CS 4476/5413 Lecture Notes

INTRODUCTION TO

NETWORK SECURITY

Ruizhong Wei

Department of Computer Science
Lakehead University

Winter, 2003

Contents

Chapter 1

Introduction

Since the inception of computer network, there have been a lot of security problems discovered, solved and developed. This is not only because of some people who have wished to demonstrate their intellectual prowess by attacking computer systems and network, but also because of people who have had some financial or political gains to perform attacks. On the other hand, there are so many different people using computer networks. There are always fault management, fault software, abuse of resources connecting to computer networks. These are the main reasons which cause security problems for a network. Today, security problem becomes one of the main problems for computer network and internet developing. There is no simple way to establish a secure computer network. In fact, we cannot find a network in the world, which does not have any security holes nowadays. It is understandable that any big complicated system, not just computer networks, has security problems. However, since the inventors of computer networks didn't consider the security of a network when they just wanted to use a network to communicate using computers from an university office to another office, and then the speed of the development of networks is beyond anyone's imagination, the security problem for computer networks is more serious.

There are many aspects of performing network security. In this book, we focus on cryptographic based network security. It should be noticed that cryptography is not the only thing required for network security. Other things such as organizations, managements, user policies, related law makings, etc. are also key things for the network security.

Recently, many people indicate that if cryptography is not used appropriately, then it will damage the security of the network instead of enhance the

security. So it is important to understand how to use cryptography correctly and what is the limitation of cryptography.

Now almost every computer is connected to some kind of network and almost every one using a computer knows there are security threats from a network. However, most people including many IT technicians do not really understand cryptography and network security protocols. There are many misunderstood of cryptographic based network securities. For examples, we can always hear wrong statements such as:

- Public key encryption is more secure than secrete key encryption.
- X.509 certificates are used to certificate computers.
- A secure hash function can be used to encrypt data.
- A firewall can prevent computer virus attack.

In this book, we will not distinguish the internet and a computer network, because the cryptographic based security consideration is similar for them. Internet is an open network so that no one knows the exact shape of the internet. A simple model of internet is demonstrated in Figure 1.1. In this model, local networks are connected to the internet through routers. This figure shows that sniffers might exist any where in the network. When a packet of a message goes through the network, any sniffer should be able to see it. For example, if you send out an email in plain text, then the sniffers on the way can read your email without any difficulty. There are many softwares which can catch all the packets on the line. For example, an open source software called `Ethereal` which is used to analysis network can be used to sniffer packets. On the other hand, a hacker can send fault messages so that it may be able to cheat other hosts in the network. So how can we trust the information from internet is a big question. A worse case is that if a router is hacked, then the hacker can change any packet come from and gone to the local network.

The main idea for using cryptography to network security is to encrypt messages in communications over the network. In this way, only the person possessing correct decryption key can understand the messages. However, we will see later that to realize this simple idea is very difficult in practice.

This book is designed as an introduction of cryptographic based network security which can serve as a textbook for a one term undergraduate computer science course.

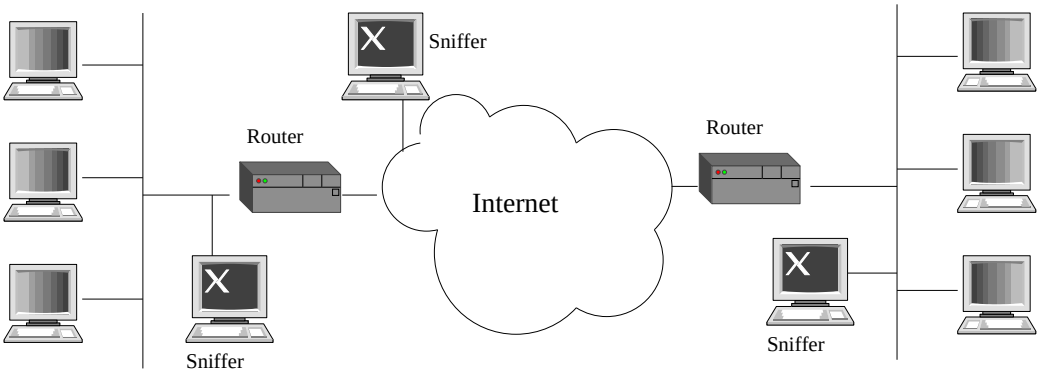


Figure 1.1: Simple model of internet

To consider the security of a network, we need to understand what are the common security attacks and what kind of security services a good network should provide to prevent against various attacks. In the rest of this chapter, we will consider these two aspects of network security.

1.1 Security attacks

Attacks on the security of network usually can be classified to four or more categories according to the functions of computer network as providing information. In the following we give a brief description of attacks by no means of an exhaustive list, but giving readers some idea of security attacks in networks. An asset of a computer system means a part of the system which can be some hardware (CPU, memory, disk space, peripherals), software (applications, operating systems, utilities), data (files, database, application input or output), etc.

- **Interruption:** An asset of the system is destroyed or becomes unavailable or unusable. Some examples are: destruction of a piece of hardware (hard disk, communication line etc.), computer worms (some independent program that does not modify other programs, but reproduces itself over and over again until it slow down or shuts down a computer system or a network), clogging (replaying some applications or using a lot of space and time of CPU to do useless computing)or flooding (a very large amount of bogus traffic is sent to a node, such as a server of router).

- **Interception:** An unauthorized party gains access to an asset. Examples include wiretapping to capture data in a network (sniffing), illicit copying of files or programs, Trojan horse virus (some programs hiding in a useful software, which collect information from the host and send the information back to the hacker).

- **Modification:** An unauthorized party not only gains access to but tampers with an asset. Examples include changing values in a data file, altering a program so that it performs differently, and modifying the content of messages being transmitted in a network, some computer virus, computer bomb (time trigger or logic trigger), salami (small alteration of numbers in a file, a small piece of an eventual large salami).

- **Fabrication:** An unauthorized party inserts counterfeit objects into the system. Examples include the insertion of spurious messages in a network or the addition of records to a file (setting a faked bank web page to collect private information, sending emails using faked addresses).

There are different kinds of attackers to perform their desired or undesired attacks to a network. Usually we may divide them into two categories as follows.

- **Passive attackers:** By eavesdropping on or monitoring of transmissions, a passive attacker will not modify the messages. The purpose of passive attackers are release of message contents or traffic analysis. An attacker may gain sensitive or confidential messages by sniffing. If all the messages are encrypted, then the attacker may difficult to understand the message. However, the attacker can do some traffic analysis to see the change of transformation amount, pattern, destinations, etc. It is hard to detect a passive attacker. The main consideration is how to prevent such attacks.

- **Active attackers:** An active attacker will modify of data stream or create a false stream. Examples include masquerade (one entity pretends to be a different entity), replay (capture a data and retransmission it), modification of message (change some portion of data), denial of service (prevents or inhibits the normal use or management of communication facilities). For active attackers, we want to detect them first. It is difficult to prevent such attackers completely.

1.2 Security services

A security service enhances the security of the data processing system and information transfers of an organization. The services are intend to counter security attacks and they use security mechanism to provide the service. Usually, we consider the following security services.

- **Confidentiality:** Ensures that the information is accessible only for reading by authorized parties. Confidentiality is the protection of transmitted data from passive attacks. Basic method for this service is encryption.
- **Authentication:** Ensures that the origin of a message is correctly identified, with an assurance that the identity is not false.
- **Integrity:** Ensures the precision, accuracy, and consistency of information. Transmitted information and computer systems only can be modified in acceptable ways by authorized entities. This service includes protection of information and detection of violation.
- **Nonrepudiation:** Requires that neither the sender nor the receiver of a message be able to deny the transmission.
- **Access control:** Requires that access to information resources be controlled by or for the target system.
- **Availability:** Requires that the system data and services be available to authorized parties when needed.

1.3 A model for network security

We will discuss a general model of network security shown in the Figure 1.2.

In this model, two principals are connected by an information channel. They will transfer information through the information channel. The information channel is open so other one can also access the channel. An opponent is connected to the information channel. Security aspects come into play to protect the information transmission from the opponent. Since the opponent is connected to the information channel, he can receives all the messages go

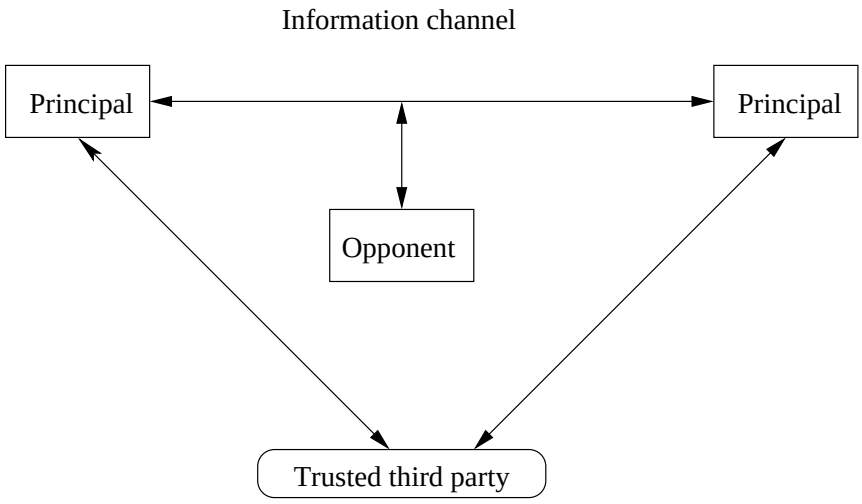


Figure 1.2: Model for network security

through the information channel and he also can send faked information to the principals.

Sometimes a trusted third party (e.g., arbiter, distributer of secret information) is needed. In this case, the opponent is supposed unable to get information communicated between the trusted third party and principals. So we suppose that there is a secret channel between the trusted third party and a principal. For example, a trust third party can be a bank and the principal be a client. Then the bank can give the client a credit card by regular mail or by hand. So suppose that there is a secure channel between the bank and the client. We will see later that it is difficult to find a secure channel in many cases related networks.

All the discussion of network security in this book will based on this model.

Network security is a subset of information security. The rapid development of internet makes the network security more and more important for the information security.

1.4 An overview

The basic idea of cryptographic based network security is that all the data going through the network is encrypted. In this way, although people can catch the data, but they will not know the meaning of the data, and where the data comes from and where to go. So the first problem for the cryptography is to find good *encryption systems*.

DES AES

Every encryption system needs some secret key for encrypting and decrypting. Since the number of users of the internet is huge, how to deliver these keys is a difficult problem. To solve this problem, researches invented *public key encryption systems*. In a public key encryption system, the encrypting key is public but the decrypting key is kept secret.

RSA Diffie-Hellman

If someone, say Bob, publishes a public key, then other people can use this key to encrypt messages when they want to send the messages to Bob. But there is a problem: how can you believe that the public key is really published by Bob? So the public key needs to be certificated.

X. 509

Another problem of network security is message authentication. We want to make sure that the message is sent really by the sender and the message is not mended by third party. For that purpose, *hash functions* and *signature schemes* are used.

MD5 SHA

Chapter 2

Conventional Cryptography

Conventional encryption, also refereed to as private-key (or single-key) encryption was used in cryptographic system for a long time. Some people also use the terminology of symmetric encryption, because in that system both encryption and decryption use the same key. In this chapter, we discuss some classical encryption systems. Although most systems mentioned in this chapter are no longer in use now, we can learn some basic ideas and problems for symmetric encryption by investigating these systems.

In this chapter, we first introduce a general model of a conventional cryptosystem. Then several cryptosystems are investigated. Some basic methods are introduced to attack these systems. These attacks (also called cryptanalysis) give us some ideas about the requirements of a good encryption function.

2.1 A General Model

A general model for the conventional cryptosystem is shown in Figure 2.1. In this model, there are a message sender called Alice and a message receiver called Bob. The message goes through a public channel. A third person, Oscar will try to get the message through the public channel. Since both Alice and Bob want to keep the message secret, they use some method to encrypt the message so that Oscar only can obtain the encrypted data. The encryption and decryption are dependent on some secret key which only Alice and Bob know. Therefore there should be a secret channel for Alice and Bob to transfer the secret key in this model. Note that in practice, a

secret channel may not exist in many cases. So in these cases, we cannot use a conventional cryptosystem directly. We will discuss that situation in Chaper 4.

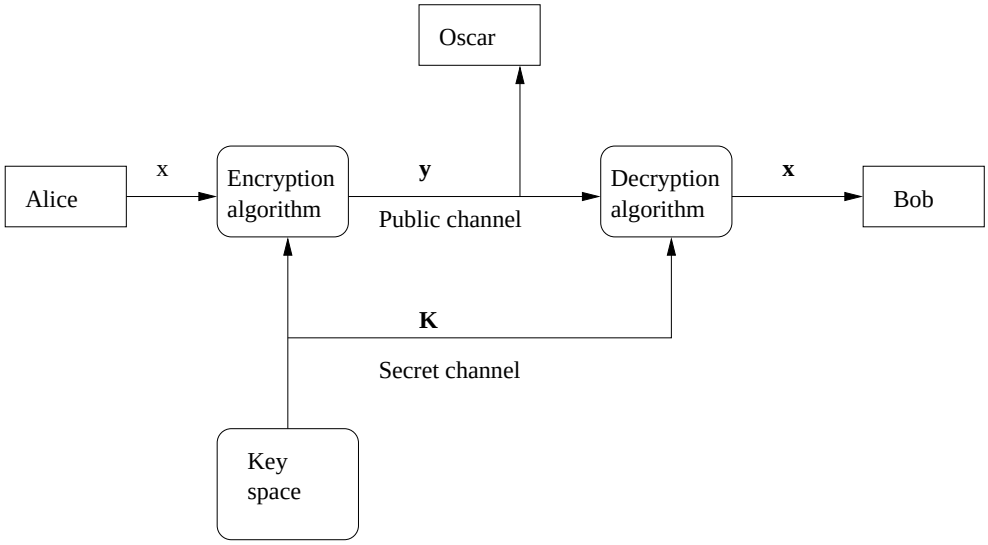


Figure 2.1: A model of conventional cryptosystem

Now we give a formal definition of a cryptosystem.

Definition 2.1.1 *A cryptosystem is a five-tuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, where the following conditions are satisfied:*

1. $\mathcal{P}$ *is a finite set of possible plaintexts.*
2. $\mathcal{C}$ *is a finite set of possible ciphertexts.*
3. $\mathcal{K}$*, the key space, is a finite set of possible keys.*
4. *For each key* $K \in \mathcal{K}$*, there is an encryption rule* $e_K \in \mathcal{E}$ *and a corresponding decryption rule* $d_K \in \mathcal{D}$*. Each* $e_K : \mathcal{P} \mapsto \mathcal{C}$ *and* $d_K : \mathcal{C} \mapsto \mathcal{P}$ *are functions such that* $d_K(e_K(x)) = x$ *for every plaintext* $x \in \mathcal{P}$*.*

In practice, a plaintext message is usually expressed as a string

$$\mathbf{x} = x_1 x_2 \cdots x_n$$

where $x_i \in \mathcal{P}, 1 \leq i \leq n$ and a ciphertext is also a string

$$\mathbf{y} = y_1 y_2 \cdots y_n,$$

where $y_i = e_K(x_i) \in \mathcal{C}, 1 \leq i \leq n$.

The procedure of communication may be roughly described as follows. When Alice and Bob want to communicate each other, they first select a suitable cryptosystem. Alice and Bob then select a random key $K \in \mathcal{K}$ secretly. When Alice wants to send a plaintext x_i to Bob, she computes and sends $y_i = e_K(x_i)$ to Bob. Bob then decrypts it by computing $x_i = d_K(y_i)$ after he receives x_i. Oscar can see y_i and he will try to find the key K or plaintext x_i. The process of attempting to discover the plaintext or the secret key is know as *cryptanalysis*.

In general, we cannot theoretically prove a cryptosystem to be secure. However, people can evaluate the system by attacking. So developing cryptanalysis technique is a very important part of cryptographic research.

To consider cryptanalysis, we need to set some conditions and divide the situations into several different levels. In this book, we will always assume that Oscar knows the encryption algorithm (which is called Kerckhoff's principle), but he does not know the key.

There are several types of attacks on encrypted messages, depending on the power of the attacker. We give a brief description of these types in the following. All types are under Kerckhoff's principle. So all the attackers know the encryption and decryption algorithms.

- Ciphertext-only: Oscar possesses a string of ciphertext $\mathbf{y}$. He wants to find the plaintext or the key.

- Known plaintext: Oscar possesses a string of plaintext and the corresponding ciphertext. He wants to find the key.

- Chosen plaintext: Oscar can choose a plaintext string and obtain the corresponding ciphertext string. That means Oscar can temporary use the encryption machine. He wants to find the key.

- Chosen ciphertext: Oscar can choose a string of ciphertext and obtain the corresponding plaintext string. In this case, Oscar can temporary use the decryption machine. He wants to find the key.

Clearly, first three levels of attacks are enumerated in increasing order of strength. The chosen ciphertext attacks are more useful in public key system which we will discuss later. In general, we will not think a cryptosystem is secure enough, if it only can tolerate ciphertext-only attacks.

Note that in the above model, there is a secure channel between Alice and Bob. In many cases, that condition is not available in computer systems. This limitation of conventional cryptosystem results the development of public-key cryptography which we will discuss later.

Next we will start to introduce some encryption methods. These methods are not secure now. However, we can learn some idea about how to encrypt and decrypt, and learn some requirements for a secure encryption system.

From the definition of a cryptosystem, we know that the encryption function should be one-to-one, because the encryption should be reversible (decryption). We need to understand why a encryption system needs a secret key. Since we want a encryption system secure, the encryption function and decryption function are usually very complicated. So it is difficult to send the algorithms through a secret channel. Moreover, we will see that if a encryption method is fixed for a long time, then it is not secure. So if the encryption system uses a secret key, then the algorithm can be used for a long time while the secret key should be changed frequently. A key is much simpler than the algorithm and relatively easy to be send through the secret channel. It is obvious that the key should have the property that the results of the encryption is total different if the key is slightly changed.

2.2 The Shift Cipher

Shift Cipher (also known as Caesar Cipher) is a very simple encryption method. Before introduce that method, we need some knowledge of modular arithmetic which is refereed to Section 2.9.

Now we present the Shift Cipher in Figure 2.2.

To use the Shift Cipher, we make use of the following correspondence.

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Let $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. For $0 \leq K \leq 25$, define

$$e_K(x) = x + K \bmod 26$$

and

$$d_K(y) = y - K \bmod 26$$

where $x, y \in \mathbb{Z}_{26}$.

Figure 2.2: The Shift Cipher

Example 2.2.1 Suppose Alice and Bob use the key $K = 10$ in the Shift Cipher. When Alice wants to send the plaintext

`iwanttomeetyou`,

Alice first converts the text to a sequence of integers:

8 22 0 13 19 19 14 12 4 4 19 24 14 20

Then she add 10 to each value, reducing each sum modulo 26:

18 6 10 23 3 3 24 22 14 14 3 8 24 4.

Therefore the ciphertext is:

`SGKXDDYWOODIYE`.

To decrypt the ciphertext, Bob first converts the ciphertext to a sequence of integers, then subtracts 10 from each value, and finally converts the sequence of integers to alphabetic characters.

Note that we used upper case letters for ciphertext and lower case letters for plaintext to improve readability. We will keep this format in rest of the book.

If a cryptosystem is "secure", then Oscar will be very difficult to find the plaintext. However, the Shift Cipher is easy to break. In fact, the key space of this system is very small (only 26 keys). Thus Oscar can try each of these keys, until he finds the meaningful plaintext. So the shift cipher is very weak. It is easy to be broken even under ciphertext-only attack.

The attack using exhaustive key search is also referred as brute-force attack.

Remark 2.2.1 *For a secure cryptosystem, the key space must be large enough so that the brute-force attack does not work.*

The value 26 in the Shift Cipher is not significant. For example, we can use $\mathbb{Z}_{27}$ for 26 alphabetic characters and space. Actually, we can use a very large key space for a shift cipher. For example, we can use a key space of size $26 \times 26 = 676$ as follows. Divide plaintext into "blocks" of size 2. Let different combination of two characters correspond to an number in $\mathbb{Z}_{676}$. So let *aa* corresponds to 0, *ab* corresponds to 1, *ac* corresponds to $2 \cdots$. However, we will see later that no matter how large the key space is, the shift cipher is not secure.

2.3 The Substitution Cipher

The Substitution Cipher can be seen as a generalization of the Shift Cipher. For simplicity, we still define the Substitution Cipher in $\mathbb{Z}_{26}$ and use the same correspondence between letters and integers as we did for the Shift Cipher.

In substitution cipher, we will use permutation of $\mathbb{Z}_{26}$. A permutation of a finite set X is a bijective function $\pi : X \to X$. Therefore each permutation has a inverse function called inverse permutation π^{-1}. They satisfy the following rule:

$$\pi(x) = x' \text{ if and only if } \pi^{-1}(x') = x.$$

Clearly, π^{-1} is also a permutation of X.

Usually, we can write a permutation as two rows of elements of X. For example, a permutation on $\mathbb{Z}_9$ can be written as

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 5 & 1 & 4 & 3 & 6 & 0 & 8 & 7 \end{pmatrix}$$

So $\pi(0) = 2, \pi(1) = 5$, etc. It is easy to see that

$$\pi^{-1} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 6 & 2 & 0 & 4 & 3 & 1 & 5 & 8 & 7 \end{pmatrix}$$

The Substitution Cipher is defined as in Figure 2.3.

Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$, $\mathcal{K}$ consists of all possible permutations of the 26 symbols $0, 1, \cdots, 25$. For each permutation $\pi \in \mathcal{K}$, define

$$e_\pi(x) = \pi(x)$$

and

$$d_\pi(y) = \pi^{-1}(y).$$

where π^{-1} is the inverse permutation to π.

Figure 2.3: The Substitution Cipher

In practice, it is not necessary to use $\mathbb{Z}_{26}$ as plaintext and ciphertext. We can directly use the permutation on 26 alphabetic characters.

Example 2.3.1 Alice and Bob choose a random permutation as follows.

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

The Alice's plaintext is the following.

> our friend from paris examined his empty glass with surprise as if evaporation had taken place while he wasnt looking i poured some more wine and he settled back in his chair face titles up towards the sun

Using the permutation, she obtains the following ciphertext.

```
YIFQFMZRWQFYVECFMDZPCVMRZWNMDZVEJBTXCDDUMJ
NDIFEFMDZCDMQZKCEYFCJMYRNCWJCSZREXCHZUNMXZ
NZUCDRJXYYSMRTMEYIFZWDYVZVYFZUMRZCRWNZDZJJ
XZWGCHSMRNMDHNCMFQCHZJMXJZWIEJYUCFWDJNZDIR
```

The permutation π^{-1} can be easily obtained by reversing the first line and the second line of π, and then sorting in alphabetical order:

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

Since Bob knows π, he can decrypt the ciphertext and get the plaintext.

There are total 26! permutations on the 26 alphabetic characters. So the key space of the Substitute Cipher is greater than 4.0×10^{26}. Thus, an exhaustive key search is infeasible.

To attack the Substitute Cipher, Oscar may use the statistical properties of the English language. From compiling statistics from numerous novels, magazines and newspapers, Beker and Piper obtained the probabilities of the frequency of the 26 letters as in Figure 2.4.

letter	probability	letter	probability	letter	probability
A	.082	J	.002	S	.063
B	.015	K	.008	T	.091
C	.028	L	.040	U	.028
D	.043	M	.024	V	.010
E	.127	N	.067	W	.023
F	.022	O	.075	X	.001
G	.020	P	.019	Y	.020
H	.061	Q	.001	Z	.001
I	.070	R	.060		

Figure 2.4: Probability of 26 letters

On the basis of the above probabilities, we can partition the 26 letters into 5 groups.

1. E, having probability about 0.120
2. T,A,O,I,N,S,H,R, each having probabilities between 0.09 to 0.06
3. D,L, each having probabilities around 0.04
4. C,U,M,W,F,G,Y,P,B, each having probabilities between 0.028 and 0.015
5. V,K,J,X,Q,Z, each having probabilities less than 0.01.

It is also useful to consider the frequency of two or three consecutive letters (called digrams and trigrams, respectively). The 30 most common digrams are (in decreasing order) *TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, NT, HA, ND, OU, EA, NG, AS, OR, TI, IS, ET, IT, AR, TE, SE, HI* and *OF*. The 12 most common trigrams are (in decreasing order) *THE, ING, AND, HER, ERE, ENT, THA, NTH, WAS, ETH, FOR* and *DTH*.

To find the plaintext and the key in Example 2.3.1, we first find the frequency of the occurrence of the 26 letters in cihpertext as follows.

letter	frequency	letter	frequency	letter	frequency
A	0	J	11	S	3
B	1	K	1	T	2
C	15	L	0	U	5
D	13	M	16	V	5
E	7	N	9	W	8
F	11	O	0	X	6
G	1	P	1	Y	10
H	4	Q	4	Z	20
I	5	R	10		

Since Z occurs significantly more often than other characters, we guess $d_K(Z) = e$.

The remaining characters that occur at least ten times are *C, D, F, J, M, R, Y*. We will think that they are encryptions of *t, a, o, i, n, s, h, r*. But we cannot decide what the correspondence might be, since their frequencies are close. So we look at digrams, especially the digrams $*\,Z$ and $Z\,*$ (remember that we already assumed $d_K(Z) = e$). In the ciphertext, *DZ* and *ZW* appear four times each, *NZ* and *ZU* appear three times each, *RZ, HZ, XZ, FZ, ZR, ZV, ZC, ZD* and *ZJ* appear two times each. Since *ZW* appears four times,

W might be encryption of *r,d,s* or *n*. On the other hand, *W* is not a frequent letter (only appears 8 times). So we decide that $d_K(W) = d$.

From *DZ*, we can guess that *D* is encrypted from *h, r, t* or *s*. Since *ZD* appears two times, *D* may be from *r, t* or *s*, but it is not clear to us which is the correct one.

We now look at the digram * *W*. *ZW* appears four times and *RW* appears two times. So we guess that $d_K(R) = n$.

Since *NZ* appears 3 times but *ZN* does not appear, we assume that $d_K(N) = h$.

By all the above assumptions, we can find a string *ne*ndhe* in the plaintext. The symbol * is from *C*. Since *C* appears 15 times in ciphertext, we think *C* is from *a* by trying *t,a,o* and *i*. So we have the following:

```
YIFQFMZRWQFYVECFMDZPCVMRZWNMDZVEJBTXCDDUMJ
******end*****a***e*a**nedh**e******a*****

NDIFEFMDZCDMQZKCEYFCJMYRNCWJCSZREXCHZUNMXZ
h*******ea***e*a***a***nhad*a*en**a*e*h**e

NZUCDRJXYYSMRTMEYIFZWDYVZVYFZUMRZCRWNZDZJJ
he*a*n******n******ed***e***e**neandhe*e**

XZWGCHSMRNMDHNCMFQCHZJMXJZWIEJYUCFWDJNZDIR
*ed*a***nh***ha***a*e****ed*****a*d**he**n
```

We now consider *M*, the second most common ciphertext character. We will think $d_K(M) \in \{t, o, i, s\}$. From the segment of ciphertext *MRNM* and the corresponding plaintext **nh**, we learnt that $d_K(M)$ does not like *t* or *s*. The digrams *CM* and *NM* in ciphertext suggest that $d_K(M) = i$.

Next we will try to determine which letter is encrypted to *o*. We guess that the corresponding ciphertext letter is one of *D, F, J, Y*. However, we know that *D* is encrypted from *r, s* or *t*. If $d_K(F) = o$, then we have *aoi* (from *CFM*). If $d_K(J) = o$, then we have *aoi* (from *CJM*). So we assume $d_K(Y) = o$. Then we consider *D, F, J* which are encrypted form *t,s,r*. The segment *NMD* suggests $d_K(D) = s$ (his). We guess $d_K(J) = t$ from *JY* (*to*) and *JN* (*th*). Therefore we assume that $d_K(F) = r$. The segment *HNCMF* could be encrypted from *chair*, which give $d_K(H) = c$.

It is easy to determine the plaintext and the key now.

In both the Shift Cipher and the Substitution Cipher, once a key is chosen, each alphabetic character is mapped to a unique alphabetic character. A cryptosystem satisfies that condition is called *monoalphabetic*.

Remark 2.3.1 *All the monoalphabetic cryptosystems can be attacked by guess-check method based on the probability of the occurrence of the alphabetic characters, digrams, trigrams, etc.*

Probabilistic methods are important tools for cryptanalysis. A good ciphertext should look like a random string.

2.4 The Permutation Cipher

Now we consider some cryptosystems which are not monoalphabetic. First we consider the Permutation Cipher (or the Transposition Cipher), which has been used for hundreds of years.

The Permutation Cipher can be described as in Figure 2.5.

Let m be some fixed positive integers. Let $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$ and let $\mathcal{K}$ consists of all permutations of $\{1, 2, \cdots, m\}$. For a key $\pi \in \mathcal{K}$, define

$$e_\pi(x_1, \cdots, x_m) = (x_{\pi(1)} \cdots, x_{\pi(m)})$$

and

$$d_\pi(y_1, \cdots, y_m) = (y_{\pi(1)^{-1}} \cdots, x_{\pi(m)^{-1}}),$$

where π^{-1} is the inverse permutation to π.

Figure 2.5: The Permutation Cipher

Lets use an example to explain how to use the Permutation Cipher.

Example 2.4.1 Suppose Alice and Bob decide that $m = 6$ and use the permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 1 & 6 & 2 & 5 \end{pmatrix}.$$

Alice wants to send the plaintext:

`he walked up and down the passage two or three times.`

Alice first divides the plaintext into groups of size 6 (we call these groups *blocks*):

`hewalk edupan ddownt hepass agetwo orthre etimes`

then performs the permutation on each of the groups and obtains the ciphertext:

`WLEHKAUADENPONDDTWPSEHSAEWGAOTTRROEHIETESM.`

When Bob received that ciphertext, he divides the text into blocks of size 6 and for each block he makes the permutation

$$\pi^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 5 & 2 & 1 & 6 & 4 \end{pmatrix}.$$

Then he obtains the plaintext.

The Permutation Cipher is not monoalphabetic. In the above example we can see that the first e is encrypted as L, the second e is encrypted as U and the third e is encrypted as S. This encryption does not change the frequency of alphabetic characters but the positions of the letters. Thus the analysis of the probability of the occurrence of letters will not give Oscar any help.

The Permutation Cipher is more difficult to break with a ciphertext-only attack. However, it succumbs easily to a known plaintext attack. In fact, if Oscar knows both plaintext and ciphertext, then it is not difficult for him to determine the length m and then find the key π.

2.5 The Vigenére Cipher

The Vigenére Cipher is also an example of cryptosystem which is not monoalphabetic. This cipher is named after Blaise de Vigenére, who lived in sixteenth century.

The Vigenére Cipher is defined in Figure 2.6.

Let m be some fixed positive integer. Define $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_{26})^m$. For a key $K = (k_1, k_2, \cdots, k_m)$, we define

$$e_K(x_1, \cdots, x_m) = (x_1 + k_1, \cdots, x_m + k_m)$$

and

$$d_K(y_1, \cdots, y_m) = (y_1 - k_1, \cdots, y_m - k_m),$$

where all operations are performed in $\mathbb{Z}_{26}$.

Figure 2.6: The Vigenére Cipher

To use the Vegenére Cipher, Alice and Bob first decide the value of m, the length of secret key and then choose a string of length m as the key. To encrypt a plaintext, Alice divides the text into blocks of size m, and encrypts the text block by block using the secret key.

Example 2.5.1 Let $m = 5$ and the secret key is *ONWAR*. Suppose the plaintext is as follows:

> the art of war teaches us to rely not on the likelihood of the enemys not coming but on our own readiness to receive him not on the chance of his not attacking but rather on the fact that we have made our position unassailable the combination of space time and strength that must be considered as the basic elements of this theory of defense makes this a fairly complicated matter consequently it is not easy to find a fixed point of departure

We first divide the plaintext into groups of size five and encrypt each group using the key *ONWAR*. The following ciphertext is obtained:

```
HUAAIHBBWRFGAATVROUJHBNECMAKTFBGDECWXALZ
VBKDFTGDEVBRIYJBBPCFAVJGSIGKNFIEKWEFRWDZ
BROSKCEACVWIAHZAAKTFBGDETVNJCVCSDIJBBPAK
HNYKZBTXUKFNPHVFBJTYSSWCKHUWTNSUWVVANZEF
```

```
IELOJWGEOEIAWSJOVHASZRPHVQBIBZBNPIFBBBSG
OPATZARWNUGGNEEUGDTYOGIUJHOACFBFEDVFRZAJ
HUABRGVYECSZANKGBBTYWFPHVCEUOWRRBEEGRIAB
SFPHZGNBAZFYUCFACHITOGADDOGPEIQBJSVEHANK
ZLETZGAKTVOFUTFTVJDRTVTEUDBENKCSZEGOEPUI
S
```

To attack Vigenére Cipher, Oscar needs to determine the length of key m (the size of blocks) and the secret key. We introduce some methods developed by Wolfe Friedman in 1920. He defined the index of coincidence as follows.

