a way that the sequences cannot be reliably reproduced (*real random*).

A simple randomness test is to try to compress a presumably random sequence; if a significant compression rate can be achieved, the sequence is not random enough. More on random generators can be found in [31].

References

[1] Schneier, B., and P. Mudge, "Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP)," *Proc. of the 5th ACM Conference on Communications and Computer Security*, San Francisco, CA, Nov. 2–5, 1998, pp. 132–141, http://www.counterpane.com/pptp.html.

[2] Rivest, R.L., "The MD5 Message-Digest Algorithm," The Internet Engineering Task Force, RFC 1321, April 1992.

[3] The National Institute of Standards and Technology, "Secure Hash Standard," FIPS PUB 180-1, April 17, 1995.

[4] Bellare, M., R., Canetti, and H. Krawczyk, "Keying Hash Functions for Message Authentication," In *Advances in Cryptology – Proc. CRYPTO '96*, pp. 1–15, N. Koblitz (ed.), LNCS 1109, Berlin: Springer-Verlag, 1996.

[5] Krawczyk, H., M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," The Internet Engineering Task Force, RFC 2104, Feb. 1997.

[6] Schneier, B., *Applied Cryptography*, 2nd edition, New York, NY: John Wiley & Sons, Inc., 1996.

[7] The American National Standards Institute, *American National Standard for Data Encryption Algorithm (DEA)*, ANSI X3.92, 1981.

[8] International Organization for Standardization, *Banking – Approved algorithms for message authentication – Part 1: DEA*, ISO 8731-1, 1987.

[9] Stallings, W., *Network and Internetwork Security*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1995.

[10] RSA Data Security, "RSA Code-Breaking Contest Again Won by Distributed.Net and Electronic Frontier Foundation (EFF)," http://www.rsa.com/pressbox/html /990119-1.html.

[11] Diffie, W., M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. on Information Theory*, Vol. IT-22, No. 6, 1976, pp. 644–654.

[12] Rivest, R. L., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM*, Vol. 21, No. 2, 1978, pp. 120–126.

[13] Centrum voor Wiskunde en Informatica, "Security of E-commerce threatened by 512-bit number factorization," Press release, Amsterdam, Aug. 26, 1999, http://www.cwi.nl/~kik/persb-UK.html.

[14] Silverman, R. D., "An Analysis of Shamir's Factoring Device," RSA Laboratories Bulletin, May 3, 1999, http://www.rsasecurity.com/rsalabs/bulletins/twinkle.html.

[15] Robshaw, M. J. B., and L. Y. Yiqun, "Elliptic Curve Cryptosystems, RSA Laboratories Technical Note, " June 1997, http://www.rsa.com/rsalabs/ecc/html/elliptic_curve .html

[16] Koblitz, N., *A Course in Number Theory and Cryptography*, Berlin: Springer-Verlag, 1994.

[17] Schoof, R., "Elliptic curves over finite fields and the computation of square roots mod p," *Mathematics of Computation*, Vol. 44, 1985, pp. 483–494.

[18] The Institute of Electrical and Electronics Engineers, Inc., "IEEE P1363: Standard Specifications for Public Key Cryptography," Draft Version 4, 1999.

[19] International Organization for Standardization, *Information technology – Security techniques – Digital signatures with appendix – Certificate-based mechanisms,* ISO/IEC 14888-3, 1998.

[20] Menezes, A., *Elliptic Curve Public Key Cryptosystems*, Dordrecht: Kluwer Academic Publishers, 1993.

[21] The National Institute of Standards and Technology, "Digital Signature Standard," FIPS PUB 186-1, Dec. 15, 1998.

[22] Horster, P., H. Petersen, and M. Michels, "Meta-ElGamal Signature Schemes," *Proc. 2nd Annual ACM Conference on Computer and Communications Security*, Fairfax, VA, Nov. 2–4, 1994, pp. 96–107.

[23] Simmons, G. J. (ed.), *Contemporary Cryptology: The Science of Information Integrity*, Piscataway, NJ: IEEE Press, 1992.

[24] Denning, D. E., *Cryptography and Data Security*, Reading, MA: Addison-Wesley, 1982.

[25] Bell, D. E., and L. J. La Padula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," MTR-2997, Mitre Corporation, Bedford, MA, 1975.

[26] Brewer, D. F. C., and M. J. Nash, "The Chinese Wall Security Policy," *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, May 1989, pp. 206–214.

[27] Clark, D. D., and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proc. 1987 Symposium on Research in Security and Privacy*, April 1987, pp. 184–194.

[28] Quisquater, J. J., et al., "How to Explain Zero-Knowledge Protocols to Your Children," In *Advances in Cryptology – Proc. CRYPTO 89*, pp. 628–631, G. Brassard (ed.), LNCS 435, Berlin: Springer-Verlag, 1990.

[29] Guillou, L. C., and J. J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," In *Advances in Cryptology – Proc. EUROCRYPT 88*, pp. 123–128, C.G. Günther (ed.), LNCS 330, Berlin: Springer-Verlag, 1988.

[30] International Organization for Standardization, *Information technology – Security techniques – Digital signatures with appendix – Identity-based mechanisms,* ISO/IEC DIS 14888-2, 1999.

[31] Eastlake, D., S. Crocker, and J. Schiller, "Randomness Recommendations for Security," The Internet Engineering Task Force, RFC 1750, Dec. 1994.