Chapter 3. Cryptography—Algorithms and protocols

3.1 Introduction

Cryptography has been a subject of much interest in early days and today is one of the most important areas of computer security. It has been an important tool for sending secret messages from the days of the Romans, to the two world wars and until these days. A fascinating history of cryptography can be found in the book by Kahn: “The code breakers” [Kahn67], and more recently in the book by Singh [Singh00].

Cryptography is the study of message concealment, while *Cryptanalysis* is the study of how to discover the encrypted message without having legal access to the key; together they are usually called: *Cryptology*. Cryptanalysis is difficult and requires good mathematical knowledge, so there aren’t many hackers trying to break codes. The equivalent to hackers in cryptography are scientists employed by the government or other organized groups.

In addition to its military value, cryptography today has much value for securing information in computer systems. It can be used for the following purposes:

- **Protection of messages** – To protect the secrecy of a message and prevent illegal modification of it.

- **Authentication** – To authenticate the identity of users, transactions, and systems.

- **Protection of software and data** – To protect sensitive data in files or databases, and using signatures to assure the authenticity and validity of software.

- **Digital signatures** – To authenticate the origin of a message and to provide for non-repudiation – i.e. the inability for a user that signed a message to deny having sent it.

- **Protection of sensitive parts of the system** – E.g. encryption of the passwords in most operating systems

Cryptography has been in general successful in solving the problems above. Cryptography is a low-level mechanism and to be used properly it should only reflect and provide assurance to higher-level policies. Security policies should not be defined at this level, although it is common doing so. Of course, there must be policies about the specific types of cryptography used, just no institution policies defined at this level. This means that it should not be up to each user to decide if to encrypt or not her messages but encryption should be decided by the system based on the specific information of the message and its information context.

In this chapter we will review the main cryptographic algorithms which were used in the past and those which are used today. Since the development of public keys and RSA (see later), emphasis has shifted into the area of cryptographic protocols, rather than algorithms. A *protocol* is a set of messages exchanged by two or more parties in order to achieve a certain common goal. We shall see how these algorithms are incorporated into different protocols such as authentication protocols, digital signature protocols, key sharing and key management protocols, and others. This chapter is only an overview of the topic, and is not intended as a full study of cryptography, especially we will not go into details of the mathematical foundations of

cryptography. Those details are covered in several books such as. [Sta02, Sti02, Sch96]. Relatively recent surveys are given in [Cal00, Lan00].

The general way in which encryption is used is shown in Figure 3.1

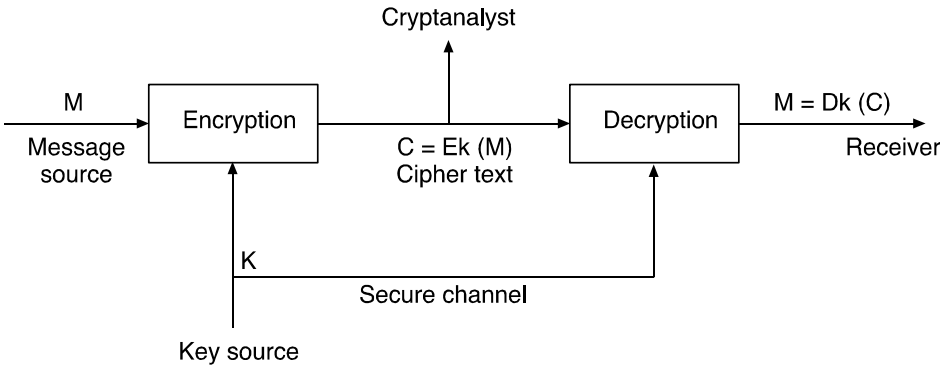


Figure 3.1: The encryption process

Figure 3.1 demonstrates the symmetric encryption process. The letter E stands for encryption, the letter D for decryption. Both functions use a single key K, which is transferred through a secure channel. The cryptanalyst tries to "break" the cipher without knowledge of the key.

The notation we usually employ is as follows:

- M – Message - Plaintext or clear text.
- C – Ciphertext - encrypted message.
- K – Key.
- E – The encryption function (or procedure) $C = E_k(M)$
- D – The decryption function $M = D_k(C)$

For any key K, and for any message M: $M = D_k(E_k(M))$

We also use the notation: C = E(M,K); M = D(C,K)

One very important principle in encryption is that the ***safety of the system does not depend on the secrecy of the algorithm, which is usually known, but only upon the secrecy of the key***. Although in the past the algorithm was generally concealed, nowadays the algorithms are well known to the public, such as the DES or RSA algorithms (see below). This principle is also an instance of the general open design principle discussed in Chapter 2.

As we shall see, a major problem of symmetric encryption is that whenever a key change is required (to protect against past exposures), a secure channel is needed for its transfer. Contrary to symmetric encryption, where the same key is used for both encryption and decryption, the main property of Asymmetric or Public-key schemes is that the encryption and decryption keys are not the same, and no secure channel is needed for key transfer. Generally the encryption key is public, while the decryption key is private and secret. Therefore, frequent change of keys and

the use of a secure channel are not necessary. Yet, key sharing, key validation, and key management may still be a problem (see Section 3.4).

The attacks on encryption usually involve some analysis of the statistics of the encrypted message plus the use of some previously known text. We can classify the cryptanalyst methods of attacks as follows:

- **Cipher text only -** the cryptanalyst has only the encrypted message to work with.

- **Known plaintext -** the cryptanalyst has a set of pairs (M, C) which he can use to compute the encryption key.

- **Chosen plaintext -** the cryptanalyst has access to the implemented encryption algorithms (e.g. the world-war II is enigma "box" that was captured by the British [Singh02]) and can generate selected pairs (M, C**).**

- **Chosen cipher text -** the cryptanalyst selects the next encrypted text based on the previous results.

We will see examples of the use of the above techniques as applied to the different encryption algorithms. Note that a common requirement from modern encryption algorithms is that they will withstand known and chosen plain-text attacks.

We can also distinguish between two types of "enemies" who are trying to attack the encryption system:

- **Passive Adversary** – Limited to eavesdropping on the unsecured (but encrypted) portion of the conversation between the sender and the receiver.

- **Active Adversary** – In addition to eavesdropping, may alter or inject messages, or perhaps even block transmission entirely.

We will deal with both types of attacks in this chapter.

Later on we will define the ***security of a cipher*** as some measure of the difficulty of the breaking it by an enemy who does not have the legal key, by using one of the attack methods mentioned above. We will define basically two types of measures:

- A ***theoretical*** measure which measures this difficulty without regards to the computational resources needed.

- A ***complexity*** measure which measures this difficulty in terms of the computational resources needed. The common complexity measures of Polynomial and NP-hard complexity are generally used in this context[1].

We start this chapter with discussing classical symmetric ciphers and their cryptanalysis. We will also mention some of Shannon's principles; Shannon was the first person to establish a theory for cryptography. We then describe the ways messages are converted into encrypted text, namely:

[1] The reader is assumed to know these terms from an Algorithms course

stream and block ciphers. We then discuss modern symmetric ciphers such as DES and AES. Next, we discuss public key schemes, hash functions, digital signatures and certificates. Finally, we describe some of the major protocols for key sharing and key management, and for authentication.

3.2 Symmetric Classical Ciphers

In the previous section we identified the two main types of encryption: symmetric and asymmetric. The main difference between them is that in symmetric encryption, one common key is used, which must be passed between the users; while with asymmetric encryption, more than one key (usually two) is used, and there is no need to pass them back and forth.

As defined earlier, the process of encryption is usually built up of an encryption function and a key such as:

$$C = E_K(M); M = D_K(C)$$

Where M and C are the original message and the encrypted message. E and D are the encryption and the decryption functions, respectively, and K is the single symmetric key.

We usually assume that plain-text messages are built from letters in some language, for example English. We will therefore assume our messages as constructed from words in the language. An example for such message may be:

THE ATTACK WILL BE MONDAY NIGHT!

The simplest encryption of such a message is the substitution of each character in the message with another character. Such simple substitution ciphers were already known to the Romans.

The Caesar cipher

This is the simplest substitution cipher (presumably it may have been used by Julius Caesar) In this cipher, each character is replaced by another character with a constant difference between them. That difference is the *Key* in the cipher. The following shows an example of this cipher:

Assume the key is 3, that is $C_i = E(p_i) = p_i + 3$

A full translation chart of Caesar cipher is:

Plaintext:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext:	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

Using this encryption, the message: *TREATY IS NOT POSSIBLE*
would be encoded as (spaces are eliminated): *WUHDWBLVQRWSRVVLEOH*

Such a cipher is usually easy to break by even a non-professional cryptanalyst. This is done by using the common letter frequencies table shown in Figure 3.2.

Letter	Percent
a	7.49
b	1.29
c	3.54
d	3.62
e	14.00
f	2.18
g	1.74
h	4.22
i	6.65
j	0.27
k	0.47
l	3.57
m	3.39
n	6.74
o	7.37
p	2.43
q	0.26
r	6.14
s	6.95
t	9.85
u	3.00
v	1.16
w	1.69
x	0.28
y	1.64
z	0.04

Figure 3.2: **Letter frequency table for English**

Similar tables exist for the most common digrams (two letter sequences) or trigrams. One can construct such tables for other languages as well as for programming languages, and for common dictionaries. For example, using the above table and the fact that t is a quite common single letter and tt is a quite common digram, one can guess that ww represent tt, and therefore the key is +3 which now can be used to break the entire cipher! Obviously, a simple generalization of this scheme is the use of a multiple letters key, rather than a single displacement. This is discussed next.

Monoalphabetic ciphers

In the monoalphabetic cipher, each character is replaced with another character from the same alphabet. A common technique is to select a short key, substitute the letters using the key, and then do the remaining substitutions using the letters which are not in the key, using their common alphabetical order. The following example shows this process using the key: **CAROLZ**

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

C A R O L Z B D E F G H I J K M N P Q S T U V W X Y

There are many other ways to select the key of mono-alphabetic cipher [Sch96]. Now, this cipher is also quite easy to break. In English, letters are not evenly distributed. That is, the frequency of appearance of some characters is higher that the others. Since some ciphers, like the mono-alphabetic one, do not change the frequency distribution of the language, by doing statistic on the cipher-text, it is then possible for the adversaries to guess back the original plaintext. Let say, if you find that the probability of the appearance of ‘q’ in the cipher-text is about 12.7%, it’s very probable that the original character in the plaintext is 'e'.

In breaking a mono-alphabetic cipher we use a frequency chart as shown in figure 3.2 Notice that the frequency in the encrypted text will match those in the language only if there is enough of the text. Therefore, we also use frequencies of digrams or trigrams.

Next we show an example of a cryptanalysis of the mono-alphabetic cipher.
Assume we have the following message:

‘QZIV XY KEO JLYYW JBRO XN KEO JKGOOK. TOK SO KX KEO AELGAE XY KBSO. KEO NBJE ZGO MLSDBYT ZYR KEO AXKKXY BJ EBTE. XLG JKZKO NZBG BJ KEO HOJK JKZKO NZBG'.

As mentioned above, the simplest effective attack on a mono-alphabetic cipher is the use of frequencies in natural languages: single letters, digrams/trigrams, small words, end/beginning of words, etc. We’ll only consider English here, and use some empirical facts about single-letter frequencies, as well as knowledge about common English words. The most common single English letters are *e* and *t*, with all others considerably less frequent.
To attack a cryptogram, we first sort the letters by frequency of appearance, for the above message we receive:

K-15, O-13, E-9, B-7, J-7, Z-6, X-6, Y-6, G-5, L-3, N-3, A-2, S-2, T-2, R-1.7 with

D, H, I, M, Q, V, W occurring much less often.

Thus, we would imagine that 'K' is either 'e' or 't', and perhaps 'O' is the other of the two. Trying first *K* = *e* and *O* = *t*, we have (in part)

QZIV XY eEt JLYYW JBRt XN eEt JeGtte. Tte St eX eEt AELGAE XY…

The 'Tte' in the second sentence immediately raises a problem: it seems unlikely that 'T' can be anything that would make this a word that could begin a sentence. So we try *K* = *t* and *O* = *e* instead, the result:

QZIV XY tEe JLYYW JBRe XN tEe JtGeet. Tet Se tX tEe AELGAE XY tBSe. tEe NBJE Zge

MLSDBYT ZYR tEe AXttXY BJ EBTE. XLG JtZte NZBG BJ tEe HeJt JtZte NZBG.

This looks better, and the substring 'tEe' suggests E = h, the ‘tX’ suggests X = o, and then 'XY' suggests Y = n. This gives:

QZIV on the *JLnnW JBRe* on the *JtGeet. Tet Se* to the *AhLGAh* on *tBSe*. The NBJh *Zge*

MLSDBnT ZnR the *Aotton BJ hBTh. oLG JtZte NZBG BJ* the *HeJt JtZte NZBG*

The 'Tet Se to the' suggests 'get me to the', so T = g and S = m. and 'JtGeet' could be 'street', so J = s, G = r, the result:

QZIV on the *sLnnW sBRe* on the street. get me to the *AhLrAh* on *tBme*. the *NBsh Zre MLmDBng* Z*nR* the Aotton *Bs hBgh. oLr stZte NZBr Bs* the Hest *stZte NZBr*.

The ending on *'MLmDBng'*, and also *'Bs hBgh'*, suggest *B = I*. Also the *'oLr'* suggests *L = u*

Now:

QZIV on the *sunnW siRe* on the street. Get me to the *AhurAh* on time. The *Nish Zre MumDing ZnR* the Aotton is high. Our *stZte NZir* is the *Hest stZte NZir*.

Then *'sunnW siRe oN'* suggests *W = y*, *R = d* and *N = f*, resulting with:
QZIV on the sunny side of the street. Get me to the *AhurAh* on time.

The rest is left as an exercise to the reader.

Polyalphabetic ciphers

The main problem with monoalphabetic ciphers as demonstrated above, is that the frequencies of letters and digrams are preserved. This is dealt with by using polyalphabetic ciphers. In these ciphers we first select a key which defines the cycle of the cipher.

Suppose the key length is *n*, then each letter will be substituted by n ***different*** letters, depending on its position with respect to the key. The simplest method uses a table called the Vigenere table, see Figure 3.3. For our example we assume the key is applied to the columns (lower case).

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*
A	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
B	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
C	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
D	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
E	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
F	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
G	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
H	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
I	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
J	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
K	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
L	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
M	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
N	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
O	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
P	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
R	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
S	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
T	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
U	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
V	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
W	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
X	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
Y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
Z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Figure 3.3: **The Vigenere Table**

Using the Vigenere table, and the key, one uses a CAESAR substitution for each of the key's letters periodically. The Caesar cipher is the row in the Vigenere table corresponding to the letter in the key. As an example assumes the key is **sharon**, then the following is a message and its encryption (only the non-blank characters are encrypted…):

s h a r o n s h a r o n s h a r o n

S E ND T H EMON EY

k l nu h uw t o e s i…

As can be seen the letter 'a" for example is encrypted by different letters in different parts of the text. Obviously, the previous methods which use the frequencies of single letters or digrams, will not work here, since these frequencies are smoothed out.