Definition 2.5.2 *Suppose* $\mathbf{x} = x_1x_2\cdots x_n$ *is a string of* n *alphabetic characters. Suppose we denote the frequencies of* $A, B, \cdots, Z$ *in* $\mathbf{x}$ *by* $f_0, f_1, \cdots, f_{25}$ *respectively. Define index of coincidence of* $\mathbf{x}$ *as*

$$I_c(\mathbf{x}) = \frac{\sum_{i=0}^{25}\binom{f_i}{2}}{\binom{n}{2}} = \frac{\sum_{i=0}^{25} f_i(f_i-1)}{n(n-1)}.$$

In fact, $I_c(\mathbf{x})$ denote the probability that two random elements of $\mathbf{x}$ are identical. The index of coincidence has the properties that if $\mathbf{x}$ is a ciphertext obtained by any monoalphabetic encryption, then

$$I_c(\mathbf{x}) \approx 0.065,$$

while if $\mathbf{x}$ is a random string, then

$$I_c(\mathbf{x}) = 0.038.$$

Using the properties of I_c, we can find the length of the key in Vigenére Cipher. Suppose that the key length is m and the ciphertext is $\mathbf{y} = y_1, y_2, \cdots, y_n$. If we write the ciphertext in columns, each column is of length m, then each row of the ciphertext is encrypted by one key letter. Thus each row is a ciphertext of a nomoalphabetic encryption and the I_c value of each row should be around 0.065.

For the Example 2.5.1, we compute the index of coincidence and obtain the following data. When $m = 2$, the values of I_c are $0.046369, 0.043824$. When $m = 3$, the values of I_c are $0.042297, 0.041457, 0.052381$. When $m = 4$, the values of I_c are $0.044944, 0.039950, 0.047690, 0.046692$. When $m = 5$, the values of Ic are $0.062207, 0.079030, 0.067684, 0.072770, 0.075117$. When

$m = 6$, the values of I_c are $0.038418, 0.035593, 0.053107, 0.046328, 0.043503$ and 0.044068.

Therefore we decide that the length of the key is five.

The second step is to determine the key. To do that we need to consider the mutual index of coincidence of two strings.

Definition 2.5.3 *Suppose* $\mathbf{x} = x_1x_2\cdots x_n$ *and* $\mathbf{y} = y_1y_2\cdots y_{n'}$ *are strings of* n *and* n' *alphabetic characters, respectively. Let* $f_0, f_1, \cdots, f_{25}$ *and* $f'_1, f'_2, \cdots, f'_{25}$ *be the frequencies of* $A, B, \cdots, Z$ *in* $\mathbf{x}$ *and* $\mathbf{y}$*, respectively. The mutual index of coincidence of* $\mathbf{x}$ *and* $\mathbf{y}$ *is defined as*

$$MI_c(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=0}^{25} f_i f'_i}{nn'}$$

The value of $MI_c(\mathbf{x}, \mathbf{y})$ is the probability that a random element of $\mathbf{x}$ is identical to a random element of $\mathbf{y}$. Suppose $\mathbf{x}$ and $\mathbf{y}$ are strings from shift cipher encryption. The value of MI_c has the property that if the related shift of $\mathbf{x}$ and $\mathbf{y}$ is zero (used the same shift), then the value of MI_c is about 0.065. Otherwise, the value estimates vary between 0.031 and 0.045.

We have hypothesized that the key length $m = 5$ in Example 2.5.1. Let the key be $(K_0, K_1, K_2, K_3, K_4)$. Now we try to use mutual index of coincidence to find the key. To do that we first write the ciphertext in columns of size 5:

```
HHFVHMBWVTBBAIIFBCWABVCBHBFFSHSA ...
UBGRBAGXBGRBVGERREIAGNSBNTNBSUUN ...
ABAONKDAKDIPJKKWOAAKDJDPYXPJWWWZ ...
AWAUETELDEYCGNWDSCHTECIAKUHTCTVE ...
IRTJCFCZFVJFSFEZKVZFTVJKZKVYKNVF ...
```

In this way, each row is an encryption of a shift cipher. Let $\mathbf{y}_i$ denote the ith row, $0 \le i \le 4$. Then we compute the values of

$$MI_c(\mathbf{y}_i, \mathbf{y}_j^g) = \frac{\sum_{k=0}^{25} f_k f'_{k-g}}{nn'},$$

for $0 \le i < j \le 4$ and $0 \le g \le 25$. The results are in Figure 2.7. From the formula we know that $\mathbf{y}_j^g$ is the string shifted g times from $\mathbf{y}_j$. Therefore if we find some g such that $MI_c(\mathbf{y}_i, \mathbf{y}_j^g) \approx 0.065$, then $K_i = K_j + g$.

i	j	values of $MI_c(\mathbf{y}_i, \mathbf{y}_j^g)$								
0	1	0.0563	**0.0675**	0.0384	0.0264	0.0336	0.0392	0.0436	0.0355	0.0401
		0.0311	0.0417	0.0282	0.0341	0.0503	0.0469	0.0380	0.0365	0.0301
		0.0258	0.0297	0.0403	0.0511	0.0338	0.0363	0.0231	0.0424	
0	2	0.0374	0.0442	0.0345	0.0349	0.0436	0.0488	0.0461	0.0476	0.0326
		0.0260	0.0276	0.0388	0.0424	0.0345	0.0347	0.0216	0.0336	0.0436
		0.0633	0.0413	0.0293	0.0297	0.0380	0.0421	0.0392	0.0446	
0	3	0.0444	0.0498	0.0382	0.0459	0.0372	0.0359	0.0351	0.0426	0.0446
		0.0282	0.0305	0.0266	0.0434	0.0430	**0.0604**	0.0355	0.0228	0.0222
		0.0380	0.0365	0.0382	0.0372	0.0316	0.0422	0.0438	0.0463	
0	4	0.0357	0.0446	0.0486	0.0368	0.0314	0.0332	0.0455	0.0363	0.0401
		0.0480	0.0378	0.0314	0.0405	0.0380	0.0268	0.0312	0.0307	0.0421
		0.0324	0.0388	0.0260	0.0388	0.0538	**0.0615**	0.0401	0.0297	
1	2	0.0324	0.0422	0.0428	0.0401	0.0380	0.0519	0.0486	0.0355	0.0336
		0.0264	0.0386	0.0278	0.0451	0.0380	0.0274	0.0228	0.0326	**0.0783**
		0.0417	0.0349	0.0326	0.0392	0.0357	0.0419	0.0471	0.0249	
1	3	0.0401	0.0444	0.0507	0.0338	0.0405	0.0276	0.0370	0.0336	0.0382
		0.0340	0.0318	0.0343	0.0324	**0.0718**	0.0451	0.0245	0.0249	0.0434
		0.0312	0.0411	0.0388	0.0289	0.0228	0.0478	0.0529	0.0484	
1	4	0.0407	0.0415	0.0446	0.0316	0.0264	0.0299	0.0392	0.0476	0.0473
		0.0380	0.0318	0.0473	0.0421	0.0326	0.0305	0.0324	0.0289	0.0307
		0.0530	0.0318	0.0228	0.0384	**0.0822**	0.0438	0.0280	0.0370	
2	3	0.0457	0.0395	0.0347	0.0355	0.0330	0.0324	0.0463	0.0577	0.0486
		0.0322	0.0309	0.0434	0.0312	0.0355	0.0262	0.0413	0.0388	0.0314
		0.0349	0.0336	0.0353	0.0349	**0.0723**	0.0465	0.0274	0.0307	
2	4	0.0318	0.0519	0.0367	0.0282	0.0411	**0.0720**	0.0430	0.0237	0.0320
		0.0392	0.0434	0.0314	0.0280	0.0299	0.0303	0.0353	0.0525	0.0509
		0.0324	0.0274	0.0494	0.0478	0.0322	0.0291	0.0403	0.0401	
3	4	0.0295	0.0382	0.0372	0.0367	0.0303	0.0513	0.0235	0.0239	0.0444
		0.0693	0.0372	0.0326	0.0307	0.0320	0.0401	0.0336	0.0291	0.0299
		0.0324	0.0355	0.0552	0.0496	0.0287	0.0403	0.0573	0.0515	

Figure 2.7: Observed Mutual Indices of Coincidence

From the data obtained we have the following equations:

$$\begin{aligned} K_0 &= K_1 + 1 \\ K_0 &= K_2 + 18 \\ K_0 &= K_3 + 14 \\ K_0 &= K_4 + 23 \\ K_1 &= K_2 + 17 \\ K_1 &= K_3 + 13 \end{aligned}$$

From these linear equations of five unknowns K_0, K_1, K_2, K_3, K_4, we can assume that the key is

$$(K_0, K_0 + 25, K_0 + 8, K_0 + 12, K_0 + 3)$$

Now we can try to decrypt the ciphertext by letting $K_0 = 0, 1, \cdots, 25$. When $K_0 = 14$, we get the plaintext. So the key is $ONWAR$.

It is easy to know that the Vigenére Cipher is not a monoalphabetic encryption. In fact, in this system, an alphabetic character can be mapped to one of m possible alphabetic characters. Such a cryptosystem is called *polyalphabetic* cryptosystem. In general, polyalphabetic cryptosystem is more secure than monoalphabetic cryptosystem.

Vergenére Cipher is based on 26 English letters. We can define a similar cipher in $\mathbb{Z}_2$ instead of $\mathbb{Z}_{26}$. In this case, the scheme is as in Figure 2.8.

Let m be some fixed positive integer. Define $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^m$. For a key $K = (k_1, k_2, \cdots, k_m)$, we define

$$e_K(x_1, \cdots, x_m) = (x_1 + k_1, \cdots, x_m + k_m)$$

and

$$d_K(y_1, \cdots, y_m) = (y_1 - k_1, \cdots, y_m - k_m),$$

where all operations are performed in $\mathbb{Z}_2$.

Figure 2.8: Binary Vergenére Cipher

In this scheme, we can think about the plaintext, ciphertext and key as binary strings of length m. In this way we can write the encryption and decryption functions as follows:

$$e_K(\mathbf{x}) = \mathbf{x} \oplus K, \quad d_K(\mathbf{y}) = \mathbf{y} \oplus K.$$

The operation $\oplus$ is called exclusive-or, or XOR, which can be easily and efficiently implemented by a computer. We can use the same program to perform both encryption and decryption.

2.6 The Hill Cipher

The Hill Cipher was invented in 1929 by Lester S. Hill. Similar to Vergenére Cipher, in this cipher $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$. The key used in this system is some kind of $m \times m$ matrix whose elements are from $\mathbb{Z}_{26}$.

Definition 2.6.1 *Suppose A is an $m \times m$ matrix over $\mathbb{Z}_{26}$,*

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,m} \end{pmatrix}.$$

If there exists an $m \times m$ matrix B over $\mathbb{Z}_{26}$,

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,m} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,m} \end{pmatrix},$$

such that $AB = I_m$, where I_m is the $m \times m$ identity matrix

$$I_m = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix},$$

then we say that A is an invertible matrix over $\mathbb{Z}_{26}$ and B is the inverse of A denoted by $B = A^{-1}$.

We will not discuss how to determine if a matrix is invertible and how to find the inverse of an invertible matrix here. These methods can be found in any linear algebra text book. The only thing need to be careful is that our computations are all in $\mathbb{Z}_{26}$.

The Hill Cipher can be defined as in Figure 2.9

Let $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$. Let $\mathcal{K}$ consists all $m \times m$ convertible matrices over $\mathbb{Z}_{26}$. For a $K \in \mathcal{K}$, define

$$e_K(x) = xK$$

and

$$d_K(y) = yK^{-1},$$

where $x, y \in (\mathbb{Z}_{26})^m$ and all the operations are performed in $\mathbb{Z}_{26}$.

Figure 2.9: The Hill Cipher

The correctness of the Hill Cipher is easy to verify. Because for any $x \in (\mathbb{Z}_{26})^m$, we have $xI_m = x$. Therefore $yK^{-1} = xKK^{-1} = xI_m = x$.

Let us see a small example.

Example 2.6.2 Suppose Alice and Bob choose $m = 2$ and use a key

$$K = \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix}.$$

When Alice wants to send a message

```
letusfly
```

to Bob, she first changes the plaintext into elements in $(\mathbb{Z}_{26})^2$ as follows (or we can say that the plaintext is divided into blocks of size 2):

$$(11, 4), (19, 20), (18, 5), (11, 24).$$

Then she computes the ciphertext as follows:

$$\begin{aligned}(11,4)K &= (3,12)\\ (19,20)K &= (9,6)\\ (18,5)K &= (5,23)\\ (11,24)K &= (11,22)\end{aligned}$$

So the ciphertext is

`DMJGFXLW`

Bob can find from K that

$$K^{-1} = \begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix}.$$

So he can decrypt the cipher and obtain the original message.

The Hill Cipher can be difficult to break with a ciphertext-only attack. However, it succumbs easily to a known plaintext attack. It involves solving linear equations. In Example 2.6.2, if Oscar knows both the plaintext and ciphertext, then he knows that

$$\begin{pmatrix} 11 & 4 \\ 18 & 5 \end{pmatrix} K = \begin{pmatrix} 3 & 12 \\ 5 & 23 \end{pmatrix}.$$

He can then compute that

$$\begin{pmatrix} 11 & 4 \\ 18 & 5 \end{pmatrix}^{-1} = \begin{pmatrix} 15 & 14 \\ 24 & 7 \end{pmatrix}.$$

Therefore he obtains

$$K = \begin{pmatrix} 15 & 14 \\ 24 & 7 \end{pmatrix} \begin{pmatrix} 3 & 12 \\ 5 & 23 \end{pmatrix}.$$

The Hill Cipher is not a monoalphabetic encryption system. In the above example, there are two "`l`" in plaintext. They are encrypted to different cipher text "`D`" and "`L`".

Remark 2.6.1 *From the attack of the Hill Cipher we learnt that if there are some "linear relationship" between plaintext and ciphertext, then the cryptosystem is not secure.*

2.7 Stream Cipher

The cryptosystems we studied so far are called *block cipher*. In a block cipher, each element of a plaintext is using a same key K, thus the ciphertext string of $\mathbf{x} = x_1x_2\cdots$ is

$$e_K(x_1)e_K(x_2)\cdots.$$

Stream Ciphers use a series of different keys instead of one key. In a Stream Cipher, we will use a key stream: $\mathbf{z} = z_1z_2\cdots$ to encrypt a plaintext. So the ciphertext will be

$$\mathbf{y} = y_1y_2\cdots = e_{z_1}(x_1)e_{z_2}(x_2)\cdots.$$

To set up a Stream Cipher, the main problem is how to generate the key stream. There are several different types of Stream Ciphers. When the key stream is related to the plaintext, the cipher is called *non-synchronous* cipher. If the key stream is independent from the plaintext, then it is called *synchronous* cipher. A stream cipher is called *periodic* if $z_{i+d} = z_i$ for some d. A Vigenére Cipher can be thought as a periodic stream cipher.

In general, stream ciphers are faster than block cipher in hardware, and have less complex hardware circuitry. They are also suitable for the cases when buffering is limited or when characters must be individually processed as they are received. A stream cipher may also used when transmission errors are highly probable, since they have less or no propagation. We will discuss this a little more in the next chapter.

Now let us consider some examples of stream cipher. The stream cipher defined in Figure 2.10 is a non-synchronous cipher called *Autokey Cipher*.

For example, suppose the plaintext is

$$networksecurity.$$

The corresponding numbers are

$$13\ 4\ 19\ 22\ 14\ 17\ 10\ 18\ 4\ 2\ 20\ 17\ 8\ 19\ 24.$$

Suppose we choose $K = 5$. Then $z_1 = 5, z_2 = x_1 = 13, z_3 = x_2 = 4\cdots$. So the result numbers are

$$18\ 17\ 23\ 15\ 10\ 5\ 1\ 2\ 22\ 6\ 22\ 11\ 25\ 1\ 17.$$

Let $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. For a $K \in \mathcal{K}$, let $z_1 = K$ and $z_i = x_{i-1}$, for $i \geq 2$. Define

$$e_z(x) = x + z \bmod 26$$

and

$$d_z(y) = y - z \bmod 26$$

Figure 2.10: Autokey Cipher

The cipher text is

$$SRXPKFBCWGWLZBR.$$

To decrypt the ciphertext, Bob first uses $K = 5$ to find the first letter of the plaintext n. Then he uses n as a key to find the second letter e, etc.

Of course, the autokey cipher is insecure since there are only 26 different keys. The autokey cipher is non-synchronous stream cipher. Next we consider some synchronous stream ciphers.

First we note that a Vigenére Cipher can been seen as a stream cipher. In this case, the key stream is period, i.e., $z_{i+m} = z_i$. We already have seen that the Vigenére Cipher can be attacked if the period is not very large. In general, we wish the period of a key stream is very large. The following method can be thought as a generalization of the binary Vigenére Cipher. One advantage of this method is obtaining a long period key stream from relatively smaller number of keys.

Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_2$. Thus we will use binary codes. The encryption and decryption operations are additions modulo 2:

$$e_z(x) = x + z \bmod 2$$

and

$$d_z(y) = y + z \bmod 2.$$

Note that in binary case, $x + z = x - z$ ($1 = -1 \pmod 2$). The key stream is formed as follows.

Suppose the first m keys are $(k_1, k_2, \cdots, k_m)$, i.e., $z_i = k_i, 1 \leq i \leq m$. We also select m element $c_0, c_1, \cdots, c_{m-1} \in \mathbb{Z}_2$. The key stream is generated by linear recurrence relation of degree m:

$$z_{i+m} = \sum_{j=0}^{m-1} c_j z_{i+j} \bmod 2.$$

In general, the period of the key stream is $2^m - 1$ which is much larger than $2m$ (We only selected $2m$ numbers $k_1, k_2, \cdots, k_m, c_0, c_1, \cdots c_{m-1}$ as the key).

Example 2.7.1 Suppose we choose $m = 4$ and the first four keys are $(1, 0, 1, 0)$. Let the constants $(c_0, c_1, c_2, c_3) = (1, 1, 0, 0)$. Then

$$z_{i+4} = z_i + z_{i+1} \bmod 2.$$

Therefore the key stream is as follows.

$$1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, \cdots.$$

Another appealing aspect of this method of key stream generation is that the key steam can be produced efficiently in hardware using a *linear feedback shift register* (LFSR). For example, we can use the LFSR in Figure 2.11 to generate the key stream in Example 2.7.1, where $\oplus$ denotes the exclusive-or operation (XOR). In fact, the key vector (k_0, k_1, k_2, k_3) can be any nonzero vector. Note that since $x \oplus x = 0$ for any x, we can use the same machine to do the encryption and decryption.

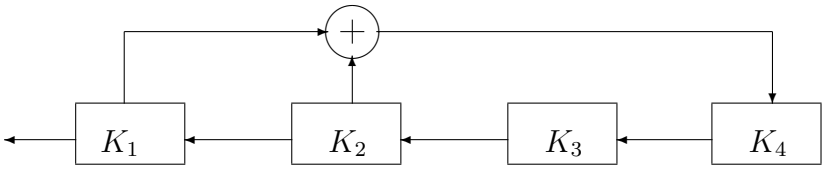


Figure 2.11: A Linear Feedback Shift Register

There are some methods to attack the LFSR based stream cipher in known-plaintext level. From plaintext and ciphertext, $y_i = x_i + z_i$, we know that $z_i = y_i - x_i$. So if we can figure out the parameters $c_0, c_1, \cdots, c_{m-1}$, then we can get the key stream. Note that there are linear relationship between

the values of c_i and z_j. If we know the value of m, then we can obtain a series of linear equations about the m unknowns $c_0, c_1, \cdots, c_{m-1}$. We might be able to solve these equations using linear algebra.

Another possible attack for the LFSR and other stream cipher is that if two plaintexts used a same key to encrypt, the XOR of the two ciphertexts is the same as the XOR of the two plaintexts. That means Oscar can easily attack the system if he can choose plaintext.

One common used stream cipher is RC4 which is a stream cipher designed in RSA laboratories by Ron Rivest in 1987. This cipher is widely used in commercial applications including Oracle SQL, Microsoft Windows and the SSL. The algorithm is kept as a trade secret until 1994. The external analysis of RC4 was invoked by the leakage of its source code in 1994 to `cypherpunks` mailing list. The key stream generated by RC4 is a stream of pseudo-random bytes (8-bit).

In the RC4 algorithm the key stream is completely independent of the plaintext used. So it is a synchronous stream cipher. The RC4 uses a S-vector $(S(0), \cdots, S(255))$, each of the entries is a byte (8 bits). S-vector is a permutation of the numbers 0 to 255, and the permutation is a function of the variable length key. There are two counters i, and j, both initialized to 0 used in the algorithm.

The S-vector is initialized as $S(0) = 0, S(1) = 1, \cdots, S(255) = 255$.

The key length of RC4 can be any number of bytes between 1 to 256. Another 256 bytes array T is then filled with the key, the key is repeated as necessary to fill the entire array. So if the key has 256 bytes, then T is same as the key. If the length of key is 8 bytes, then T contains 32 copies of key, and so on.

The index j is then set to 0. The algorithm in Figure 2.12 is used to initialize the S-vector. This algorithm does some permutation of the S-vector, which depends on the key (the array T).

The algorithm in Figure 2.13 is then used to generate a key.

K is then XORed with the plaintext to produce the ciphertext. The operations used in RC4 are additions and swaps. So RC4 is a fast encryption which can be implemented easily by a software. So it has some advantages than LFSR which is more efficient using hardware implementation.

RSA claims that the algorithm is immune to differential and linear cryptanalysis (we will discuss these attacks in the next chapter). The algorithm can also be changed from the 8-bit used above to 16-bit by using a 16-bit word.

for $i = 0$ **to** 255 **do**
$j = (j + S(i) + T(i)) \bmod 256$
Swap $S(i)$ and $S(j)$
end for

Figure 2.12: RC4 Key initialization

$i = (i + 1) \bmod 256$
$j = (j + S(i)) \bmod 256$
Swap $S(i)$ and $S(j)$
$t = (S(i) + S(j)) \bmod 256$
$K = S(t)$

Figure 2.13: Key stream of RC4

2.8 Product Cryptosystems

Because of the rapid development of computer, the cryptosystem requires more complicated encryption functions and larger key spaces. One method called product cryptosystems, innovated by Shannon, is an important idea for modern cryptosystems. This method allows us to build "large" cryptosystems from "small" ones.

Suppose we have two cryptosystems $\mathbf{S}_1 = (\mathcal{P}_1, \mathcal{C}_1, \mathcal{K}_1, \mathcal{E}_1, \mathcal{D}_1)$ and $\mathbf{S}_2 = (\mathcal{P}_2, \mathcal{C}_2, \mathcal{K}_2, \mathcal{E}_2, \mathcal{D}_2)$. If $\mathcal{C}_1 = \mathcal{P}_2$, then the product of $\mathbf{S}_1$ and $\mathbf{S}_2$, $(\mathbf{S}_1 \times \mathbf{S}_2)$, is defined as follows:

$$(\mathcal{P}_1, \mathcal{C}_2, \mathcal{K}_1 \times \mathcal{K}_2, \mathcal{E}, \mathcal{D}),$$

where for a key $(K_1, K_2) \in \mathcal{K}_1 \times \mathcal{K}_2$,

$$e_{(K_1,K_2)}(x) = e_{K_2}(e_{K_1}(x))$$

and

$$d_{(K_1,K_2)}(y) = d_{K_1}(d_{K_2}(y)).$$

The product of cryptosystems is also called a combination of cryptosystems. Two cryptosystem can be product if and only if the cipertexts of the first system is contained in the plaintexts of the second system. However, sometimes a product of cryptosystems will not result a new crptosystem. For example, suppose $\mathbf{S}_1$ is a Vigenére Cipher and $\mathbf{S}_2$ is a Shift Cipher. Then $\mathbf{S}_1 \times \mathbf{S}_2$ is still a Vigenére Cipher. Only the key is shifted in the product system. Therefore such a product is meaningless. In some cases, however, the product of cryptosystems does form a new cryptosystem.

Example 2.8.1 Suppose $\mathbf{S}_1$ is a substitution cipher and $\mathbf{S}_2$ is a permutation cipher. Then $\mathbf{S}_1 \times \mathbf{S}_2$ is a new cryptosystem. The key space of the new system is $26! \times m!$.

Sometimes one crytosystem combines itself will create a new system. In that case, we just need to use the encryption algorithm two times. This method gives us a economical way to enlarge the key space. A cryptosystem $\mathbf{S}$ is called idempotent if $\mathbf{S} \times \mathbf{S} = \mathbf{S}$. It is easy to check that the Shift Cipher, the Substitution Cipher, the Hill Cipher, the Vigenére and the Permutation Ciphers are examples of idempotent ciphers. The cryptosystem obtained from Example 2.8.1 is not idempotent. If a system $\mathbf{S}$ is not idempotent, then we can construct a system as follows:

$$\underbrace{\mathbf{S} \times \mathbf{S} \times \cdots \times \mathbf{S}}_{n} = \mathbf{S}^n,$$

which is called an *iterated* cryptosystem. Iterated method is used in modern block encryption systems.

2.9 Modular Arithmetics

In this section, we display some knowledge of modular arithmetic used in this chapter.

Definition 2.9.1 *Suppose a and b are integers, and m is a positive integer. Then we write $a \equiv b \pmod{m}$ if m divides $b-a$. (Equivalently, if $a = mt+b$ for some integer t).*

$a \equiv b \pmod{m}$ is read as " a is congruent to b modulo m." The integer m is referred as modulus. The following properties are easy to check.

If $x \equiv a \pmod{m}$ and $y \equiv b \pmod{m}$, then $x + y \equiv a + b \pmod{m}$ and $xy \equiv ab \pmod{m}$.

For example, since $13 \equiv 3 \pmod{5}$ and $7 \equiv 2 \pmod{5}$, we have $13 + 7 \equiv 3 + 2 \equiv 0 \pmod{5}, 13 \cdot 7 \equiv 3 \cdot 2 \equiv 6 \equiv 1 \pmod{5}$.

Suppose $m > 1$ is an integer. We can assume that the remainder of an integer a divided by m is $b, 0 \leq b \leq m - 1$, i.e., $a \equiv b \pmod{m}, 0 \leq b \leq m - 1$. We say that a is reduced to b modulo m.

We now define arithmetic modulo m: $\mathbb{Z}_m$ is defined to be the set $\{0, \cdots, m-1\}$, equipped with operations $+$ and $\times$. Addition and multiplication work exactly like real addition and multiplication, except that the results are reduced modulo m.

For example, in $\mathbb{Z}_5$, we have $2+4 = 1, 3+2 = 0, 2 \times 4 = 3, 3 \times 2 = 1$, etc.

Suppose that $a, b, c \in \mathbb{Z}_m$. The addition and multiplication in $\mathbb{Z}_m$ has the following properties:

1. addition is closed: $a + b \in \mathbb{Z}_m$
2. addition is commutative: $a + b = b + a$
3. addition is associative: $(a + b) + c = a + (b + c)$
4. 0 is an additive identity: $a + 0 = 0 + a = a$
5. the additive inverse of a is $m - a$: $a + (m - a) = (m - a) + a = 0$
6. multiplication is closed: $ab \in \mathbb{Z}_m$
7. multiplication is commutative: $ab = ba$
8. multiplication is associative: $(ab)c = a(bc)$
9. 1 is multiplicative identity: $a = 1 \times a = a$
10. multiplication distributes over addition: $(a + b)c = ac + bc, a(b + c) = ab + ac$.

Properties 1, 3 – 5 say that $\mathbb{Z}_m$ is a *group* with respect to the addition operation. Properties 1 – 10 establish that $\mathbb{Z}_m$ is a *commutative ring*. Rings and groups are useful algebraic structures.

It is not necessary that an element in $\mathbb{Z}_m$ has a multiplicative inverse. In fact, an element $a \in \mathbb{Z}_m$ has a multiplicative inverse if and only if $\gcd(a, m) = 1$. Particularly, for a prime number p, each nonzero element in $\mathbb{Z}_p$ has a multiplicative inverse. When every nonzero element in a commutative ring has a multiplicative inverse, it is called a *field*. $\mathbb{Z}_p$ is an example of finite field.

Chapter 3

Modern Block Ciphers

In this chapter, we examine modern conventional cryptosystems. Since the explosive growth of computer systems, now people have very powerful facilities to perform attacks for a cryptosystems. Therefore the modern conventional cryptosystems are very complicated.

As a good encryption system, we need to consider both security and efficiency. However, in general there is a trade-off between security and efficiency. For example, we already observed that the key space of a cryptosystem should be large enough otherwise a key exhausted search can break the system. On the other hand, a large key space means more storage space and more computation time.

Although modern block ciphers are more complicated, we can see that techniques of classic block ciphers discussed in previous chapter are still used. In this chapter, we mainly discuss two most important block ciphers: DES and AES.

3.1 The Data Encryption Standard

The Data Encryption Standard, or DES, is the most widely used cryptosystem in the world. DES was developed at IBM and first published in the Federal Register of March 17, 1975. In 1977, this system was approved as a Federal Information Processing Standard. Although DES now was proved to be insecure and a new encryption standard was announced on November 26, 2001 (FIPS PUB 197), DES is still an important modern cryptosystem.

DES is an iterated block cipher. The three operations: XOR, substitution

and permutation form the backbone of the encryption.

DES encrypts a plaintext bitstring x of length 64 using a key K which is a bitstring of length 56. The resulting ciphertext is again a bitstring of length 64.

The algorithm can described as follows:

1. A fixed initial permutation $\mathbb{P}$ is use to permuting the bits of the plaintext x. The resulting 64 bitstring is divided into two parts L_0 and R_0, each comprised 32 bits.

2. 16 iterations of Feistel type cipher are then performed. For $1 \leq i \leq 16$, L_iR_i is computed according to the following rule:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i), \end{aligned}$$

   where $\oplus$ denotes the XOR (exclusive-or) of two bitstrings. And

$$f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i)),$$

   with the operations E (expansion), S (S-box lookup), and P (permutation) discussed later. $K_1, K_2, \cdots, K_{16}$ are each bitstrings of length 48 computed as a function of the key K. The selections of these subkeys, or "key schedule" will be discussed later.

3. Apply the inverse of initial permutation $\mathbb{P}$ to $R_{16}L_{16}$ and obtain the ciphertxt.

Figure 3.1 describes the algorithm of DES.

The function $f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i))$ works as follows. First $E(R_{i-1})$ expands 32 bits of R_{i-1} to 48 bits in a certain way (16 bits appears twice). The expansion is specified by the following table.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

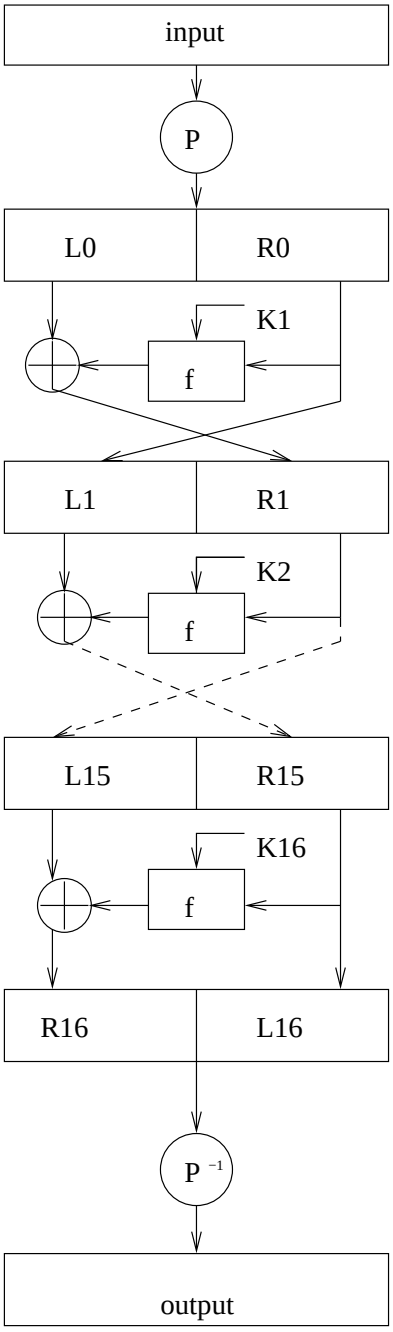


Figure 3.1: The Data Encryption Standard

For a 32-bit string $b_1b_2\cdots b_{32}$, the 48-bit output is $b_{32}b_1b_2b_3b_4b_5b_4\cdots b_1$.

Then the round subkey K_i and the expanded data are XORed together. The result is divided into eight 6-bit strings $B = B_1B_2\cdots B_8$. These strings are then passed through the eight "S-boxes" $S_1, S_2, \cdots, S_8$. Each S-box takes input of six bits and outputs four bits.

The S-boxes are the source of DES's complexity. We can write an S-box as a 4×16 table. The definition of S-boxes are listed in Table 3.1

Suppose the input is $b_1b_2b_3b_4b_5b_6$. The bits b_1, b_6 determine the row, while the bits b_2, b_3, b_4, b_5 determine the column. The output is the entry in the intersection. Note that each possible four-bit entry $0, \cdots, 15$ appears in each row of the S-box output. For example, suppose the input of S_2 is 111010. Then $b_1b_6 = 10$ which is 2 in decimal and $b_2b_3b_4b_5 = 1101$ which is 13 in decimal. Therefore the output is 0011 (number 3).

Finally, the total 32-bit output is permuted according to a fixed permutation P described as follows.

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

The f function is depicted in Figure 3.2

Now we need to describe the computation of key schedule from the key K. Actually, K is a bitstring of length 64, but only 56 bits are used. The other 8 bits are used for parity-check (for error-detection). Thus the size of key space is 2^{56}. The 56 bits are chosen as follows.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56

The 56-bit key is permuted according to the follow table of permuted

$$S_1 = \begin{bmatrix} 14 & 4 & 13 & 1 & 2 & 15 & 11 & 8 & 3 & 10 & 6 & 12 & 5 & 9 & 0 & 7 \\ 0 & 15 & 7 & 4 & 14 & 2 & 13 & 1 & 10 & 6 & 12 & 11 & 9 & 5 & 3 & 8 \\ 4 & 1 & 14 & 8 & 13 & 6 & 2 & 11 & 15 & 12 & 9 & 7 & 3 & 10 & 5 & 0 \\ 15 & 12 & 8 & 2 & 4 & 9 & 1 & 7 & 5 & 11 & 3 & 14 & 10 & 0 & 6 & 13 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} 15 & 1 & 8 & 14 & 6 & 11 & 3 & 4 & 9 & 7 & 2 & 13 & 12 & 0 & 5 & 10 \\ 3 & 13 & 4 & 7 & 15 & 2 & 8 & 14 & 12 & 0 & 1 & 10 & 6 & 9 & 11 & 5 \\ 0 & 14 & 7 & 11 & 10 & 4 & 13 & 1 & 5 & 8 & 12 & 6 & 9 & 3 & 2 & 15 \\ 13 & 8 & 10 & 1 & 3 & 15 & 4 & 2 & 11 & 6 & 7 & 12 & 0 & 5 & 14 & 9 \end{bmatrix}$$

$$S_3 = \begin{bmatrix} 10 & 0 & 9 & 14 & 6 & 3 & 15 & 5 & 1 & 13 & 12 & 7 & 11 & 4 & 2 & 8 \\ 13 & 7 & 0 & 9 & 3 & 4 & 6 & 10 & 2 & 8 & 5 & 14 & 12 & 11 & 15 & 1 \\ 13 & 6 & 4 & 9 & 8 & 15 & 3 & 0 & 11 & 1 & 2 & 12 & 5 & 10 & 14 & 7 \\ 1 & 10 & 13 & 0 & 6 & 9 & 8 & 7 & 4 & 15 & 14 & 3 & 11 & 5 & 2 & 12 \end{bmatrix}$$

$$S_4 = \begin{bmatrix} 7 & 13 & 14 & 3 & 0 & 6 & 9 & 10 & 1 & 2 & 8 & 5 & 11 & 12 & 4 & 15 \\ 13 & 8 & 11 & 5 & 6 & 15 & 0 & 3 & 4 & 7 & 2 & 12 & 1 & 10 & 14 & 9 \\ 10 & 6 & 9 & 0 & 12 & 11 & 7 & 13 & 15 & 1 & 3 & 14 & 5 & 2 & 8 & 4 \\ 3 & 15 & 0 & 6 & 10 & 1 & 13 & 8 & 9 & 4 & 5 & 11 & 12 & 7 & 2 & 14 \end{bmatrix}$$

$$S_5 = \begin{bmatrix} 2 & 12 & 4 & 1 & 7 & 10 & 11 & 6 & 8 & 5 & 3 & 15 & 13 & 0 & 14 & 9 \\ 14 & 11 & 2 & 12 & 4 & 7 & 13 & 1 & 5 & 0 & 15 & 10 & 3 & 9 & 8 & 6 \\ 4 & 2 & 1 & 11 & 10 & 13 & 7 & 8 & 15 & 9 & 12 & 5 & 6 & 3 & 0 & 14 \\ 11 & 8 & 12 & 7 & 1 & 14 & 2 & 13 & 6 & 15 & 0 & 9 & 10 & 4 & 5 & 3 \end{bmatrix}$$

$$S_6 = \begin{bmatrix} 12 & 1 & 10 & 15 & 9 & 2 & 6 & 8 & 0 & 13 & 3 & 4 & 14 & 7 & 5 & 11 \\ 10 & 15 & 4 & 2 & 7 & 12 & 9 & 5 & 6 & 1 & 13 & 14 & 0 & 11 & 3 & 8 \\ 9 & 14 & 15 & 5 & 2 & 8 & 12 & 3 & 7 & 0 & 4 & 10 & 1 & 13 & 11 & 6 \\ 4 & 3 & 2 & 12 & 9 & 5 & 15 & 10 & 11 & 14 & 1 & 7 & 6 & 0 & 8 & 13 \end{bmatrix}$$

$$S_7 = \begin{bmatrix} 4 & 11 & 2 & 14 & 15 & 0 & 8 & 13 & 3 & 12 & 9 & 7 & 5 & 10 & 6 & 1 \\ 13 & 0 & 11 & 7 & 4 & 9 & 1 & 10 & 14 & 3 & 5 & 12 & 2 & 15 & 8 & 6 \\ 1 & 4 & 11 & 13 & 12 & 3 & 7 & 14 & 10 & 15 & 6 & 8 & 0 & 5 & 9 & 2 \\ 6 & 11 & 13 & 8 & 1 & 4 & 10 & 7 & 9 & 5 & 0 & 15 & 14 & 2 & 3 & 12 \end{bmatrix}$$

$$S_8 = \begin{bmatrix} 13 & 2 & 8 & 4 & 6 & 15 & 11 & 1 & 10 & 9 & 3 & 14 & 5 & 0 & 12 & 7 \\ 1 & 15 & 13 & 8 & 10 & 3 & 7 & 4 & 12 & 5 & 6 & 11 & 0 & 14 & 9 & 2 \\ 7 & 11 & 4 & 1 & 9 & 12 & 14 & 2 & 0 & 6 & 10 & 13 & 15 & 3 & 5 & 8 \\ 2 & 1 & 14 & 7 & 4 & 10 & 8 & 13 & 15 & 12 & 9 & 0 & 3 & 5 & 6 & 11 \end{bmatrix}$$

Table 3.1: S-boxes of DES

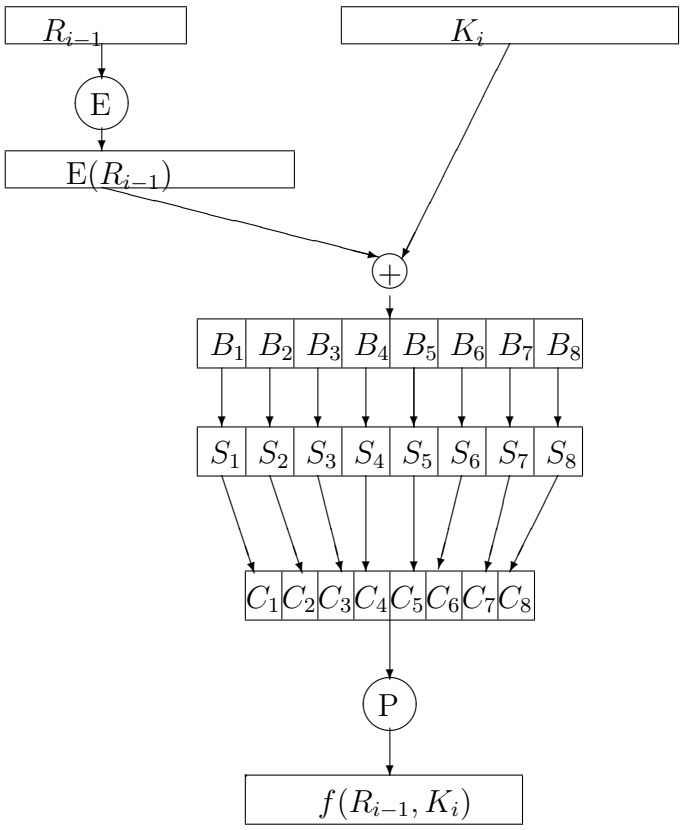


Figure 3.2: The DES f function

choice one (PC-1):

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Then the 56-bit is split into two 28-bit halves and each half rotated (shifted) one or two bits each round (one bit in rounds $1, 2, 9$ and 16; two bits otherwise). In each round, the two halves are put back together, and then 48 particular bits are chosen and put in the order as follows (PC-2):

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

So the 14th bit is put in the first place, 17th bit is put in second place, etc. The output is the round key.

Decryption is done using the same algorithm as encryption, starting with grouping ciphertext into 64-bit strings. This is one advantage of the Feistel type cipher. Note that the DES algorithm has the following properties:

$$\begin{aligned} R_{i-1} &= L_i \\ L_{i-1} &= L_{i-1} \oplus f(R_{i-1}, K_i) \oplus f(R_{i-1}, K_i) \\ &= R_i \oplus f(R_{i-1}, K_i) \end{aligned}$$

Therefore using the key schedule $K_{16}, \cdots, K_1$ in reverse order, the output will be the plaintext.

DES can be implemented very efficiently, either in hardware or in software.

3.2 Attacks on DES

When DES was proposed as a standard, there was considerable criticism and quickly followed by attacks. Some researchers objected to the system's small key space. There were even rumours that NSA (National Security Agency) had pressed for shorter key length. Another objection to DES concerned the S-boxes. Several people have suggested that the S-boxes might contain hidden "trapdoors" which would allow the NSA to decrypt messages. There have been many attacks to DES. Most of them are known plaintext attacks or chosen-plaintext attacks.

One well-known attack on DES is the method called *differential* cryptanalysis introduced by Biham and Shamir. Although the S-boxes have balanced output (each possible output appears four times, once in each row),

the output of differences of inputs has an uneven distribution. More precisely, suppose $(B_1, B_1'), (B_2, B_2'), \cdots, (B_{64}, B_{64}')$ are 64 pairs in $(\mathbb{Z}_2)^6$ such that $B_j \oplus B_j' = B_i \oplus B_i'$ for each $1 \leq i \leq j \leq 64$ (the pairs with same difference). Then the "differences" of the output of the S-box $S(B_i) \oplus S(B_i')$ have non-uniform distributions. Therefore we are able to find the differences in input pairs that have high probability of causing certain differences in output pairs in an iterate round. From this fact, we can get some information about the key from a chosen-plaintext attack. We will not discuss the details of difference cryptanalysis here, but mention that Biham and Shamir indicated in 1990 that using difference cryptanalysis requires only 2^{47} inputs, fewer than the 2^{56} that required by key exhaustive search.

Another method used to attack DES is called *linear cryptanalysis* discovered by Matsui. This attack examines sums of plaintext and ciphertext bits to reveal information about sums of key bits. Here "sum" means XORs. Matsui's known-plaintext attack on DES required studying 2^{43} encrypted texts.

Although the difference cryptanalysis and linear cryptanalysis do not break DES, these attacks are very important. These attacks actually work against any block cipher.

On the other hand, people tried to construct efficient key exhaustive search machine to break DES. In 1998, the Electronic Frontier Foundation (EFF) built "DES Craker" using custom-designed chips and a personal computer. Costing less than $ 250,000 and taking less than a year to build, the DES Craker broke a message in 56 hours. In 1999, this result was improved to 22 hours using a combination of 100,000 networked PCs and the EFF machine.

3.3 DES Modes and Triple-DES

DES has had a wide applications in the world. To apply DES in a variety of applications, four modes have been developed (FIPS PUB 81). Another mode is included in NIST (National Institute of Standards and Technology) Special Publication 800-38A. In this section, we give a brief description for these modes. Note that these modes are allocatable for other block ciphers such as AES which we will discuss later.

ECB (Electronic Codebook mode): ECB mode corresponding to the usual use of a block cipher. The plaintext are grouped into blocks of 64-bit

and each block is encrypted with the same key K.

CBC (cipher block chaining mode): In CBC mode, each ciphertext block y_i is XORed with the next plaintext block x_{i+1} before x_{i+1} being encrypted by the key K. An initialized vector IV $= y_0$ is chosen before encryption. This mode is used some idea similar to the autokey cipher. Using this mode, the encryption can be described as follows.

$$\begin{aligned} y_1 &= e_K(y_0 \oplus x_1), \\ y_2 &= e_K(y_1 \oplus x_2), \\ &\cdots\cdots \\ y_n &= e_K(y_{n-1} \oplus x_n). \end{aligned}$$

The decryption of CBC mode is as follows.

$$\begin{aligned} x_1 &= d_K(y_1) \oplus y_0, \\ x_2 &= d_K(y_2) \oplus y_1, \\ &\cdots\cdots \\ x_n &= d_K(y_n) \oplus y_{n-1}. \end{aligned}$$

CFB (Cipher Feedback mode): In CFB mode, we start with an initialization vector y_0 =IV and produce the key stream $z_i = e_K(y_{i-1}), i \geq 1$. The ciphertext blocks are $y_i = x_i \oplus z_i, i \geq 1$. So the encryption is as follows.

$$\begin{aligned} y_1 &= x_1 \oplus e_K(y_0), \\ y_2 &= x_2 \oplus e_K(y_1), \\ &\cdots\cdots \\ y_n &= x_n \oplus e_K(y_{n-1}). \end{aligned}$$

In this mode, we do not use the decryption function to decrypt a ciphertext:

$$\begin{aligned} x_1 &= y_1 \oplus e_K(y_0), \\ x_2 &= y_2 \oplus e_K(y_1), \\ &\cdots\cdots \\ x_n &= y_n \oplus e_K(y_{n-1}). \end{aligned}$$

OFB (Output Feedback mode): In OFB mode, let z_0 =IV be an initialization vector. The key stream is $z_i = e_K(z_{i-1}), i \geq 1$ and the ciphertext blocks are $y_i = x_i \oplus z_i, i \geq 1$. The OFB mode is similar to a synchronous stream cipher.

CTR (Counter mode): In CTR mode, a counter c is selected, which has the same size of a plaintext block (64-bit in DES). The encryption is as follows.

$$\begin{aligned} &y_1 = x_1 \oplus e_K(c), \\ &y_2 = x_2 \oplus e_K(c+1), \\ &\cdots\cdots \\ &y_n = x_n \oplus e_K(c+n-1). \end{aligned}$$

In DES, the size of blocks in both plaintext and cyphertext is 64-bit. However, when we use CFB, OFB or CTR mode, the block size of plaintext can be any number less than or equal to 64-bit. For example, if there is a plaintext block with 16-bit in CFB mode, then the encryption can be $y_i = x_i \oplus s_{16}(z_i)$, where $s_j(z_i)$ means the j most significant bits of z_i. In this way, we can avoid to add padding to the plaintext.

The different modes of operations have different advantages and disadvantages. ECB is usually used for encrypting short message. In ECB and OFB modes, changing one plaintext block only causes the changing the corresponding ciphertext block. Other ciphertext blocks will not be effected. This property is desired for transmission over noisy channel (e.g., satellite communication). However, in ECB mode same plaintext blocks will produce same ciphertext blocks, so one might find some patterns in the ciphertext if same blocks repeat several times in a long plaintext.

On the other hand, if a plaintext block is changed in CBC and CFB modes, then the according ciphertext block and all subsequent ciphertext blocks will be affected. This property makes CBC and CFB useful for purposes of authentication. We will discuss message authentication code later.

CFB, OFB and CTR modes use encryption function for both encryption and decryption, that simplifies the cryptosystem. However, CTR can do parallel encryptions, i.e., several blocks can be encrypted at the same time. But CFB and OFB modes only can do sequential encryptions.

Since there are serious concern about the key size of DES, we will think about using product of DES to enlarge key space. It was proved in 1992 that DES is not idempotent. So we can try to use the product method for DES.

First we will try to use double DES. So we may choose two keys K_1 and K_2 to encrypt a plaintext block x as follows

$$y = e_{K_2}(e_{K_1}(x)).$$

However, there is a method called meet-in-the-middle attack to break this system. Let

$$m = e_{K_1}(x) = d_{K_2}(y).$$

Then we can perform known-plaintext attack as follows. Suppose we know the values of x and y. First we use 2^{56} keys to encrypt the plaintext x and store these values (sorted) in a table. Then we use 2^{56} possible keys to decrypt the ciphertext y and check with the table. In this way we might find m and the two keys. Because there are efficient sort and search algorithms, the double DES dose not give much improvement to the DES.

Next we consider triple DES. An obvious way is to use three keys and three rounds. In 1979, Tuchman proposed a triple encryption method that uses only two keys as follows:

$$y = e_{K_1}(d_{K_2}(e_{K_1}(x))).$$

Triple DES with two keys has been adopted for use in the key management standards. One advantage of using d_{K_2} instead of e_{K_2} is that if we let $K_2 = K_1$, then the triple DES can be used as single DES:

$$y = e_{K_1}(d_{K_1}(e_{K_1}(x))) = e_{K_1}(x).$$

There is also triple DES with three keys defined as follows.

$$y = e_{K_3}(d_{K_2}(e_{K_1}(x))).$$

Three-key triple DES are applied in some internet-based applications. Although triple DES has larger key space, its running time is also tripled. Another disadvantage for 3-DES is that its block size is 64-bit. For the security reason, larger block size is desired.

3.4 The Advanced Encryption Standard

The National Institute of Standards and Technology (NIST) announced the Advanced Encryption Standard (AES) on November 26, 2001 (FIPS PUB 197, see `http://cscr.nist.gov/publications/`). As the successor of DES, AES applies a much larger key space. AES has three settings. The Key-Block-Round combinations of this standard are as in Figure 3.3

	Key Length	Block Size	Number of Rounds
AES-128	128 bits	128 bits	10
AES-192	192 bits	128 bits	12
AES-256	256 bits	128 bits	14

Figure 3.3: Key-Block-Round Combinations

AES was developed by two Belgian cryptographers Joan Daemen and Vincent Rijmen. This cryptosystem relies more directly on algebraic constructions than do the other modern cryptosystems. The original cryptosystem proposed by Daemen and Rijmen (They call it Rijndael) allowed three different block size. The AES used the fixed 128-bit block to simplify the system.

In Section 2.2 we defined commutative ring. If any non-zero element in a commutative ring has a multiplicative inverse, then the ring is a *field*. A field with finite elements is called a finite field or Galois field, and denoted as $GF(q)$, where q is the number of the elements. The following theorem is well-known (see Section 3.6 for more materials about finite fields).

Theorem 3.4.1 *A $GF(q)$ exists if and only if q is a power of prime.*

AES uses $GF(2^8)$ (with irreducible polynomial $x^8+x^4+x^3+x+1$ which determines the operations in $GF(2^8)$) in which each element can be expressed as a byte (8-bit string). In AES, the 128 bits of plaintext block is written as 16 bytes and is placed in a 4×4 array of elements of $GF(2^8)$ as follows (arranged column by column).

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

For convenience, a byte is also expressed using hexadecimal notations. The hexadecimals are denoted as $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}$. One

byte can be written as two hexadecimals. For example, a byte {10110101} can be written as {$b5$} (1011 and 0101).

AES is also an iterated cryptosystem. AES does not use a Feistel structure. So it put whole block, not half block, to S-boxes. In this way, a inverse algorithm, decryption algorithm, must be provided.

In the encryption algorithm of AES, each round consists of four operations (transformations): *SubBytes, ShiftRows, MixColumns* and *AddRoundKey* (the last round skips the MixColumn operation).

The SubBytes transformation is a non-linear substitution using a substitution table (S-box). The S-box of transformation is presented in hexadecimal form as in Figure 3.4.

$$\begin{bmatrix}
63 & 7c & 77 & 7b & f2 & 6b & 6f & c5 & 30 & 01 & 67 & 2b & fe & d7 & ab & 76 \\
ca & 82 & c9 & 7d & fa & 59 & 47 & f0 & ad & d4 & a2 & af & 9c & a4 & 72 & c0 \\
b7 & fd & 93 & 26 & 36 & 3f & f7 & cc & 34 & a5 & e5 & f1 & 71 & d8 & 31 & 15 \\
04 & c7 & 23 & c3 & 18 & 96 & 05 & 9a & 07 & 12 & 80 & e2 & eb & 27 & b2 & 75 \\
09 & 83 & 2c & 1a & 1b & 6e & 5a & a0 & 52 & 3b & d6 & b3 & 29 & e3 & 2f & 84 \\
53 & d1 & 00 & ed & 20 & fc & b1 & 5b & 6a & cb & be & 39 & 4a & 4c & 58 & cf \\
d0 & ef & aa & fb & 43 & 4d & 33 & 85 & 45 & fa & 02 & 7f & 50 & 3c & 9f & a8 \\
51 & a3 & 40 & 8f & 92 & 9d & 38 & f5 & bc & b6 & da & 21 & 10 & ff & f3 & d2 \\
cd & 0c & 13 & ec & 5f & 97 & 44 & 17 & c4 & a7 & 7e & 3d & 64 & 5d & 19 & 73 \\
60 & 81 & 4f & dc & 22 & 2a & 90 & 88 & 46 & ee & b8 & 14 & de & 5e & 0b & db \\
e0 & 32 & 3a & 0a & 49 & 06 & 24 & 5c & c2 & d3 & ac & 62 & 91 & 95 & e4 & 79 \\
e7 & c8 & 37 & 6d & 8d & d5 & 4e & a9 & 6c & 56 & f4 & ea & 65 & 7a & ae & 08 \\
ba & 78 & 25 & 2e & 1c & a6 & b4 & c6 & e8 & dd & 74 & 1f & 4b & bd & 8b & 8a \\
70 & 3e & b5 & 66 & 48 & 03 & f6 & 0e & 61 & 35 & 57 & b9 & 86 & c1 & 1d & 9e \\
e1 & f8 & 98 & 11 & 69 & d9 & 8e & 94 & 9b & 1e & 87 & e9 & ce & 55 & 28 & df \\
8c & a1 & 89 & 0d & bf & e6 & 42 & 68 & 41 & 99 & 2d & 0f & b0 & 54 & bb & 16
\end{bmatrix}$$

Figure 3.4: S-Box in AES

Suppose a byte in the input array is 01011010. Then we can write it as two hexadecimals $5a$ (0101 and 1010). The output is the element in 5th row and ath column in the table, i.e., be. So the output is 10111110. Different from DES's S-boxes, this S-box is formed by algebraic operations. First each inputted element is replaced by its multiplicative inverse in $GF(2^8)$ (while 00 is mapped to its self). Then the array undergoes a fixed affine transformation.

This transformation can be described as follows. Suppose the inverse of the element is $b_7b_6b_5b_4b_3b_2b_1b_0$ and the output is $b'_7b'_6b'_5b'_4b'_3b'_2b'_1b'_0$. Then

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Note that the computation of the above equation is in $\mathbb{Z}_2$.

In the ShiftRows transformation, the operation cyclically shifts the elements of the ith row of the array i elements to the right, where $i = 0, 1, 2, 3$. After ShifRows, the block looks as follows.

in_0	in_4	in_8	in_{12}
in_5	in_9	in_{13}	in_1
in_{10}	in_{14}	in_2	in_6
in_{15}	in_3	in_7	in_{11}

The MixColumns transformation operates on the input column-by-column. Suppose the input column is

S_0
S_1
S_2
S_3

Then the output

S'_0
S'_1
S'_2
S'_3

are as follows.

$$\begin{aligned} S'_0 &= (\{02\} \cdot S_0) \oplus (\{03\} \cdot S_1) \oplus S_2 \oplus S_3 \\ S'_1 &= S_0 \oplus (\{02\} \cdot S_1) \oplus (\{03\} \cdot S_2) \oplus S_3 \\ S'_2 &= S_0 \oplus S_1 \oplus (\{02\} \cdot S_2) \oplus (\{03\} \cdot S_3) \\ S'_3 &= (\{03\} \cdot S_0 \oplus S_1 \oplus S_2 \oplus (\{02\} \cdot S_3), \end{aligned}$$

where $(\cdot)$ is the multiplication in $GF(2^8)$.

Finally, the AddRoundKey transformation adds a round key to each column of the input using bit XOR operation. The size of a round key is 128-bit.

We can describe the AES encryption algorithm as follows. Suppose there are `Nr` iteration round. Then a key achedule algorithm will create `Nr` + 1 round keys $k_i, i = 0, 1, \cdots,$ `Nr`. A block of plaintext will go through the following.

- Add round key using k_0.
- From round 1 to round `Nr` $- 1$, perform transformations SubBytes, ShiftRows, MixColumns, AddRoundKey using $k_i, i = 1, \cdots,$ `Nr` $- 1$.
- Perform SubBytes, ShiftRows and AddRoundKey using $k_{\texttt{Nr}}$.

The key schedule of AES can be obtained from KeyExpansion() algorithm shown in Figure 3.5, where a byte is a 8-bit string and a word consists 4 bytes. In the algorithm, `Nk` is the number of words in the key and `Nr` is the number of rounds. `SubWord()` is a function that takes a four-byte input word and applied the S-box to each of the four bytes to produce an output word. The function `RotWord()` performs a cyclic permutation on the input four bytes of a word. The round constant word array, `Rcon[i]`, contains the values given by $[y_i, \{00\}, \{00\}, \{00\}]$, where

$$y_i = 1\underbrace{0 \cdots 0}_{i-1} \in GF(2^8)$$

The decryption algorithm of AES is a straightforward of inverse of the encryption algorithm. We can use the inverted transformations of encryption: *InvShiftRows, InvSubBytes, InvMixColumns* and *AddRoundKey*. The details of the decryption algorithm is omitted here.

From the algorithms of AES, we can see that AES takes advantage of 32-bit processors (while DES is for 8-bit processors). The S-boxes in AES are proved to be good against the differential and linear cryptanalysis. The implementation of AES both in hardware and software are efficient.

3.5 Some Other Block Ciphers

In 1997, the National Institute of Standards and Technology (NIST) announced a competition for the algorithm of Advanced Encryption Standard

```
KeyExpansion(byte key[4 * Nk], word w[4(Nr + 1)], Nk)
begin
  word temp
  i = 0
  while (i < Nk)
    w[i] =word(key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3])
    i = i + 1
  end while
  i = Nk
  while (i < 4(Nr + 1))
    temp = w[i − 1]
    if (i  (mod Nk) = 0)
      temp = SubWord(RotWord(temp)) ⊕ Rcon[i/Nk]
    else if (Nk > 6 and i  (mod Nk) = 4)
      temp = SubWord(temp)
    end if
  w[i] = w[i − Nk]⊕ temp
  i = i + 1
  end while
end
```

Figure 3.5: Pseudo Code for Key Expansion

(AES) which will replace DES. This time, NIST allowed foreign submission and foreign viewing of the candidates.

NIST published evaluation criteria for AES candidates. The criteria contains 3 parts which can be seen as the requirements for modern block ciphers.

- **Security:** Randomness of the output, sound mathematical basis and other security considerations.
- **Cost:** Computational efficiency, memory requirement.
- **Algorithm and implementation characteristics:** Flexibility (key and block sizes, different platforms and applications, implementations),

hardware and software suitability and simplicity.

AES candidates were due June 15, 1998. Of the twenty-one submissions, fifteen met NIST's criteria. In August 1999, NIST announced the five final candidates. One of the five finalist, Rijndael, was selected to be AES in 2001.

Now we briefly describe the other four final candidates.

MARS is developed by IBM, which breaks the 128-bit input block into four 32-bit words. MARS uses 32-round unbalanced Feistel type algorithm. In each of these rounds MARS uses two S-boxes with one distinguished word. The S-boxes used in MARS were built using some hash function applied to some fixed constant.

RC6 is developed by RSA Security Inc., which is modified from RC5. The structure of RC6 is simpler than others. Basically, the algorithm uses XOR, addition and rotation operations. RC6 operates on four 32-bit words and treats these words in pairs. RC6 uses 20-round cipher. In each round, data-dependent rotations are used for the cryptographic complexity (instead of using S-boxes).

Twofish was proposed by Counterpane Systems, a U.S.A. based cryptographic consulting firm. Twofish uses 16-round Feistel algorithms. The 128-bit input is broken into four 32-bit words. In each round, four distinct S-boxes are used. These S-boxes (8-bit to 8-bit) are key dependent, which is different from S-box in other systems. Twofish also uses matrix multiplication, addition and rotation in a round.

Serpent was created by three cryptographers from the United Kingdom, Israel, and Denmark. Serpent contains a 32-round algorithms. Each round consists of XORing the key and input, a pass through S-boxes, and a linear function that combines fixed rotations and XOR. 32 identical S-boxes (4-bit to 4-bit) are used in a round (but different round uses different ones).

There are also several encryption systems used or still used in internet. We simply list them as follows without details for references.

IDEA (International Data Encryption Algorithm) was developed by Xuejia Lai and James Massey of the Swiss Federal Institute of Technology. IDEA uses a 128-bit key and encrypts data in blocks of 64 bits.

Blowfish was developed by Bruce Schneier. Blowfish uses a key ranged from 32 bits to 448 bits and encrypts blocks of 64 bits.

RC5 was developed by Ron Rivest. The key range of RC5 is from 0 to 2040 bits. The length of a data block is 32, 64 or 128.

CAST-128 was developed by Carlisle Adams and Stafford Tavares. The

key size of CAST varies from 40 bits to 128 bits in 8-bit increments. The length of a block is 64 bits.

RC2 was also developed by Ron Rivest. The key range of RC2 is from 8 to 1024 bits. The length of a data block is 64.

3.5.1 CMVP

The Cryptographic Module Validation Program (CMVP) is a joint American and Canadian security accreditation program for cryptographic modules. The program is available to any vendors who seek to have their products certified for use by the U.S. Government and regulated industries (such as financial and health-care institutions) that collect, store, transfer, share and disseminate "sensitive, but not classified" information. All of the tests under the CMVP are handled by third-party laboratories that are accredited as Cryptographic Module Testing Laboratories by the National Voluntary Laboratory Accreditation Program (NVLAP). Product certifications under the CMVP are performed in accordance with the requirements of FIPS 140-2.

The Government of Canada also recommends the use of FIPS 140 validated cryptographic modules in unclassified applications of its departments.

The CMVP was established by the U.S. National Institute of Standards and Technology (NIST) and the Communications Security Establishment (CSE) of the Government of Canada in July 1995.

3.6 Finite Fields

We already defined finite fields in Section 2.9. As examples, we have seen that $\mathbb{Z}_p$ is a finite fields. For the existence of finite fields, we have the following theorem.

Theorem 3.6.1 *There exists a finite field of order m if and only if $m = p^n$, where p is a prime number and n is a positive integer.*

A finite field with p^n elements is called Galois field denoted by $GF(p^n)$. A $GF(p)$ for a prime p is equivalent to $\mathbb{Z}_p$. A $GF(p^n)$, $n > 1$, can be built from $GF(p)$ using an irreducible polynomial of degree n on $GF(p)$. In the following, we briefly discuss the structure of $GF(p^n)$.

A polynomial of degree n on $GF(p)$ can be expressed as:

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

where $0 \leq a_i \leq p-1, a_n \neq 0$ (or we say that $a_i \in GF(p), a_n \neq 0$). The polynomial $P(x)$ is irreducible if $P(x)$ cannot be written as a product of two polynomials with both of degree less than n.

For two polynomials, we can define addition and multiplication.

Example 3.6.2 Suppose $P_1(x) = x^3 + x^2 + 1, P_2(x) = x^3 + x + 2$ are two polynomials defined on $GF(3)$. Then

$$P_1(x) + P_2(x) = 2x^3 + x^2 + x$$

and

$$P_1(x)P_2(x) = x^6 + x^5 + x^4 + x^3 + 2x^2 + x + 2.$$

Note that the coefficients are computed in $GF(3)$.

It is not difficult to verify that the polynomials defined on $GF(p)$ is a commutative ring under the polynomial addition and multiplication. Now we define *modular polynomial arithmetic*. Suppose $m(x)$ is a polynomial defined on $GF(p)$. Now for a polynomial $P(x)$ on $GF(p)$, if

$$P(x) = q(x)m(x) + r(x)$$

where the degree of $r(x)$ is lower than $m(x)$, then we write

$$P(x) \equiv r(x) \bmod m(x),$$

and say that $P(x)$ is congruent to $r(x)$ modulo $m(x)$.