How then one breaks this cipher? The secret is to find first the length of the key. Once the key length is known, suppose it is k, we can now count frequencies every k letters! With enough text, this can lead to breaking the cipher. For example, if the ciphered text is long enough, then there is a high probability that a stream of ciphered characters representing a very common phrase will occur several times in the ciphered text. One then, can compute the distances in which equal encryptions of such plaintext occur. These distances must be multiplications of the key length. However, if the key length k can be discovered, then monoalphabetic frequency analysis can be used on the submessages formed from letters that are a multiple of k apart in the ciphertext. This is called the Kasiski method of breaking a polyalphabetical cipher.

Example: assume we have the message:

offset	cliphertext									
0	UPVWB	GVUPN	KKFOL	OGAKU	FBTKF	LFXUJ	VIPWV	KFWXO	FIDLO	ONLUP
50	KKFUW	OMQFQ	MQXKU	AFIUP	VVVVK	KFDEP	DNFIU	PVVFI	WVTMU	XDBWY
100	FVVYF	WTHBA	WQHEY	LTXVU	JVXFM	IDRSQ	EJNCI	PVWWQ	HOEYJ	BWQHB
150	YHTWL	OUMND	OLVUJ	VREWA	JHPWW	VPTWW	VLVDM	TROPV	XWIMW	KJBVE
200	FITKV	WRQEL	FWOBY	HSMND	TVFOJ	DWQHB	YLOOW	OTQXK	UISLS	LNLUP
250	RESWB	HOEZQ	HERVC	XRMJV	XWINR	LSISR	WMIHF	TWOHN	CXUBV	UJVXF
300	JWTOJ	VXGJA	REMMU	GPEEG	PEEWP	BYHXI	KHS			

You can see that ‘WQH’ appears in several places with offset 110, 138 and 226. It is likely that the keyword length k divides the difference of any two of these offsets.

The differences (28 and 88) are divided only by 2 or 4. The cryptanalyst may then examine frequency distribution for each candidate language, in this case 2 and 4. From the distribution, one can see that the k = 4 case, is closer to the normal character distribution. We will then suspect that one of the higher frequency cipher text letters, in each of the k = 4 lines, corresponds to plaintext letter 'e', and continue from there.

An important polyalphabetic cipher is the Vernam cipher which uses encryption based on the XOR function and on a key made up of a bit stream which is generated in a random way (actually pseudo-random). A pseudo-random number generator is a function which accepts a "seed" and generates from it numbers in a certain range, whose probability distribution in that range is very close to uniform (see example below). Such generators are present in all common programming languages. Usually, we generate numbers and then we convert them to bits. If the stream of bits is really random, (e.g. generated by a fair coin), then the encryption is perfect (i.e. even the legal decipherer wouldn’t be able to break it). This kind of encryption is called a **one-time pad**.

Shannon [Shan49] investigated encryption in depth with concepts of information theory as early as 1949 (see next section). Shannon explains “perfect secrecy” as a cipher in which the **entropy** or the uncertainty of the enciphered message is not different than that of the original message’s entropy, meaning that we didn’t receive any information at all by capturing the enciphered message! (See formal definition in Section 3.1.5)

Shannon showed therefore that for a one-time pad cipher there is “perfect secrecy”. In reality, of course, we cannot use truly random keys, and instead, we use, for both the encryption and the

decryption, the same generator of random numbers using the same seed (to make it possible to decipher the message legally) so the generated numbers are not random but they are derived from the same specific seed by a known algorithm. (See later for a version of a "perfect cipher" suggested by Rabin.). An example of encryption and decryption using Vernam cipher is shown below.

Vernam Cipher

For example, the binary number:

1 0 1 1 0 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1 0 1

Can be encrypted with the pseudorandom binary key and an exclusive OR function:

1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 1 1 0 0 0 1

To produce the following cipher text:

0 0 0 0 1 0 1 1 1 0 1 1 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 0 0

A very common method of generating pseudo-random numbers is the method based on modulo large numbers. For example, the *i*+1 number is computed from the *i*'th number by using the following equation:

$$N_{i+1} = (a*N_i + b) \bmod n$$

The problem is that if some plain text is known, it is possible to compute the coefficients *a* and *b* and break the cipher (see below). In spite of this, the Vernam cipher is quite effective against non-professionals and since it is also very simple and fast, it is quite popular. Notice that in the Vernam cipher we have the following properties:

$$C = Key \oplus M$$

$$M = C \oplus Key$$

However, we also have

$$Key = C \oplus M$$

This is the reason why this cipher cannot withstand a clear-text attack.

Vernam Ciper – Cryptanalysis

The problem with the modulo form of random number generator is its dependency on very few coefficients. Because each number depends only on the previous number, you can determine the algorithm coefficients by solving a series of equations:

$$r_1 = a * r_0 + b \bmod n$$
$$r_2 = a * r_1 + b \bmod n$$

$$r_3 = a * r_2 + b \bmod n$$

An interceptor who has r_0, r_1, r_2 and r_3 can solve these equations for *a*, *b*, and *n*.

For example, an interceptor can get r_0, r_1, r_2 and r_3 by a **probable word attack**. With a Vernam cipher, each cipher text letter comes from the formula:

$$c_i = r_i + p_i \bmod n$$

If an interceptor of the cipher text guesses that the message starts with HELLO (with the encoding*: H* = 8, *E* = 5, *L* = 12), the interceptor can try to substitute probable values of p_i and solve for values of r_i *as follows*:

$$r_0 = c_0 - 8 \bmod n$$

$$r_1 = c_1 - 5 \bmod n$$

$$r_2 = c_2 - 12 \bmod n$$

$$r_3 = c_3 - 12 \bmod n$$

With these values of r_0 to r_3, the interceptor may be able to solve the three equations for *a*, *b*, and *n*. Given those, the interceptor can generate the full sequence of random numbers and obtain plaintext directly.
Another disadvantage of this method is of course the need of a joint seed (this is though a common a disadvantage of all the symmetric ciphers…). The following suggestion tries to overcome the common seed disadvantage. Remember that Vernam is also a commutative and an associative cipher, i.e.:

$$C = (M + Key_1) + Key_2 = (M + Key_2) + Key_1$$

This implies two users (Bob & Alice) who want to communicate with each other without having a joint seed, may use the following protocol (we assume that each of them has its own secret seed or key and the common seed they want to share is M):

Bob sends Alice the common seed encrypted using his key. Alice encrypts it with her secret key and sends it back to Bob. Meaning that Bob gets back:

$$C = (M \oplus Key_{Bob}) \oplus Key_{Alice}$$

Now Bob decipher the message he got with his key:

$$C' = C \oplus Key_{Bob}$$

And sends C' to Alice. Alice than does:

$$M = C' \oplus Key_{Alice}$$

Now both have a common seed - M. Of course the enemy has to know at least one of the secret keys in order to decipher the message. Likewise, Bob and Alice do not need to know the keys of each other in order to share the secret key M.

* There is a **serious bug** in this method! Find it!

Despite the bug mentioned in the method above, the idea of a cipher as a “closed box” that users exchange between themselves in order to share a common key, is a very important idea, which we discuss later in this chapter (see the Diffie-Hellman method).

Permutation ciphers

In this type of cipher the ***location*** of each character is changed! This will cause the message to look as "garbage" but will not change the individual letter frequencies (although digram frequencies will of-course change!)
As an example, we can write the plaintext message as:

H	E	R	E	I
S	A	M	E	S
S	A	G	E	T
O	S	H	O	W
H	O	W	A	C
O	L	U	M	N
A	R	T	R	A
N	S	P	O	S
I	T	I	O	N
W	O	R	K	S

And the ciphered message is:

H S S O H O A N I W E A A S O L R S T O …

The key in this case is the respective *sizes* of the permutation rectangle (in our case 5x10). With enough text, this cipher is quite easy to break by shifting the ciphered text on itself and looking for common digrams or trigrams (try it!). (There are some easy modifications of the above scheme, like setting some predefined "holes" in the rectangle, but still the cipher is not difficult to break…)

Nevertheless, it has been found that even though the permutation cipher and the substitution cipher, each separately, are quite easily broken; their combination (a *product cipher*) makes quite a strong cipher, which is not easily breakable. This is the reason why it was used in the Enigma machine in World War II, and a product cipher is also used for most of the symmetric ciphers such as DES (Section 3.2).

Overview of Shannon's theory

The first person that published a formal theory of encryption was C. Shannon in his famous paper from 1949 ([Shan49]). In that paper, Shannon used the concept of **Entropy** which was borrowed from Physics as a measure of information, actually a measure of lack of information or uncertainty.

Applying this to cryptography, Shannon defined the entropy of a system of messages before encryption and before anything was sent out, as:

$$H(M) = -\sum P(M_i)\log(P(M_i))$$

Where $P(M_i)$ is the a priori probability of the message Mi. For example, if we have a 2- bit message with the possibilities:

00, 01, 10, 11

Then if the probability of each of these messages is ¼, then the entropy of the system is exactly 2, meaning the uncertainty of which message will be sent is exactly 2 bits of information.

When messages are encrypted, then the uncertainty in the message may decrease depending on how difficult is to break the cipher. We can therefore define the following terms:

$H(C) = -\sum P(C_i)\log(P(C_i))$ – entropy of the encrypted messages.

$H(M|C) = -\sum P(M_i|C)\log(P(M_i|C))$ – entropy of the clear message given the encrypted message.

$H(K|C) = -\sum P(K_j|C)\log(P(K_j|C))$ – entropy of the key used in the encryption given the encrypted message.

Since if we know the key we know also the message, then obviously H(M|C) is less or equal to H(K|C) (the opposite is not true, why?)

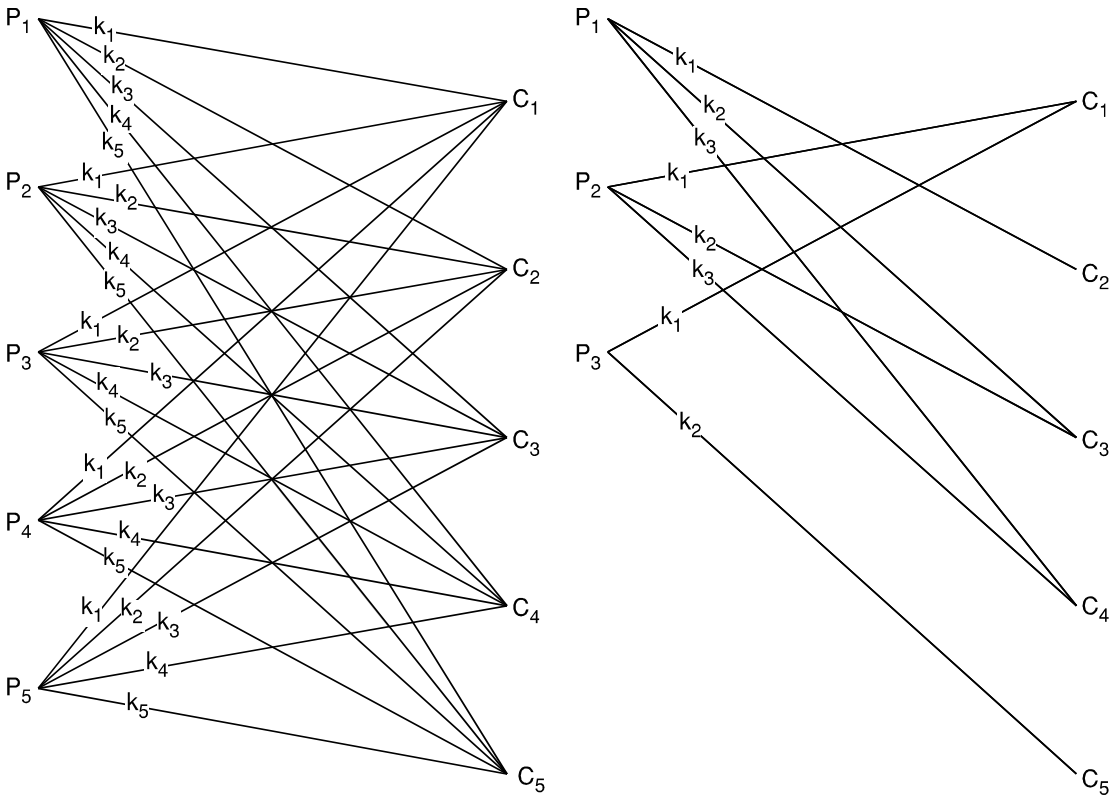
Using the above definitions, one may compute the entropy of various encryption algorithms. One important class of ciphers is the following:

$$H(M|C) = H(M)$$

This is the perfect cipher we mentioned earlier! Shannon explains “perfect secrecy” as a cipher in which the entropy or the uncertainty of the encrypted message is not different than that of the original message’s entropy, meaning that we didn’t receive any information at all by capturing the encrypted message! In general, an algorithm is a *perfect cipher* iff:

- There is a unique key for each message.
- All keys are equally likely.

Figure 3.4 also shows the difference between a perfect and a non-perfect cipher where for the first, for each encrypted message every clear message (and key) is possible, while for the second for some encrypted messages, some clear messages are impossible. This is obvious for natural languages such as English for which only a few hundred thousand messages (of one word) are possible out of a huge possible space of messages.



Error: Reference source not found

Figure 3.4: **Perfect vs. Non perfect cipher**

Since perfect secrecy is not achievable, Shannon investigated properties of non perfect ciphers and tried to give a measure for their strength. He also defined a set of guidelines for designing good ciphers shown below:

Shannon's Principles for a Good Cipher:

1. The amount of secrecy needed should determine the labor required for encryption/decryption.
2. The keys and ciphering algorithm should be "free" from complexity.
3. The implementation of the cipher algorithm should be simple and effective.
4. Errors in ciphering should not propagate to the entire message.
5. The size of the enciphered text should not be much larger then the size of the clear text.

Shannon also defined two important concepts by which we can characterize various ciphers:

Confusion – a complex functional relationship between the key, plaintext and ciphertext.

Diffusion – Information from one plaintext bit is diffused over all bits of the cipher (block).

The ***confusion*** insures that the conversion of each component is complex enough and the diffusion assures that the information from one bit in the original message, is in almost every bit (in the block of) the ciphered message.

We would like all ciphers to have high confusion and high diffusion, but that's is usually not the case. For example, the monoalphabetic cipher has both low confusion and low diffusion. Poly-

alphabetic ciphers have higher confusion, while permutation ciphers have higher diffusion, and their product has both higher confusion and diffusion.

Shannon also defined the concept of *Unicity Distance* = H(K)/D where H(K) is the entropy of the key and D is the redundancy of the language, (about 2.6 for English) Unicity Distance is a measure of the minimal amount of encrypted text needed to theoretically break a cipher (based on the message language redundancy). However, the theoretical possibility of breaking the cipher may still not be feasible if it requires a huge amount of computational resources. Therefore, today, we use ***computational complexity*** as the meaasure for a cipher security and the unicity distance measure is not so useful any more.

Rabin "perfect" cipher

In 2002, Michael Rabin announced a scheme for what he called "everlasting security" [Ding02]. Here, he used a bounded storage model, a refinement of computational security, in which it is assumed that any adversary has a bounded amount of storage available, but no assumption is made about computational power.