All the polynomials defined on $GF(p)$ with polynomial addition and multiplication reduced by $m(x)$ form the modular polynomial arithmetic on $GF(p)$ with modulus $m(x)$. In fact, in that modular polynomial arithmetic, the degree of each polynomial is less than the degree of $m(x)$. The addition is just the polynomial addition over $GF(p)$. The result of the multiplication of two polynomials is reduced by $m(x)$.

For a nonzero polynomial $P(x)$, if there is a polynomial $f(x)$ such that

$$P(x)f(x) \equiv 1 \bmod m(x),$$

then we say that $f(x)$ is the multiplicative inverse of $P(x)$ in the modular polynomial arithmetic and sometimes denote it as $f(x) = P^{-1}(x)$. Obviously, 0 and 1 are the additive identity and multiplicative identity, respectively, in the modular polynomial arithmetic. It is not difficult to check that the modular polynomial arithmetic is a communicative ring.

It can be proved that if $m(x)$ is an irreducible polynomial, then every nonzero element in the modular polynomial arithmetic has multiplication inverse. Therefore it becomes a finite field of order p^n.

Now we can construct the $GF(2^8)$ used in AES. The following irreducible polynomial defined on $\mathbb{Z}_2$ is used to construct the finite field:

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Each element in $GF(2^8)$ can be expressed as a polynomial or a 8-bit binary string. For example the following notations are used to denote the same element of $GF(2^8)$.

$$\begin{array}{l} x^6 + x^4 + x^2 + x + 1 \\ 01010111 \end{array}$$

Suppose there is another element $x^7 + x^4 + x$ (10010010). The addition in that field is simple.

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x^4 + x) = x^7 + x^6 + x^2 + 1.$$

Note that in binary notation, the addition is simply XOR:

$$(01010111) \oplus (10010010) = 11000101.$$

The multiplication in $GF(2^8)$ is a little complicated. But there is still an efficient way to do that. Note that

$$x^8 \equiv x^4 + x^3 + x + 1 \bmod m(x).$$

Suppose an element in $GF(2^8)$ is $P(x) = \sum_{i=0}^{7} b_i x^i$, where $b_i = 0$ or $1, i = 0, 1, \cdots, 7$. Then

$$\begin{aligned} xP(x) &= \sum_{i=0}^{7} b_i x^{i+1} \\ &\equiv \begin{cases} \sum_{i=0}^{6} b_i x^{i+1} & \text{if } b_7 = 0, \\ (\sum_{i=0}^{6} b_i x^{i+1}) + x^4 + x^3 + x + 1 & \text{if } b_7 = 1. \end{cases} \end{aligned}$$

Using the binary notation, this operation can be expressed as:

$$xP(x) \quad = \quad \begin{cases} b_6b_5b_4b_3b_2b_1b_00 & \text{if } b_7 = 0, \\ (b_6b_5b_4b_3b_2b_1b_00) \oplus (00011011) & \text{if } b_7 = 1. \end{cases}$$

So to compute $xP(x)$, we just need to do a shift and an XOR. Repeat i times of these operations we get $x^iP(x)$. Basically, we can use shifts and XORs to do the multiplications in $GF(2^8)$.

Chapter 4

Public Key Encryption

In a conventional cryptosystem, Alice and Bob need a secret channel to distribute a key. This usually can be done by physically delivered from Alice to Bob or a third party decides and sends the key to both Alice and Bob secretly.

Now let us consider the communications at a network. Suppose there are N hosts on the net. Then each pair of users requires a secret key. So the number of keys required by the system is $N(N-1)/2$ and every user needs to store $N-1$ keys. How to distribute and restore these secret keys is a big problem because the number of users in the internet is huge.

One method can be used to reduce the number of keys is to establish a key distribution centre (KDC) on the net. Let each host have a (secret) master key with the KDC. So there are only N keys need to be distributed. When a host i wants to communicate to a host j, the KDC sends a session key to i and j by request. Since the KDC knows the hosts' master keys, the session key can be encrypted by using i and j's master keys. After i and j received the encrypted session key, they can perform the decryption and then use the session key as their shared secret key. However, in this case there are still N secret master keys to be distributed through secret channels and the existence of KDC is not satisfied by privacy. In fact, in many circumstances KDC is not adoptable. That gives us a question: Can we create an encryption system that does not require to distribute secret keys? This is why people invent public key cryptosystems.

4.1 Some Math Facts in Number Theory

Before introduce public-key cryptosystems we need to review some mathematical facts, since public-key systems rely on more mathematics.

The following facts can be found in any number theory text book. We will just list these facts without proofs.

A prime number is a positive integer which only has positive divisors 1 and itself. Any positive integer a can be uniquely written in a product of primes:

$$a = \prod p^{\alpha_p}, \alpha_p \geq 0, p \text{ primes.}$$

We use $\gcd(a, b)$ to denote the greatest common divisor of integers a and b. Then we say that a pair of integers a and b are relatively prime (or coprime) if $\gcd(a, b) = 1$. For an integer n, define Euler's totient function $\phi(n)$ to be the numbers of positive integers which are less than n and relatively prime to n. For examples, $\phi(6) = 2$, $\phi(7) = 6$ and $\phi(12) = 4$. It is easy to see that for a prime p, $\phi(p) = p - 1$ because each positive number which is less then p is coprime with p. The following fact will be useful: If p and q are different primes, then $\phi(pq) = (p - 1)(q - 1) = pq - p - q + 1$. This comes from the fact that the positive integers which are less than pq and not coprime to pq are $1 \cdot p, 2 \cdot p, \cdots, (q - 1) \cdot p, 1 \cdot q, 2 \cdot q, \cdots (p - 1) \cdot q$.

The following theorem is called Euler's Theorem.

Theorem 4.1.1 *Suppose positive integers m and a are relatively prime, then*

$$a^{\phi(m)} \equiv 1 \pmod{m}.$$

Since $\phi(6) = 2$ and $\phi(12) = 4$, we have $5^2 \equiv 1 \pmod 6$ and $7^4 \equiv 1 \pmod{12}$ by Theorem 4.1.1.

If p is a prime, then we mentioned that $\phi(p) = p - 1$. Therefore we have the following Fermat's Theorem from Theorem 4.1.1.

Theorem 4.1.2 *If p is a prime and a is a positive integer, then*

$$a^p \equiv a \pmod{p}.$$

To find the greatest common divisor of two integers r_0 and r_1, we can use the Euclidean algorithm described as follows.

$$\begin{array}{rcll} r_0 & = & q_1 r_1 + r_2, & 0 < r_2 < r_1 \\ r_1 & = & q_2 r_2 + r_3, & 0 < r_3 < r_2 \\ \cdots & & \cdots\cdots & \\ r_{m-2} & = & q_{m-1} r_{m-1} + r_m, & 0 < r_m < r_{m-1} \\ r_{m-1} & = & q_m r_m. & \end{array}$$

Then $\gcd(r_0, r_1) = r_m$. Note that $r_1 > r_2 > r_3 \cdots$, so $r_m \geq 1$ must exist.

$\gcd(r_0, r_1) = r_m$ follows from the fact that $\gcd(r_0, r_1) = \gcd(r_1, r_2) = \cdots = \gcd(r_{m-1}, r_m) = r_m$.

For example, we want to find $\gcd(75, 28)$, then we can compute as follows

$$\begin{array}{rcl} 75 & = & 2 \cdot 28 + 19 \\ 28 & = & 1 \cdot 19 + 9 \\ 19 & = & 2 \cdot 9 + 1 \\ 9 & = & 9 \cdot 1 \end{array}$$

So $\gcd(75, 28) = 1$.

For $b \in \mathbb{Z}_m$, if $\gcd(b, m) = 1$ then b^{-1} (multiplicative inverse) exists. We can use Euclidean algorithm to compute b^{-1} efficiently.

We first need the following theorem which can be proved by induction on j.

Theorem 4.1.3 *For $0 \leq j \leq m$, let $t_0 = 0, t_1 = 1, t_j = t_{j-2} - q_{j-1} t_{j-1} \pmod{r_0}, j \geq 2$, where q_j, r_j are defined as in the Euclidean algorithm. Then*

$$r_j \equiv t_j r_1 \pmod{r_0}.$$

When $\gcd(r_0, r_1) = 1$, we have $r_m = 1$. Therefore from this theorem we have the following corollary.

Corollary 4.1.4 *Suppose* $\gcd(r_0, r_1) = 1$. *Then* $t_m = r_1^{-1} \pmod{r_0}$.

Combine the Euclidean algorithm and the above results, we obtain an extended Euclidean algorithm which can be used to find the inverse in $\mathbb{Z}_m$. In Figure 4.1 we display the algorithm which compute $b^{-1} \pmod{n}$. In the algorithm, $\lfloor x \rfloor$ means the greatest integer which is less than or equal to x.

$n_0 = n$
$b_0 = b$
$t_0 = 0$
$t = 1$
$q = \lfloor \frac{n_0}{b_0} \rfloor$
$r = n_0 - q \times b_0$
while $r > 0$ **do**
    $temp = t_0 - q \times t \bmod n$
    $t_0 = t$
    $t = temp$
    $n_0 = b_0$
    $b_0 = r$
    $q = \lfloor \frac{n_0}{b_0} \rfloor$
    $r = n_0 - q \times b_0$
end do
if $b_0 \neq 1$ **then**
    b has no inverse modulo n
else
    $b^{-1} = t \bmod n$

Figure 4.1: Extended Euclidean algorithm for $b^{-1} \pmod{n}$

Using the Extended Euclidean algorithm, we can find that $28^{-1} \pmod{75} = 67$. The parameters in the algorithm are changed as follows.

n_0	b_0	t_0	t	q	r	$temp$
75	28	0	1	2	19	
28	19	1	73	1	9	73
19	9	73	3	2	1	3
9	1	3	67	9	0	67

4.2 RSA Public-key System

In a public-key encryption system (PKS), the encryption key and the decryption key are different. The encryption key is public so that other people can use this key to encrypt texts. The decryption key is kept in secret. So only the person has the decryption key can decrypt cipher texts.

The basic idea of PKS is that the encryption function $e_K(x)$ is a "one-way" function which means that from e_K it is difficult to find the inverse of e_K (i.e., d_K).

In a PKS, Alice has two keys: one public key and one private key. Alice publishes the public key (and the encryption algorithm) so that anyone can use that key to encrypt messages. The encryption function is a one-way function, so it is difficult to find the decryption method by the knowledge of public key and encryption algorithm. But the inverse of the encryption function can be easily found if the private key is known. So only Alice is able to decrypt the ciphertext.

RSA was invented by Rivest, Shamir and Adleman in 1977. The security of RAS is based on the difficulty of factoring large integers (but it is easy to product integers). The description of RSA is in Figure 4.2

Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$ and define

$$\mathcal{K} = \{(n, p, q, a, b) : n = pq, ab \equiv 1 \bmod \phi(n), p, q \text{ primes}\}.$$

For $K = (n, p, q, a, b)$, define

$$e_K(x) = x^b \bmod n$$

and

$$d_K(y) = y^a \bmod n$$

$(x, y) \in \mathbb{Z}_n$. The values n and b are public, and the values p, q, a are secret.

Figure 4.2: RSA Cryptosystem

In the system the pair (n, b) is the public-key, while (p, q, a) is the private key.

The encryption algorithm only uses the public-key, but the decryption needs the private key. Since $ab \equiv 1 \pmod{\phi(n)}$, we have

$$d_K(e_K(x)) = (x^b)^a = x^{ab} \equiv x \pmod{n}$$

by Euler's theorem.

Note that $\phi(pq) = (p-1)(q-1)$ because p, q are primes. If Oscar knows p, q, then he can easily compute a from $ab \equiv 1 \pmod{(p-1)(q-1)}$ by extended Euclidean algorithm. For this reason, the number of p and q should be very large so that pq is difficult to be factored. Current factoring algorithms are able to factor numbers having up 130 decimal digits. Hence it is recommended that one should choose both p and q to be more than 150 decimal digits (so n will be more than 300 decimal digits). Currently, RSA uses binary version of primes with 512 bits. However, some people recommend to use 1024-bit primes. We will expect to use larger primes when the computer is faster.

Now we consider the implementation of RSA. To establish an RSA system, we need the following algorithms.

1. Generate large primes p and q.
2. Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.
3. Choose a random number b, $0 \leq b \leq \phi(n)$ such that $\gcd(b, \phi(n)) = 1$.
4. Compute $a \equiv b^{-1} \pmod{\phi(n)}$.
5. Compute $x^b \pmod{n}$ and $y^a \pmod{n}$.

Using Euclidean algorithm and its extension, we can get most of the above algorithms except the algorithm of computing $x^b \pmod{n}$ and generating large primes.

To compute $x^b \pmod{n}$, we can use square-and-multiply algorithm. First we write b in its binary version.

$$b = \sum_{i=0}^{l-1} b_i 2^i, \text{ where } b_i = 0 \text{ or } 1.$$

```
z = 1
for  i = l − 1 downto  0 do
        z = z^2 mod n
        if  b_i = 1 then  z = z × x mod n
end do
```

Figure 4.3: The square-and-multiply algorithm

So b is of length l in its binary expression. Then we can use the algorithm displayed in Figure 4.3.

In the square-and-multiply algorithm $l \approx \log_2 b$. So this algorithm performs about $2 \log b$ times multiplication which is much efficient than b times multiplication.

As an example, let us compute $3^{13} \pmod{10}$. Since the binary version of 13 is 1101, we have

$$\begin{aligned} 3^{13} &\equiv 3^{2^3+2^2+1} \pmod{10} \\ &= ((3^2 \cdot 3)^2)^2 \cdot 3 \pmod{10} \\ &= (7^2)^2 \cdot 3 \pmod{10} \\ &= 9^2 \cdot 3 \pmod{10} \\ &= 1 \cdot 3 \pmod{10} \\ &= 3 \pmod{10}. \end{aligned}$$

The square-and-multiply algorithm works in the same way.

Next we consider how to find large primes. We will use some Monte Carlo algorithm to do that.

Definition 4.2.1 *A yes-biased Monte Carlo algorithm is a probabilistic algorithm for a decision problem in which a "yes" answer is always correct, but a "no" answer may be incorrect. We say that a Monte Carlo algorithm has error provability ϵ, if the algorithm will give the incorrect answer "no" with probability at most ϵ.*

The idea to generate a large prime is to choose a random large number first and then use some Monte Carlo algorithm to test whether that number

is a prime. There are several Monte Carlo algorithms can be used. Here we just introduce Miller-Rabin algorithm which is described in Figure 4.4

write $n-1 = 2^k m$, where m is odd
choose a random integer $a, 1 \leq a \leq n-1$
compute $b = a^m \bmod n$
if $b \equiv 1 \bmod n$ **then**
    answer "n is prime" and **quit**
for $i = 0$ **to** $k-1$ **do**
    if $b \equiv -1 \bmod n$ **then**
        answer "n is prime" and **quit**
    else
        $b = b^2 \bmod n$
answer "n is composite"

Figure 4.4: Miller-Rabin primality test algorithm

Miller-Rabin algorithm is a "yes-biased" composite test for odd numbers Monte Carlo algorithm. So if the algorithm answers "n is composite", then n must be a composite. This can be proved as follows.

Suppose the answer of the program is "n is composite". Then from the program we know that $a^m \not\equiv 1 \pmod{n}$ and $a^{2^i m} \not\equiv -1 \pmod{n}, 0 \leq i \leq k-1$. However, if n is a prime then $a^{n-1} = a^{2^k m} \equiv 1 \pmod{n}$ by Euler's Theorem. Therefore $(a^{2^{k-1}m} - 1)(a^{2^{k-1}m} + 1) \equiv 0 \pmod{n}$. Since n is a prime and $a^{2^{k-1}m} \not\equiv -1 \pmod{n}$, we must have $(a^{2^{k-1}m} - 1) \equiv 0 \pmod{n}$. Following this way, finally we will get $a^m - 1 \equiv 0 \pmod{n}$ which is a contradiction.

Using some mathematics, it can be proved that the error probability of this algorithm is at most 1/4. So when the algorithm answers "n is a prime", there is 1/4 probability that n actually is a composite. To reduce the error probability, we can repeat the algorithm several times. Each time the algorithm uses different random number a. For example, we can choose t random values of a and run the algorithm t times. If the answer is always "n is a prime", then the probability that n is a composite is $(1/4)^t$. Let t be

large enough, then the error probability will be very small.

Lets look at a small example to illustrate the RSA system. Suppose Bob chooses primes $p = 101$ and $q = 113$. Then $n = 11413$ and $\phi(n) = 11200$. Bob chooses a random number $b = 3533$ which is coprime to 11200. Then $b^{-1} \bmod 11200 = 6597$. Bob publishes $n = 11413$ and $b = 3533$. Suppose Alice wants to send Bob a plaintext 9726. She will compute

$$9725^{3533} \bmod 11413 = 5761.$$

and send 5761 to Bob. When Bob receives the ciphertext 5761, he uses his secret key $a = 6597$ to decrypt

$$5761^{6597} \bmod 11413 = 9726.$$

4.3 ElGamal Cryptosystem

Another popular public-key system is ElGamal Cryptosystem which is based on the discrete logarithm problem. Sometimes this cryptosystem is also called Diffie-Hellman type cryptosystem, because this system is based on some idea from Diffie-Hellman key exchange scheme which we will discuss later.

Suppose p is a prime. Then we can find some $\alpha \in \mathbb{Z}_p$ such that each number $\beta \in \mathbb{Z}_p^* (= \mathbb{Z}_p \backslash \{0\})$ can be written as $\beta \equiv \alpha^a \pmod{p}$ for some a, α is called a primitive element of $\mathbb{Z}_p$ (or equivalently, $GF(p)$). For example, if $p = 7$, then $\alpha = 3$ is a primitive element of $\mathbb{Z}_7$. This is because in $\mathbb{Z}_7$ we have:

$$1 = 3^6, 2 = 3^2, 3 = 3^1, 4 = 3^4, 5 = 3^5, 6 = 3^3.$$

The discrete logarithm problem is that : In the equation

$$\beta \equiv \alpha^a \pmod{p} \ (\beta \neq 1),$$

find the value of a if β, α and p are known.

There are several methods to attack the discrete logarithm problem. To thwart known attacks, p should be at least 300 decimal digits, and $p - 1$ should have at least one large prime factor.

The ElGamal system is described in Figure 4.5.

The correctness of the system is easy to check:

Let p be a prime such that the discrete logarithm problem in $\mathbb{Z}_p$ is intractable and let $\alpha \in \mathbb{Z}_p^*$ be a primitive element. Let $\mathcal{P} = \mathbb{Z}_p^*, \mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and

$$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \bmod p\}.$$

The values of p, α and β are public and a is secrete.
For $K = (p, \alpha, a, \beta) \in \mathcal{K}$ and for a secret random number $k \in \mathbb{Z}_{p-1}$ define

$$e_K(x, k) = (y_1, y_2),$$

where $y_1 = \alpha^k \pmod p, y_2 = x\beta^k \pmod p$.
For $y_1, y_2 \in \mathbb{Z}_p^*$, define

$$d_K(y_1, y_2) = y_2(y_1^a)^{-1} \pmod p.$$

Figure 4.5: ElGamal Cryptosystem

$$\begin{aligned} d_K(y_1, y_2) &= y_2(y_1^a)^{-1} \\ &= x\beta^k(\alpha^{ak})^{-1} \\ &\equiv x \pmod p \end{aligned}$$

The public key of ElGamal cryptosystem contains the values of p, α and β, and the private key of the system is the value of a. Note that the ElGamal system is non-deterministic, since the ciphertext depends not only on plaintext and the public key, but also on the random number k. A plaintext may have different ciphertexts, which is good for the security of the system. On the other hand, the length of a ciphertext is twice of the length of its plaintext, which is not good for the network traffic.

When using the ElGamal system, the value of k should be kept in secret. Otherwise, x can be revealed from k and y_2 by computing $x = y_2\beta^{-k} \pmod p$.

To implement the system, we need the following algorithms:

1. Find a large prime p such that $p-1$ has at least a large prime factor.
2. Find a primitive element α of $\mathbb{Z}_p$.
3. Choose a random number $k \in \mathbb{Z}_{p-1}$.
4. Compute $\alpha^k \pmod{p}$ and $x\beta^k \pmod{p}$.
5. Compute $(y_1^\alpha)^{-1} \pmod{p}$.

We already have some method to find a large prime. So we can first find a large prime q. Then we try some random number r such that $qr+1=p$ is a prime. To find a primitive element, we can choose a random α and let $g=\alpha^r$. If $g \not\equiv 1 \pmod{p}$, then we use α in the ElGamal system. Otherwise try another value of α. In fact, α is not necessary a primitive element of $\mathbb{Z}_p^*$ by using this method. But it is guaranteed that α has a order not less than q. To ensure that the α is a primitive element of $\mathbb{Z}_p^*$, one needs to check $\alpha^{tq} \not\equiv 1 \pmod{p}$ for each factor t of r, where $t \neq r$. The other algorithms can be obtained from the previous section.

It is easy to know that if we can solve the discrete logarithm problem, then the ElGamal system is broken.

Let's see an example. Suppose $p=2579$ and $\alpha=2$. α is a primitive element in $\mathbb{Z}_p$. Let $a=765$, so

$$\beta = 2^{765} \bmod 2579 = 949.$$

Suppose Alice wants to send the message 1299 to Bob. She chooses a random number $k=853$. Then she computes

$$y_1 = 2^{853} \bmod 2579 = 435,$$

and

$$y_2 = 1299 \times 949^{853} \bmod 2579 = 2396.$$

When Bob receives the ciphertext $(435, 2396)$, he computes

$$x = 2396 \times (435^{765})^{-1} \bmod 2579 = 1299.$$

ElGamal system can be established in any cyclic group. In this section, we used $\mathbb{Z}_p^*$, a cyclic group of order $p-1$, for the system. In practice, other cyclic groups are used. For examples, for a prime p, $\mathbb{Z}_{p^n}$, n a positive integer and $\mathbb{Z}_q$, q a prime factor of $p-1$ are used for different systems.

4.4 Other Public-key Cryptosystems

There are other public-key cryptosystems. Most of them are based on the difficulty of factoring problem or discrete logarithm problem. In this section, we just list a few examples briefly.

The Rabin Cryptosystem also uses a number $n = pq$, where p, q are large primes such that $p, q \equiv 3 \pmod 4$. The encryption function is

$$e_K(x) = x^2 \bmod n$$

and the decryption function is

$$d_K(y) = \sqrt{y} \bmod n.$$

The security of the Rabin system is based on some number theory facts. In general, to find the value $\sqrt{y} \bmod n$ is very difficult unless n is a prime. However, if p and q are known, then there is some easy way to compute $\sqrt{y} \bmod n$ for $n = pq$.

There are some variant forms of RSA system. For examples, some people suggested to use $n = pqr$, a production of three primes. Some people suggested to use $n = p^2q$, where p, q are primes, etc.

The Elliptic curve cryptosystem uses the idea from the ElGamal system. In the ElGamal system, each element in $\mathbb{Z}_p^*$ can be written as α^i for some i. $\mathbb{Z}_p^*$ is a cyclic group generated by α. The Elliptic curve system uses a group different from $\mathbb{Z}_p^*$. The system uses a group of the solutions of $y^2 \equiv x^3 + ax + b \pmod p$ instead of the group $\mathbb{Z}_p^*$. Elliptic curve systems depend on more mathematics. The decryption in an Elliptic curve system is more efficient than that in an RSA system.

Another interesting public-key system is NTRU which uses polynomial rings and depends on some problems in a mathematical structure called lattices. There is some relationship between factoring problem and discrete logarithm problem. However, lattice problem seems to be different from these problems.

4.5 Public-key Systems and Secret-key Systems

We should indicate here that in general a public key cryptosystem is much slower than a secret key cryptosystem. For example, the RSA system is about

1500 times slower than DES. So it is not practical to use public key system for large plaintext. Usually, we will use public key system to send a key of some secret key cryptosystem and then use the secret key cryptosystem to encrypt the plaintexts. So one of the important applications of public-key systems is distribute secret keys, which we will discuss later.

We can use the following simple example to explain the combination of a public-key system and a secret-key system. Suppose Bob has an RSA system. So he published his public key K_{pub} and stored his private key K_{pri} in a secure place. Now Alice wants to communicate with Bob. She first chooses a random key K for AES system. Then she send Bob the value of $e_{K_{pub}}(K)$. Since only Bob has the key K_{pri}, he can obtain K securely. After that Alice and Bob can use AES with common key K to encrypt their communication.

For the security of a public key cryptosystem, we need to consider the chosen plaintext attacks. Since the encryption function is public in the system, Oscar can choose any plaintext and obtain the according ciphertext.

All the existing public key cryptosystems are based on some difficult mathematical problems. Usually we call such a system a computational secure cryptosystem, because we don't know efficient algorithms to solve these problems. If some efficient algorithm is invented in the future, then the system will be no longer secure.

Chapter 5

Information Authentication

Message authentication is one of the important security problems in network security. Suppose Bob received some message from Alice over internet. How can he believe that the message is really sent by Alice? How can he be sure that the message has not been modified by Oscar? These are basic problems for information authentication.

Information authentication on a network needs new techniques and algorithms. In this chapter we discuss digital signature schemes, massage authentication and hash functions which are among most important authentication techniques.

5.1 Signature Schemes

Handwritten signatures are used to sign paper documents for a long history. However, for an electronic file, we cannot just put a name on the file since the name can be easily forged by other people. Note that if there are only two parties Alice and Bob in the scheme and they share a common secret key, then Alice can use the key to encrypt the whole file and Bob will know that the file is from Alice since only he and Alice know the key. However, Bob cannot know whether the cipher text is complete. Oscar might cut part of the cipher text and Bob still can decrypt the cipher text and get wrong information. And in many cases, a signature needs to be checked by several people. Also as we discussed before that it is not easy to distribute secret keys.

The idea of public key system therefore is used for signatures. A signature

scheme is used to create an electronic signature such that any other party cannot forge the signature while they can verify the truth of the signature. So a signature scheme consists of two components, a signing algorithm and a verification algorithm.

In a signature scheme when Alice wants to send Bob a message x, she will send a pair $(x, sig(x))$ to Bob. Bob can check if $sig(x)$ is correct and decides to accept x or not. Oscar wants to construct a message (y, z) and hopes that Bob will accept the message y as sent by Alice.

We give a formal definition of signature scheme as follows.

Definition 5.1.1 *A signature scheme is a five-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$, where the following conditions are satisfied:*

1. *$\mathcal{P}$ is a finite set of possible messages.*
2. *$\mathcal{A}$ is a finite set of possible signatures.*
3. *$\mathcal{K}$, the key space, is a finite set of possible keys.*
4. *For each key $K \in \mathcal{K}$, there is a signature algorithm $sig_K \in \mathcal{S}$ and a corresponding verification algorithm $ver_K \in \mathcal{V}$. Each $sig_K : \mathcal{P} \mapsto \mathcal{A}$ and $ver_K : \mathcal{P} \times \mathcal{A} \mapsto \{true, false\}$ are functions such that the following equation is satisfied for every message $x \in \mathcal{P}$ and for every signature $y \in \mathcal{A}$:*

$$ver_K(x, y) = \begin{cases} true & if\ y = sig_K(x) \\ false & if\ y \neq sig_K(x). \end{cases}$$

In general, the key $K \in \mathcal{K}$ includes a pair of keys. One is a public key and the other is a secret key. The security of a signature scheme requires that the $ver_K()$ (the verification algorithm and the public key) is public known and for any $x \in \mathcal{P}$, it is difficult to forge a $sig_K(x)$ without knowing the secret key.

RSA signature scheme

The RSA signature scheme is based on the RSA public-key system. The scheme is described in Figure 5.1.

The RSA signature scheme actually uses the decryption function of the RSA as sig_K and uses encryption function of RSA as ver_K. This is because

Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$ and

$$\mathcal{K} = \{(n, p, q, a, b) : n = pq, p, q \text{ prime}, ab \equiv 1 \pmod{\phi(n)}\}.$$

The values n and b are public and the values of p, q, a are secret. For $K = (n, p, q, a, b)$, define

$$sig_K(x) = x^a \bmod n$$

and

$$ver_K(x, y) = true \Leftrightarrow x \equiv y^b \pmod{n}$$

$(x, y \in \mathbb{Z}_n)$.

Figure 5.1: RSA Signature Scheme

the encryption and decryption functions of the RSA is "symmetric". It is easy to prove the correctness of the RSA signature scheme.

Now we consider some possible attacks for a signature scheme. The security of a signature scheme is based on how to prevent against forging a signature. For the RAS signature scheme, Oscar might forge Alice's signature as follows. Since ver_K is public, Oscar can compute $x = ver_K(y) = y^b \pmod{n}$ for some value of y and send (x, y) to Bob. In that case, Bob will accept the signature as Alice's signature since the verification algorithm will output *true*. However, it is difficult for Oscar to choose the value of y such that $x = y^b \pmod{n}$ is meaningful. In fact, for given x, b, n, it is difficult to find y such that $x = y^b \pmod{n}$. Therefore that kind of attacks is not very useful. However, this attack tells us that signing a random string by RSA system is meaningless. Another possible attack comes from a property of RSA signature scheme that $sig_K(x) \cdot sig_K(y) = sig_K(xy)$. So if Oscar observed $(x, sig_K(x))$ and $(y, sig_K(y))$, he can then sends $(xy, sig_K(x) \cdot sig_K(y))$ to Bob and Bob will accept it if xy is meaningful.

For a signature scheme, Oscar can always use the following attack. Oscar requests Alice to sign several messages $(x_1, y_1), (x_2, y_2), \cdots, (x_t, y_t)$, where

$y_i = sig_K(x_i), i = 1, 2, \cdots, t$. Then Oscar sends a subset of the pairs $(x_{j_1}, y_{j_1}), (x_{j_2}, y_{j_2}), \cdots, (x_{j_s}, y_{j_s})$ to Bob. Bob will accept the message since the verification algorithm will always output *true*. By choosing the values of $x_{j_1}, x_{j_2}, \cdots, x_{j_s}$, Oscar can send some wrong information to Bob and Bob will think that the information is from Alice. To avoid that kind of attacks, one method is to use hash functions or MAC which we will discuss in the next section. Another method we can use is to encrypt the messages x_i.

Note that the message x is not encrypted in the signature scheme. To keep the message secret, we have to encrypt x. In this case, we recommend signing message x before encryption to avoid a possible attack from Oscar. In fact, suppose Alice first encrypts the message x to get the ciphertext y and then signs y using her scheme: $z = sig_{Alice}(y)$. (y, z) is then sent to Bob. If Oscar obtains the pair (y, z) in the middle way, then he can change z with $z' = sig_{Oscar}(y)$ and transmit (y, z') to Bob. Bob may verify the message and think that the plaintext x originated with Oscar. So usually Alice will encrypt the pair $(x, sig_{Alice}(x))$ and sends the cipher text to Bob. When Bob receives the cipher text, he first decrypts it to obtain the pair $(x, sig_{Alice}(x))$, then verifies the signature.

ElGamal signature scheme

The ElGamal signature scheme is also widely used, which is described in Figure 5.2. The security of this signature scheme is based on the difficulty of discrete logarithm problem.

The correctness of ElGamal signature scheme can be proved as follows.

$$\begin{aligned} \beta^{\gamma}\gamma^{\delta} &\equiv \alpha^{a\gamma}\alpha^{k\delta} \\ &= \alpha^{a\gamma+k\delta} \\ &\equiv \alpha^{x} \pmod{p}, \end{aligned}$$

where we use the fact that

$$a\gamma + k\delta \equiv x \pmod{p-1}$$

and hence

$$\alpha^{a\gamma+k\delta} \equiv \alpha^{x} \pmod{p}.$$

Similar to ElGamal public-key encryption, the random value of k should be kept in secret. If Oscar knows k, then he can find the secret key a easily:

$$a = (x - k\delta)\gamma^{-1} \pmod{p-1}.$$

Let p be a prime such that the discrete logarithm problem in $\mathbb{Z}_p$ is intractable and let $\alpha \in \mathbb{Z}_p^*$ be a primitive element. Let $\mathcal{P} = \mathbb{Z}_p^*, \mathcal{A} = \mathbb{Z}_p^* \times \mathbb{Z}_{p-1}$, and define

$$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \bmod p\}.$$

The values of p, α and β are public and a is secret.
For $K = (p, \alpha, a, \beta) \in \mathcal{K}$ and for a secret random number $k \in \mathbb{Z}_{p-1}^*$ define

$$sig_K(x, k) = (\gamma, \delta),$$

where $\gamma = \alpha^k \pmod p$ and

$$\delta = (x - a\gamma)k^{-1} \pmod{p-1}.$$

For $x, \gamma \in \mathbb{Z}_p^*$ and $\delta \in \mathbb{Z}_{p-1}$, define

$$ver_K(x, \gamma, \delta) = true \Leftrightarrow \beta^\gamma \gamma^\delta \equiv \alpha^x \pmod p.$$

Figure 5.2: ElGamal signature scheme

Moreover, we cannot use a same k to sign two different messages. If k is used to sign two different messages x_1 and x_2, then Oscar knows

$$\beta^\gamma \gamma^{\delta_1} \equiv \alpha^{x_1} \pmod p$$

and

$$\beta^\gamma \gamma^{\delta_2} \equiv \alpha^{x_2} \pmod p.$$

So he can compute

$$\begin{aligned} \alpha^{x_1 - x_2} &\equiv \gamma^{\delta_1 - \delta_2} \\ &\equiv \alpha^{k(\delta_1 - \delta_2)} \pmod p. \end{aligned}$$

Therefore

$$x_1 - x_2 \equiv k(\delta_1 - \delta_2) \pmod{p-1}.$$

In this way, Oscar can find k by using some knowledge of number theory and then find the secret key a.

Another attack Oscar can perform is similar to an attack to RSA signature scheme discussed before. Oscar selects two numbers u, v such that $\gcd(v, p-1) = 1$. Let

$$\gamma = \alpha^u \beta^v \pmod{p},\ \delta = -\gamma v^{-1} \pmod{p-1}.$$

Then it is easy to check that (γ, δ) is a valid signature of $x = u\delta$, since $\beta^\gamma \gamma^\delta \equiv \alpha^{u\delta}$. Although, it is difficult for Oscar to find a useful value of x.

Digital Signature Standard

The Digital Signature Standard (or DSS) was published in 1994, which was modified from ElGamal signature scheme (FIPS PUB 186). We describe the DSS in Figure 5.3.

The most important modification of DSS from ElGamal signature scheme is the use of $\mathbb{Z}_q$, a subgroup of $\mathbb{Z}_p^*$. In this way, the length of signature is reduced to 320 bits (while that is at least 1024 bits t in ElGamal signature scheme). On the other hand, the main computations are still in 512-bit modulo as same as that in ElGamal scheme.

Let us use a small example to illustrate the DSS.

Example 5.1.2 Let $q = 29$ and $p = 29 \cdot 8 + 1 = 233$. Let $h = 3$ so

$$\alpha = 3^8 = 6561 \equiv 37 \pmod{233}.$$

Therefore $\mathcal{P} = \mathbb{Z}_{233}^*, \mathcal{A} = \mathbb{Z}_{29} \times \mathbb{Z}_{29}$. Let $a = 2$ and

$$\beta = \alpha^a = 37^2 \equiv 204 \bmod 223.$$

p, q, α, β are public and $a = 2$ is secret.

Suppose the plaintext $x = 10$. Let the random number $k = 4$ $(1 \le k \le 28)$. Then the signature $sig_K(10, 4) = (\gamma, \delta)$, where

$$\begin{aligned} \gamma &= (37^4 \bmod 233) \bmod 29 \\ &\equiv (204^2 \bmod 233) \bmod 29 \\ &\equiv 142 \bmod 29 \\ &= 26 \end{aligned}$$

Let p be a prime number such that $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L a multiple of 64, and let q be a 160-bit prime that divides $p-1$. Let $\alpha = h^{(p-1)/q} \pmod{p}$, where h is any integer with $1 < h < p-1$ such that $h^{(p-1)/q} \pmod{p} > 1$. Let $\mathcal{P} = \mathbb{Z}_p^*, \mathcal{A} = \mathbb{Z}_q \times \mathbb{Z}_q$, and define

$$\mathcal{K} = \{(p, q, \alpha, a, \beta) : \beta \equiv \alpha^a \bmod p\}.$$

The values of p, q, α and β are public and a is secret.
For $K = (p, q, \alpha, a, \beta) \in \mathcal{K}$ and for a secret random number $k, 1 \leq k \leq q-1$, define

$$sig_K(x, k) = (\gamma, \delta),$$

where

$$\gamma = (\alpha^k \bmod p) \bmod q$$

and

$$\delta = (x + a\gamma)k^{-1} \bmod q.$$

For $x \in \mathbb{Z}_p^*$ and $\gamma, \delta \in \mathbb{Z}_q$, verification is done by performing the following computations:

$$\begin{aligned} e_1 &= x\delta^{-1} \bmod q \\ e_2 &= \gamma\delta^{-1} \bmod q \end{aligned}$$

and

$$ver_K(x, \gamma, \delta) = true \Leftrightarrow (\alpha^{e_1}\beta^{e_2} \bmod p) \bmod q = \gamma.$$

Figure 5.3: Digital Signature Standard

and

$$\begin{aligned}\delta &\equiv (10 + 2 \cdot 26) \cdot 4^{-1} \bmod 29 \\ &\equiv 4 \cdot 4^{-1} \bmod 29 \\ &= 1.\end{aligned}$$

So for $x = 10$, the signature is $(26, 1)$.

To compute $ver_K(x, \gamma, \delta) = ver_K(10, 26, 1)$, we have

$$\begin{aligned}e_1 &\equiv 10 \cdot 1^{-1} \bmod 29 = 10 \\ e_2 &\equiv 26 \cdot 1^{-1} \bmod 29 = 26\end{aligned}$$

and

$$\begin{aligned}(37^{10} \cdot 204^{26} \bmod 233) \bmod 29 &\equiv (74 \cdot 46 \bmod 233) \bmod 29 \\ &\equiv 142 \bmod 29 \\ &= 26 \\ &= \gamma\end{aligned}$$

So the verification algorithm will output *true*.

There are other signature schemes. Basically, from a public key cryptosystem, we can find a signature scheme. One widely used signature scheme is Elliptic Curve Digital Signature Algorithm (ECDSA) which is approved as FIPS 186-2 in 2000.

5.2 Message Authentication and Hash Functions

A signature scheme can only be used to sign a short message. As we mentioned in previous section, if a key is used to sign several messages, then Oscar might select and rearrange the messages to cheat Alice. Another problem is that the size of the signature is equal to or larger than the message that will cause traffic problem for a network. Therefore we need some method to authenticate long messages. There are basically two methods to authenticate messages. One method uses hash functions and signature schemes. An

alternative authentication technique involves the use of a secret key called message authentication code (or MAC).

Suppose Alice and Bob shared a common secret key K. We need some function $C_K(x)$ which depends on the secret key K. We also hope that the input x of the function can be very large but the output of the function is small. When Alice wants to send a message x to bob, she first computes $C_K(x)$ and then sends $(x, C_K(x))$ to Bob. Usually, the size of $C_K(x)$ is much smaller than x. Since Bob knows K, he can compute $y = C_K(x)$ after he received x and check whether y is the same value of $C_K(x)$ he received. The formal definition of a MAC is as follows.

Definition 5.2.1 *A MAC is a four-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{H})$, where the following conditions are satisfied:*

1. *$\mathcal{P}$ is a set of possible messages.*

2. *$\mathcal{A}$ is a finite set of possible authentication tags.*

3. *$\mathcal{K}$, the key space, is a finite set of possible keys.*

4. *For any $K \in \mathcal{K}$ and $C \in \mathcal{H}$, $C_K : \mathcal{P} \mapsto \mathcal{A}$.*

Note that in the above definition, the set $\mathcal{P}$ can be a infinite set. A MAC function $C_K()$ is similar to an encryption function, but it needs not be reversible (do not need to decrypt). This property makes it possible that the size of $C_K(x)$ is much smaller than that of x.

A MAC function is not a signature. Since both Alice and Bob know the secret key, they can forge the signature each other. On the other hand, only Bob can verify the correctness of the message in this case while every one can do the verification in signature scheme.

A MAC function should have the property: If Oscar knows x and $C_K(x)$, it should be computational infeasible to construct a message x' such that $C_K(x') = C_K(x)$. This is called collision-free property. We will discuss collision-free property a little bit later when we consider hash functions.

As an example of MAC, we describe the Data Authentication Algorithm which is based on DES (FIPS PUB 113). Suppose the message is x. First we divide the message into groups of 64-bit blocks: $x = x_1 x_2 \cdots x_N$ where x_i is 64-bit string, $1 \leq i \leq N$. The algorithm is similar to the CBC mode of

DES. Suppose e_K is the DES encryption with the secret key K. Let

$$\begin{aligned} O_1 &= e_K(x_1) \\ O_2 &= e_K(x_2 \oplus O_1) \\ O_3 &= e_K(x_3 \oplus O_2) \\ &\vdots \\ O_N &= e_K(x_N \oplus O_{N-1}). \end{aligned}$$

Then O_N is the result of the MAC, i.e., $C_K(x) = O_N$.

The length of the input message of this MAC function can be anything bigger than 64 bits (with some padding bits if necessary). The output of the function is always 64-bit.

Another common technique used in authentication is applying hash functions. We already noted that a signature scheme is not convenient for signing a large message. If we break the message into chunks first and then sign each piece of the chunks, Oscar might remove some chunks from the sequence and thus change the message. Hash functions can be used to prevent against such attacks.

Definition 5.2.2 *A hash family is a triple $(\mathcal{P}, \mathcal{D}, \mathcal{H})$, where the following conditions are satisfied:*

1. *$\mathcal{P}$ is a set of possible messages.*
2. *$\mathcal{D}$ is a finite set of possible message digests.*
3. *For any $h \in \mathcal{H}$, $h : \mathcal{P} \mapsto \mathcal{D}$.*

A hash function will take a message of arbitrary length and produce a message digest of a specified size (for example, 160-bit). A hash function does not use any keys. So anyone can compute a hash value provided the hash algorithm is public. One method using a hash function to authenticate a message is as follows. For a large message x, we can first use a hash function $h()$ to get a digest: $z = h(x)$. Then the digest is signed using some signature scheme: $\gamma = sig_K(z)$. The pair (x, γ) is the signed message. Since hash function is published, anyone can compute z from x and check the correctness of the signature.

A MAC function or a hash function h should satisfy some properties for the security reasons. The following properties can be considered.

- Weakly collision-free: Given a message x, it is computationally infeasible to find a message $x' \neq x$ such that $h(x') = h(x)$. This property can be used to prevent Oscar from forging a signature (Oscar sends $(x', sig(h(x))$ to Bob and Bob believes x').

- One-way property: Given a message digest z, it is computationally infeasible to find a message x such that $h(x) = z$. Since to forge a signature of a random value is easier than a signature of a meaningful message, this property can used to prevent forging a signature.

- Strong collision-free: It is computationally infeasible to find messages x and x' such that $h(x') = h(x)$.

It can be proved that strong collision-free implies one-way property and weak collision property.

The main difference between a MAC and a hash function is that a hash function does not use a secret key. Since to establish a secret key through internet is not easy, hash function has some advantages.

We can use a block cipher to construct a hash function as follows. Suppose the encryption function is $e_K()$ and the message is x. Write the message as blocks of fixed size: $x = x_1 x_2 \cdots x_N$. Then we use each block as a key of encryption. First let h_0 be some initial value. Then compute

$$\begin{aligned} h_1 &= e_{x_1}(h_0) \\ h_2 &= e_{x_2}(h_1) \\ &\vdots \\ h_N &= e_{x_N}(h_{N-1}). \end{aligned}$$

Define $h(x) = h_N$. This method is called a block chaining technique. However, this method is not secure. There are some methods to attack the function. There are several ways to improve the security of the function. For example, let

$$h_i = e_{h_{i-1}}(x_i) \oplus x_i \oplus h_{i-1}.$$

A hash function created by a block chaining technique and a block cipher is not very efficient. A block cipher is usually very complicated and the encryption takes time. So researchers tried to find more efficient hash functions.

Since the input of the hash function is large and the output is small, a hash function is many-to-one, or even infinite-to-one function. In general, to construct a collision resistance hash function (strong collision free) is not easy.

MD5

MD5 (Message Digest, RFC 1321) is one of the common used hash functions, which was developed by Rivest. MD5 can be described as follows.

- The first step of MD5 is appending some padding bits to the message so that it is congruent to 448 modulo 512. Padding is always added, even if the message is already 448 modulo 512. The size of padding bits is between 1 to 512. The padding starts with a bit 1 followed by desired number of bit 0's.

- Then a 64-bit representation of the length in bits of the original message is appended. If the length of the message is more than 2^{64}, then use the length modulo 2^{64}. In this way, the message becomes an integer multiple of 512 bits in length.

- A 128-bit buffer (four 32-bit registers) is used to hold intermediate and final results of the hash function. The buffer is initialized to four 32-bit integers.

- Then each 512-bit block of the message is processed to a compression function which consists of four rounds of hashing. All the operations in the compression function are very efficient. Basically, the function uses bitwise "and", "or", "XOR", complement, circular left shift and integer addition modulo 2^{32}. The function also uses a table of values from sine function (abs(sin(i))). The four rounds have a similar structure, but each uses a different primitive logical function. In each round, by inputting the current 512-bit block and the 128-bit buffer value, the contents of the buffer is updated. Each round consists of 16 steps. The output of the fourth round is added to the input to the first round. The addition is done for four words using modulo 2^{32}.

- The output of MD5 is 128-bit digest.

MD5 is modified from MD4 which used three rounds.

SHA

The secure hash algorithm (SHA) was developed by the NIST, along with NSA. A revised version referred as SHA-1 was published in 1995 (FIPS PUB 180-1). SHA-1 is also derived from MD4 and quite similar to MD5. However, the input of SHA-1 is a message of any length $< 2^{64}$ bits and the output is a 160-bit message digest. The Secure Hash Standard (SHS) suggested use SHA-1 to produce a message digest and then use Digital Signature Standard (specified in DSS).

SHA-1 uses a similar method as in MD5 to append some padding bits to the message and a 64-bit integer indicating the length of the original message, so that the message can be written as $M_1 M_2 \cdots M_n$, each M_i is a 16 words (512 bits). Since the digest of SHA-1 is 160 bits, a 160-bit buffer is used (five 32-bit registers).

A sequence of logical functions $f_0, f_1, ..., f_{79}$ is used in the SHA-1. Each $f_t, 0 \leq t \leq 79$, operates on three 32-bit words B, C, D and produces a 32-bit word as output.

SHA-1 uses the following operations:

$X \wedge Y$	bitwise "and " of X and Y
$X \vee Y$	bitwise "or " of X and Y
$X \oplus Y$	bitwise "XOR" of X and Y
$\neg X$	bitwise complement of X
$X + Y$	integer addition modulo 2^{32}
$S^t(X)$	circular left shift of X by t positions

A function $f_t(B, C, D)$ is defined as follows: for words B, C, D,

$$\begin{aligned} f_t(B,C,D) &= (B \wedge C) \vee ((\neg B) \wedge D), (0 \leq t \leq 19) \\ f_t(B,C,D) &= B \oplus C \oplus D, (20 \leq t \leq 39) \\ f_t(B,C,D) &= (B \wedge C) \vee (B \wedge D) \vee (C \wedge D), (40 \leq t \leq 59) \\ f_t(B,C,D) &= B \oplus C \oplus D, (60 \leq t \leq 79). \end{aligned}$$

A sequence of constant words $K_0, K_1, \cdots, K_{79}$ is used in the SHA-1. In hex these are given by

$$K_t = 5A827999, (0 \leq t \leq 19)$$

$$\begin{aligned} K_t &= 6ED9EBA1, (20 \leq t \leq 39) \\ K_t &= 8F1BBCDC, (40 \leq t \leq 59) \\ K_t &= CA62C1D6, (60 \leq t \leq 79). \end{aligned}$$

The message digest is computed using the final padded message. The computation uses two buffers, each consisting of five 32-bit words, and a sequence of eighty 32-bit words. The words of the first 5-word buffer are labelled A, B, C, D, E. The words of the second 5-word buffer are labelled H_0, H_1, H_2, H_3, H_4. The words of the 80-word sequence are labelled $W_0, W_1, ..., W_{79}$. A single word buffer $TEMP$ is also employed.

To generate the message digest, the 16-word blocks $M_1, M_2, ..., M_n$ are processed in order. The processing of each M_i involves 80 steps.

Before processing any blocks, the H_i are initialized as follows: in hex,

$$\begin{aligned} H_0 &= 67452301 \\ H_1 &= EFCDAB89 \\ H_2 &= 98BADCFE \\ H_3 &= 10325476 \\ H_4 &= C3D2E1F0. \end{aligned}$$

Now $M_1, M_2, ..., M_n$ are processed. To process M_i, we proceed as follows:

- Divide M_i into 16 words $W_0, W_1, \cdots, W_{15}$, where W_0 is the left-most word.
- For $t = 16$ to 79 let $W_t = S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$.
- Let $A = H_0, B = H_1, C = H_2, D = H_3, E = H_4$.
- For $t = 0$ to 79 do

$$\begin{aligned} TEMP &= S^5(A) + f_t(B, C, D) + E + W_t + K_t; \\ E &= D; \\ D &= C; \\ C &= S^{30}(B); \\ B &= A; \\ A &= TEMP; \end{aligned}$$

- Let $H_0 = H_0+A, H_1 = H_1+B, H_2 = H_2+C, H_3 = H_3+D, H_4 = H_4+E$.

After processing M_n, the message digest is the 160-bit string represented by the 5 words

$$H_0H_1H_2H_3H_4.$$

In 2001, NIST announced that a draft version of FIPS 180-2 was available for public comments and review. This proposed standard includes SHA-1 as well as three new hash functions, which are named SHA-256, SHA-384 and SHA-512. The suffixes "256", "384" and "512" refer to the sizes of the message digests. In 2002, NIST added SHA-224 into FIPS 180-2.

It is not difficult to see that the longer of the size of digest the securer the hash function. For example, if the message digest is of 64 bits, then we can always find a collision from any $2^{64} + 1$ different messages. However, there is a more efficient attack called *birthday attack*. The name of the attack is from the birthday paradox: The probability that at least 2 people in a room of 23 people have the same birthday is more than 0.5 (≈ 0.507). In general, it can be proved that if a random variable that is an integer with uniform distribution between 1 and n, then more than $1.18\sqrt{n}$ random values will have a collision with probability 0.5.

For a hash function with m-bit message digest, Oscar can perform the following birthday attack.

1. Oscar generates $2^{m/2}$ variations of a message and computes message digests of these variations. These data are recorded in a list L sorted by the message digests.

2. Oscar generates variations of another fault message. For each variation, he computes its digest and tries to find a same digest in the list L.

3. If Oscar finds the same message digest for two different messages, then he can substitute one message with other one using the same hash value.

It is easy to generate variations of a message by a computer. One can just add some not significant codes, such as space, backspace, enter etc., to the message. The probability of success is more than 0.5, if the number of variations in step 2 is $2^{m/2}$.

Under birthday attack, we can find a collision by 2^{64} inputs for MD5 and 2^{80} inputs for SHA-1. Recently, MD5 and subsequently SHA-1 have been

broken. Here "broken" means that some algorithms can be used to find a collision by less inputs. In 2004, a group of researchers reported that they find an algorithm which can find a collision using 2^{69} inputs. On February 28, 2005, NIST published a comment on that attack. The comment indicated that "Due to advances in computing power, NIST already planed to phase our SHA-1 in favor of the larger and stronger hash functions (SHA-224, SHA-256, SHA-384 and SHA-512) by 2010. New developments should use the larger and stronger hash functions."

HMAC

A hash function was not designed for use as a MAC and cannot be used directly for that purpose because it does not use a secret key. To incorporate a secret key into an existing hash algorithm, people considered HMAC (keyed-hash authentication code). HMAC was issued as RFC 2104 and was generalized as an FIPS PUB 198 in March, 2002. HMAC is used in combination with a FIPS proved cryptographic hash function and a secret key. The algorithm can be described as follows:

$$MAC(x)_t = HMAC(K, x)_t = h((K_0 \oplus opad||h((K_0 \oplus ipad)||x))_t$$

where x is the message, h is some FIPS proved hash function, K is a secret key. K_0 is modified from K to fit the hash function. *ipad* and *opad* are inner pad and outer pad. And $||$ is the concatenation operation. The output is the leftmost t bytes.

The HMAC Algorithm can be described step by step as follows.

1. If the length of $K = B$ (the block size of the input to the hash function): set $K_0 = K$. Go to step 4.

2. If the length of $K > B$: apply hash function to K to obtain L (block size of output of the hash function) bytes string, then append $(B - L)$ zeros to create a B-byte string K_0. Go to step 4.

3. If the length of $K < B$: append zeros to the end of K to create a B-byte string K_0.

4. Exclusive-Or K_0 with *ipad* to produce a B-byte string: $K_o \oplus ipad$, where *ipad* is the byte 36 (hexadecimal notation) repeated B times.

5. Append the stream of data x to the string resulting from step 4: $(K_0 \oplus ipad)||x$.

6. Apply h to the stream generated in step 5: $h((K_0 \oplus ipad)||x)$.

7. Exclusive-Or K_0 with $opad$: $K_0 \oplus opad$, where $opad$ is the byte $5c$ (hexadecimal notation)repeated B times.

8. Append the result from step 6 to step 7: $(K_0 \oplus opad)||h((K_0 \oplus ipad)||x)$.

9. Apply h to the result from step 8: $h(K_0 \oplus opad)||h((K_0 \oplus ipad)||x)$.

10. Select the leftmost t bytes of the result of step 9 as the MAC.

If the same key is used for authentication of several messages, then the values of $K_0 \oplus ipad$ and $K_0 \oplus opad$ can be stored some where for reuse.

Note that in HMAC (or MAC) the output should be large enough. In general, the probability of an HMAC verification algorithm to accept a random digest of t-bit is $(1/2)^t$. The limitation is magnified if a verification algorithm permits different digests of a same message to be repeatedly presented for verification or permits a digest to be presented with different messages for verification. Therefore, if the output of the hash function in an HMAC is truncated, then the length t should be chosen as large as in practical with at least half as many bits as the output.

Although many hash functions, including MD5 and SHA-1, have been attacked successfully, these attacks will not effect the security of an HMAC.

5.3 Key Distribution

Since a public-key system is usually much slower than a private-key system, we need some method to distribute secret keys. In fact, the idea of a public key system is first considered for key exchange protocol by Diffie and Hellman. We will introduce some key distribution methods in this subsection.

For the security reason, a secret key cannot be used for a long time. In practice, we usually use session keys. A session key is used for limited time which enhances the security of the key. Since session keys needs refresh frequently, it is important to develop method to distribute session keys.

On July 6, 2005 NIST published a Draft Special Publication 800-56, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography". In this section, we discuss some basic key establishment schemes based on discrete logarithm problem.

Diffie-Hellman key exchange

The Diffie-Hellman key exchange algorithm is used for two parties to exchange a key. This algorithm was invented in 1976, which is based on discrete logarithm problem.

The Diffie-Hellman key exchange algorithm is described in Figure 5.4.

Let p be a prime such that the logarithm problem in $\mathbb{Z}_p$ is infeasible. Let α be a primitive element of $\mathbb{Z}_p$. Then Alice and Bob do the following.

1. Alice chooses $x_A \in \mathbb{Z}_p^*$. Then she computes and sends Bob $y_A = \alpha^{x_A} \bmod p$.
2. Bob chooses $x_B \in \mathbb{Z}_p^*$. Then he computes and sends Alice $y_B = \alpha^{x_B} \bmod p$.
3. The common key of Alice and Bob is

$$K = y_B^{x_A} \bmod p = y_A^{x_B} \bmod p$$

Figure 5.4: Diffie-Hellman Key Exchange Algorithm

The Diffie-Hellman algorithm is simple and the secret keys are created only when needed. However, there are some weaknesses which Oscar can use to attack the protocol. One attack is called man-in-the-middle attack. In this attack, Oscar impersonates Bob while communicating with Alice and impersonates Alice while communicating with Bob. To do that, he sends Alice and Bob a public value α^{x_O} using Bob's ID and Alice's ID, respectively. Then two common keys are created: $K_1 = \alpha^{x_A x_O}$ is between Alice and Oscar,

and $K_2 = \alpha^{x_O x_B}$ is between Bob and Oscar. However, Alice and Bob think they have established a common key.

Another kind of attack Oscar can do is sending Bob a lot of random numbers as his public value y_O. Then Bob has to do a lot of computations of exponentiation which wastes considerable computing resources. This attack is sometimes called clogging attack.

The above two attacks actually based on the fact that the information exchanged in the protocol are not authenticated.

Oakley key exchange

The *Oakley Key Determination Protocol* is a refinement of the Diffie-Hellman key exchange algorithm. Oakley uses cookie exchange which requires each party send a pseudorandom number in initial message and use the cookie in the subsequent communications. Cookies are used to thwart clogging attacks. Usually, a cookie is a hash value of the IP addresses of the source and destination, the UDP or TCP ports and some local secret value. The protocol requires user ID and data authentication to against man-in-the-middle attack. When cookies are used, Oscar may use a replay attack as follows. He just send the old message again and again. So the protocol also uses nonce (pseudorandom numbers) in each message exchange to ensure against replay attacks.

The Oakley specification includes a number of examples of key exchange. One example is called aggressive key exchange which only has three message exchanges. Let I be the initiator and R be the receiver. The message exchanges are as follows.

- I sends a cookie, the group to be used (value of $(\mathbb{Z}_p, \alpha)$), the value of $y_I = \alpha^{x_I} \bmod p$, I's nonce, I's identifier and R's identifier. I also indicates the public key encryption, hash and authentication algorithms to be used in this exchange. Then I appends a signature that signs the two identifiers, the nonce, (p, α), y_I, and the offered algorithms.

- R verifies I's signature. Then R echoing back I's cookie, identifier, nonce and (p, α). R also includes in the message a cookie, $y_R = \alpha^{x_R} \bmod p$, the selected algorithms (which must be among the offered algorithms), R's identifier, and R's nonce. R appends a signature that signs the two identifiers, the two nonces, p, α, y_I, y_R, and the selected algorithms.

- I verifies R's signature. The nonce values in the message assure that this is not a replay of an old message. To complete the exchange, I send a message back to R to verify that I has received R's message. This message contains two cookies, (p, α), y_I, two identifiers, two nonces the selected algorithms and a signature of the information.

The detailed message exchanges is described in Figure 5.5.

$I \rightarrow R$:	$CKY_I, KTP, GRP, \alpha^{x_I}, EHAO, NIDP, ID_I, ID_R, N_I,$ $sig_I(ID_I\|\|ID_R\|\|N_I\|\|GRP\|\|\alpha^{x_I}\|\|EHAO)$
$R \rightarrow I$:	$CKY_R, CKY_I, KTP, GRP, \alpha^{x_R}, EHAS, NIDP, ID_R, ID_I, N_R,$ $N_I, sig_R(ID_R\|\|ID_I\|\|N_R\|\|N_I\|\|GRP\|\|\alpha^{x_R}\|\|\alpha^{x_I}\|\|EHAS)$
$I \rightarrow R$:	$CKY_I, CKY_R, KTP, GRP, \alpha^{x_I}, EHAS, NIDP, ID_I, ID_R, N_I,$ $N_R, sig_I(ID_I\|\|ID_R\|\|N_I\|\|N_R\|\|GRP\|\|\alpha^{x_I}\|\|\alpha^{x_R}\|\|EHAS)$

Notation:

CKY_I:	I's cookie
KTP:	Key exchange message type
GRP:	Name of Diffie-Hellman group
$EHAO, EHAS$:	Encryption, hash, authentication, offered and selected
$NIDP$:	Indicates encryption is not used for the remainders
N_I:	Nonce of I.

Figure 5.5: Aggressive Oakley Key Exchange

In the above key exchange, the main parts of each message is signed by signature schemes. In this way, Oscar cannot forge Alice's message. However, in this case, Bob must be sure that he knows Alice's public key which is not faked by Oscar. We will discuss how to certificate a public key later.

Kerberos

The Diffie-Hellman algorithm and Oakley key exchange are for two parties to exchange a secret key.

In a network we need other key distribution service. We mentioned that to reduce the number of total keys in a network, it is desirable that there

is a key distribution center (KDC) and that session keys are used. To use KDC and session keys, we need some special key services. The *Kerberos* is a popular key serving system developed by MIT. In the Kerberos system, there is an Authentication server (AS). Each user U on the net shares a secret DES key K_U with AS (For example, a key created from password). When two users U and V need to communicate each other (here V might be some network server), they request a session key. The Kerberos is used to transmit a session key. A simplified version of the Kerberos can be described as follows.

1. U asked AS for a session key to communicate with V: U sends ID_U, TS_1 to AS, where ID_U is user U's identifier and TS_1 is time which is used to check the time synchronization.

2. AS chooses a random key K, a time stamp T (also called a ticket), and a lifetime L (lifetime for the session key). So that the session key will be valid from time T to $T + L$. Then AS sends U: $m_1 = e_{K_U}(K||ID_V||T||L)$ and $m_2 = e_{K_V}(K||ID_U||T||L)$.

3. U decrypts m_1 to obtain K, T, L and ID_V. Then U computes $m_3 = e_K(ID_U||T)$ and sends V the value of m_2 and m_3.

4. V decrypts m_2 and obtains K, ID_U, T and L. Then V can computes T and ID_V from m_3. V checks whether the two values of T and the two values of ID_U are the same. If so, V computes $m_4 = e_K(T + 1)$ and sends m_4 to U.

5. U decrypts m_4 and verifies the correctness of $T + 1$.

If everything is correct, then U and V use K as a session key.

The above description of Kerberos is based on version 4. Version 5 of Kerberos is specified in RFC 1510. Version 5 has several improvements so that the protocol is more flexible and more secure. The latest version is Kerberos 5 release 1.4.3.

5.4 Public Key Infrastructure

Kerberos uses conventional cryptosystem. So we still need to establish secret keys in the beginning of the protocol. An alternative method is to use public-key systems. Using a public key system, Bob can publish his public key

and let Alice to use that key to encryption or verification of his signature. However, how can Alice be sure that the key published is not impersonated by Oscar? That means that a public key should be authenticated. The general solution is to use a *public key infrastructure (PKI)*.

The basic idea of PKI is to let users know which public key belongs to whom. In PKI, there is a *Certificate Authority (CA)*. The CA can certificate the users public keys. However, if we look at the details of PKI, then we will find there are a lot of security problems to be considered. And it seems there is no ideal solution for PKI.

The tern PKI can be very confusing, even to a technologist, because it is used for several different things. The PKI may mean the methods, technologies and techniques that together provide a secure infrastructure. But in some cases, the PKI may just mean the use of a public key and private key pair for authentication and proof of content. People use different definitions for PKI. In general, the PKI means using public cryptosystem to protect electronic communications and electronic files for authentication, message integrity access control, identity verification, nonrepudiation, etc. We have already seen many issues about public key systems. The main uncertain problem is how to certificate the public key.

NIST developed a document called "The Certificate Issuing and Management Components Protection Profile" (version 1.0) in 2001. This document specifies the functional and assurance security requirements for a CIMC. The intent of this family of Protection Profiles is to ensure specification of the complete set of requirements for a CIMC and not the specification of a subset of requirements implemented in a specific CIMC subcomponent. It includes all the technical features of a CIMC, regardless of which CIMC subcomponent performs the function. The document does not differentiate between functions that are typically performed by a CA and functions that are typically performed by a RA (Registration Authorities).

Figure 5.6 displays an example of PKI. In this example, there are 3 CIMCs with different structures and hierarchies. One CIMC contains several registration authorities, other CIMC contains a OCSP (Online Certificate Status Protocol) server. There are two repositories to distribute certificates and certificate revocation lists (CRLs).