To encrypt using this scheme, the sender A and the receiver B use a publicly accessible sequence á of random bits. This sequence should be longer than their adversary's storage capacity and can come, for example, from an intercepted digital stream from a TV channel. The key that A and B share is an algorithm for picking out bits from á to use for a one-time pad. To solve the problem of perfect synchrony, it will suffice for A and B to use a self-synchronizing stream, such as that from a digital TV broadcast or a satellite signal.

To see why this works, suppose an adversary captures the cipher-text and later captures the key. But since the adversary lacks the ability to store á, she will not be able to construct the one-time pad, and so, will not be able to decrypt the message.

The main criticism of this method is that with storage systems of today and using delay technology, the assumption that á cannot be stored by the adversary (for any reasonable length source message and key) is unrealistic. (see e.g.
http://www.greatencryption.com/why_use_great_encryption.html)

Stream and Block ciphers

Most ciphers use an encrypting function whose input is restricted in length. Therefore, there are commonly two ways to encrypt large messages: stream ciphers and block ciphers. In the *stream cipher*, we encipher the message with a key which advances with the message such as Vernam. That is, the message is decomposed into small components (usually single characters or even single bits), and each component is enciphered with the next component of the key. The decryption is done similarly.

In the *block cipher*, the message is divided into components (a block such as 32, 64 or 128 bits). We encipher each block separately with the *same key*. So if the text appears twice in different blocks, it also will also appear in the same ciphered form. That is the reason why we have to make sure that the block will be long enough, so it will not be possible to use the “chosen text” attack to break it.

Sometimes in a block cipher, we may use the previous encrypted block as input to the next block encryption. Such a cipher is called: *Chained block cipher* or CBC.

The two ways to use a block cipher are shown in Figures 3.5a and 3.5b (M_i is a block of the message M).

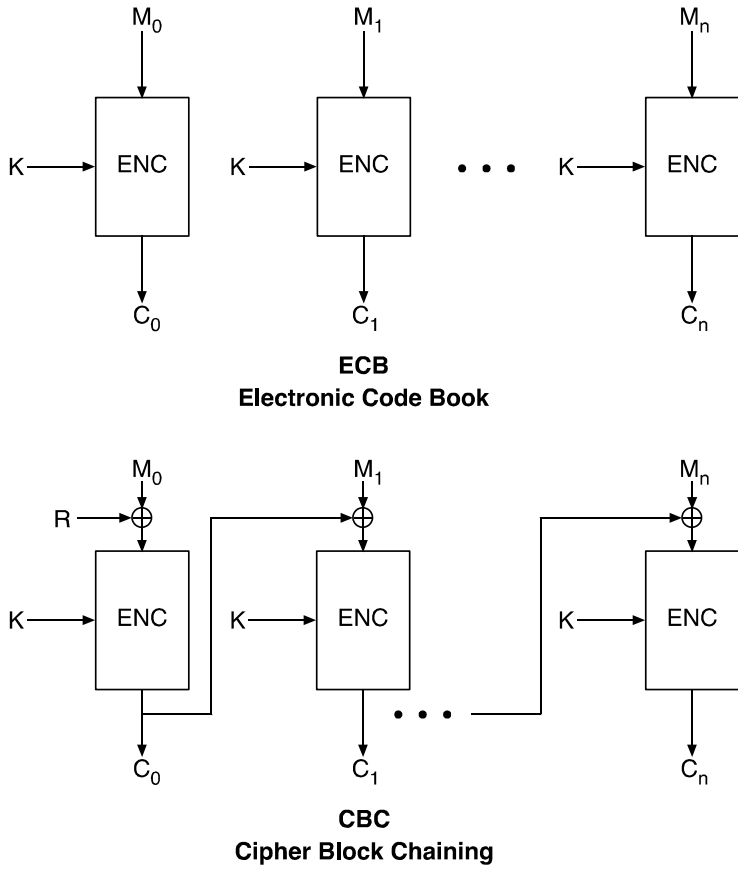


Figure 3.5: **Two modes of using block ciphers**

The CBC cipher uses an initial random number R to encrypt the first block and then the encryption of each successive block is dependent on the previous block. This way, recurring text will not be encrypted in the same way in different blocks, another version of this idea called: Cipher feedback is commonly used for DES [rfc2952].

The main advantage of the Stream cipher is ***speed***. It is usually very fast. Furthermore, each character can be encrypted in real time without the need to accumulate a full block. Another advantage is low error propagation, since an error in one ciphered character (bit) affects only that character. The main disadvantage of stream cipher is low diffusion, since each symbol is enciphered separately. Another disadvantage is susceptibility to malicious insertions. Since the key stream has to be synchronized with the ciphered stream, any insertion of a ciphered character will completely offset this synchronization and will make legal decryption almost impossible.

The main advantage of block ciphers is their high diffusion, especially with modern symmetric ciphers such as DES. This cipher is also immune to malicious insertions, since each block is separately encrypted, so a malicious insertion will affect one block only. On the other hand,

block ciphers are usually slower than stream ciphers. Also, an error in a ciphered block will cause the entire block to become useless.

Today, most of the systems use block ciphers and in particular use the modern block ciphers such as: DES, AES and IDEA. We'll discuss them in section 3.3.

Evaluating ciphers

Ciphers may be evaluated using different criteria (see also Shannon principles above). In particular we use criteria such as: Security, speed and convenience.
In terms of security we should note the following:

- Any algorithm can be eventually broken by trying all possible keys; a secure algorithm just requires such a large number of tries that it is computationally unfeasible to find the key in a reasonable time.

- If the algorithm has shortcuts, it is not necessary to try all the possible keys. For example, if the statistical frequency of letters is kept in the cipher text, the work to find the key is considerably smaller as was already discussed in explaining Shannon's Unicity distance.

A *practical secure encryption algorithm* should have the following properties:

- No method of recovering the key is known other than trying all possible keys.

- Trying all the possible keys is not feasible with a limited cost and limited time.

As we shall see later, some ciphers may be broken not only by trying an exhaustive search which is usually of exponential complexity, but may require solving a difficult problem such as an NP complete problem. We'll see examples of such ciphers later in Section 3.3.

The computational difficulty of breaking a cipher is not enough however, a secure algorithm must be implemented properly and used properly or other attacks may bypass the cryptographic defenses; for example, through the operating system, through timing attacks, by power consumption estimates, and other ways [Sch01]. The way an encryption algorithm is implemented is called the *encryption protocol*. In addition, it is important that ciphers will resist breaking when the enemy has additional information such as cleartext (cleartext attack), or may select the text to encrypt (chosen text attack).

To summarize this section, we can classify ciphers into the following classes:

- Unconditionally secure - One time pad

- Computationally secure (using computational complexity measures)--Breaking it is provably at least as difficult as solving a mathematical problem believed to be intractable such as the well know NP-complete problems.

- Exhaustive search secure--Ciphers for which the only known method to break them is exhaustive search, but there is no proof that this is the only method.

3.3 Modern Symmetric Ciphers

As was mentioned in the introduction, interest in cryptography increased considerably as computer systems started to penetrate every segment of society. In the 1970s it was realized that encryption is a valuable technology for security in the civilian sector as well, and this lead to the development of the first standard cipher - the DES cipher.

The Data Encryption Standard (DES) Cipher

The DES cipher [Nist93] was developed by the USA government for needs of standard encryption, back in 1976. Such standard became important so that the different organizations, including different government institutions, could exchange information between them in a secure way using a standard encryption algorithm. At the time, only symmetric ciphers were in use, and therefore a block symmetric cipher which was based on the Lucifer cipher developed at IBM, was selected from the various proposals. This cipher is a block cipher (up to 64-bit blocks) and uses a chain of substitutions and permutations, which as a result generate almost perfect "confusion" and "diffusion". Therefore it is very hard to break it except by searching all the possible keys (i.e. exhaustive search or "brute-force").

This cipher uses an initial and final permutation, and in between, 16 iterations of complex product cipher based on the key and the input message. The iterations are depicted in Figure 3.6 and each iteration is depicted in Figure 3.7. Notice that in each iteration the key is not used as is, instead it goes under a transformation! This transformation first shift the bits of the key 1 or 2 bits depending on the iteration number, and then transform it using a complex substitution table as well. After this transformation, the key is x-ored with the text and goes again through various substitutions and permutations. In each iteration, one part of the text (left or right) is encrypted under the complex transformation, and the other part is transformed in the next iteration.

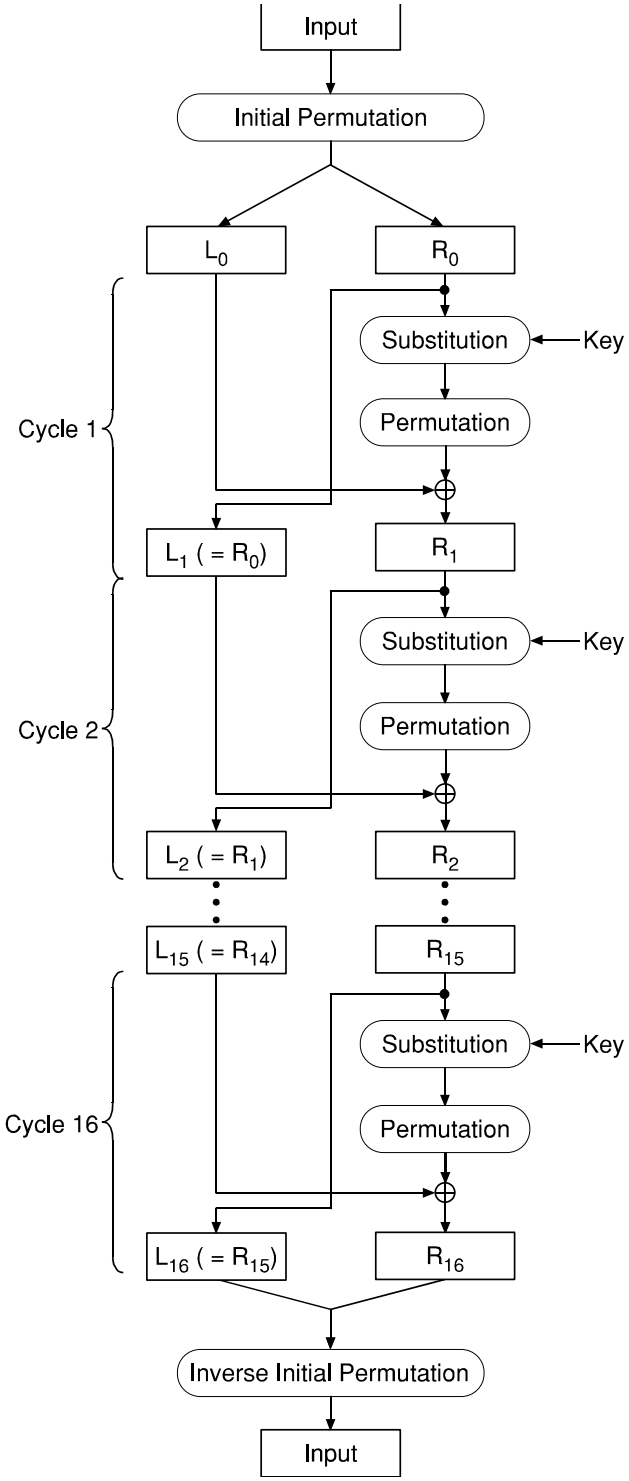


Figure 3.6: **The DES Cipher Iterations**

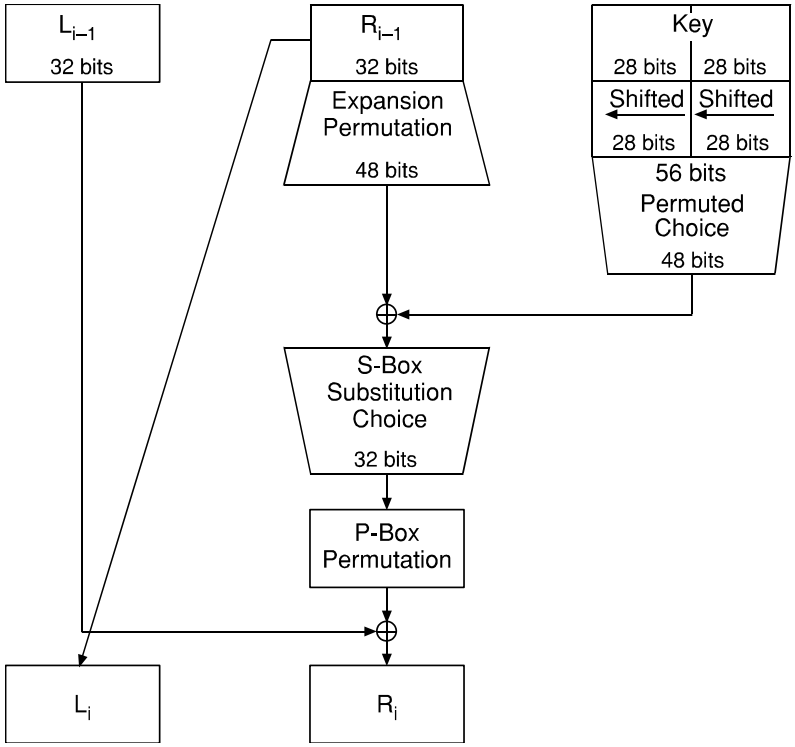


Figure 3.7: **DES single iteration**

The main steps in this transformation are:

– Expand R (the right part) to 48 bits R' using permutation and repetition

– ⊕ R' with K

– Subdivide the result into 8 blocks of 6 bits each

– Each 6-bit block goes through S-boxes to produce 4 bits (The S boxes cause the main diffusion in this cipher)

– Finally, permute the 32-bit string and xor it with the left part

The above iteration is repeated 16 times, each time with a different set of the key bits. This makes it very hard to break the cipher analytically or by using target source and ciphered text, so even the plain-text or chosen plain-text attacks usually do not help (however, see discussion on differential cryptanalysis below).

The DES cipher has two important advantages:

1. It is relatively quick to calculate and it is possible to map it into a hardware chip.

2. It is a good defense against the open text attack. Even if M and C are known the function Key = f(M,C) for the DES cipher is hard (compared to many other symmetric ciphers).

Another advantage of the DES cipher is that the cipher is invertible using exactly the same key, but using the iterations in the opposite order (from 16^{th} to 1^{st})! This is because the following property holds:

$$Rj\text{-}1 = Lj$$

$$Lj\text{-}1 = Rj \oplus f(Lj,Kj)$$

Therefore, the legal decryptor can use the same hardware chip or software package for decryption.

Because the security of the cipher is in the key and not in the algorithm itself, which is public, the length of the key is crucial. DES was developed for keys as long as 56 bits (64 bits including 8 bits of parity). Many researchers claimed at the time, that this length of the key is enough in order to protect against a typical enemy, but not enough to protect from government institutions, which use supercomputers (such as the NSA). This caused great controversy and debate. Nevertheless, the standard was adopted, and both software and hardware products have implemented it, and it also became the standard for SSL, the common secure Internet protocol (see Chapter 7).

However, two adverse developments happened. First, the DES cipher was broken using several thousands of machines on the Internet [Des99]. Second, a new cryptanalysis technique called ***differential cryptanalysis*** was developed by Biham and Shamir [Bih90], and using this technique and chosen-text attacks, the DES cipher was broken. This raised the need to develop a new standard; the AES cipher (see below).

Some methods were suggested to overcome the weakness of DES towards the brute-force attack by using multiple rounds of DES encryption. These are denoted as: *Double and Triple DES*. They are shown below:

Double DES: $C = E_{k1}(E_{k2}(M))$; $M = D_{k1}(D_{k2}(C))$

Triple DES: $C = E_{k1}((D_{k2}(E_{k3}(M)))$; $M = D_{k1}(E_{k2} (D_{k3}(C)))$

DES may also be used in various block cipher modes (ECB, CBC). This is especially useful for message authentication. For example, to authenticate a message using DES, Alice applies the block cipher in CBC mode, and sends the message M along with its signature, known as message authentication code (MAC). Bob computes the CBC-MAC (as Alice did), and accepts the message as genuine if this value agrees with the received MAC. We will discuss more about MAC and signatures later.

The Advanced Encryption Standard (AES) cipher

As the result of recognizing DES weaknesses, the NIST in the late 90s published a call for a new standard to replace DES as the symmetric cipher for the next 20 years. Several proposals were submitted, and after a fierce competition and thorough evaluation the Rijndael algorithm was selected. This algorithm was selected both because of its strength and its flexibility.

The AES [Nist01] uses block length of 128 bits and key sizes of 16, 24, or 32 bytes. It uses 10, 12, or 14 rounds. Each round applies four operations: byte substitution, row shift, column mixing, and key addition. Briefly the four operations are:

Byte substitution. Bytes are transformed using invertible substitutions (to add nonlinearity). The substitution table based on inverses in a field of (256) assures that each input byte is substituted into a unique output byte.

Shift row. Each byte is shifted a number of bytes depending on its location in the block, and on the key length. Row shifts provide diffusion over multiple rounds.

Mix column. This is the most complex operation. Bytes in columns are linearly combined for diffusion.. Each column is multiplied by a matrix which represents a polynomial mod 256; that is, each column is considered a polynomial: $-a_j(x) = a_{0j}x3 + a_{1j}x2 + a_{2j}x + a_{3j}$ –multiplied modulo $x^4 + 1$ with a fixed **Key addition**. Key addition makes each round key dependent. Before the first round, the key is expanded into Nk bytes where Nk is the size of the block times the number of rounds. Then, in each round the next required number of key bytes are extracted, shifted and xored between them resulting in a great key diffusion.

The entire process is depicted in Figure 3.9. Because of the "Mix column" operation which is a modulus arithmetic operation and not a simple substitution or permutation, the differential cryptanalysis method cannot succeed, and the result is a cipher which so far is unbreakable except for the "exhaustive search" method.

Other symmetric ciphers

Here we should mention IDEA [Lai90] which is based on modulus arithmetic and its key size is 128 bits, and RC5 which was invented by Rivest et al [Riv94], and uses a variable size key and message (between 1 and 255 bytes!), and is based on simple transformations such as: xors, shifts and rotations. The RC4 and RC5 are particularly important in the new standards for wireless security such as WEP (see [Bhag2004]).

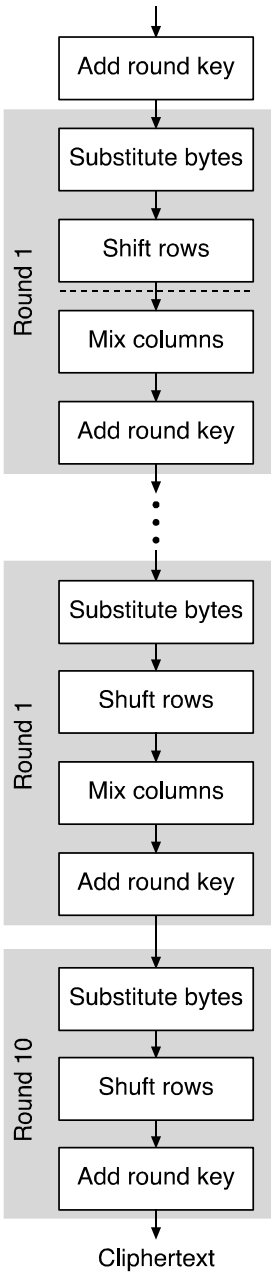


Figure 3.9: **Main steps of the AES**

3.4 Public key ciphers

The idea of public-key schemes was introduced by Diffie and Hellman in 1976 [Diff76]. The main problem with symmetric encryption is that to be secure, it requires periodic change of the common encryption key. However, in symmetric cryptographic systems, a secure channel is required for key exchange, which makes the scheme inconvenient (the "knight on a horse" problem). The main feature that distinguishes public-key from symmetric-key schemes is the separation of encryption and decryption capabilities.

The public key algorithms use two keys, one of which is public and the other secret. The approach is based on the infeasibility of determining the decryption key given the algorithm and the public key. Instead of permutations and substitutions these algorithms use properties of mathematical functions. In particular, they use the theory of NP functions, those for which there is no known polynomial time solution algorithm. The most famous public key cipher is the RSA cipher developed by Rivest, Shamir, and Adelman [Riv78], which is used in most current systems. This cipher takes advantage of the difficulty of factoring a large number into primes.

In the public key scheme the public key is known (or can be disclosed) to all users requiring communication (like a telephone number) Therefore, in a system with N users, and a request for private communication between each pair, the public key scheme requires only 2N keys, while the symmetric scheme requires N(N-1)/2 keys!

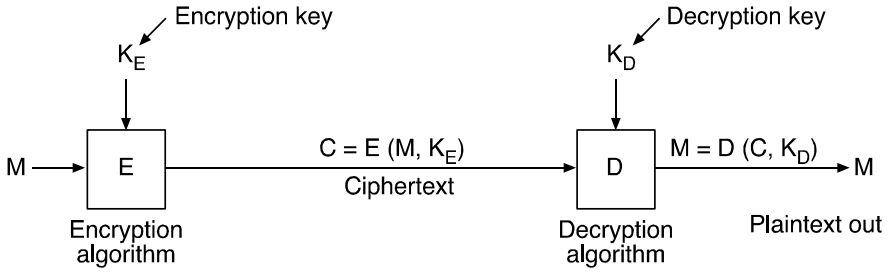


Figure 3.10: **Public key Encryption/Decryption**

As shown in Figure 3.10, in a public key system we use two keys, a public and a secret (private) key. In the encryption process we use the public key and in the decryption process we use the secret key.

$C = E(M, Key_{pub})$ or $C = E(M,K_E)$

$M = D(C, Key_{priv})$ or $M = D(C,K_D)$

While the idea of a public key was first suggested in 1976, no efficient implementation of the idea was discovered at that time. Today, there are several different public key methods, where RSA is the most famous, but we would like to review at least one more method. One of the first proposed methods was developed by Merkel and Hellman [Merk78]. Although the method is not very useful today because of some flaws that were found in the method by Shamir [Sham82], it is very elegant and therefore worth studying. The method is based on the **Knapsack** problem. The knapsack problem is a well known packing problem, in which one has to pack into a given box as many as possible various size boxes, and leave minimal wasted space. This problem is known to be an NP-complete problem. In the Knapsack-based method the sender encrypts the message with a very difficult knapsack problem; however, the receiver can break it by solving an easy knapsack problem. The enemy, on the other hand, has to solve a difficult knapsack problem, which is known as a problem from the NP-hard class of problems, and from here the safety of the cipher. Next we describe it in more detail.

The Merkle-Hellman Technique

The Merkle-Hellman [Merk78] encryption technique is a public key cryptosystem. The cipher uses a set of integers defining a knapsack problem, which solving it, is known to be NP complete. However, there is a class of knapsack problems called: simple (or super increasing) knapsacks, whose solution is linear!.

In a simple knapsack each integer is greater than the sum of all previous integers, thus at every stage, there is only one choice to make, either select the next integer, or skip it!

An example of a simple knapsack is shown in Figure 3.11 In this example the problem is posed as, find the combination of the integers:

1, 4, 11, 17, 38, 73 such that their sum is: 97 or 95

As can be seen, its very simple to solve a simple knapsack problem, by scanning the set of integers, one at a time.

97:	73? Yes	95:	73? Yes
96–73 = 24:	38? No	95–73 = 22:	38? No
24:	17? Yes	22:	17? Yes
24–17 = 7:	11? No	22–17 = 5:	11? No
7:	4? Yes	5:	4? Yes
7–4 = 3:	1? Yes	5–4 = 1:	1? Yes
3–1 = 2:	No solution	1–1 = 0:	solution

Figure 3.11: **Example of Solving a Simple Knapsack**

The main idea of Merkle and Hellman was the design of a technique for converting a superincreasing knapsack into a regular one. The trick is to change the numbers in a nonobvious but reversible way.

The steps are as follows:

Public Key Using the Knapsack Problem

1. Select a simple (super-increasing) Knapsack S.

2. Convert the problem to a hard Knapsack (select w and n relatively prime)
   H = w*S mod n

3. Encrypt: C = H*M mod n - i.e. multiply M by a hard knapsack

4. Since H is hard C is hard to break

5. Decrypt:

$$C' = w^{-1} * C = w^{-1}w\ S\ M = S\ M \bmod n$$

Since S is simple; M can be computed easily by solving S!

In this scheme, the public key is the pair: H and n, so that anyone can encrypt messages using this key. The private key is *w*, since with *w* the legal receiver can convert the hard knapsack into a simple one, and then solve it linearly!
Let us see an example:

Knapsack Example

1. Assume the simple knapsack is: = (1, 2, 4, 10).
2. W = 15, n = 17, 15, 8 = 1 mod 17, i.e. the inverse of W is 8.
   The Hard knapsack: is computed by W*S mod n = (15, 13, 9, 14).
3. The Message is = 1100 1010 1011 0100
4. Encryption - each 4 bits is multiplied by the hard knapsack:

$$P = 1100\ 1010\ 1011\ 0100$$

$$[1, 1, 0, 0]*[15, 13, 9, 14] = 28$$

$$[1, 0, 1, 0]*[15, 13, 9, 14] = 24$$

$$[1, 0, 1, 1]*[15, 13, 9, 14] = 38$$

$$[0, 1, 0, 0]*[15, 13, 9, 14] = 13$$

5. Decryption - we multiply the encrypted message by the private key i.e.
   Inverse (W) mod n:

$$28 * 8 = 224 \bmod 17 = 3$$

$$24 * 8 = 320 \bmod 17 = 5$$

$$38 * 8 = 304 \bmod 17 = 15$$

$$13 * 8 = 104 \bmod 17 = 2$$

6. Now we have to solve each of the resulted simple knapsacks how?
   We have to find M that solves C = S*M

$$C = 3, S = (1, 2, 4, 10) \quad M = (1, 1, 0, 0)$$

$$C = 5, S = (1, 2, 4, 10) \quad M = (1, 0, 1, 0)$$

$$C = 15, S = (1, 2, 4, 10) \quad M = (1, 0, 1, 1)$$

Etc.
And the recovered message is thus 1100101010110100

In-spite of the elegance of the above method it has two problems:

1. With large numbers it's quite inefficient
2. Recently researchers have found some weaknesses that enable the breaking of this cipher. [Sham82]

The RSA cipher

The next method is the RSA one. The RSA cipher was developed by Rivest, Shamir, and Adelman and it is used in most current systems. The RSA cipher is based on number theory and takes advantage of the difficulty of factoring a large number into primes. There are two keys, e (public) and d (private) so that:

$$C = P^e \bmod n$$

$$P = C^d \bmod n$$

Therefore $P^{de} \bmod n = P^{ed} \bmod n = p \bmod n$
This cipher is based on the following theorems from number theory:

Fermat's Little Theorem:

Given a prime number P, then for every integer not equal to P or 1

$$a^{p-1} = 1 \bmod P \quad or$$

$$a^p = a \bmod P$$

Euler's Generalization of Fermat theorem:

Let n be a composite, denote φ(n) the number of integers less than n which are relatively prime to n (do not have a common factor larger than 1)
Then for any *a* relatively prime to n

$$a^{\varphi(n)} \equiv 1 \pmod n$$

Now suppose N is a multiplication of two (large) primes: P and Q
Then its easy to show (do it!) that the number of integers less than N and relatively prime to N is exactly: (P-1)*(Q-1).

Therefore for every a relatively prime to N $a^{(P-1)(*Q-1)} \equiv 1 \pmod N$

Now suppose one selects an integer e relatively prime to (P-1) (Q-1), then according to Euler theorem, there exists a d, such that:

$$ed \equiv 1 \pmod{(P-1)*(Q-1)}$$

Now take any integer P < n then: $P^{ed} \equiv 1 \pmod N$ (see proof next.)

Proof of RSA

We need to show; $X^{ed} \bmod N = X$

$$ed = 1 \bmod (p-1)(q-1)$$
$$ed = 1 + K(p-1)(q-1)$$

Now, from Euler generalization $X^{k(q-1)(p-1)} \equiv 1 \bmod p = rp + 1$

Similarly, $X^{k(p-1)(q-1)} \equiv 1 \bmod q = sq + 1$. Equating both expressions:

$$rp + 1 = sq + 1 = t \cdot pq + 1 \equiv 1 \bmod pq \quad \text{(this is so because p and q are prime!)}$$

$$\text{Therefore, } X^{ed} = X^{1+k(p-1)(q-1)} \equiv X \bmod pq = X \bmod N$$

From this we can design the following cipher:

Take N the product of two large primes: P and Q. compute (P-1)*(Q-1).

Select e relatively prime to (P-1)(Q-1). Find the inverse d.

The public key is chosen as the pair (N, e)

The private key is the pair (N, d)

The encryption of a message M is : $M^e \bmod N$

The decryption of a message C is : $C^d \bmod N$

since $M = M^{ed} \bmod N$

Obviously, the entire security of RSA lies with the fact that factoring a large number (N) into its multiplicative terms (P, Q) is a very difficult computational problem. Clearly, If one can factor N to get p and q, one can then get (p-1)(q-1), then one can get d from e and (p-1)(q-1) easily.

RSA Example

Example:

Let $p = 5$ and $q = 7$, hence $n = pq = 35$ and $(p-1)(q-1) = (5-1)(7-1) = 24$.

Pick $d = 11$. Then $e = inv(11, 24) = 11$

Suppose $M = 2$.

$$C = M^e \bmod n = 2^{11} \bmod 35 = 2048 \bmod 35 = 18$$

And

$$C^d \bmod n = 18^{11} \bmod 35 = 2 = M$$

Another Example:

Let $p = 53$ and $q = 61$, hence $n = 53*61 = 3233$ and $(p-1)(q-1) = 52*60 = 3120$.
Letting $d = 791$, we get $e = 71$. To encipher the message M = RENAISSANCE, we

break it into blocks of 4 digits each, where A = 00, B = 01,…, Z = 25, and blank = 26 (in practice, characters would be represented by their 8-bit ASCII codes).
We thus get

M = R E N A I S S A N C E
= 1704 1300 0818 1800 1302 0426

The first block is enciphered as 170471 = 3106. The entire message is enciphered as:

C = 3106 0100 0931 2691 1984 2927

Note that in reality we do not need to work with very large numbers. For example X**71 Mod N is actually ((X**8) Mod N)**8 Mod N * X**3 which is much easier to compute! (try this method in computing 18^{11} Mod 35…)

We like to make an important comment here. In the example above, although the cipher looks complex, it is a simple duo-alphabetic cipher and very easy to break! The reason is that we used only two characters (16 bits) to encrypt each time. In reality we'll choose at least 128 bits keys and numbers; therefore we will encrypt at least 64 bits (8 bytes) at once, which makes any exhaustive search or statistical analysis prohibitive.