NIST's PKI and S/MIME programs have been merged, reflecting NIST's increased attention to PKI-aware applications. Secure mail is a priority application for nearly every organization, whether in the private sector or government. We will discuss S/MIME later. NIST is also pursuing XML

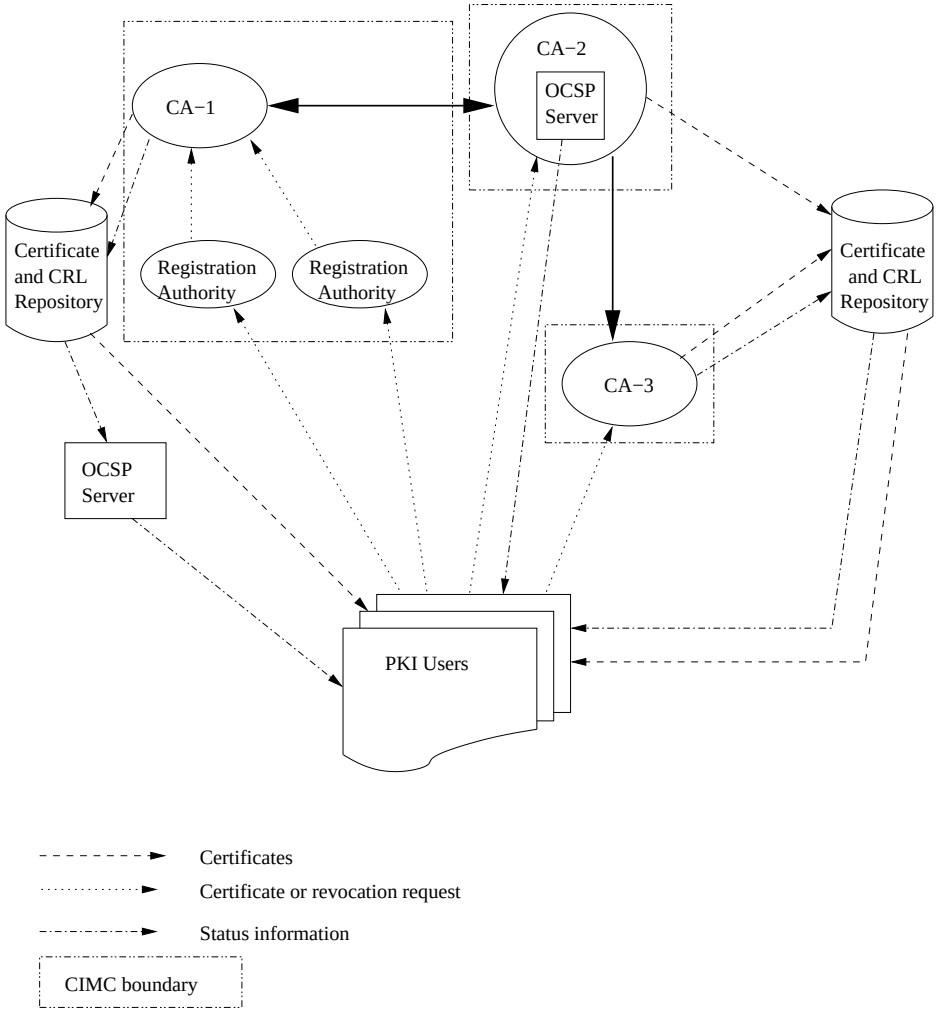


Figure 5.6: A PKI with three CAs

digital signatures using PKI to verify the identity of the signer, based on the IETF/W3C draft specifications.

There are many groups in the world working on PKI and different PKI profiles are developed. Here we give a brief introduction of internet X. 509 public key infrastructure. X.509 defines a framework for the provision of authentication services. An important part of the X.509 scheme is the public-key certificate associated with each user. The scheme assumes CA who creates user certificates and places certificates in the directory. X.509 defines certificate format, certification paths and trust, certificate revocation list, authentication procedures, etc.

All X.509 certificates have the following data, in addition to the signature:

Version: This identifies which version of the X.509 standard applies to this certificate, which affects what information can be specified in it. Thus far, three versions are defined. (A software keytool is a java based tool which can import and export v1, v2, and v3 certificates. It generates v1 certificates.)

Serial Number: The entity that created the certificate is responsible for assigning it a serial number to distinguish it from other certificates it issues. This information is used in numerous ways, for example when a certificate is revoked its serial number is placed in a Certificate Revocation List (CRL).

Signature Algorithm Identifier: This identifies the algorithm used by the CA to sign the certificate.

Issuer Name: The X.500 Distinguished Name of the entity that signed the certificate. This is normally a CA. Using this certificate implies trusting the entity that signed this certificate. (Note that in some cases, such as root or top-level CA certificates, the issuer signs its own certificate.)

Validity Period: Each certificate is valid only for a limited amount of time. This period is described by a start date and time and an end date and time, and can be as short as a few seconds or almost as long as a century. The validity period chosen depends on a number of factors, such as the strength of the private key used to sign the certificate or the amount one is willing to pay for a certificate. This is the expected period that entities can rely on the public value, if the associated private key has not been compromised.

Subject Name: The name of the entity whose public key the certificate identifies. This name uses the X.500 standard, so it is intended to be unique across the Internet. This is the X.500 Distinguished Name (DN) of the entity, for example,

CN=Java Duke, OU=Java Software Division, O=Sun Microsystems Inc,

C=US

(These refer to the subject's Common Name, Organizational Unit, Organization, and Country.)

Subject Public Key Information: This is the public key of the entity being named, together with an algorithm identifier which specifies which public key cryptosystem this key belongs to and any associated key parameters.

X.509 certificates are used in SSL protocol, IPSec, S/MIME, SET, that we will discuss later.

Another type of certificate of public keys is PGP.

5.5 Quantum Techniques in Cryptography

Quantum computing is a new area of research which is difficult for non-physicists to fully understanding. However, some quantum techniques are very important to cryptography systems. In this section, we will give some very brief explanation of some quantum techniques that are used for cryptography.

Quantum mechanics is a particle-level physics. We need particles that we are able to observe. Photons are the particles that make up light and are therefore observable. Light is an example of an electromagnetic wave, meaning that it consists of an electric field that travels orthogonally to a corresponding magnetic field. Therefore we can define the concept of polarization of light.

We will represent a photon's polarization by a unit vector in the two dimensional complex vector space. This vector space has a dot product given by $(a,b)\cdot(c,d) = a\bar{c}+b\bar{d}$, where $\bar{c}$ and $\bar{d}$ denote the complex conjugates of c and d. The square of the length of vector (a,b) is then $(a,b)\cdot(a,b) = |a|^2+|b|^2$. Choose a basis for this vector space, which we will denote $|\uparrow\rangle$ and $|\rightarrow\rangle$. We can think of $|\uparrow\rangle$ as being the vertical direction and $|\rightarrow\rangle$ as being horizontal. So an arbitrary polarization may be represented as $a|\uparrow\rangle + b|\rightarrow\rangle$, where a and b are complex numbers. Since we are working with unit vectors, $|a|^2+|b|^2 = 1$. We could also chosen a different orthogonal basis, for example one corresponding to a 45° rotation: $|\nwarrow\rangle$ and $|\nearrow\rangle$.

The Polaroid filters perform a measurement of the polarity of the photon. There are two possible outcomes: either the photon is aligned with the filter, or it is perpendicular to the direction of the filter. If the vector $a|\uparrow\rangle + b|\rightarrow\rangle$

is measured by a vertical filter, then the probability that the photon has vertical polarity after passing through the filter is $|a|^2$. The probability that it will have horizontal polarity is $|b|^2$.

Similarly, suppose we measure a vertically aligned photon with respect to a 45° filter. Since

$$|\uparrow\rangle = \frac{1}{\sqrt{2}}|\nearrow\rangle + \frac{1}{\sqrt{2}}|\nwarrow\rangle,$$

the probability that the photon passes through the filter (it is measured as being aligned at 45°) is $(1/\sqrt{2})^2 = 1/2$. Similarly, the probability that it does not pass through the filter is also 1/2. One important property of quantum mechanics is that such a measurement forces the photon into a definite state. Therefore after being measured, the state of the photon will be changed to the result of the measurement. For example, if we measured the state of $a|\uparrow\rangle + b|\rightarrow\rangle$ as $|\rightarrow\rangle$, then the photon will have the state $|\rightarrow\rangle$. If we then measure with a $|\uparrow\rangle$ filter, we will never observe that the photon is in the $|\uparrow\rangle$ state.

5.5.1 Quantum key distribution BB84

Researchers have used the quantum mechanics to design key distribution schemes which are not depending on computational security. We introduce one of such schemes BB84 developed by Charles Bennett and Gilles Brassard in 1984.

We need to define a quantum bit, known as a qubit. In a two-dimensional complex vector space, choose a pair of orthogonal vectors of length one, call them $|0\rangle$ and $|1\rangle$. A qubit is a unit vector in this vector space. For the demonstration, we can think of a qubit as a polarized photon. We have chosen $|0\rangle$ and $|1\rangle$ as notation to conveniently represent the 0 and 1 bit, respectively. The other qubits are linear combinations of these two bits. So a qubit can be represented as $a|0\rangle + b|1\rangle$, where a and b are complex numbers such that $|a|^2 + |b|^2 = 1$.

To establishing a secret key, Alice and Bob need two channels: on quantum channel and one classical communication channel. Both channels are public. A quantum channel is one through which they can exchange polarized photons. The classic channel will be used to send ordinary messages to each other.

Two bases are chosen: $B_1 = \{|\uparrow\rangle, |\rightarrow\rangle\}$ and $B_2 = \{|\nwarrow\rangle, |\nearrow\rangle\}$. Alice starts the establishment of a message by sending a sequence of bits to Bob.

Alice's bit	0	1	1	0	1	0	0	1	0	1	0	1
Alice's basis	+	+	×	+	×	×	×	+	+	×	+	×
Photon polarization	↑	→	↗	↑	↗	↖	↖	→	↑	↗	↑	↗
Bob's measure	+	×	×	+	+	×	+	×	+	×	×	+
Shared key	0		1	0		0			0	1		

Table 5.1: BB84 key distribution

For each bit, Alice will randomly choose a base. If B_1 is chosen, then she encodes 0 as $|\uparrow\rangle$ and 1 as $|\rightarrow\rangle$. If B_2 is chosen, then she encodes 0 and 1 using the two elements of B_2.

When Alice sends a photon, Bob randomly chooses measure with respect either basis B_1 or B_2. In this way, Bob obtains an element of that choice of basis as the result of his measurement. Bob records the measurements has made and keeps them secret. He then tells Alice the basis with which he measured each photon. Alice responds to Bob by telling him which bases were the correct bases for the polarity of the photons that she sent. They keep the bits that uses the same bases and discard the other bits. Since two bases were used, Alice and Bob will agree on roughly half of the amount of bits that Alice sent. They can then use these bits as the key for a cryptographic system.

We now use a small example to explain the idea. To simplify the discussion, we use the following notations. So we use + to denote basis B_1 is chosen to encode (× denote basis B_2 is chosen).

Basis	0	1
+	↑	→
×	↖	↗

Now suppose Alice creates random bits: 011010010101 and randomly selects bases as in table 5.1. Bob measured the photon using randomly selected bases. Then both of them send the basis sequences they used. In this way, they established the shared key 010001.

Now if Oscar tries to eavesdrop the communication. To get the information about the photons Alice sent to Bob, Oscar also chooses random bases to measure the polarization of photons. Then Bob may obtain wrong information about the key. We explain that situation in Table 5.2.

Therefore after Alice and Bob obtained the common key, they will select partial key bits at random positions and check the correctness of these bits.

Alice's bit	0	1	1	0	1	0	0	1	0	1	0	1
Alice's basis	+	+	×	+	×	×	×	+	+	×	+	×
Photon polarization	↑	→	↗	↑	↗	↖	↖	→	↑	↗	↑	↗
Oscar's measure	+	×	+	×	+	×	×	+	+	×	+	×
Photon polarization	↑	↑	→	↖	→	↖	↖	→	↑	↗	↑	↗
Bob's measure	+	×	×	+	+	×	+	×	+	×	×	+
Shared key	0		0	1		0			0	1		
Errors			?	?								

Table 5.2: BB84 key distribution with attackers

If there is error, then the key will be discarded. Otherwise, the remained bits (unreleased bits) will be used as the key. In this example, if the random bits including the second or third bit, they will found that the key is wrong.

If the length of the key is long enough and the randomness that Alice and Bob used are good, then the probability that they cannot find the error is very small.

Note that the BB84 does not include the communication authentication. It means that the man-in-the-middle attacks needs to be considered. There are other quantum key distribution proposals. The security of quantum key distribution does not depend on difficult mathematical problems, that is a very important advantage comparing to the public key distribution schemes. On the other hand, there are still many kinds of possible attacks for quantum key distributions which have not been fully discussed in cryptography society.

5.5.2 Shor's factoring algorithm

Chapter 6

Remote Access Control

We have learnt a lot of cryptosystems, authentication schemes and other security programs. Now we start to consider how to apply these techniques to network security. In fact, there are still a lot of things need to consider when we try to apply the security programs discussed in precious chapters.

In this chapter we discuss remote access controls. However, we first need to discuss general access controls. We will mainly discuss access control by passwords. Although there are other access control methods such as biometric authentications and behavioral authentications. Examples of biometric authentications are using fingerprinting scanner, hand geometry scanner, retinal scanner iris scanner etc. Examples for behavioral authentication are handwritten signature verifications, voice-to-print technologies etc. Using password to control accessing computer and network is a most popular method.

6.1 UNIX Password Systems

A password system is an important tool to control the access of a host computer. Since the UNIX system is a multi-user operating system, it applied a password system. The purpose of a password is used to protect a user's privacy, so that other users cannot access his account. So a password should be kept in secret that other users cannot see it. On the other hand, when the user uses the password, the computer should be able to check the correctness of the password. So there should be some information stored in computer for the verification of the passwords.

The UNIX system uses function `crypt(3)` to encrypt user's password. `crypt(3)` function is based on DES. It takes user's password (8 7-bit ASCII characters) as the encryption key and uses it to encrypt a 64-bit block of zeros. Then the resulting ciphertext is encrypted using the password as a key again. The process is repeated a total of 25 times. The procedure can be described as follows. Let the password be PW. The algorithm of password encryption is as in Figure 6.1.

$O_0 = 00 \cdots 0$.
For $i = 1$ to 25 do
$O_i = e_{PW}(O_{i-1})$

Figure 6.1: Password Encryption

The final 64 bits are unpacked into a string of printable characters that are stored in the `etc/passwd` file or `etc/shadow` file. For example the following is a record in `etc/passwd`:

```
guest:x:1001:10:limited user:/export/home/guest:/bin/sh
```

This record indicates that the user name is `guest`, password is requested, user directory is `/home/guest` and the shell he used is `/bin/bash`.

A record in `etc/shadow` looks like:

```
guest:541e3S7LBx03E:12109:5:30:5:10:12144:
```

The string `541e3S7LBx03E` is the encrypted password record.

When a user logs in, the computer (`login`) calls `crypt(3)` and encrypts with the user's password as a key. The output is compared with the record of the `passwd`. The user is accepted only if the two values are the same.

In practice, `crypt(3)` also adds some random number as salt of the password when a user creates a password. The password will concatenate with the salt before it is encrypted. The salt is also recorded in the file. Next time, when a user logs in using user's ID, the computer first gets the salt from the file and then encrypts the password provided by the user. In this way, the salt will modify the password record so that even two users use the

same passwords, their records are different. The salt also increases the length of a password.

There are some password cracker programs which can search password by guessing the password or using a password dictionary. So a password must be meaningless and it should be changed frequently. The password shadow file (`/etc/shadow`) is used to control password aging. In the above example, numbers `5:30:5:10` are used to control password lifetime. A good password should meet the following criteria:

- Be at least 8 characters in length.
- Contain both upper and lower case characters.
- Contain at least one number.
- Contain at least one special character.
- Not based upon a dictionary word.

Some method can be used to create password met the above rules, which is also easy to remember. For example you may think about a sentence (but it is kind of unique) such as "My girl friend Patricia is always asking me for help". Then you can use the following password:

`MgfPia?m4h`.

In some cases, hash function is used for password. For example MD5 is used in Linux so that the password can be any length. The resulting 128-bit digest is used as a key for encryption. In this case, a long nonsense sentence or phrase can be used as a password. For example: `I jumped to the Moon and saw many beautiful ladies swimming there!` might be a good password. Using a long sentence or phrase as a password need consider an attack called racing attack which we will discuss in next section when we consider one time password.

Some UNIX system now uses `bigcrypt()` or `crypt16()` that uses 16 or more significant characters as a password.

6.2 One Time Password

It is more difficult to defend the password over a network, since some sniffer program can capture and record characters sent over the network. So some-

one can eavesdrop on network connections to obtain login IDs and passwords of legitimate users.

One method used for protecting password is to use one time password (OPT). A one time password only can be used one login. So when someone catches a password over the net, the password is already expired.

To apply one time password, we need to change login program. Another way used in UNIX system is to replace the user's login shell with a specialized program to prompt for the one time password. In this case, when a user logs in, he is first asked for the password and then is asked for the one time password. If an incorrect password is entered, the program will log the user out.

Now we need some method to create one-time password. Several methods were used for one-time password.

One method is to use a *token card*. A token card is a small card or calculator with a built-in programmed authentication functions and a serial number. A token card can be used to generate one-time password. The following two cards are examples.

- SECURID: A small card displays a number that changes every 30-90 seconds. The number that is displayed is a function of the current time and the ID of that particular card, and is synchronized with the remote server. Some version has a keypad which is used to enter a personal identification number code (PIN). This card is simple and small. But the server should keep the time synchronized with the card.

- SecureNet key: A small device looks like a calculator. When the user contact the remote server, the server displays a number as a challenge. The user then type the challenge number into the card, along with its PIN. The card will display the one-time password. The SecureNet key card can be programmed to self-destruct if incorrect PIN is entered more than a number of times.

Another method to create one-time password is to use a codebook. This is a list of passwords that are used, one at a time, and then never reused. For example, a program called S/Key can used to create a codebook. It can either run the program to generate a sequence passwords on a portable computer or print out a listing as a paper codebook.

In 1998, A one-time password system was published as RFC 2289 (which is a revision of RFC 1938). This system uses a standard hash function such

as MD4, MD5 or SHA-1. To create a one-time password, server sends a challenge message to user. The syntax of the challenge is

`otp-<algorithm ID><sequence integer><seed>`

where `seed` consists of 1 to 16 purely alphanumeric characters. An example of an OPT challenge is : `otp-md5 487 dog2`.

Then the user chooses a secret pass-phrase which consists at least 10 characters. The pass phrase is concatenated with the seed. The result of the concatenation is passed through the secure hash function N times, where N is specified by the user. The resulting digest is the one-time password record. The next one-time password to be used is generated by passing through the secure hash function $N - 1$ times.

To authenticate the user, the server passes the password through the secure hash function once and compares the result with the stored previous OTP. If the result of the operation matches the previous OPT, the authentication is successful and the accepted one-time password is stored for future use. In this way, a pass-phrases can be used for $N - 1$ times.

The security of this system depends on the hash function's one-way property. The `seed` used here enables the user to use the same secret pass-phrase for different machines.

Since one-time password only can be used once, it does not need to protect against eavesdropping. However, it is possible for an attacker to listen to most of a one-time password, guess the remainder, and then race the legitimate user to complete the authentication. The speed of human typing is much slower than the computer generating. Multiple guesses against the last word of the six-word format are likely to succeed. So we need some method to protect against the race attack. One method is to prevent a user from starting multiple simultaneous authentication sessions. This means that once the legitimate user has initiated authentication, an attacker would be blocked until the first authentication process has completed. In this case, a timeout is necessary to thwart a denial of service attack.

6.3 Secure Shell

Recently, another approach for remotely accessing a computer is to use SSH (secure shell) protocol. SSH provides support for secure remote login, secure file transfer, and secure TCP/IP and X11 forwarding. It can automatically

encrypt, authenticate, and compress transmitted data.

The main idea of SSH is to establish a common key between a client and a server using secure key exchange technique first. The followed communications are then encrypted and authenticated. So the main idea is not very complicated. However, as a real application, we will see that many things need to be considered more careful in details.

SSH consists of three major components:

- The Transport Layer Protocol provides server authentication, confidentiality and integrity with perfect forward secrecy.
- The User Authentication Protocol authenticates the client to the server.
- The Connection Protocol multiplexes the encrypted tunnel into several logical channels.

SSH is widely used now although it is still in a developing stage. In most cases, old protocols such as telnet and ftp are substituted by SSH. We will not discuss the details of these three protocols, but give some description of the main security considerations of SSH. These descriptions are based on Internet-Draft written by `secsh` group. Recently, SSH has been published as RFC 4251-4256 (January 2006).

The Transport Layer Protocol can be described as follows. In SSH, the server listens for connections (on port 22). The client initiates a connection. When the connection has been established, both sides do the following. In what follows, C denote the client and S denote the server.

- Send an identification string to each other. The main contents of the string is the version of SSH and the version of software they used. An example is as follows.

  `SSH-2.0-billsSSH_3.6.3q3<CR><LF>`

  In this example, the user uses protocol version 2.0 and a software `billsSSH_3.6.3q3`. The identification string must be terminated by a single Carriage Return (CR) and a single Line Feed (LF) character (ASCII 13 and 10, respectively).

- Both side send out a KEXINIT packet. This packet includes: cookie (random bytes), list of algorithms supported by the machine such as key

exchange algorithms, encryption algorithms, MAC algorithms, compression algorithms, languages. All the algorithms are listed in order of preference. This packet is used for each side to choose the same algorithm they will use later. The purpose of the cookie is to make it impossible for either side to fully determine the keys and the session identifier.

- Run key exchange program. For example the following Diffie-Hellman key exchange can be used (p, q, α is defined as in Section 5.3).

  1. C generates a random number $x, (1 < x < q)$ and sends the value $e = \alpha^x \bmod p$ to S.
  2. S generates a random value $y, (0 < y < q)$ and computes $f = \alpha^y \bmod p$, $K = e^y = \alpha^{xy} \bmod p$ and

     $$H = hash(V_C||V_S||I_C||I_S||K_S||e||f||K),$$

     where V_C, V_S are the version strings for C and S respectively, I_C (I_S) is the payload of C's (respectively S's) KEXINIT, K_S is S's public key used to verify the signature. A payload means the useful contents of the packet. Then S computes the signature s on the message H and sends $K_S||f||s$ to C.
  3. C checks K_S from a local database or some trusted certification authority. C computes $K = f^x \bmod p$ and

     $$H = hash(V_C||V_S||I_C||I_S||K_S||e||f||K).$$

     Then C verifies the signature s.

  K is the session key. A session key should be re-changed after some time. It is recommended that the keys are changed after each gigbyte of transmitted data or after each hour of connection time, whichever comes sooner.

- User Authentication protocol and connection protocol may start after the key exchange.

- C requests a service from S and S provides the service. In this stage, all the communications should be encrypted and authenticated.

- Either party sends out a disconnection message.

In each step of the communication, if any party finds something wrong, then the connection will be broken.

All packets following the identification string use the following binary packet protocol.

PKL	PDL	Payload	Padding	MAC

The fields of the packet is as follows. The total size of the packet is $35,000$ bytes or less.

- PKL (32 bits): The length of the packet (in bytes), not including MAC and PKL field itself.
- PDL (8 bits): The Length of padding (in bytes).
- Payload (n_1 bytes): The useful contents of the packet. If compression has been negotiated, this field is compressed. n_1 = PKL-PDL - 1.
- Padding (PDL bytes): Added random padding bytes, such that the total length of the packet is a multiple of the cipher block size or 8, whichever is larger. The length of the padding (PDL) is between 4 bytes to 255 bytes.
- MAC: If message authentication has been negotiated, this field contains the MAC bytes. Initially, the MAC algorithm is "none".

The encryption method required in SSH is 3-DES (3 keys) of CBC mode. Other method recommended for SSH are AES-128, AES-192, AES-256. Optional encryption algorithms can be used in SSH such as: Blowfish, Twofish, Serpend, IDEA, CAST. The compression method currently defined is zlib. The message authentication used in SSH is HMAC. The hash function used is SHA-1, but the MD5 is still an option. So we first use HMAC to get the authenticated digest of a message m. Then the message m is encrypted by the decided encryption method. The actually transmitted data is the encrypted message together with the authenticated digest. The signature scheme used in SSH is DSS.

SSH authentication protocol runs on top of the SSH transport layer protocol and provides a single authenticated tunnel for the SSH connection protocol. The service name for this protocol is "ssh-userauth". Basically, the server sends authentication requests using the following format:

`SSH-MSG-USERAUTH-RQUEST` (code 50)
`user name`
`service name`
`method name`
`method specific fields.`

The server should have a timeout for authentication, and disconnect if the authentication has not been accepted within the timeout period. If the authentication is successful, then the server sends out a response:

`SSH-MSG-USERAUTH-SUCCESS` (code 52)

Otherwise the server responds:

`SSH-MSG-USERAUTH-FAILURE` (code 51)
`authentications that can continue`
`partial success`

where `authentications that can continue` is a comma-separated list of authentication method names that may productively continue the authentication dialog. `partial success` is a boolean value of true or false.

There are three authentication methods used in SSH. One is the public key authentication method. In this method, the user uses a public key signature scheme to sign on a message that contains session identifier, user name, public key algorithm name, public key to be used for authentication etc. When the server receives this message, it checks whether the supplied key is acceptable for authentication, and if so, it then check whether the signature is correct.

The second method is password authentication method. In this method, the user needs to transmit the password to server. Since this transmitted packet is on the transport layer, it is encrypted. In this case, both the server and the client should check whether the underlying transport layer provides confidentiality (i.e., if encryption is being used).

The third method is host-based authentication. This form of authentication is optional, since it is not suitable for high-security sites. It is similar to the UNIX `rhosts` and `hosts.equiv` styles of authentication, except that the identity of the client host is checked more rigorously. In this method, the client sends a public key signature with the key of the client host. The message signed contains session identifier, user name, public key algorithm

for host key, public host key and certificates for client host, client host name, etc. The server verifies that the host key actually belongs to the client host name in the message, that the given user on that host is allowed to log in, and that the signature is a valid signature on the appropriate value by the given host key. If it is possible, the server performs additional checks to verify that the network address obtained from the network matches the given client name.

The SSH connection protocol has been designed to run on top of the SSH transport layer and user authentication protocols. It provides interactive login session, remote execution of commands, forward TCP/IP connections, and forwarded X11 connections. All of these channels are multiplexed into a single encryption tunnel. We will omit the details of this protocol, since the most security considerations are addressed in transport layer protocol and user authentication protocol.

The design of protocols of SSH considered security, efficiency and flexibility. It is intended to be implemented at the application level.

Chapter 7

E-Mail Security

Electronic mail is one of the most heavily used network-based applications. With the explosively growing reliance on e-mail, there grows a demand for security e-mail systems. In an e-mail system, there are a sender and a receiver. However, usually the receiver is not on-line. So in an e-mail system, usually there is no massage interchange when the sender sends an e-mail. On the other hand, some e-mail system (e.g., SMTP) only can deliver ASCII codes. We need to consider how to provide authentication and confidentiality services in this situation. We will examine two most widely used systems.

7.1 Pretty Good Privacy

Pretty good privacy or PGP was developed by Phil Zimmermann. PGP uses public key encryption, signature scheme, hash function, secret key encryption, compression function and e-mail compatibility. We can outline the algorithm as follows.

When Alice (A) wants to send a message M to Bob (B), she does the following.

1. Computes $H(M)$, where H is a hash function.
2. Signs the digest. So A computes $sig_A(H(M))$.
3. Compression the message. A computes $CM = zip(sig_A(H(M))||M)$.
4. Chooses a session key K (random number) and encrypts CM using the session key. A computes $e_K(CM)$.

5. Uses B's public key to encrypt K. She computes $e_{K_B}(K)$.
6. Concatenates $e_K(CM)||e_{K_B}(K)$.
7. Uses Radix-64 conversion (explained later) to convert $e_K(CM)||e_{K_B}(K)$ to printable characters.
8. Sends out the above message. The message is split into segment before sending, if necessary.

Bob does the following, when he received the message.

1. Uses Radix-64 to convert the message into binary version.
2. Decrypts $e_{K_B}(K)$ to obtain session key K.
3. Decrypts $e_K(CM)$ to obtain CM.
4. Unzips CM to obtain $sig_A(H(M))||M$.
5. Computes $H(M)$ from M
6. Verifies $sig_A(H(M))$.

The PGP has many nice features. We give some further discussions about the operations of PGP.

- Confidentiality. PGP uses secret session key to encrypt (CAST, IDEA, AES, Blowfish, 3-key DES etc.) and the session key actually is a one-time key. The session key is sent by public key encryption (RSA or ElGamal). So key exchange is not needed and we don't need hand-shaking to assure that both sides have the same session key.
- Authentication. PGP uses a hash function (SHA-1 or MD5) to obtain digest of the message. Then a public key signature scheme (RSA or DSS) is used to sign the digest.
- Compression. PGP uses zip (or zlib) to compression the message and the signature. A message encryption is applied after compression to strengthen the security. This is because the compressed message has less redundancy than the original text. The basic idea of zip compression is to replace a repeated string by a short code. For example, consider the following text

`the more he read the letter the more he confused by the letter they wrote`

The zip program will search the text to find repeated sequences. The second appearance of "`the more he` " will be replaced as a code $(18, 10)$, where 18 is a pointer pointed to 18 characters before and 10 is the length of the sequence. Similarly, the second appearance of "`the letter the`" will be replaced as a code $(38, 16)$. In this way, the compressed message will be shorter and have less redundancy than the original message.

- E-mail compatibility. Since many email system only permits the use of block consisting of ASCII text, PGP uses Radix-64 (also called base 64) to convert the raw 8-bit binary stream to a stream of ASCII characters as follows.

  Suppose there is a 24-bit:

  $$001000\ 110101\ 110010\ 010001$$

  which is divided into four 6-bit. Each of the 6-bit can be converted into a number between 0 and 63. In the above example, the 24-bit is converted to numbers $8, 53, 50, 17$. Then the following correspondence is used. The numbers from 0 to 25 are corresponding to the characters from A to Z. The numbers from 26 to 51 are corresponding to the characters from a to z. The numbers from 52 to 61 are corresponding to the characters from 0 to 9. The number 62 is corresponding to the character + and the number 63 is corresponding to the character /. Thus the numbers $8, 53, 50, 17$ are corresponding to $I1yR$. If the binary stream is not divided by 6, then four or two "0" need to padded. In these cases, two or one "=" are padded as a indicator. Finally, radix-64 outputs the characters as 8-bit ASCII codes. So radix-64 expands 24-bit to 32-bit. However, since PGP uses zip compression which is of an average compression rate of about 2.0, the overall compression is about one-third.

Alice in PGP involves three types of keys: a session key, her keys of a public key system and Bob's public key. So we need to consider key management and public key certificates.

For a session key, PGP defined a random number generation algorithm. CAST-128 uses a key of size 128 bits.

Since a public key system has two keys, one public key and one private key, how to securely store a private key is a problem. In PGP, when the system generates a new public/private key pair, it asks the user for a passphrase. The SHA-1 is then used to the passphrase to get a 160-bit digest. This digest is used as a key to encrypt the private key of the public system. The ciphertext is then stored in a file (called secret-key ring or private key ring). Subsequently, when the user wants to retrieve the private key, he or she must supply the passphrase. PGP will retrieve the encrypted private key, generate the digest of the passphrase, and decrypt the ciphertext to obtain the private key.

The structure of private-key ring is as follows, where the records are indexed by Key ID or User ID.

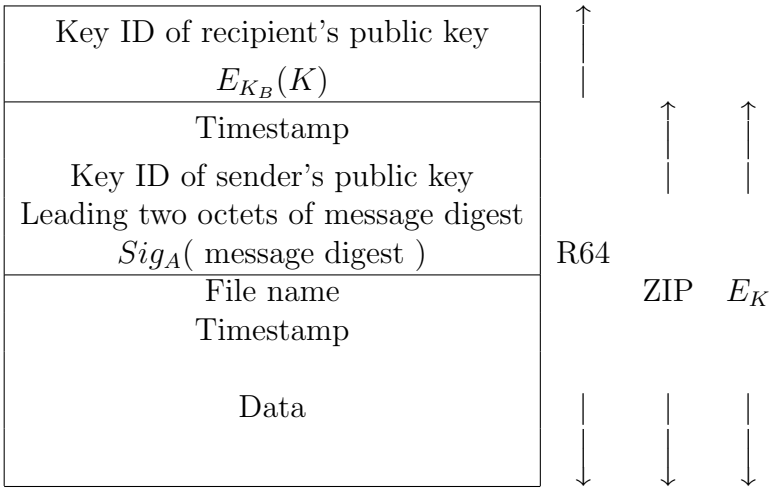
Timestamp	Key ID	Public Key	Encrypted private key	User ID
⋮	⋮	⋮	⋮	⋮

For the public key, there are more things we need to consider. First a public key ID is used in PGP to indicate which public key is used. This is necessary if Bob has more than one public keys. PGP uses the last 64 bits of the public key as key ID. Most difficult and complicated thing for the public key is how to certificate it. PGP uses a referral method for certification. PGP uses a public-key ring to record public keys of other users that are known to this user. In each entry of the record, several fields are listed which include timestamp (the date/time of the generating of this entry), key ID, public key, user ID (the owner of this key), owner trust (a trust flag indicating the user's assessment of the trust to the owner of the public key), key legitimacy, signature(s), signature trust, etc. A certificate of PGP can be signed by several users. These signatures are collected in signature fields. For each signature there is a trust flag of the signature recorded in signature trust field. Based on these information, PGP computes a value of the trust level to the public key and records the value at the legitimacy field. The higher the level of trust, the stronger is the binding of this user ID to this key. The structure of a public-key ring is as follows.

Time-stamp	Key ID	Public key	Owner trust	User ID	Key legitimacy	Signa-tures	Signature trusts
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

The PGP also considers how to revoke public keys. Usually a user will revoke a key after some time or by some reasons. On the other hand, the system will not let a user revoke other users' public keys.