Exercise:

Let denote z as (p-1)(q-1). What is the importance of the demand that d and z do not have a joint factor?

Answer:

This demand insures that there is a number e for which e*d = 1 mod z. For example, for p = 7 and q = 3, meaning z = 12. If we had chosen d = 6, than there does not exist e number such that 6e = 1 mod 12.

Next comes the problem of finding large prime numbers. Remember that the method depends on finding large P and Q which are prime. How we can insure that a very large integer is a prime? Trying to factor it is too expensive computationally. Several techniques exist to do that, the most famous one uses the Miller-Rabin test [Rab76]. In this test, ones use the concept of a ***Witness***. Suppose we generate a large number (e.g. 512 bits!) S. Now we start the test, we generate another number *a* and compute a^{S-1}. If the result is different than 1 then we know that S is not a prime, otherwise we find another witness. it can be shown that with K witnesses we can reduce the probability that S is not a prime to 2^{-K} !. (There is actually a small set of numbers call Charmichael numbers, for which this test doesn't work, see [Corman02] for details). Recently, a new method was invented that finds whether a number is a prime deterministically (and not probabilistically as Rabin-Miller test) in polynomial time, see [Mah2003], but it is quite .complex, so the miller-Rabin test is the most popular

In terms of the difficulty of factoring large numbers, there is a large amount of research on the problem, but no efficient method was yet discovered. The largest number that has been factored was a number of 129 digits (467 bits) which was factored out by 1600 computers working in parallel. No factoring of 512 bits numbers has been discovered yet.

Although there have been many attempts to break it, the RSA cipher proved itself very safe, which is the reason it is very popular these days. However, because calculations with big numbers are slow, this method is used mostly to generate a symmetric key in a secure way and then using a symmetric key for encrypted messages. We will see it for example in chapter 7, when we discuss the Kerberos protocol. We summarize below the main properties of RSA.

RSA Properties:

1. Very secure! In order to find d from e or vice-versa, you need to know p and q. This means you need to find the two terms which divide n, and this requires solving the factorization problem.

2. A relatively slow cipher since arithmetic in modulus for large numbers is very slow!

3. The cipher can be used for digital signatures as will be discussed in the next section.

Another method for Public key encryption is called: **El Gamal** [Elgamal84]. It is based on the following principles:

1. Instead of using difficulty of factoring, we use difficulty of taking discrete logarithms. That is, in Modulus arithmetic, even if we know X and $Z = X^{\wedge y}$, it is very difficult to compute Y from X and Z!

2. The prime number of which modulus arithmetic is done has a known generator g.

3. The secret key is a random number s, and the public key is Kp = g**s.

Briefly, the method works as follows:

1. To encrypt a message Bobs picks a blinding exponent (e), multiply the message by Kp^e, and also sends g^e;: i.e. (g^e, $M \cdot Kp^e$)

2. Alice computes $(g^{\wedge}e)^{\wedge}s = Kp^e$ since she knows the secret key s, and divides to recover the message M.

The El Gamal algorithm although very secure, is used more for signing messages than for encryption, because it doubles the length of encrypted messages.

Recently there have been work on another public key cipher called **elliptic curves**. Their main properties are:

1. Elliptic curves are more efficient in their key-length (i.e. provides better security for the same key length). For example, an elliptic curve algorithm with a key length of 150 bits takes 3.8 x 10^{10} MIPS-years to break in exhaustive search, while RSA with a key length of 512 bits takes only 3 x 10^{4} MIPS years.

2. Because of their small memory requirements, elliptic curve algorithms are appealing for portable devices.

Man in the middle attack

One problem facing all public key schemes is the fact that a user wanting to send a message to another user needs to know her public key. This may lead to a famous attack called "Man in the Middle Attack". This is depicted in Figure 3.1

1. Alice sends Bob the message “I am Alice” with her public key (which is not known), encrypted by Bob’s public key (assuming it is known), meaning, Alic Msg = E (K_B, K_A|| I am Alice)

2. Bob sends Alice the message “I am Bob” encrypted by Alice’s public key, and this is the way they believe each other.

Now assume that Matt disguises herself as Alice and sends Bob the message
Matt_msg = E (K_B, K_M|| I am Alice)
where K_M is Matt's public key. Now, Bob will encrypt the message with K_M and Matt will decrypt the communication from Bob to Alice. Both Alice and Bob will think they talk to each other, while they are actually talking to Matt. An attack of this type is called “Man in the Middle Attack” and it is prevented by authentication of each side using ***Digital Signatures*****.** This will be discussed next. In more detail.

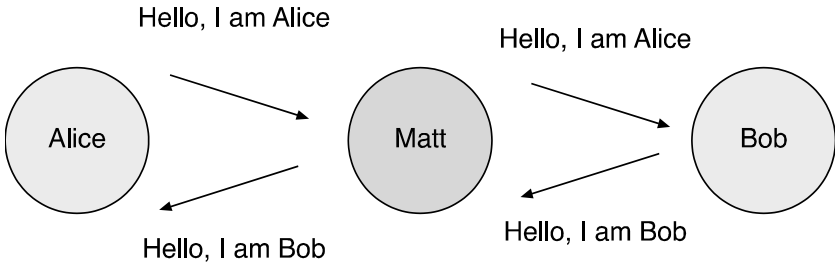


Figure 3.12: **Man in the Middle Attack**

The above was just an example, other forms of "man in the middle" attack are possible.

Digital Signatures and Hash Functions

Digital signatures have become a very important issue in our electronic world. Since many day to day transactions are done electronically, signing them and making sure the signature is authentic becomes extremely important! Not surprisingly, Cryptography plays here a major role. In general we require the following requirements from a digital signature:

a) Authenticity - the signature must be authentic, i.e. belongs only to the signing user

b) Unforgeable - no other user can sign on behalf of the valid user, or forge his signature

c) Nonrepudiation - the signing user, once he has signed, cannot deny his signature. That is, it is impossible to claim that a certain message was “made up” by someone else and has the sender’s authentic signature.

In theory one can use a symmetric key for signatures, but this process is not very convenient, since it requires the use of a third party. The sender S and the third party have a common secret symmetric key. The sender S encrypts the message using their common key and sends it to the third party. The third parties decrypts it and send both the decrypted and encrypted messages to the receiver R possibly encrypting them with a common key with R. R trusts the third party for the authenticity of the signature, and S cannot deny it because only him and the third party know the encryption key used by S. Unfortunately, this process involves a third party. Using public keys one can avoid the third party as described below.

Because of the symmetric roles of encryption and decryption in Public Key cryptography, i.e !D(E(M)) = E(D(M)) = M, this symmetry becomes the basis for the public key signature scheme :The signature process done by the sender is simply the encryption D(M); that is

$$SIG = D_{\text{private key}}(M)$$

Now using E which is known, everybody can recover M. However, only the sender could have :signed it, since only he knows the private key, that is, the signature verification scheme is

$$M =? E_{\text{public key}}(SIG)$$

meaning, we check whether the two sides are indeed equal. This way, the three requirements above are guaranteed! The signature is authentic, nobody can forge it and the sender cannot deny .signing it

This all relies on the fact that its very hard to compute the private key from the publicly known public key. The simplicity and elegance of this signature scheme is obvious! However, it has two :major disadvantages

1. The signature is at least as long as the message itself, which doubles the length of each signed message. This happens because both M and sig(M) are sent in one message.
2. It is impossible to distinguish between the original message and a copy of it; that is we are susceptible to replays.

We next show how to overcome the first problem, the second is solved with the help of certificates or time-stamps.

The first problem is solved by the technique of **hashing**. Instead of signing the original message we sign a ***Digest*** of it which is obtained by hashing. Hashing is a mapping from a large source space to a smaller target space, such that it is very difficult to find the source, by knowing the target. It is commonly used in database indexes and also in encrypting passwords. Obviously, a basic requirement is that it is unlikely for two different messages to have the same hash. How is this obtained?

Hashing is usually performed with one of several well-known cryptographic hash functions. Such functions involve a complex combination of the bits of the original message which results with a much shorter non-invertible digest. Because of the complexity of the hash functions, even though it is not free of collisions, it is extremely difficult to find a "twin" from a given hash value, i.e. given K1 and its hash value H(K1), its difficult to find another K2, s.t. H(K1) = H(K2).

This resembles the "Birthday paradox". The birthday paradox implies that in a group larger than 23 people, there is a high probability (above -0.5) that two people will have the same birthdates. Yet, to find a collision for a particular person means searching linearly the entire group. For a 128 bits key it means searching a space of size 2^{128} !.

There are several standard hashing functions. The best known ones are: MD4, MD5 and SHA. All of them rely on the following principles:

1. The message is divided into 16-word blocks. Each block generates 5 words H0… H4 using a complex computation using a series of 'ands', 'ors', and 'xor' with some given constants (see [Riv92] for details), and the final five words are xored with the earlier group of five words. Overall we obtain a hash word of size 128 (MD5) or 160 bits (SHA).

2. There is a very strong diffusion. And therefore the signature looks completely random. Also, because of the strong diffusion, even a change of one bit in the original message, will result with a completely different signature. This guarantees the use of such hashing for authenticating the messages (i.e. that nobody modified them even by 1 bit!).

The algorithm recommended by NIST is today a version of SHA called SHA-1. It has the advantage that input bits are used more often then they are in the other Hash functions like MD4. (creates more confusion). Thus it creates a more secure function although it is a bit slower.

With the use of Hashing, one not only can sign messages more efficiently, but can also prevent the replay of old messages by including a 'time-stamp' in the signature. One method divides the time stamp into parts and sticks each part to a different encrypted block. Another option is to perform the hash function both on the message and on the time stamp at the same time:

$$CT = H(C+T),$$

where T itself has to be known to the receiver (generated by a synchronized clock).

Apart from signing messages, hashing is also used to assure message integrity. See the discussion below on MAC - message authentication code.

Other signature schemes

Previously, we showed how to use any public key scheme, and in particular RSA, for generating signatures. Here we like to note that there are schemes which are mainly used for signing and generally not used for encryption. Two of these schemes are the El-Gamal cipher which is based on the difficulty of computing a discrete logarithm (see section 3.3.2), and the DSA scheme, which is based on El-Gamal and has been adopted as a standard in the US [DSS94].

One technique which is an elegant application of electronic signatures is called a ***blind signature***. This technique is used for users to ensure that a document is signed by a well-known authority, e.g. certifying a check, where the authority is really not interested in the content of the message. The method is as follows:

Blind signatures

Alice wants the Bank to sign M without the Bank knowing M (that's the reason for the term "blind").The Bank keys are: (e, d)

First Alice sends: $x = M r^e \bmod n$ where r- is a random number.

The Bank then computes: $x^d = rM^d$ and sends the result to Alice. Finally, Alice computes: $x^d / r = \underline{M^d}$. The end result is M signed by the Bank!

The Digital Signature Standard

In 1991 the US standards body NIST announced a new proposed digital signature standard called DSS [DSS94]. The DSS is essentially the El-Gamal signature scheme but works using a smaller prime-order group, and therefore takes less space. The signature itself is hashed by the standard hash function SHA-1 to get a standard size signature (e.g. 160 bits).

MAC - Message Authentication Codes

Earlier, we discussed the goals of digital signatures as both providing authenticity, and non-repudiation. However, the problem of authenticity of messages, especially against malicious users trying to modify them, received attention even before the era of electronic signatures. Such authenticity was achieved using MACs - Message authentication codes.

MACS were usually generated by using a symmetric key (e.g. DES), and were attached to the original message to ensure its authenticity and integrity.

The Message Authentication Code Based on DES is constructed as is depicted in Figure 3.13.)The Algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17))

The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES with an initialization vector of zero. The data (e.g., message, record, file, or program) to be authenticated is grouped into contiguous 64-bit blocks: D_1, D_2, …, D_N. If necessary, the final block is padded on the right with zeroes to form a full 64-bit block. Using the DES encryption algorithm, E, and a secret key, K, a data authentication code (DAC) is calculated as shown in Figure 3.13.

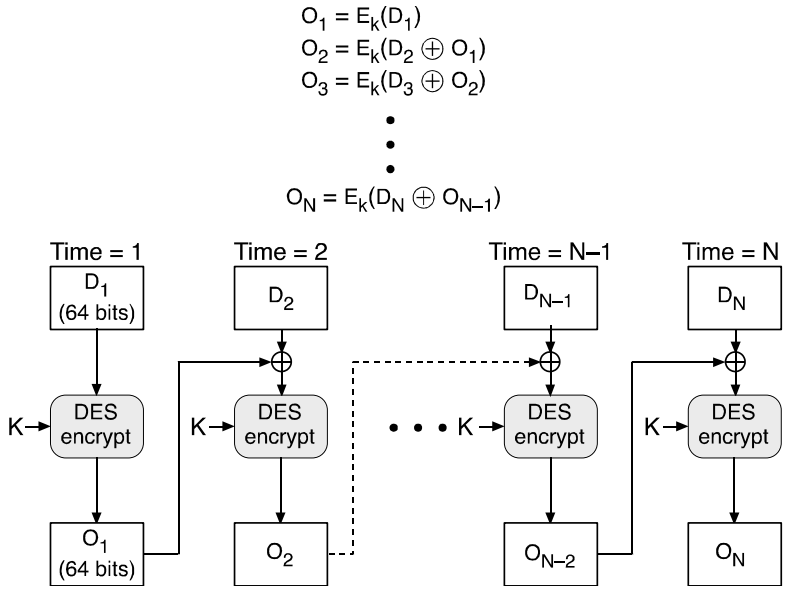


Figure 3.13: **Generation of DAC based on DES**

The DAC is then attached to the message to become a MAC. Although DES is being phased out, there are still current protocols which use this scheme for authenticity and integrity.

Certificates

Earlier we discussed the problem of signed messages being copied by illegal users. To overcome this problem, and to make sure that public keys are valid, and to prevent the "man in the Middle" :attacks, one uses the notion of ***Certificate*s**. Certificates use the following principles

ㅎ ㄴ The public keys are normally registered with a ***certification authority*** (CA). This authority distributes ***certificates***, which are public keys with the signature of the CA.

ㅎ ㅂ There are authentication and attribute certificates. Attribute certificates assert that certain properties are true of the owner of some authentication certificate. Attribute certificates are also used in SSL and other protocols (see chapter 7).

The idea of certificates is not new at all. Actually in every ID card such as a social-security card, or a passport, there is someone that signs that the card is authentic. Usually we are content with one authority level, for example for SSN - the signature of the Social Security administration. But, if we had seen a student card from a university that we are not familiar with, we might have asked for a proof that this university does exist and its signature is authentic, that is two authority levels.

Assume initially that we have one authority level. It is usually called the CA for Certificate Authority. We can look at the content of a certificate as composed of two parts: CertA and CertB, where CertA = {Attr, Key} - where Attr is the set of attributes of the user carrying the certificate (such as name, Degree, etc.) and Key is his public key, and CertB is the signature of the authority issuing the certificate that CertA is authentic. The signature is constructed as follows:

CertB = Sig = $D(H(CertA), K_D)$ where K_D is the private key of the issuing authority.

Since the public key of the issuing authority is known, the receiver of the certificate can easily verify it by performing:

$H(CertA) = ? E(Sig, K_E)$ and check for the equality.

Because decentralization exists in all the computer systems and in particular in the Internet, the process of giving the certificate usually involves several levels. There is usually one global authority which is recognized by everyone, and whose public key is well known (e.g. on the Web, almost every Browser is familiar with Verisign and its public key…). The first level is then signed by this global authority, and the level below is signed by the first sub-authority, etc.

Each such level has the ID, Name and Public key of the certificate holder, and it is signed by the private key of the level above. This idea is well-explained in Figure 3.14.

The receiver of the certificate, can perform the hash function on the clear information, then encrypt it with the public key of the higher authority and finally compare it with the certificate signature and verify its correctness, and repeatedly do it for all the certificate levels. Notice that the public key in the highest level must be known to the user. For example it is saved within the web browser. (usually a standard Web browser comes installed with several such keys, e.g. the one by Verisign.)

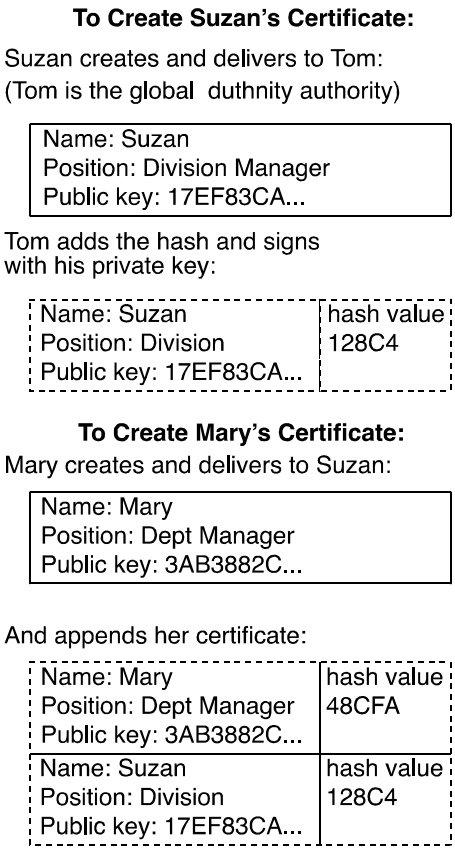


Figure 3.14: **Example for Certificates Hierarchy**

Today certificates are an essential part of any PKI (Public key Infrastructure) scheme, and therefore there is a well known standard to define them called: X509. The main attributes in a X509 certificates are shown below:

Fields of the X.509 Certificate

- **Version** – which version of the standard
- **Serial number** – unique, used for Revocation
- **Algorithm Id** – which algorithm is used for the signature
- **Issuer** – the CA issuing the certificate

- **Period of validity**
- **The subject** – attributes describing the subject (name, position, etc.)
- **Public key** – of the subject
- **Signature** – of the above by the private key of the CA

In a dynamic system, it may be necessary to revoke certificates from some users. Revocation of certificates is a very active area of research. One popular method is the use of revocation lists (CRL) which a verifier must consult before accepting a certificate. In general there are two methods, the "push" method which "pushes" the CRLs to various servers, and the "pull" method which requires the verifiers to pull the CRLs for verification. The interested reader may refer for example to [Wohl00].

Key Escrow - Clipper

The possibility of the ease of encrypting messages by both legal users and criminals has become a real problem for law enforcement agencies such as the Police. Usually, it is possible for the police to get a legal permit to “listen” to conversations, but what about conversations over the Internet or digital technology such as cellular phones when the messages are encrypted? It must be possible for the police to decrypt such messages with a court order. Since it is assumed that standard telephone or internet channels are used, it was suggested to introduce an encrypting chip to each node in the network called: ***Clipper***. Each encrypted message is attached with an encrypted part (called 'leaf') which allows deciphering it with the right key which will be obtained by a court order. Although the Clipper suggestion was not accepted eventually, it is useful to understand the idea, since it presents some balance between privacy and law enforcement needs.

The structure of the suggested clipper is depicted in Figure 3.15. Each message is encrypted with a symmetric key (e.g. DES) which is dynamic and known only to the two nodes transferring the message. The node that sends the message with identifier ID, encrypts the symmetric key with the help of a network node key K_{ID} not known to anybody, and creates the leaf which is the ciphered K and its identifier concatenated. The node then creates an encrypted form of the leaf by encrypting it by a special key known only to the police called Kpolice. The message is encrypted with K and concatenated with the encrypted leaf. Obviously, nobody can decrypt this message, since he does not know K nor K_{ID}. When the police likes to listen to a conversation, it decrypts the leaf, obtaining the node number ID. Then the police requests the court to issue it the key K_{ID} of the network node ID, and then can use it to decrypt the symmetric key K, and then and decrypt the message.

The subject of Clipper and key trustee is very “hot” and “political” and received a lot of echoes from the public, the government and the congress. Eventually it was dropped, but the issue remains an important one.

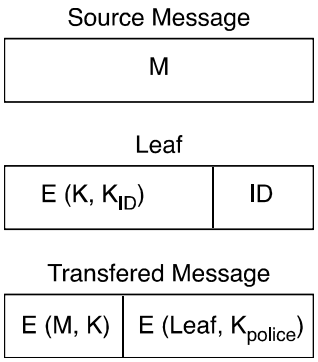


Figure 3.15: **The Clipper Operation**

Cryptography in practice

Cryptography has become an extremely active area, and many companies are producing both cryptographic hardware and software. Some of the known products are:

1) Java incorporates in its Java 2 version the Java Cryptography Architecture (JCA). This is a framework (class library) that includes several classes implementing PK functions for authentication, digital signatures, certificates, and one-way hash functions. The JCA is complemented by the Java Cryptography Extension (JCE), which includes cipher-based encryption (including DES, triple DES, Blowfish, and others), key exchange, and keyed hash functions. .NET also has a complete set of cryptographic functions. Figure 3,16 shows the class structure of their symmetric algorithms; they also have classes for asymmetric cryptography.

2) Encryption products from companies such as: Baltimore, Entrust (Northern Telecom).

3) PKI products provided by companies such as: RSA or Verisign.
In terms of performance, with Java 2 version the Java Cryptography Architecture, a common computer can achieve the following performance:

- Public-key (RSA 1024bits) - 20 signatures or 100 verifications/second
- Shared-key (DES, RC ons/second or 1MByte/second
- One-way hashing (MD s hashes/second or

15 MBytes/second, i.e. network speed

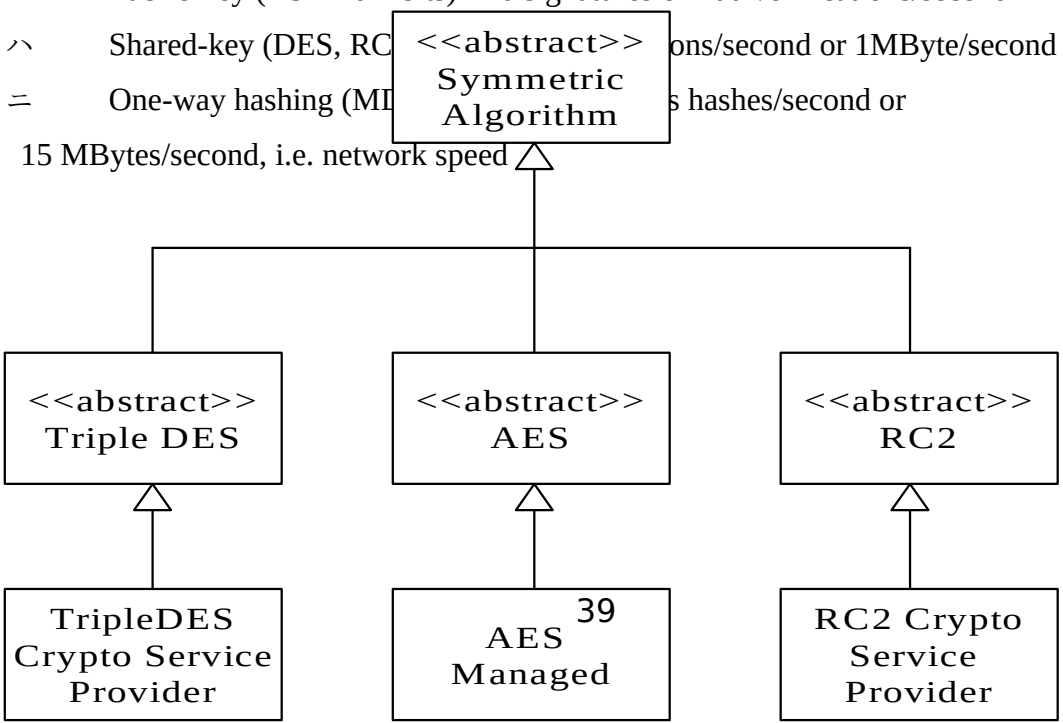


Figure 3.16 Classes for symmetric encryption in .NET

3.5 Cryptographic Protocols

Until now we discussed mainly the various cryptographic algorithms. In this section we discuss the important topic of ***Cryptographic Protocols***. Protocols are used for various purposes; the most important ones are for sharing keys, and for authentication purposes. The protocols we describe here usually form the basis for the more complex protocols which are used as standard such as the Kerberose and the SSL protocols and are which are discussed in Chapter 7.

In general, a protocol is an agreement, which the participants agree to fulfill, and if it is being fulfilled than it must (and it is possible to prove it) give the needed assurance. We can distinguish between several types of protocols:

1. Protocols which involve only both (or more) participants.
2. Protocols which involve a third party whom all sides trust and it is a part of the protocol.
   Protocols which involve a third party, which is not a part of the protocol but is able to "judge" if one of the sides cheated or did not obey the protocol.

Usually when describing the protocols we will use "names" such as Bob and Alice for participants, and some other names for third parties or adversaries.

Note that in all protocols we talk about two types of encryption algorithms: symmetric and asymmetric (public key), but we do not require a specific algorithm.

3.5.1 Protocols For Key Sharing

One of the hardest problems existing in each encryption based system is the problem of key sharing and key management. Key management is important because keys are replaced often. Especially in the case of symmetric encryption, then the changing of the keys is essential for better protection. We assume that in all protocols the goal is to share a given symmetric key.

Some protocols will only use symmetric encryption, while others will use also asymmetric encryption. We will start with the simple ones and proceed with more complex protocols.

Key sharing with symmetric keys without third party.

This is the simplest protocol. Whenever two users require sharing a key, they encrypt it with the previous symmetric key. This, however, does not provide a solution to the initial key sharing. It is also not safe if the previous symmetric key was exposed.

Key sharing with symmetric keys with a third party.

In this protocol we assume the existence of a third party called: KDC - Key Distribution Center. It is assumed that both Bob and Alice have a secret symmetric key shared with the KDC, let us call them Kb and Ka respectively. The protocol is as follows:

1. Bob asks the KDC for the shared key to communicate with Alice.
2. The KDC sends Bob two parts of a message:
   E(Key, Kb), E(Key,Ka)
3. Bob decrypts the first part to obtain Key, and send the second part to Alice who decrypts it to obtain the same Key.
4. Now Bob and Alice can communicate using Key.

The problem with this protocol is that it requires the use of the KDC for every key sharing operation and that none of the sides involved is authenticated, i.e Bob cannot be sure he is talking to the KDC, nor Alice can be sure she is talking to Bob. We'll see later how to overcome these problems.

Key sharing without third party - the Diffie-Helman Protocol

The Diffie-Hellman protocol is one of the most elegant solutions for key sharing without the use of a third party. It is often used as part of other protocols. This protocol uses the difficulty of calculating discrete logarithms. We shall define the primitive root of a prime number P as the number α such that: $\alpha \bmod P$, $\alpha^2 \bmod P$,… $\alpha^{p-1} \bmod P$ generates all the integers between 1 and P.

When we are given $b = \alpha^i \bmod P$, i is the discrete logarithm of b. Now, when P and α are very large, calculating the discrete logarithm is very hard!

Creating the joint key according to D&H looks like in Figure 3.17:

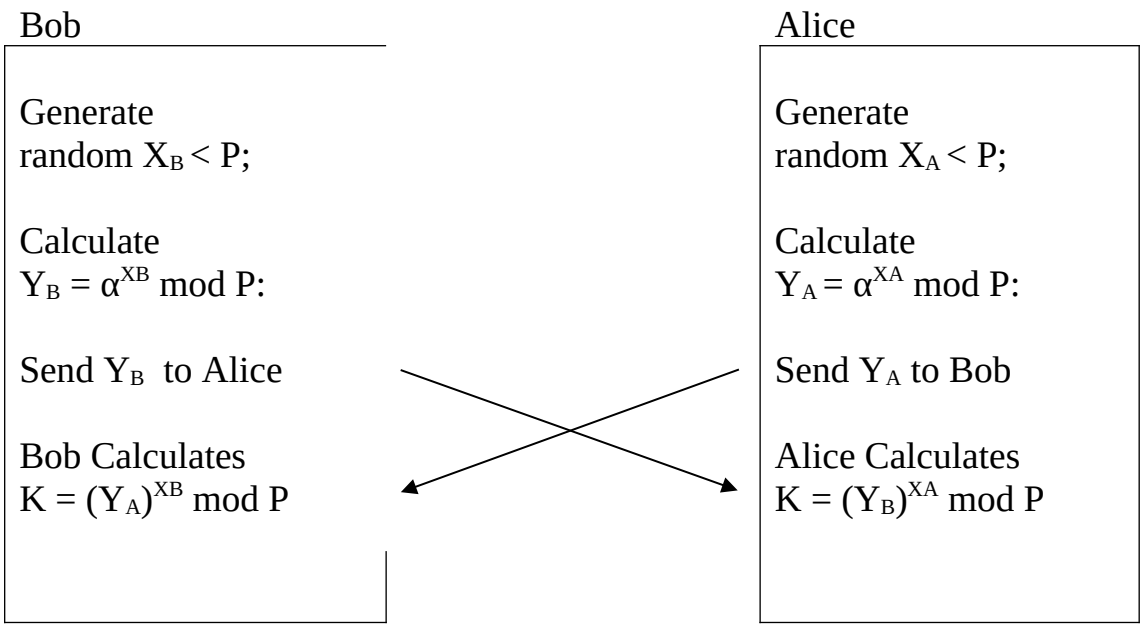


Figure 3.17: **The Diffie-Hellman protocol**

Both Bob and Alice agree on P and α (they are not considered secrets!). Each one of the sides generates part of the key (X_A or X_B) and the joint key which is α^{XAXB} . As seen in the digram, creating the key is fast and easy, but in order to find the joint key, the enemy has to calculate the discrete logarithm, for example to find X_B he has to find the discrete logarithm of Y_B according to α mod P and that could be very hard!