The format of PGP message is described in Figure 7.1



Notation:
K_B: Receiver's public key
K: Session key
Sig_A: Sender's signature
R64: Radix-64

Figure 7.1: General format of PGP message

This format includes three components: session key, signature, and message.

The timestamp indicates the time at which the signature (or the message) was made. The leading two octects of message digest is used for the receiver to check the signature.

The message component and optional signature component may be compressed using ZIP and may be encrypted using a session key. The entire block is usually encoded with the redix-64.

In the implementation of PGP, several requirements are considered such as the flexibility which allows the user using or not using encryption or authentication, the revoking public keys etc. We omitted the details here.

RFC 2440 described a standard of PGP called OpenPGP. RFC 3156 describes how the OpenPGP Message Format can be used to provide privacy and authentication using the Multipurpose Internet Mail Extensions (MIME) security content types described in RFC 1847.

7.2 S/MIME

Secure/Multipurpose Internet Mail Extension (S/MIME) is another security enhanced email system. S/MIME is similar to PGP which uses signature scheme, session key and secret key encryption. S/MIME version 3.1 message specification is given in RFC 3851. It appears likely that S/MIME will emerge as the industry standard for commercial and organizational use, while PGP will remain the choice for personal use.

MIME is specified in RFCs 2045 through 2049. MIME is an extension of Simple Mail Transfer Protocol (SMTP) that is specified in RFC 822. MIME is more flexible and powerful than SMTP. MIME uses different transfer encodings such as 7-bit, 8-bit, binary, base64, etc. So MIME can be used to send texts, mixed messages, images, video and audio files, executable files, etc, while SMTP only can transfer ASCII codes.

S/MIME is very similar to PGP. It also offers the ability to encrypt and authenticate messages. The hash functions used in S/MIME are SHA-1 and MD5. The signature schemes used in S/MIME are DSS and RSA signature scheme. The public key encryption system used is ElGamal cryptosystem or RSA encryption with key sizes 512 bits to 1024 bits. The data encryption method used is triple DES or RC2. In S/MINE version 3.1, more advanced security algorithms are included, such as SHA-256, SHA-384, SHA-512, AES, etc.

S/MIME provides the following functions

- Enveloped data: This consists of encrypted content of any type and encrypted content encryption keys for one or more recipients. To prepare an enveloped data, the sender generates a pseudo random session key and then for each recipient, encrypted the session key with the recipient's public RSA key. For each recipient, the sender prepares an RecipientInfo that contains the sender's public key certificate, an identifier of public key system used to encrypt the session key, and the encrypted session key. The message content is encrypted using the ses-

sion key. The RecipientInfo blocks followed by the encrypted message content constitute the enveloped data. it is then encoded into base64.

- Signed data: A digital signature is formed for the message. The sender selects a hash function and computes the digest of the message content. Then the sender sign the message digest using a signature scheme. A block known as SignerInfo is formed, which contains the signer's public key certificate, an identifier of the hash function, an identifier of the signature scheme and the signature. The content and the SignerInfo are then encoded into base64.

- Clear-signed data: Use the same method of signed data to form a signature. However, in this case, only the digital signature is encoded into base64. The message content is not encoded so that the recipients without S/MIME can view the message, although they cannot verify the signature. The data consist two part, one part is the message content and the other part is an attached digital signature.

- Signed and enveloped data: Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

Since there is no interaction between a sender and a recipient when a sender sends an e-mail, there are some decisions about the algorithm used for the content to be made by the sender. First, the sending agent must determine if the receiving agent is able to decrypt the message using a given encryption algorithm. Second, if the receiving agent is only capable of accepting weakly encrypted content, the sending agent must decide if it is acceptable to send using weak encryption. To support this decision process, a sending agent may announce its decrypting capabilities in order of preference. A receiving agent may store that information for future use. Usually, the sending agent should use the first method in the intended receiver's list or the method used on the last message received from that intended recipient. If the sending agent has no knowledge about the decryption capabilities of the intended recipient, then the sending agent uses triple DES when the sender is willing to risk that the recipient may not be able to decrypt the message, or uses RC2 otherwise.

The certificates used in S/MIME is X.509.

RFC 2634 defined Enhanced Security Services for S/MIME, which is a set of extensions to S/MIME to allow signed receipts, security labels, and secure mailing lists.

Chapter 8

Web Security

Web security includes three parts: security of server, security of client, and network traffic security between a browser and a server. Security of server and security of client are problems of computer security. In this chapter we consider the traffic security.

Network security can considered at different levels, for example:

- Network level: Use IPSec.
- Transport level: Use SSL (Secure Socket Layer) or TLS (Transport Layer Security).
- Application level: Use PGP, S/MIME, SET(Secure Electronic Transaction).

In this chapter we discuss some schemes related to the web security.

8.1 SSL

Secure Socket Layer (SSL) is developed by Netscape. The main parts of SSL contains several protocols: SSL Handshake Protocol, SSL Change Cipher Spec Protocol, SSL Alert Protocol, SSL Record Protocol. We give the outline of these protocols in the follows.

(a) SSL Record Protocol

This protocol defines how to transmit an application message in SSL. The sender of a message does the following:

- Fragmentation: The message is fragmented into blocks of 2^{14} bytes or less.

- Compression: (optional).

- Add MAC: Compute a MAC using a shared secret key K and add the resulting MAC to the fragment. The MAC function used in SSL is similar to HMAC. A hash function h and a shared secret key K is used. The MAC value is computed as follows.

$$h(K||pad_2||h(K||pad_1||seqnum||type||length||M))$$

  where *seqnum* is the sequence number of the message, *type* is the high-level protocol used to process this fragment, *length* is the length of the fragment and M is the content of the fragment. pad_1 and pad_2 are fixed paddings which are repeating of fixed bytes.

- Encrypt: Encrypt the compresses message plus the MAC using symmetric encryption (block cipher or stream cipher). The block cipher used in SSL are IDEA, DES, 3-DES, RC4, Fortezza. The stream cipher used are RC4s.

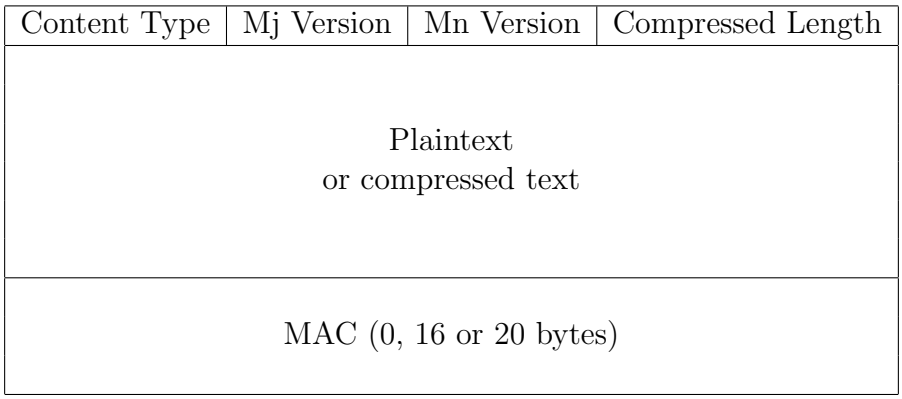
- Append SSL record header: The header consists: Content Type (8 bits), Major Version (8 bits), minor version (8 bits) and Compressed Length (16 bits). The compressed length is the length of the plaintext (or compressed plaintext) fragment in bytes. The maximum value is $2^{14} + 2048$. The content types are change_cipher_spec, alert, handshake and application_data.

The SSL record format is illustrated in Figure 8.1.

The following Change Cipher Spec Protocol and Alert Protocol are encapsulated by Record Protocol.

(b) Change Cipher Spec Protocol

This protocol consists of a single byte with the value 1. This message causes the pending state to be copied into the current state, which updates the cipher suite. Usually, it is sent during the handshake sequence (we will discuss Handshake protocol later) after key exchange and certificate certificate verification (optional).



Note: The message except the header is encrypted.

Figure 8.1: SSL Record Format

(c) Alert Protocol

Each message of this protocol consists of two bytes. The first byte takes the value waning (1) or fatal (2). The second byte contains a code which indicates the specific alert such as unexpected_message, bad_record_mac, bad_certificate, etc.

(d) Handshake Protocol

This protocol is more complicated than other SSL protocols. The protocol consists of a series of messages exchanged by client and server. Each message has three fields:

- Type (1 byte): Indicates the message type such as hello_request, client_hello, server_key_exchange, certificate_verify, etc.
- Length (3 bytes): the length of the message in bytes.
- Content (≥ 1 byte): The parameters associated with this message.

The protocol can be viewed as having four phases of exchanges. Now we give description of these exchanges.

Phase 1. Establish Security Capabilities

The client initiates the exchange by sending a client_hello message (in SSL record format) which contains: version, random number (nonce), session ID,

CipherSuite which is a list of cryptographic algorithms supported by the client, compression method which is a list of compression methods the client supports.

After sending the client_hello message, the client waits for the server_hello message which contains the same parameters as the client_hello message but only one CipherSuite and one compression method are chosen.

The elements of Cipher Suite are key exchange method and CipherSpec which includes cipher algorithm, MAC algorithm, cipher type (stream or block), hash size, key material, IV size etc.

Phase 2. Server Authentication and Key Exchange

Server first sends its certificate message. Then a server_key_exchange message may be sent if it is required. The certificate_request message may be followed to request the client's certificate. Finally, server sends the server_done message which indicates the end of server hello.

Phase 3. Client Authentication and Key Exchange

Upon receipt of the server_done message, the client should verify the certificate and server_hello parameters. If everything is fine, then the client sends back messages to the server. First the client sends certificate message or no_certificate alert according whether the server requested a certificate. Next the client sends the client_key_exchange message which contains a 48-byte pre-master secret if RSA is used or public parameters of Diffie-Hellman scheme. We will explain pre-master secret later. Finally, the client may sends a certificate_verify message which provides verification of a client certificate.

Phase 4. Finish

This phase completes the setting up of a secure connection. The client sends a change_cipher_spec message and copies the pending CiperSpec into the current CipherSpec. Then the client sends the finished message under the new algorithms, keys, and secrets. The finished message is hash value of master_secret, handshake_message (which consists of all of the data from handshake messages up to but not including this message) and some other codes. The master_secret is computed from the pre_master secret. The master_secret is then used to generate the session keys which are used for encryption and authentication.

In response to these messages, the server sends its own change_cipher_spec

message and its finished message. At this point, the handshake is complete and the client and the server may begin to exchange application layer data using the master_secret as session key.

In SSL, the master_secret is created as follows. First the client and the server establish a pre_master_secret. There are two methods to do that. One method is using RSA system. In this case, the client generates a 48-byte pre_master_secret, encrypts that with the server's public RSA public key and sends the ciphertext to the server. Another method is using Diffie-Hellman key exchange scheme to create the pre_master_secret.

After both sides have the pre_master_secret, they compute the master_secret as follows.

```
MD5(pre_master_secret||SHA('A'||pre_master_secret||
ClientHello.random||serverHello.random))||
MD5(pre_master_secret||SHA('BB'||pre_master_secret||
ClientHello.random||serverHello.random))||
MD5(pre_master_secret||SHA('CCC'||pre_master_secret||
ClientHello.random||serverHello.random))
```

where ClientHello.radom and serverHello.random are two nonce values exchanged in the initial hello messages.

The common secret keys used for authentication and block encryption are formed from master secret as follows.

```
MD5(master_secret||SHA('A'||master_secret||
ClientHello.random||serverHello.random))||
MD5(master_secret||SHA('BB'||master_secret||
ClientHello.random||serverHello.random))||
MD5(master_secret||SHA('CCC'||master_secret||
ClientHello.random||serverHello.random))||...
```

until enough output for the key size.

TLS (Transport Layer Security) is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL. TLS is very similar to SSL. Netscape products also support TLS. Some difference of TLS from SSL are TLS using standard HMAC, TLS not supporting Fortezza, etc. TLS is described in RFC 3456.

8.2 Secure Electronic Transaction (SET)

SET is an open encryption and security specification designed to protect credit card transactions on the Internet. SSL secures communications between a client and a server. However, if we use SSL for credit card transaction, some problems might occur. For example, the information of a credit card might be used by the server for some purpose not desired for the client. On the other hand, a client might supply an invalid credit card number. SET is designed as a secure credit card payment system over internet, which protect both customers and merchants. SET is more complicated than SSL. A full description of SET needs a hundred pages. In this section, we just give a brief description of SET.

SET participants include the following:

- Cardholder: who hold a payment card (credit card)issued by an issuer. A cardholder will use the card to purchase over the internet.

- Merchant: A person or organization that has goods or service to sell to the cardholder. A merchant that accepts payment cards must have a relationship with an acquirer.

- Issuer: A financial institution that provides the cardholder with the payment card.

- Acquirer: A financial institute that establishes an account with a merchant and processes payment card authorizations and payments. Merchants will usually accept more then one credit card brand but do not want to deal with multiple bankcard associations or with multiple issuers. The acquirer provides authorization to the merchant that a given card is fine. The acquirer also pays the merchant's account and then reimbursed by the issuer.

- Payment gateway: A function operated by the acquirer or a third party that processes merchant payment messages. The payment gateway interfaces between SET and the existing bankcard payment network.

- Certification authority (CA): An entity that is trust to issue public key certificates (X. 509) for card holders, merchants, and payment gateways.

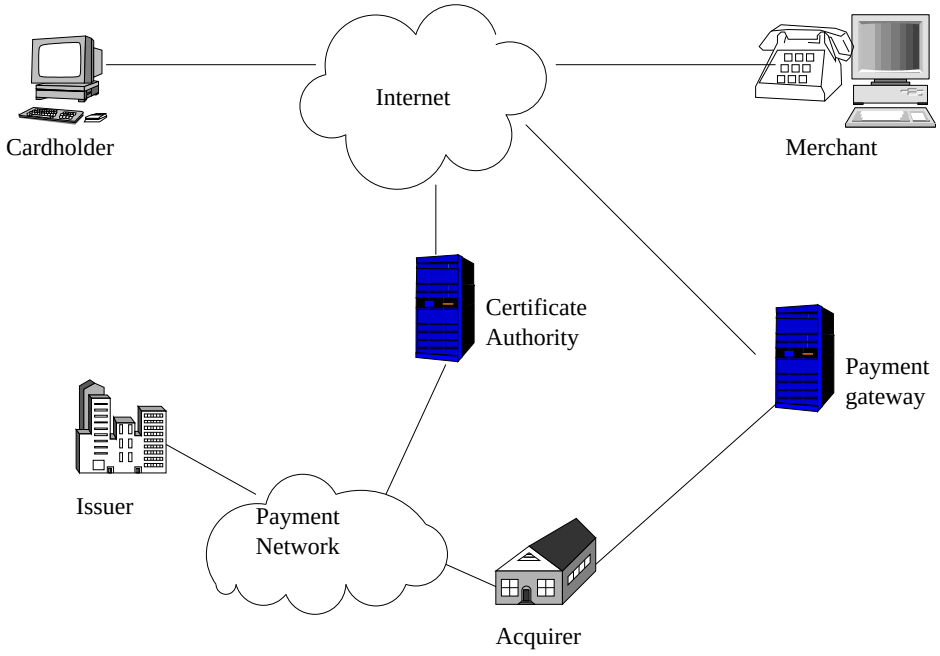


Figure 8.2: SET components

The components of SET can be described as in Figure 8.2.

Next we give an outline of the sequence of the events for a transaction using SET.

1. The customer opens an account. The customer obtains a credit card that supports electronic payment and SET.

2. The customer receives a certificate. The customer receives an digital certificate (X. 509) signed by the bank. The certificate verifies the customer's RSA public key and its expiration date. It also establishes a relationship between the customer's key pair and his/her credit card.

3. Merchants have their own certificates. A merchant must have two certificates for two public keys: one for signing messages and one for key exchange. The merchant also needs a copy of the payment gateway's public-key certificate.

4. The customer places an order. The customer sends a list of the items to

be purchased to the merchant through web site, who returns an order form containing the list of items, their price, and an order number.

5. The merchant is verified. The merchant sends a copy of its certificate, so that the customer can verify that he is dealing with a valid store.

6. The order and payment are sent. The customer sends both order and payment information to the merchant, along with the customer's certificate. The payment is encrypted in such a way that it cannot be read by the merchant. The customer's certificate enables the merchant to verify the customer.

7. The merchant requests payment authentication. The merchant sends the payment information to the payment gateway, requesting the authorization for this purchase.

8. The merchant confirms the order. The merchant sends confirmation of the order to the customer.

9. The merchant provides the goods or service.

10. The merchant requests payment. This request is sent to the payment gateway, which handles all of the payment processing.

Now we discuss some interesting innovation introduced in SET: the dual signature. The purpose of dual signature is to link two messages that are intended for two different recipients. In this case, the customer has two messages: the order information (OI) and the payment information (PI). OI is sent to merchant while PI is sent to the bank. However, there must be a link between OI and PI. For example, a customer does not want the merchant to know the PI but the merchant need to know that the customer provided the PI for the specific OI. SET uses a dual signature to solve that problem. Let h be a hash function (SHA-1). The dual signature (DS) is as follows.

$$\text{DS} = Sig_{K_c}(h(h(\text{PI})||h(\text{OI}))$$

where K_c is the customer's private signature key. Then customer can give OI, h(PI) and DS to the merchant and give PI, h(OI) and DS to the bank. Both merchant and bank can verify the signature since they know the customer's public key of the signature scheme. However, the merchant cannot substitute

another OI in this transaction for its advantage, since it is difficult to find another OI which has the same hash digest.

To encrypt the PI, the customer will choose a session key and uses that key for encryption. The session key is then encrypted using the payment gateway's public key. In this way, the merchant cannot decrypt the PI. However, the payment gateway can get the information after the merchant forwards these information to it.

SET provided 14 transaction types. We omit the details here. Unlike SSL and SSH, the cryptographic algorithms used in SET are fixed. It uses DES, RSA signature using SHA-1, HMAC based on SHA-1 and X. 509v3 digital certificate.

SET is a very comprehensive and complicated security protocol. It addresses all the parties involved in typical credit card transactions. To realize SET, every party needs to have some part of the software, even very expensive hardware. That is why SET is not widely spread although people believe that it is safe. How to simplify the SET while keep the main security features is still an open problem.

Chapter 9

IP Secure

Network architecture is usually explained as a stack of different layers. Figure 9.1 explains the OSI (Open System Interconnect) model stack and IP (Internet Protocol) model stack. TCP is Transmission Control Protocol, IP is Internet Protocol, UDP is User Datagram Protocol, data link including Ethernet, PPP, FDDI etc.

<table>
<tr><th>OSI model</th><th>IP model</th></tr>
<tr><td>Application</td><td rowspan="2">Application</td></tr>
<tr><td>Presentation</td></tr>
<tr><td>Session</td><td rowspan="2">TCP/UDP</td></tr>
<tr><td>transport</td></tr>
<tr><td>Network</td><td>IP</td></tr>
<tr><td>Data Link</td><td>Data link</td></tr>
<tr><td>Physical</td><td>Physical</td></tr>
</table>

Figure 9.1: Network layer stack

We have seen some security protocols in application level (PGP, S/MIME, etc.) and Transport level (SSL, TLS). In this chapter, we investigate IP level security.

9.1 TCP/IP Protocol

The method of communication on the network is to send and receive chunks of data called *packets*. A packet is comprised of small chunks of data that each layer appends onto that packet data it received from the layer directly above it.

A TCP/IP packet can be described as follows.

Link-H	IP-H	TCP-H	Data	Link-T

Where H means header of that layer, Link-T is the tail of the link. Usually, when a sending packet is formed, the application data is first generated, then different headers for different layers are added. To secure communications, we need to encrypt the packets.

Network security encryption can be classified into two types.

- End-to-end encryption: The encryption process is carried out at the two end systems.
- Link encryption: The encryption process is carried in each link.

Figure 9.2 is a simple example of these two types of encryption.

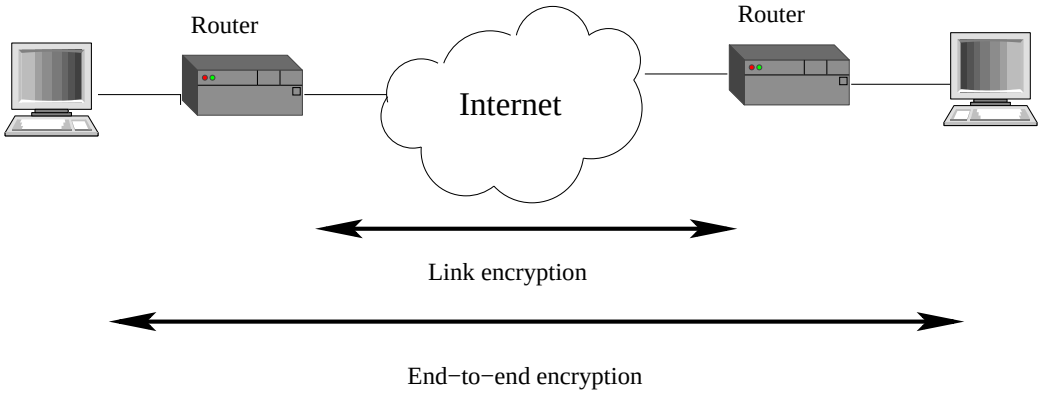


Figure 9.2: Types of Encryption

End-to-end encryption is simple, but it cannot perform at a low level of the communication hierarchy. The address of the message cannot be encrypted, otherwise the packet-switching node cannot route the packet. So end-to-end encryption cannot protect against the traffic analysis attack.

The link encryption can encrypt most data except the link control protocol header. However, the router has to decrypt the data and then encrypt it again.

IPSec considers the security at IP layer. It can be used in a firewall or router. It also can be used for individual users. So IPSec can be used for both end-to-end encryption or link encryption. Since IPSec is below the transport layer, it can be transparent to end users. So there is no need to change the application software or train users on security mechanisms.

Before discussing the IPSec, we need some knowledge of IP. An internet protocol (IP) is used for transmit packets across multiple networks. The main internet protocol is IPv4. The IP header of IPv4 is shown in Figure 9.3

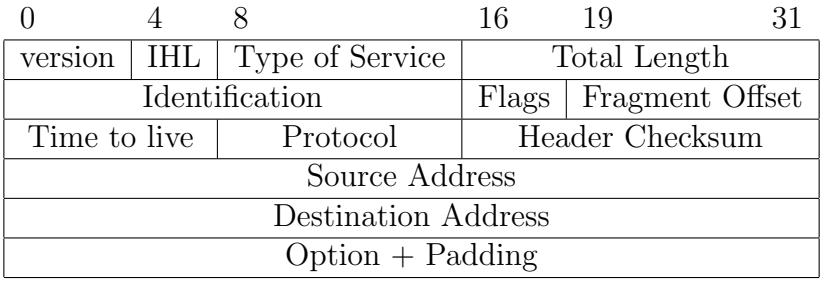


Figure 9.3: IPv4 header

The size of the IPv4 header is a minimum of 20 octets, or 160 bits. The items in IPv4 header are as follows.

- Version (4 bits): The version of IP, the value is 4.
- Internet Header Length (IHL) (4 bits): Length of header in 32-bit words. The minimum value is 5.
- Type of Service (8 bits): Provides guidance to end IP modules and to routers along the packet's path about the packet's relative priority.
- Total length (16 bits): Total IP packet length, in octets.
- Identification (16 bits): A sequence number identifies the packet, together with the source address, destination address and user protocol.
- Flags (3 bits): Indicates whether it is the last fragment of the original packet.

- Fragment Offset (13 bits): Indicate where in the original packet this fragment belongs, measured in 64-bit units.

- Time to live (8 bits): Specifies how long a packet is allowed to remain in the internet.

- Protocol (8 bits): Indicates the next higher level protocol.

- Header Checksum (16 bits): An error-detecting code (for the header only). Since some header fields may change during transit, this is reverified and recomputed at each router.

- Source Address (32 bits): Coded to allow a variable allocation of bits to specify the network and the end system attached to the specified network.

- Destination Address (32 bits): Same characteristics as source address.

- Options (variable): Encodes the options requested by the sending user, such as security label, source routing, record routing, and timestamping.

- Padding (variable): Used to ensure that the packet header is a multiple of 32 bits in length.

A new version of IP was developed as a standard by IETF (the Internet Engineering Task Force), which is known as IPv6. IPv6 header uses fewer fields than IPv4, that lets the router treat the packet faster. IPv6 provides more space for source and destination addresses, which uses 16 bytes each (128 bits). An IPv6 also includes zero or more extension headers such as hop-by-hop option header, router header, fragment header, authentication header, encapsulating security payload header, destination option header, etc. Separated extension headers may be placed between the IPv6 header and the upper- layer header in a packet. The IP header of IPv6 is shown in Figure 9.4.

IPv6 is still in developing.

IP-level security encompasses three functional areas: authentication, confidentiality and key management.

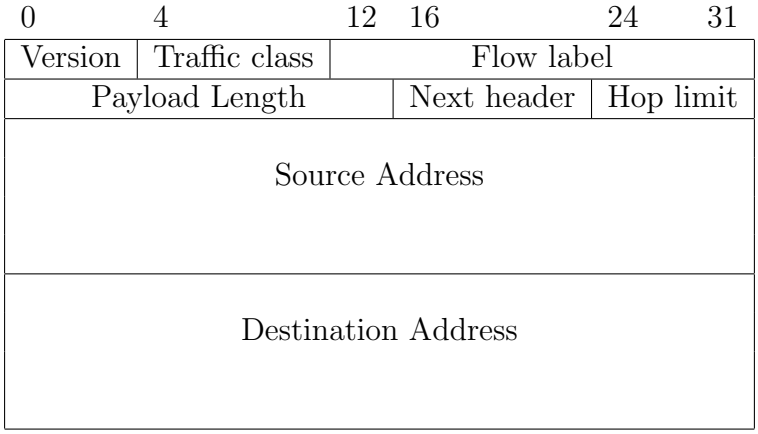


Figure 9.4: IPv6 header

9.2 IPSec documents

Documents of IPSec are developed by IETF as RFC standard. The documents are divided into seven groups as follows.

- Architecture: Covers general concepts, security requirements, definitions and technology.
- Encapsulating Security Payload(ESP): Covers the packet format, packet encryption and, optionally, authentication.
- Authentication Head (AH): Covers the packet format and packet authentication.
- Encryption Algorithm: Describes various encryption algorithms used for ESP.
- Authentication Algorithm: Describes various authentication algorithms for AH and ESP.
- Key Management: Describes key management schemes.
- Domain of Interpretation (DOI): Contains values needed for the other documents to related to each other.

We will not discuss all these documents, but select several most important specifications.

An important concept that appears in IPSec is the security association (SA). An association is a one-way relationship between a sender and a receiver. If a two-way exchange is needed, then two SA are required. An SA is uniquely identified by three parameters.

- Security Parameters Index (SPI): A bit string assigned to this SA, which is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.
- IP Destination Address: This is the address of the destination endpoint of the SA (end user, firewall or router).
- Security Protocol Identifier: This indicates whether the SA is an AH or ESP security association.

9.3 Authentication Header

The Authentication Header (AH) provides support for data integrity and authentication of IP packets. The authentication header is defined as in Figure 9.5

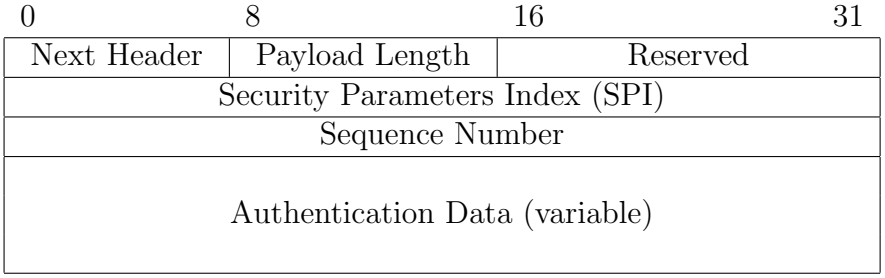


Figure 9.5: IPSec Authentication Header

The fields of AH are as follows.

- Next Header (8 bits): Identifies the type of header immediately following this header.

- Payload Length (8 bits): Length of AH in 32-bit words, minus 2. The default length of the authentication data field is three 32-bit words. So the default value of Payload Length is 4.
- Reserved (16 bits): For future use.
- Security Parameters Index (32 bits): Identifies a security association.
- Sequence Number (32 bits): A monotonically increasing counter value to prevent replay attack.
- Authentication Data (variable, must be an integral number of 32-bit words): Contains the Integrity Check Value (ICV), or MAC, for this packet. The default length of this field is 3 32-bit word.

Sequence numbers are used for anti-replay service. An attacker might obtain a copy of a packet and later transmits it to the destination. Thus when a new SA established, sender initializes a sequence number counter to 0. Each time that a packet is sent on this SA, the sender increases the counter and place the value in the Sequence Number field. If the number reaches to $2^{32} - 1$, a new SA with a new key needs to be set.

Note that IP is connectionless. So the packets may not be delivered in order and some packets might be missing. So the IPSec authentication document dictates that the receiver should implement a window of size W (default value of W is 64). The right edge of the window represents the largest sequence number, N, received so far. For any packet with a sequence number in the range from $N - W + 1$ to N that has been correctly received, the corresponding slot in the window is marked. When a packet is received, the receiver does the following.

- If the received packet falls within the window and is new, the MAC is checked and the corresponding slot in the window is marked if the authentication is good.
- If the received packet is to the right of the window, The MAC is check. If the MAC is good, the window is advanced so that this sequence number is the right edge of the window.
- If the received packet is to the left of the window, or if the authentication fails, the packet is discarded.

A receiver's window is shown in Figure 9.6

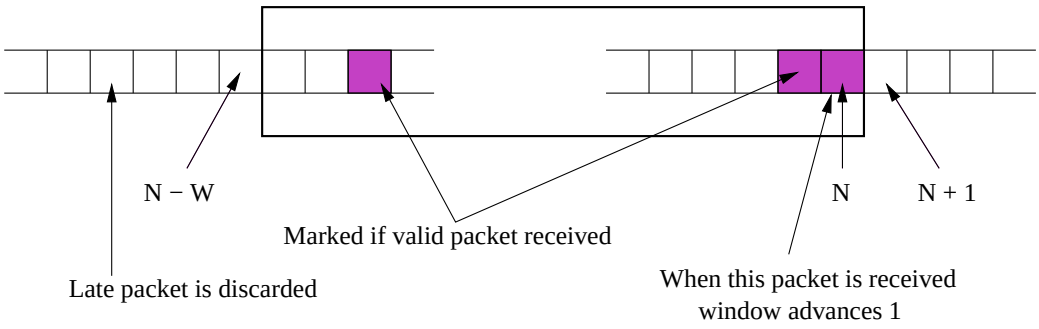


Figure 9.6: A Receiver's Window

An ICV is a MAC (or HMAC) or a truncated version of a code produced by a MAC (or HMAC) algorithm. For example, an HMAC-MD5 or HMAC-SHA-1 is used to produce the code and then the first 96 bits is truncated. The input of the MAC algorithm includes those fields which will not be changed in transit (immutable) or that are predictable in value upon arrival at the endpoint. That includes immutable fields in IP header, AH header other than Authentication data and upper-level protocol data. The mutable fields are set to zero when an ICV is computed.

For example, in IPv4 header, the Internet Header Length (IHL) and Source Address are immutable. The Destination Address is predictable field. The Time to Live and Header Checksum fields are mutable fields, which are zeroed prior to compute ICV.

Note that the AH uses MAC, so there is a shared secret key. We will discuss the key management later.

There are two modes in which the IPSec authentication service can be used.

- Transport Mode: Used in end-to-end authentication (e.g., server to client).
- Tunnel Mode: Used in end-to-intermediate authentication (e.g., workstation to firewall).