The problem with this protocol is that it is exposed to the "Man in the middle attack" we mentioned before. As we'll see later, it can be solved by using certificates. The next protocols attempt to solve this problem.

Key sharing with Asymmetric keys without third party.

In this protocol we have two cases. First, we assume that the public keys are known. If this is the case, then the protocol is simple:

1. Bob send to Alice E(Key,Ka)
   Alice decrypts the key and send to Bob E(n, Key) where n is a random number generated by Alice.
2. Bob sends back E(n+1,Key).

The last two steps are called: **Challenge-Response** and are used to ensure that both sides talk to each other and not to somebody else.

This protocol is very simple; however, if the two sides don't know the respective public keys there is a problem. It can be solved with the help of a KDC.

Key sharing with Asymmetric keys with a third party.

Now, Bob and Alice gets the respective public keys from the KDC. We do need to assume however, that they share a symmetric key with the KDC, or alternatively, the KDC know their respective public key, and also Bob and Alice know the KDC public key. The protocol is as follows:

1. Bob sends a request to KDC asking for public key of Alice
   KDC sends E(Ka,Kb) - public key of Alice encrypted by public key of Bob
   Bob decrypts the message and sends to Alice his public key encrypted with Alice public key i.e. E(Kb, Ka).
2. They continue as in the previous protocol.

Again, the main problem is that none of the messages above is authenticated, and that the protocol is exposed to the "man in the middle" attack.

The only solution to the above problems is the use of electronic signatures and certificates. Protocols that use that are more complicated and are called Authentication protocols and are discussed next.

Key sharing with Authentication Protocols

The first two protocols use symmetric keys. The first protocol was suggested by Needham and Schroeder [Need78]:

1. Sender A asks KDC for a secret key in order to call B. The message includes a time stamp or challenge N_1.

2. KDC reacts to the message with an encrypted message (Using A's symmetric key) which contains:

   a) The secret key (asked by A).

   b) The original A's message including N1.

   c) The message E (KS, IA, Kb) which A will send and only B would be able to decrypt it using D_{KB}. Kb is the symmetric key common to KDC and B.

3. A stores the secret key K_S and sends B the message sent by KDC. B decrypts the message and finds K_S.
4. B sends the ID of A and a challenge N_2 to A and encrypts it with K_S.
5. A sends back $f(N_2)$ encrypted with K_S.

Now both sides are positive that they are "speaking" with each other:
A is positive with K_S because he got it from KDC (with the challenge he sent).
A is positive with B because he received his challenged encrypted in K_S.
B is positive with A because he got his challenge back.

However, in this protocol, it is still possible to "cheat" if A will send an old message (using an old K_S) in part 3.

Denning [Denn81] changed the protocol so that the KDC will send the K_S including a time stamp, and the time stamp will be sent from A to B. Now the protocol looks like this:

1. $A \rightarrow KDC$: $ID_A \| ID_B$
2. $KDC \rightarrow A$: $E_{Ka}[ID_B \| K_s \| T] \| E_{Kb}[ID_A \| K_S \| T]$
3. $A \rightarrow B$: $E_{Kb}[ID_a \| K_s \| T]$
3 $B \rightarrow A$: $E_{Ks}[N1]$
3. $A \rightarrow B$: $E_{Ks}[f(N1)]$

Now assuming the clocks of B and KDC are synchronized, B can compare the timestamp T sent by KDC to his own clock, and if the difference is not too high, can accept the message as authentic.

Neuman [Neu93] improved the above protocol by using challenges in addition to timestamps as follows:

1. $A \rightarrow B$: $ID_A \| N_A$
2. $B \rightarrow KDC$: $ID_B \| N_B \| E_{Kb}[ID_A \| N_a \| T_b]]$
3. $KDC \rightarrow A$: $E_{Ka}[ID_B \| N_a \| K_s \| T_b \| N_b] \| E_{Kb}[ID_A \| K_S \| T_b]$
4. $A \rightarrow B$: $E_{Kb}[ID_a \| K_s \| T_b] \| E_{Ks}[N_b]$

In this protocol, the timestamps and the challenges assure that no replay will be possible.

Finally, let us see one such protocol using Public key encryption The protocol was suggested by Woo and Lam [WooLam92]. (Notice, in these protocols
KD – is a private key, KE – is a public key):

1. $A \rightarrow KDC$: $ID_A \| ID_B$
2. $KDC \rightarrow A$: $E_{KEA}[E_{KPKDC}]ID_B \| KEB]]$
3. $A \rightarrow B$: $E_{KEB}[N_a \| ID_A]$
4. $B \rightarrow KDC$: $ID_B \| ID_A \| E_{KEKDC}[N_A]$
5. $KDC \rightarrow B$: $E_{KEB}[E_{KPKDC}[ID_A \| KPA]] \| E_{KEB}[E_{KPKDCt}[N_a \| K_s \| ID_B]]$
6. $B \rightarrow A$: $E_{KPA}[E_{KPKDC}[N_a \| K_s \| ID_B] \| N_b]$
7. $A \rightarrow B$: $E_{Ka}[N_b]$

Notice that steps 3,4,5,6 assure A that the communication between B and KDC is authentic and immediate because the challenge that A sent B was sent back. Also notice, that KDC uses digital signatures to assure that the messages indeed came from KDC.

As was mentioned before, the above protocols are the basis for the more complex protocols used today. See for example the description of the Kerberos protocol in Chapter 7.

Key generation and protection

Obviously, a critical component of any protocol is the generation and protection of keys. This topic has received much attention in recent years. The generation of keys is usually performed as part of a Public-key infra-structure scheme (PKI). There are several standards and quite a few products for providing PKI. See [NSS200] for a good survey of the subject. Key protection is also a problem. When the key is stored in a hardware device like a smart-card, then it is reasonably secure, but if it is stored as part of a software scheme, as for example in PGP (secure mail - see chapter 7), extra care must be taken to assure its protection.

Other Protocols

In addition to the basic protocols for identification and signatures that we have seen up to now, lately different encryption protocols (most of them nice and elegant) have been developed for different problem solutions. Below, we will mention some of them.

The first protocol we like to mention is the “mental poker” protocol for “card exchanging” through electronic messages. One of the practical uses of this protocol is the division of pairs of public and private keys in a way that the person dividing the keys will not be aware which pair was chosen by the participant. Another similar protocol is used for anonymous electronic voting.

A very important protocol is used to exchange messages between two or more participants in such a way that the sender won’t know exactly which message was selected by the receiver. This is called the ***Oblivious Transfer*** protocol. For example, it is possible to use this protocol in order to assure a "coin flipping" in a fair way. In this protocol, Alice sends Bob two public keys, of which Bob selects one in random, encrypt with it a random key and sends to Alice. Alice then guesses the private decryption key, and encrypts a known message with it. If she guesses the index chosen by Bob, right, she wins, otherwise Bob wins. A similar protocol is used in the *Private Information Retrieval* Area. This protocol is very important in the theory of cryptography today

Finally, we mention a protocol named “Secret Sharing”. Here there are n users, each of which knows just a part of the secret. It is possible to know the whole secret only if at least k users from the n users cooperate. This kind of a protocol is very useful for cooperation between users who do not trust each other.

These protocols are described in great detail in several books on cryptography, see e.g. [Schn88].

3.7 Steganography

Steganography (literally meaning covered writing) is the science of hiding information within other information such as text or images. Steganography dates back to ancient Greece, where common practices consisted of etching messages in wooden tablets and covering them with wax, and tattooing a shaved messenger's head, letting his hair grow back, then shaving it again when he arrived at his contact point. For example, a watermark "hides" an image on a piece of paper. If you look at most paper currency at a low angle or if you hold it up to a bright light, you can see a

ghostly image in the paper. When you look at the currency straight on in normal light, you cannot see the image. Because this example is so easy to understand, steganography is often called "watermarking." Watermarking is a common technique to prevent forgery or illegal copying.

Today steganography is often used to hide copyright information in an image, movie, or audio file. The information is carefully encrypted and hidden so you cannot easily find it. At any point though, the original author may prove the originality of the document or image by extracting the hidden copyright information. A common technique is to use redundant less significant bits in an image representation; for example, in the least significant bits of pseudo-randomly selected pixels. In DVD movies or software programs, it is common to use the concept of "Easter eggs". In the context of software, an Easter Egg is a hidden feature or novelty that the programmers have put in their software. In general, it is any hidden, entertaining thing that a creator hides in their creation only for their own personal reasons. This can be anything from a hidden list of the developers, to hidden commands, to jokes, to funny animations.

The problem of illegal copying and violating copyrights is a very serious problem in today Internet world. The amount lost by companies which distribute movies or music due to illegal copying by Internet users is billions of dollars, so Steganography and Watermarking are very important topics. Their relationship to the current chapter is that they often use Encryption as their basic building block.

Figure 3.18 illustrates a typical steganography (or stego) application scenario. The application receives the data to hide as input—text, audio, video, or image—and the file in which data will be hidden, called the *cover file*. The *stego file* is the result of the process. Although it contains the original cover file data as well as the hidden stenographic information, the stego file is *virtually* identical to the cover file. Normally, there is also another application which verifies the existence of the hidden message in the Stego file, like verifying a watermark. Similar techniques may be used for hiding messages for intelligence or defense purposes. This is often the case when, in contrast to the use of encryption, one wants to conceal the existence of the message itself.

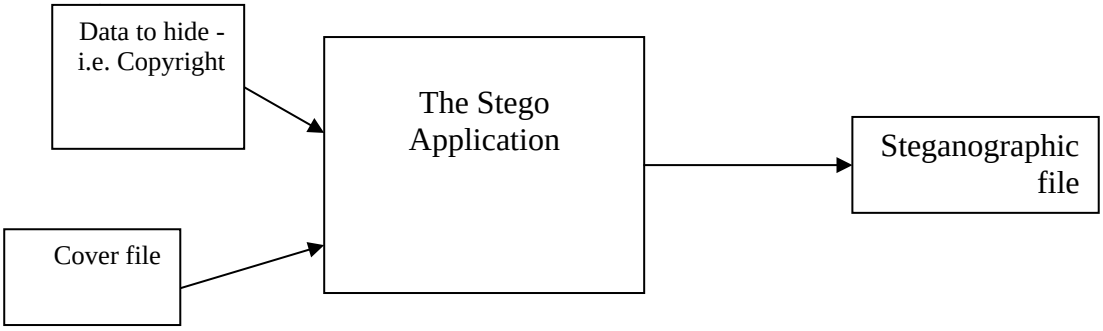


Figure 3.18: **Steganography application scenario**

There are several different techniques one can use to hide information in a cover file:

Injection (or insertion). Using this technique, you store the data you want to hide in sections of a file that are ignored by the processing application. By doing this you avoid modifying those file bits that are relevant to an end-user—leaving the cover file perfectly usable. For example, you can add additional *harmless* bytes in an executable or binary file. Because those bytes don't affect the process, the end-user may not even realize that the file contains additional hidden information. However, using an insertion technique changes file size according to the amount of data hidden and therefore, if the file looks unusually large, it may arouse suspicion.

Substitution. Using this approach, you replace the least significant bits of information that determine the meaningful content of the original file with new data in a way that causes the least amount of distortion. The main advantage of that technique is that the cover file size *does not change* after the execution of the algorithm. On the other hand, the approach has at least two drawbacks. First, the resulting stego file may be adversely affected by quality degradation—and that may arouse suspicion. Second, substitution limits the amount of data that you can hide to the number of insignificant bits in the file.

Generation. Unlike injection and substitution, this technique doesn't require an existing cover file, this technique generates a cover file for the sole purpose of hiding the message. The main flaw of the insertion and substitution techniques is that people can compare the stego file with any pre-existing copy of the cover file (which is supposed to be the *same* file) and discover differences between the two.

Among the substitution techniques, a very popular methodology is the LSB (Least Significant Bit) algorithm, which replaces the least significant bit in some bytes of the cover file to hide a sequence of bytes containing the hidden data. That's usually an effective technique in cases where the LSB substitution doesn't cause significant quality degradation, such as in 24-bit bitmaps.

For example, to hide the letter "b" (ASCII code 98, that is 01100010) inside eight bytes of a cover, you can set the LSB of each byte like this:

10010010
01010011
10011011
11010010
10001010
00000010
01110011
00101010

The application decoding the cover reads the eight Least Significant Bits of those bytes to re-create the hidden byte—that is *0110010*—the letter "b." As you may realize, using this technique let you hide a byte every eight bytes of the cover. Note that there's a fifty percent chance that the bit you're replacing is the same as its replacement, in other words, half the time, the bit doesn't change, which helps to minimize quality degradation.

The above technique is most commonly used to hide information in digital images which are represented as RGB pixels. For example, in an RGB based 24 bit representation, changing the least significant bits will create a very small change in colors, so the human eye will usually will not catch it, For example:

A simplified example with a 24-bit image

```
1 pixel:

        (00100111 11101001 11001000)

Insert 101:

        (00100111 11101000 11001001)

            red   green   blue
```

LSB insertion works well with gray-scale images as well. It is possible to hide data in the least and second least significant bits and the human eye would still not be able to discern it. Unfortunately LSB insertion is vulnerable to slight image manipulation such as cropping and compression. For example, converting a GIF or a BMP image, which reconstructs the original message exactly (lossless compression), to a JPEG format, which does not (lossy compression), and then converting back, can destroy the data in the LSBs.

There are other ways to hide information in digital images like:

- masking and filtering

- algorithms and transformations

Masking techniques hide information in such a way that the hidden message is more integral to the cover image than simply hiding data in the "noise" level. Masking adds redundancy to the hidden information. This makes the masking technique more suitable than LSB with lossy JPEG images. It may also help protect against some image processing such as cropping and rotating.

Another steganography technique is to hide data in mathematical functions that are in compression algorithms. The idea is to hide the data bits in the least significant coefficients.

A key advantage of JPEG images over other formats is its lossy compression methods. It enables high quality images to be stored in relatively small files. JPEG images use the discrete cosine transform (DCT) technique to achieve image compression. The DCT is a technique for expressing a waveform as a weighted sum of cosines. In a JPEG file, the image is made up of DCT coefficients. When a file is steganographically embedded into a JPEG image, the relation of these coefficients is altered. Instead of actual bits in the image being changed as in LSB steganography, it is the relation of the coefficients to one another that is altered.

Another method called the Wavelet transform is a transformation to basis functions that are localized in frequency. The wavelet compression methods are better at representing transients, such as an image of stars on a night sky. Wavelet compressions are good for transient signal characteristics but not for smooth, periodic signals.

Many transform domain methods are not dependent on the image format so that the hidden message is retained after conversion between lossless and lossly formats. The steps are to take the DCT or wavelet transform of the cover image and find the coefficients below a specific threshold. The reason being that the coefficients of the low frequencies are the most important, and will not be easily affected by image processing methods. The DCT or Wavelet method is popular since it hardly distorts the image, and can withstand many image processing attacks. An example using the DCT method is shown in Figure 3.19 which shows two images, one with the watermark, the other without it. As can be seen it is impossible to see visual differences between the two images. Furthermore, even after processing the image with the watermark by for example adding noise, or editing manually, it was still possible to identify the watermark in the DCT coefficients!

Validating the Watermark

In the DCT method the steps for validation are as follows:
1. Convert the image with the watermark to the frequency plain X
2. Convert the suspicious image to the frequency plain X*
3. Selection of the first N significant coefficients in both images.
4. Computing the similarity between the vectors of coefficients using the similarity measure:

$$\mathrm{sim}(X, X^*) = \frac{X^* \cdot X}{\sqrt{X^* \cdot X^*}}.$$

If the similarity is above a threshold its clear that the image contains the watermark!

Figure 3.19 - The DCT method

Applications of Watermarking

Currently, watermarking is used for

- Copyright protection - to prevent third parties from claiming the ownership of the digital media.
- Fingerprinting - to convey information about the recipient of the digital media (rather than the owner) in order to track distributed copies of the media.
- Copy protection - to control data copying devices and prevent them from copying the digital media if the media is copy-protected.
- Image authentication - to check the authenticity of the digital media.

Watermarks are embedded directly into a file data, usually by making minor variations to pixel brightness. Watermarks are not text that is included in the file description "Comments" field. The variations in the data bits are subtle and cannot be detected by the human eye. The patterns are repeated many times, allowing the information contained in the watermark to be recovered even if the image is cropped. Some watermarks can survive a limited amount of image manipulation, such as contrast adjustments and filtering. In general successful watermarking requires the following:

- Hiding - the watermark should be hidden in a way that will not cause significant change in the image.
- Survivability - the sign should withstand various image processing operations like compression, zooming, color changes, etc.
- Unambiguousness - the sign should be unique and verified

- Practicality - the sign should be implemented efficiently and in a simple enough way.

It is difficult to design signs that fulfill all the above criteria. Generally, there are four classes of attacks on watermarking schemes: robustness attacks, presentation attacks, interpretation attacks, and legal attacks.

Robustness Attacks

Robustness attacks attempt to diminish or remove the presence of watermarks in a suspect image without rendering the image useless. These attacks can be classified into two types: signal processing attacks, and analytic and algorithmic attacks. A signal processing attack include common processing operations such as compression, filtering, resizing, printing, and scanning. Analytic and algorithmic attacks involve removal or weakening of watermarks in images based on the specific methods of watermark insertion and detection. An example is the collusion attack where different watermarked versions of the same image are combined to generate a new image, reducing the strength of the watermark.

Presentation Attacks

In a presentation attack the watermarked content is manipulated so a detector cannot find it. For example, misaligning a watermarked image can sometimes fool an automated detector such as a Web-crawler, even though the underlying pixel values are not changed. Other examples of presentation attacks include rotation and enlargement. In a presentation attack the watermark does not need to be removed or diminished.

Interpretation Attacks

Interpretation attacks seek to falsify invalid or multiple interpretations of a watermark. For example, an attacker can attempt to make another watermark appear in the same watermarked image with strength equal to that of the owner's watermark, creating an ownership deadlock. The pixel values may or may not change.

A truly robust watermarking scheme has to minimize an attacker's ability to cast doubt on technical evidences presented in court.

References

Two important references describing Steganography and Watermarking are [Katzen00] and [Petitcolas99]. The DCT method is described in [Cox96]. The theoretical issues regarding the security of a watermarking method are studied in [Cachin04].

3.7 Summary

In this chapter we presented the principles of cryptography, and described the most important algorithms and protocols. Cryptography is a very important protection mechanism today, especially for network and Internet security, and although the chapter is not intended as a full coverage of this fascinating subject, it provides the necessary background for understanding the later chapters.

Exercises

1. Complete the decryption of the mono-alphabetic cipher in Section 3.2.

2. In a poly-alphabetic cipher it is possible to find quite easily the length of a key by using the Kasisky method. Assume that you know that a certain string of a larger length than the length of the key is in the original text (but you don't know where - this is called clear text attack). Describe a simple algorithm to find the key and to break the cipher. What do you think of the cipher's resistance from the clear text attack? Assume that the cipher is based upon the Vigenere chart.

3. Find and explain the "bug" in the method described when discussing the Vernam cipher.

4. Explain in detail the equation for Unicity distance - UD, and the relation to redundancy. Find the D for a language of 26 letters, N = 3 and the number of legal words with 3 letters is 1000.

5. Give a numeric example for breaking a Vernam cipher using the chosen text attack.

6. Assume that M is a number with 6 digits ranged [0-999999] encrypted with CASESAR cipher with a key range [0-9], for example if M = 123456 and A = 1 than C = 234567. Find H(M), H(K), H(C), H(C/M), H(C/K).

7. Compare a block cipher and a stream cipher and give their advantages and disadvantages.

8. Explain in detail Knuth's algorithm to find inverses.

9. Assume that the public key for the Merkle-Hellman is A = (17, 34, 2, 21, 41) and the message is C = 72. Assume that the secret key is given by n = 50, w = 17. Find message M.

10. Assume in the RSA scheme n = 55, e = 7, find q, p and d by factoring Encipher the message M = 10, decipher the message C = 35.

11. Assume that Alice chooses the prime numbers p = 113, q = 157 and e = 113.

    a) Find both Alice's public and private keys.

    b) Assume that Alice uses the coding equation

    $X = X_3*26^2+X_2*26^1+X_1$ (iencoding 3 letters at once). decipher the next messages:

        a. 13667

        b. 11220

        c. Encipher the next message using the above cipher:

        The message is: TOBEORNOTTOBE.

12. Find a description of a Rabin algorithm for checking if a number is a prime one and summarize it.

13. Find information in the Internet about the development of the new symmetric cipher AES and summarize it.

14. Find and summarize the material about Clipper and the politics surrounding it.

15. Assume that in protocol DH in figure 3.16, $\alpha = 5$, $q = 97$, $X_a = 36$, $X_B = 58$. Find the joint key.

16. Explain in details the three protocols we presented for exchanging a common key in Section 3.4.2 and discuss their advantages and disadvantages.

17. Find information about the "oblivious transfer" protocol and suggest two different applications for it.

18. The "millionaires" problem is the following: two rich people want to know who is the richer of them but without disclosing to each other their exact fortune. An elegant cryptographic method to solve this problem is called ***Secure computation***. Find information about secure computation and describe a practical application for it.

19. Give your opinion on the next hash function:

    I. The message is divided into blocks $b_1, b_2 \ldots b_k$

    II. Each block is divided into bits: $x_1, \ldots x_8$

    III. The hash function is defined as:

    .

    $$f(x_1 \ldots x_8) = x_2 x_1 x_4 x_3 \ldots x_8 x_7$$

    $$C_1 = f(b_1) \qquad C_2 = f(C_1 + b_2)$$

    $$H = C_K = f(C_{k-1} + b_k)$$

    .

    Show how it is possible to arrive at two different messages which collide into the same hash value.

20. Improved DES ciphers are the double and triple DES. Describe these ciphers and compare their security properties to that of DES.

21. The next algorithm calculates efficiently exponent in modulo. Explain the algorithm and give a non-trivial numeric example for its use.
    Fast exponentiation
    Algorithm fastexp (a,z,n)

    Begin "return $x = a^2 \bmod n$"

    $a_1 := a; z_1 := z;$

    $x := 1;$

    while z1 # 0 do "$x(a1^{z1} \bmod n) = a^2 \bmod n$"

```
        begin
            while z1 mod 2 = 0 do
                begin                                   "square a[1] while z[1] is even"
                    z1 = z1 div 2;
                    a1 : = (a1*a1) mod n
                end;
            z1 : = z1-1;
            x : = (x * a1) mod n                "multiply"
        end;
    fastexp : = x
end
```

22. Many companies have a set of databases, maybe from different vendors. Indicate a way to access securely a set of corporate databases from a smart phone. By securely we mean that authorized employees can perform specific accesses to information in the databases.

Programming exercises

23 Encipher the next message:

a. using a poly-alphabetic cipher (in length of 4-6 digits)
b. using a permutation cipher (assume a rectangle of 4*6)

For the enciphered text you received, explain how you can find the type of cipher in each one of the cases and the length of the key. Do you have enough text in order to "break" the cipher and to find the key? Below is the message:

"There was a pilot flying a small single engine charter planer, with a couple of very important executives on board. He was coming into a large city airport through thick fog with less than 10m visibility when his instruments went out. So he began circling around looking for a landmark. After an hour or so, he starts running pretty low on fuel and the passengers are getting very nervous. Finally, a small opening in the fog appears and he sees a tall building with one guy working alone on the fifth floor. The pilot banks the plane around, rolls down the window and shouts to the guy "Hey, where am I? To this, the solitary office worker replies, "You're in a plane". The pilot rolls up the window, executes a 275-degree turn and proceeds to execute a perfect blind landing on the runway of the airport 5 miles away. Just as the plane stopped, so does the engine as the fuel has run out. The passengers are amazed and one asks how he did it. "Simple" replies the pilot, "I asked the guy in that building a simple question. The answer he gave me was 100 percent correct but absolute useless, therefore that must be the building of the well known software company support office and from there the airport is just a mile away"

24 . Implement the RSA cipher using 128 bits arithmetic's (4 words of 32 bits each). Your program is supposed to perform the following:

a. Reading a seed from the user and creating two prime numbers of 64 bits. Use the "Rabin test" to find primary.

b. Receiving e from the user and finding d as a reaction.

c. Receiving an original message from the user and enciphering it. Receiving an enciphered message and deciphering it (it is possible to check the results with a hex dump program…)

d. For testing, receive both p and q from the user.

24. Implement simulation of the WooLAM protocol in section 3.4.2 for a secret key exchanging.

25. Implement one of the known hash functions (such as MD4 or MD5) and find at least three keys which have the same hash value

26. Describe in detail one of the steganography methods discussed in this chapter.

References

[Bhag2004] W.C. Bhagyavati, and A. Summers, "Wireless Security Techniques: An Overview", *InfoSecCD Conference'04*, Kennesaw, GA,USA, September 17-18, 2004

[Bih90] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems", *Proceedings of Crypto conference, 1990*, . 2-21.

[Corman02] T. H. Cormen, C. E. Leiserson, R. L. Rivest, Clifford Stein, *Introduction to Algorithms*, Second Edition, The MIT Press, 2002.

[Cachin04] C. Cachin. An information-theoretic model for steganography. *Information and Computation*, 192(1):41-56, July 2004.

[Cox96] Ingemar J. Cox, Joe Kilian, Frank Thomson Leighton, Talal Shamoon: A Secure, Robust Watermark for Multimedia. Information Hiding 1996, 185-206

[Denn81] D. Denning, "Timestamps in key distribution protocols," *Comm. of the ACM*, August, 1981

[DES99] The DES Challenge, http://www.distributed.net/des/

[Diff76] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information theory*, 22(6), 1976, pp. 644-654

[Ding02] Y. Z. Ding and M. O. Rabin**. "**Hyper-encryption and everlasting security". In 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS), volume 2285 of Lecture Notes in Computer Science, pp. 1--26. Springer-Verlag, 2002.

[DSS94] The Digital signature standard, FIPS PUB 186 , 1994

[Elgamal84] T. El Gamal: “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. *CRYPTO 1984*: 10-18

K. Hashizume, E.B.Fernandez, and S. Huang, "Digital Signature with Hashing and XML Signature patterns", *Procs. of the 14th European Conf. on Pattern Languages of Programs, EuroPLoP 2009*.

K. Hashizume and E.B.Fernandez, "Symmetric Encryption and XML Encryption Patterns", *Procs. of the 16th Conf. on Pattern Languages of Programs (PLoP 2009)* http://portal.acm.org/citation.cfm?doid=1943226.1943243

Xuejia Lai, James L. Massey: A Proposal for a New Block Encryption Standard. [Lai90] *Proceedings of EUROCRYPT 1990*: 389-404

[Kahn67] Kahn, The Code breakers**,** Macmillan, 1967

[Kat00] S. Katzenbeisser and F. A. P. Petitcolas (Editors)**,** *Information hiding techniques for steganography and digital watermarking*, Artech House Books, January 2000

[Mah03] M. Agrawal, N. Kayal, and N. Saxena, "PRIMES is in P." Technical report, Computer Science, Indian Institute of Technology, 2003

[Merk78] R. Merkle and M. Hellman, "Hiding information and signatures in trapdoor knapsacks," *IEEE Trans. on Information Theory*, 24(5), 1978, pp. 525-530

[Need78] R. Needham and M. Schroeder, "Using encryption for authentication in large network of computers,", *Comm. of the ACM*, December, 1978.

[Nist93] NIST, DATA ENCRYPTION STANDARD (DES), FIPS PUB 46-2, 1993.

[Nist01] NIST, Advanced Encryption Standard (AES), FIPS PUB-197, 2001

[Neum93] Neuman B., and S. Stubblebine, "A note on the use of timestamps as nonces", Operating system review, April, 1993.

[NSS00] http://www.nss.co.uk/Articles/PKI%20Market%20Survey.htm

[Rab76] Rabin, M. O. "Probabilistic Algorithm for Testing Primality." *J. Number Th.* 12, 128-138, 1980.

[Petitcolas99] -Petitcolas F. A., Anderson, R. and M. G. Kuhn, Information hiding - a survey,
Proceedings of the IEEE, special issue on protection of multimedia content, 87(7):1062-1078, July 1999.

[rfc2952] http://www.faqs.org/rfcs/rfc2952.html

[Riv78] R. L. Rivest, A. Shamir, and L. M. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM 21*(2): 120-126 (1978)

[Riv92] R. L. Rivest, The MD5 Message-digest algorithm. RFC 1321, Internet activities board, 1992.

[Riv94] R. L. Rivest: The RC5 Encryption Algorithm. Fast Software Encryption 1994: pp. 86-96

[Sham82] A. Shamir, "A polynomial time algorithm for breaking the basic Merkle-Hellman Cryptosystem," *Proceedings of the Crypto conference*, 1982, pp. 279-288

[Singh00] S. Singh, *The Science of Secrecy: The Secret History of Codes and Code-breaking*, HarperCollins Publishers, 2000

[Sch96] B. Schneier, *Applied Cryptography*, 2nd edition, John Willey&Sons, 1996 ISBN0-471-11709-9

[Shan49] C. E. Shannon, "*Communication Theory of Secrecy Systems*," The Bell Labs Technical Journal, pp. 656--715, May, vol. 28, No 4 1949.

[Stallings02] Cryptography and Network Security: Principles and Practice (3rd Edition), Prentice-hall, 2002

[Stinton02] *Cryptography: Theory and Practice*, Second Edition
Chapman&Hall, 2002

[Wohl00] P. Wohlmacher: Digital certificates: a survey of revocation methods. ACM Multimedia Workshop 2000: pp. 111-114

[WooLam92] T. Woo and S. Lam, "Authentication for distributed systems", *IEEE Computer*, January, 1992, pp 39-52.