These two modes are explained in Figure 9.7

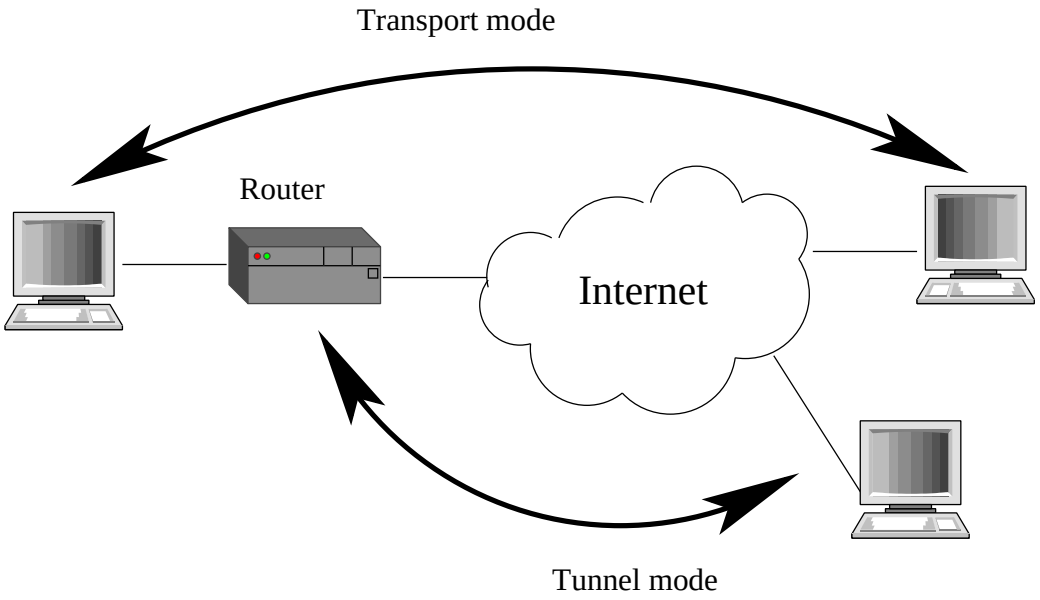


Figure 9.7: IPSec modes

The position of AH at the packet is as follows. A simple packet before applying AH looks like the following.

IPv4	IP-H	TCP-H	Data	
IPv6	IP-H	Extension H	TCP-H	Data

Then the packet after applying AH in transport mode is

IPv4	IP-H	AH	TCP-H	Data	
IPv6	IP-H	ext H	AH	dest	TCP-H	Data

For the tunnel mode, the packet after applying AH looks as

IPv4	New IP-H	AH	orig IP-H	TCP-H	Data	
IPv6	New IP-H	ext H	AH	orig IP-H	dest	TCP	Data

Basically, all the fields are authenticated except for mutable fields which are set to be zero values before using HMAC.

9.4 Encapsulating Security Payload (ESP)

The ESP provides confidentiality services. As an optional feature, ESP can also provide authentication service.

The format of ESP is as in Figure 9.8

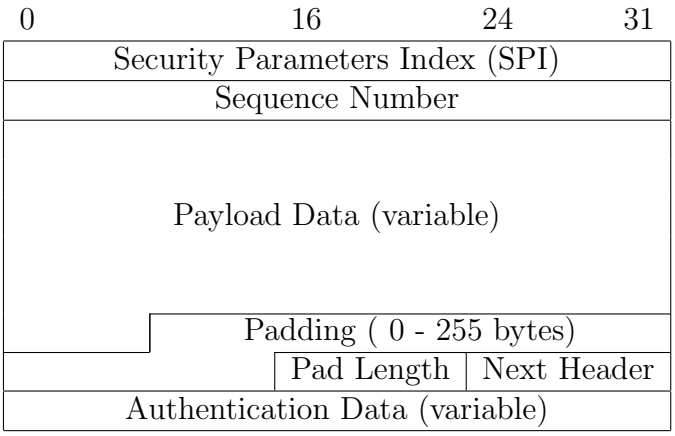


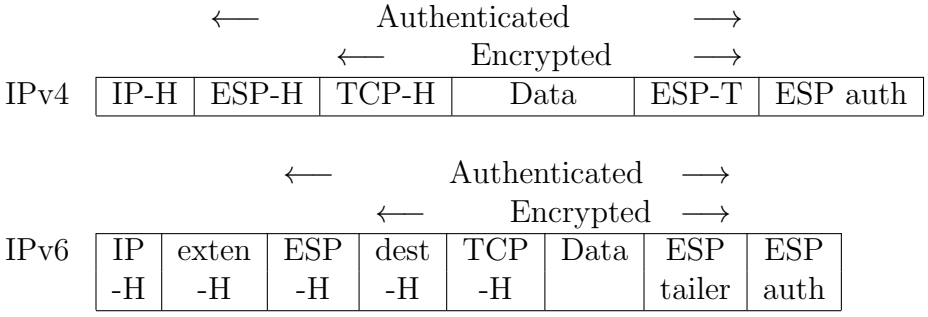
Figure 9.8: IPSec ESP Format

SPI identifies a security association. Sequence number is similar to that of AH. These two 32-bit words are the head of ESP. Payload Data is a transport level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption. The length of payload data is variable. The length of padding is between 0 to 255 bytes. The padding is used to satisfies the requirement of encryption function and the requirement of the alignment of the ESP format. Sometimes, padding also can be used to provide partial traffic flow confidentiality by concealing the actual length of the payload. Pad length (8 bites) indicates the number of pad bytes. Next header (8 bits) identifies the type of data contained in the payload data field (first header in that payload). The Authentication Data field contains the ICV.

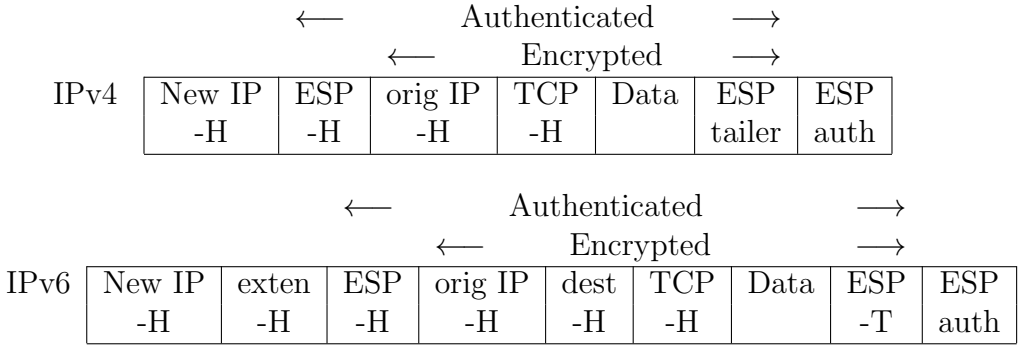
The Payload Data, Padding, Pad Length and Next Header fields are encrypted. If the encryption algorithm needs some initialization vector (IV),

then this data may be carried explicitly at the beginning of the Payload Data field. In that case, the encryption is usually started right after IV.

Then the packet after applying ESP in transport mode is



For the tunnel mode, the packet after applying ESP looks as



When the authentication option is chosen, the authenticated part covers from the ESP header to the ESP trailer.

In practice of using IPSec, we can use different combinations of AH and ESP. We can use ESP with authentication. In this case, the authentication is applied to the ciphertext instead of plaintext. We also can use two SAs. One is for AH and one is for ESP. The inner is used for ESP without authentication and the outer is for AH. Sometimes, we can use several SAs. The IPSec architecture documents lists four examples of combinations of SAs that must be supported by compliant IPSec hosts.

Example 1 implements IPSec for both end systems. Several SA may be used. They can use AH in transport mode, ESP in transport mode, AH followed by ESP in transport mode, etc. (see Figure 9.9).

Example 2 implements IPSec only for gateways such as routers, firewalls etc. In this case, usually only a single tunnel SA is used which supports AH,

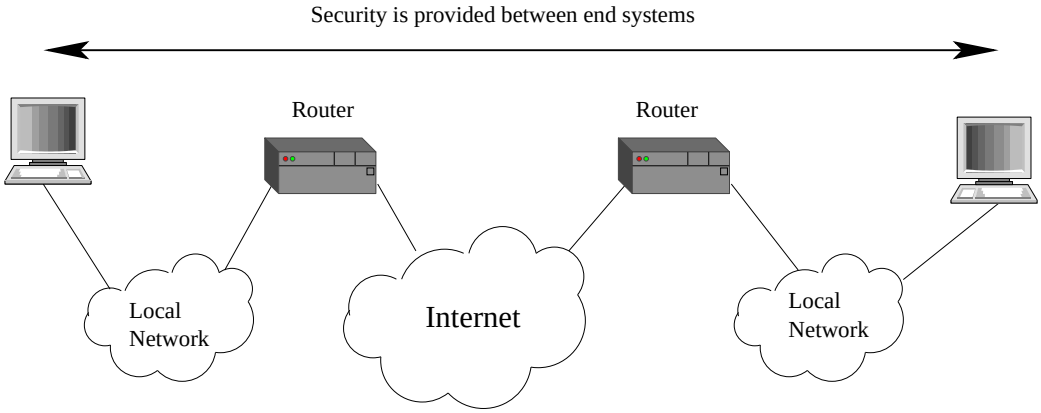


Figure 9.9: Example 1

ESP or ESP with the authentication option. This implementation can be used to support simple virtual private network (see Figure 9.10).

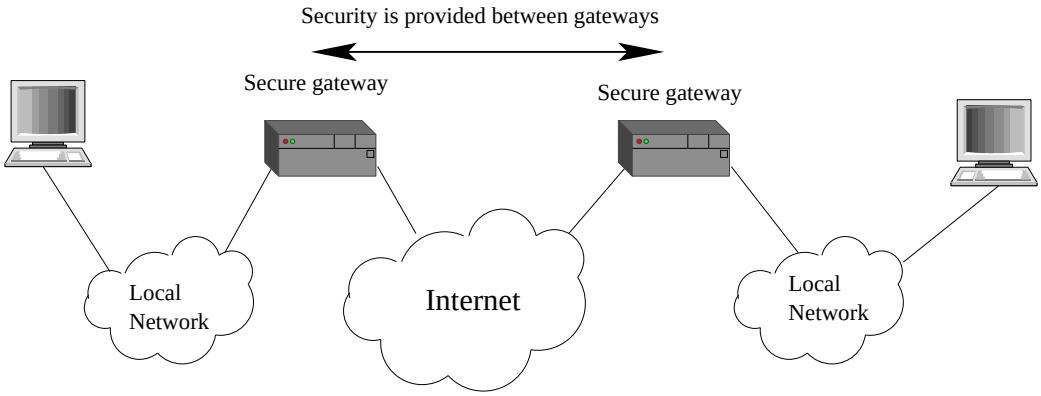


Figure 9.10: Example 2

Example 3 combines example 2 with example 1. One or several end-to-end SA is added to the gateway-to-gateway security (see Figure 9.11).

Example 4 supports a remote host to reach an organization's firewall and then to access the end system behind the firewall. A tunnel mode is used between the remote host and the firewall. (see Figure 9.12).

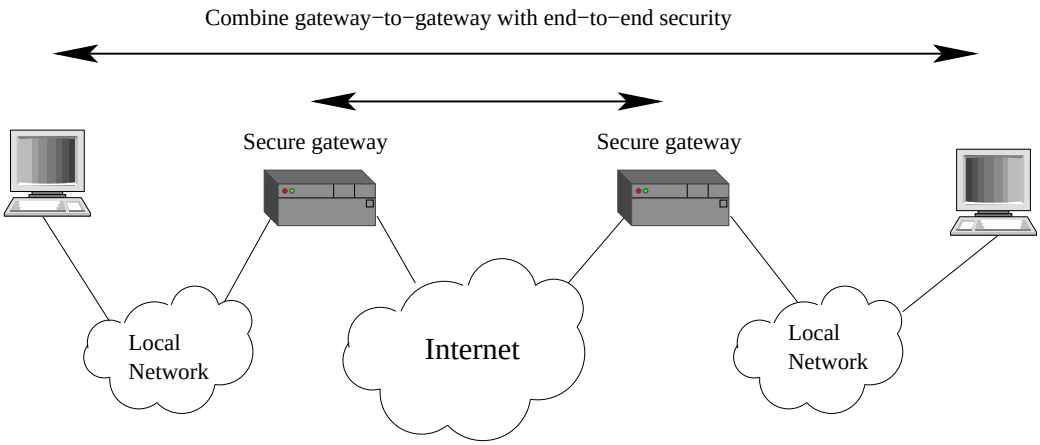


Figure 9.11: Example 3

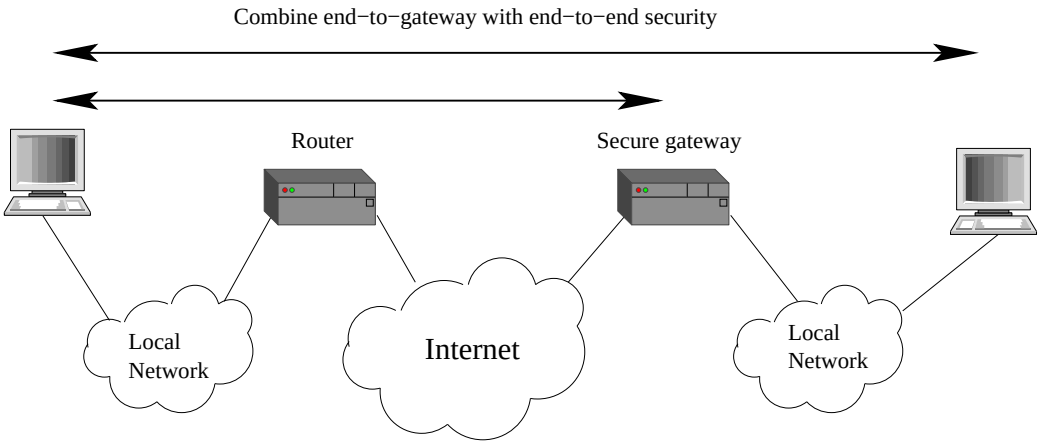


Figure 9.12: Example 4

9.5 Key Management

In AH and ESP, secret keys are required for communications. An important part of IPSec is key management.

IPSec supports two types of key management for both AH and ESP:

- Manual: A system administrator manually configures each system with its own keys and with the keys of other communicating systems.
- Automated: An automated system enables the on-demand creation of keys for SA and facilities the use of keys in a large system.

IPSec's default automated key management protocol is ISAKMP/Oakley. The Oakley key exchange protocol was discussed before, which is based on Deffie-Hellman key exchange scheme. Now we introduce ISAKMP key management. ISAKMP provides a framework for internet key management. It does not dictate a specific key exchange algorithm. Other key exchange algorithms can also be used with ISAKMP.

ISAKMP (Internet Security Association and Key Management Protocol) defines procedures and packet formats to establish, negotiate, modify and delete security associations.

The ISAKMP header format is shown in Figure 9.13

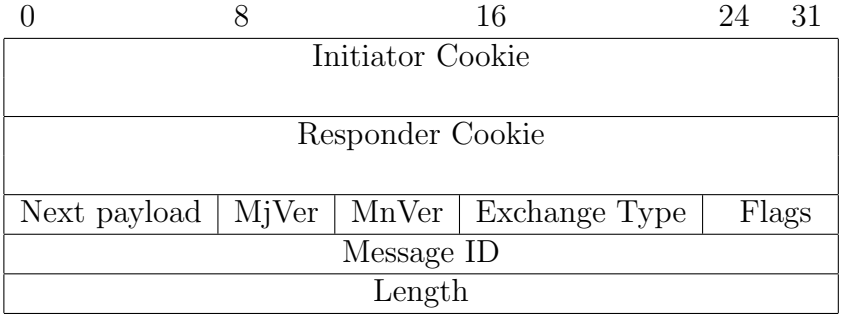


Figure 9.13: ISAKMP header

Initiator Cookie (64 bits) initiated SA establishment, SA notification, or SA deletion. Responder Cookie(64 bits) is used for responding; null in the first message from initiator. Next Payload (8 bits) indicates the type of the first payload in the message, which is followed the ISAKMP header. Major Version (4 bits) indicates major version of ISAKMP in use and Minor Version

(4 bits) indicates minor version in use. Exchange Type (8 bits) indicates the type of exchange. Flags (8 bits) indicates specific options set for this ISAKMP exchange. Message ID (32 bits) is the unique ID for this message. Length (32 bits) is the length of total message (header plus all payloads) in octets.

ISAKMP Payload is followed the ISAKMP header. All ISAKMP payloads begin with the same generic payload header which is shown in Figure 9.14

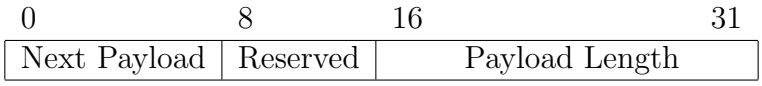


Figure 9.14: ISAKMP Payload Header

Next Payload field uses 8 bits which has value 0 if this is the last payload in the message, otherwise its value is the type of the next payload. The Payload Length field indicates the length in octets of this payload, including the generic payload header.

The payload types defined for ISAKMP are as follows.

- SA payload (SA): Used to begin the establishment of an SA. Parameters include Domain of Interpretation (e.g., IPSec DOI), and a situation parameter which defines the security policy for this negotiation.

- Proposal payload (P): Contains information used during SA negotiation. The payload indicates the protocol for this SA (ESP or AH), entity's SPI and the number of transforms.

- Transform payload (T): Includes a transform number which identifies this particular payload. The payload also contains the transform ID and Attributes to specify the transform (e.g., 3DES for ESP, HMAC-SHA-1-96 for AH) with its associated attribution (e.g., hash length).

- Key Exchange payload (KE): Used for key exchange (Oakley, PGP etc.). The payload contains the data required to generate a session key.

- Identification payload (ID): Used to identify the communicating peers. The Data field will contain an IPv4 of IPv6 address.

- Certificate payload (CERT): Transfers public-key certificate. The payload indicates the type of certificate of certificate-related information.
- Certificate Request payload (CR): Requests the certificate of the other communicating entity. Lists certificate types and certificate authorities that are acceptable.
- Hash payload (HASH): Contains data generated by a hash function over some message.
- Signature payload (SIG): Contains data generated by a digital signature scheme.
- Nonce payload (NONCE): Contains random data to protect against replay attacks.
- Notification payload (N): Contains either error or status information associated with this SA.
- Delete payload (D): Indicates one or more SAs that sender has deleted from its database.

RFC 2408 lists five default exchange types which are described in Figure 9.15. In the table SA refers to an SA payload with associated Protocol and transform payload. NONCE is a random number used to ensure against replay attacks. AUTH payload is used to authenticate keys, identities and the nonce. In the Identity Protection Exchange, the two parties identities are protected by encryption. The Aggressive Exchange minimizes the number of exchanges at the expense of not providing identity protection.

We use the diagram in Figure 9.16 to illustrate the payloads exchanged between the two parties in the first round trip exchange. In this example, the initiator propose two proposals. The responder should reply with one proposal. This round is to initiate an SA. This example is the Identity Protection Exchange in ISAKMP.

The second round trip exchange is to generate key and send nonce. We explain it in Figure 9.17

We omit the third trip round. It is not difficult to describe this round now.

The Internet Key Exchange (IKE, RFC 2409) further detailed the key exchange scheme. The IKE uses part of Oakley and part of SKEME (Secure Key Exchange MEchanism protocol, a versatile key exchange technique

(a) Base Exchange

(1) $I \rightarrow R$: SA; NONCE	Begin ISAKMP-SA or Proxy negotiation
(2) $I \leftarrow R$: SA; NONCE	Basic SA agreed upon
(3) $I \rightarrow R$: KE;IDi; AUTH	Key Generated (by responder) Initiator Identity Verified by Responder
(4) $I \leftarrow R$: KE; IDr; AUTH	Responder Identity Verified by Initiator Key Generated (by initiator) SA established

(b) Identity Protection Exchange

(1) $I \rightarrow R$: SA	Begin ISAKMP-SA or Proxy negotiation
(2) $I \leftarrow R$: SA	Basic SA agreed upon
(3) $I \rightarrow R$: KE; NONCE	Key generated
(4) $I \leftarrow R$: KE; NONCE	Key generated
(5)*$I \rightarrow R$: IDi; AUTH	Initiator Identity Verified by Responder
(6)*$I \leftarrow R$: IDr; AUTH	Responder Identity Verified by Initiator SA established

(c) Authentication Only Exchange

(1) $I \rightarrow R$: SA;NONCE	Begin ISKMP-SA or Proxy negotiation
(2) $I \leftarrow R$: SA;NONCE;IDr;AUTH	Basic SA agree upon Responder identity verified by Initiator
(3) $I \rightarrow R$: IDi;AUTH	Initiator identity verified by responder; SA established

(d) Aggressive Exchange

(1) $I \rightarrow R$: SA;KE;NONCE;IDi	Begin ISKMP-SA or Proxy negotiation and key exchange
(2) $I \leftarrow R$: SA;KE;NONCE;IDr;AUTH	Initiator identity verified by responder; Key generated; Basic SA agreed upon
(3)* $I \rightarrow R$: AUTH	Responder identity verified by initiator; SA established

(e) Informational Exchange

(1) $I \rightarrow R$: N/D	Error notification or deletion

Notation:
I = initiator
R = responder
* = payload encryption after the ISAKMP header

Figure 9.15: ISAKMP Exchange Types

ISAKMP Header with Exchange Type of Main Mode and Next Payload of ISA-SA		
0	Reserved	Payload Length
Domain of Interpreting		
Situation		
0	Reserved	Payload Length
Proposal # 1, PROTO-ISAKMP, SPI size=0 || # of Transforms		
ISA-TRANS	Reserved	Payload Length
Transform # 1, KEY-OAKLEY, preferred SA attributes		
0	Reserved	Payload Length
Transform # 2, KEY-OAKLEY, preferred SA attributes		

Figure 9.16: ISAKMP exchange round 1

ISAKMP Header with Exchange Type of Main Mode and Next Payload of ISA-KE		
ISA-NONCE	Reserved	Payload Length
D-H public Value (α^x from initiator α^y from responder)		
0	Reserved	Payload Length
$NONCE_I$ or $NONCE_R$		

Figure 9.17: ISAKMP exchange round 2

which provides anonymity, repudiability and quick key refreshment) in conjunction with ISAKMP to obtain authenticated keying material. Two phases of exchange are defined in IKE. Phase 1 is where the two ISAKMP peers establish a secure, authenticated channel with which to communicate. Phase 2 is where Security Associations are negotiated on behalf of services such as IPsec or any other service which needs key material and/or parameter negotiation. Two modes of phase 1, Main mode and Aggressive mode, are described in IKE. For phase 2, IKE describes Quick mode. There are different authentication options for each mode. We will not discuss all the details, but give some examples to explain.

An example of Main Mode is authenticated with a revised mode of public key encryption. This mode is defined as follows.

1. $I \rightarrow R$: HDR, SA.

   Where HDR is the header of ISAMKP whose exchange type is the mode.

2. $I \leftarrow R$: HDR, SA.

3. $I \rightarrow R$: HDR, $[HASH(1)]$, $E_{Pub_R}(NONCE_I)$, $E_{K_I}(KE)$, $E_{K_I}(ID_I)$, $[E_{K_I}(CERT_I)]$.

   Where [x] indicates that x is optional. $HASH(1)$ is a hash (using the negotiated hash function) of the certificate which the initiator is using to encrypt the nonce and identity. E_{Pub_R} is the encryption function using R's public key Pub_R. $NONCE_I$ is I's nonce payload. E_{K_I} is a symmetric encryption algorithm (from the SA payload) with I's secret key K_I which is derived from the nonce.

4. $I \leftarrow R$: HDR, $E_{Pub_I}(NONCE_R)$, $E_{K_R}(KE)$, $E_{K_R}(ID_R)$.

   E_{Pub_I} is the encryption function using I's public key Pub_I. $NONCE_R$ is I's nonce payload. E_{K_R} is a symmetric encryption algorithm (from the SA payload) with R's secret key K_R which is derived from the nonce.

5. $I \rightarrow R$: HDR*, $HASH_I$.

   HDR* indicates payload encryption. $HASH_I$ is the hash payload produced from HMAC. The key is derived from I's cookie and the nonce. The hash function is performed on the Diffie-Hellman key exchange values, cookies, SA and I's ID.

6. $I \leftarrow R$: HDR*, $HASH_R$.

   $HASH_R$ is similar to $HASH_I$.

The Quick Mode is defined as follows. The payload in this mode is encrypted. Since this mode is basically in phase 2, the encryption is possible.

1. $I \rightarrow R$: HDR*, $HASH(1)$, SA, $NONCE_I$, [, KE], [, ID_{cI}, ID_{cR}].

   If ISAKMP is acting as a client negotiator on behalf of another party, then the identities are passed as ID_{cI} and ID_{cR}.

2. $I \leftarrow R$: HDR*, $HASH(2)$, SA, $NONCE_R$, [, KE], [, ID_{cI}, ID_{cR}].

   $HASH(2)$ is the hash payload of message ID, SA, $NONCE_R$ and optional KE, ID_{cI}, ID_{cR}

3. $I \rightarrow R$: HDR*, $HASH(3)$

   $HASH(3)$ is the hash payload of message ID, $NONCE_I$ and $NONCE_R$.

ISAKMP is not only for IPSec. It can be used for other security issues of internet. For example, is it used for VPN (virtue private network).

Chapter 10

Firewall

Firewalls are devices used to protect a local network from network based security threats while at the same time affording access to the wide area network and the internet. Basically, firewall provides access control of a local system according to specific policies. In this chapter we give some overview of firewalls. Figure 10.1 explains the position of a firewall.

10.1 Some Characteristics of firewall

The definition of a firewall depends on how and to what extent a firewall is used in a network. In general, the design goals for a firewall are:

- All traffic from inside to outside, and vice versa, must pass through the firewall.
- Only authorized traffic, defined by the local security policy, will be allowed to pass the firewall. A firewall usually has a good logging facility and notification abilities.
- The firewall itself is immune to penetration. This implies that use of a secure operating system, keep patching the system regularly, secure administrative access, etc.

Some general techniques that firewalls used are as follows.

- Service control: Determines the types of internet services that can be accessed. The firewall may filter traffic on the basis of IP address and TCP port number.

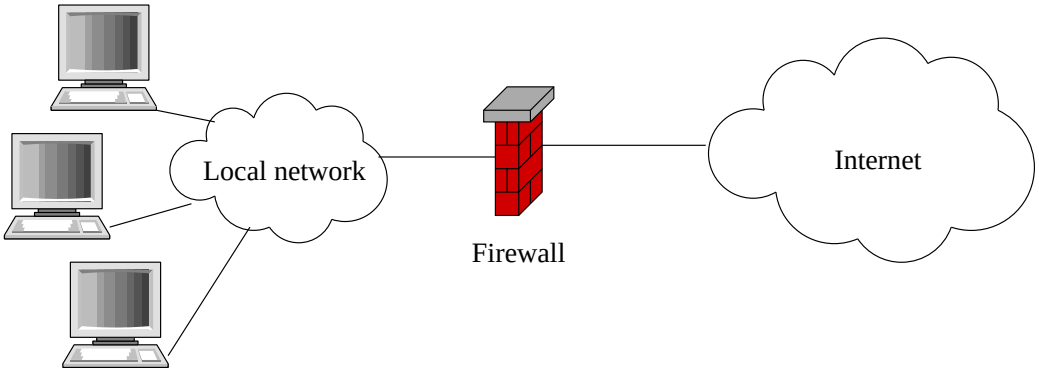


Figure 10.1: A Firewall

- Direct control: Determines the direction in which particular services requests may be initiated and allowed to go through the firewall.

- User control: Controls access to a service according to which user is attempting to access it. This is usually applied to inside users. For incoming traffic from outside of the firewall, some protocols are required such as IPSec.

- Behaviour control: Controls how particular services are used. For example, it may enable external access to only a portion of the information on a local Web server.

It should be noted that firewalls only can protect certain kind of attacks from the internet. They have their limitations as follows.

- The firewall cannot protect against attacks that bypass the firewall. For example, dial-out connection will not go though the firewall.

- The firewall does not protect against internal threats.

- The firewall cannot protect against the transfer of virus-infected programs of files. It would be impractical for the firewall to scan all incoming files, e-mails, etc.

10.2 Common Types of Firewall

A basic function for a firewall is to check the TCP(UDP) and IP headers of a packet according to security rules. A typical TCP header is shown in Figure 10.2.

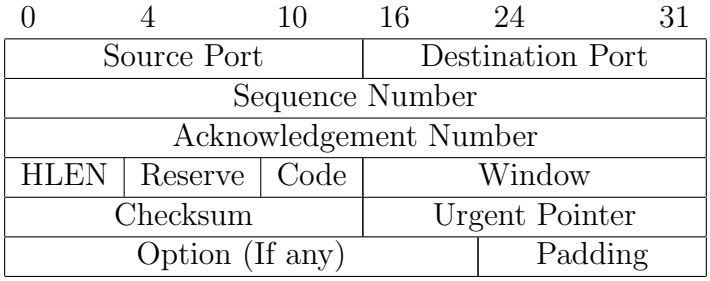


Figure 10.2: TCP header

The source port and destination port fields contain the TCP port numbers that identify the application program at the two ends of the connection. The sequence number identifies the position in the sender's bytes stream of the data in the segment. The acknowledgment number identifies the number of the byte the source expects to receive next. HLEN specifies the length of the segment header. The window field contains the buffer size that limits the data TCP software willing to accept every time it sends a segment. The code field uses a 6-bit code to determine the purpose and contents of the segment.

There are several types of firewalls. In most cases, a firewall is a combination of software and hardware. Now we describe some common types of firewalls below.

(a) *Packet-filtering*

A basic firewall uses packet-filtering routers. The router applies a set of rules to each incoming IP packet and then forwards or discards the packet. It is usually designed to filter packets going in both directions. Filtering rules are based on fields in the IP or transport header, including source and destination IP addresses and TCP or UDP port numbers. The filter is set up as a list of rules to determine whether to permit or block a packet. When a packet comes, the router checks whether it matches one of the rules. The rules are checked from top to bottom on the list. If a rule is matched, then

the rule is invoked. Otherwise, a default action is called.

Two possible default policies can be set: discard or forward. The default discard policy is more conservative.

Figure 10.3 shows the algorithm of the packet filtering.

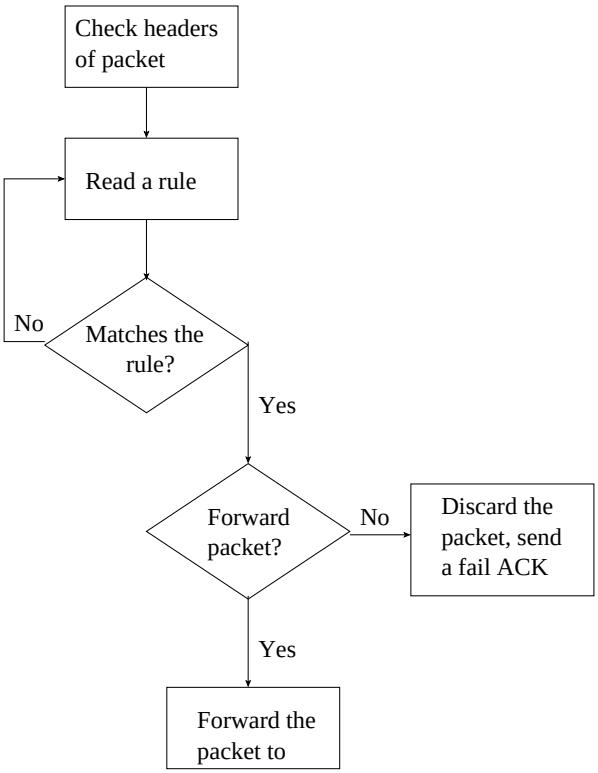


Figure 10.3: Packet Filtering Algorithm

Now let us see some examples of packet filtering rules. In each set, the rules are applied top to down.

Example A.

action	ourhost	port	theirhost	port	comment
block	*	*	SPIGOT	*	Don't trust these people
allow	OUR-GW	25	*	*	connect to our SMPT
block	*	*	*	*	default

In this example, the inbound mail is allowed, but only to a gateway host (OUR-GW). Mail from a particular host, SPIGOT, is blocked. The default policy is discard.

Example B.

action	ourhost	port	theirhost	port	comment
allow	*	*	*	25	connect to their SMTP

This rule set is intended to specify that any inside host can send mail to the outside.

Example C.

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	25		to their SMTP
allow	*	25	*	*	ACK	their replies

This set of rules takes advantage of a feature of TCP connections. Once a connection is set up, the ACK flag of a TCP segment is set to acknowledge segments sent from the other side. The rules allow our hosts to send packet to destination with TCP port 25 and allow incoming packets with a source port number of 25 that include the ACK flag in the TCP segment.

Example D.

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calles
allow	*	*	*	> 1024		traffic to nonservers

This rule set is used to handling FTP connections. With FTP, usually two TCP connections are used. Port 20 or 21 is used to set up the file transfer. A data connection uses a different port number that is dynamically assigned for the transfer. Since most servers live on low-numbered ports, most outgoing calls tend to use a port above 1023. This rule set allows our hosts to call external machine and receive the reply packets. And other incoming packets for high-numbered port on an internal machine are allowed. This scheme requires that the systems are configured so that only appropriate port numbers are in use.

On a Cisco router, the access control lists (ACL) are typed in. For example, the following two lines of ACL allows any packet with a destination IP address of 216.211.73.222 and port 80 (HTTP) but denies the other IP packets with that IP address:

```
access-list 101 permit tcp any 216.211.73.222 0.0.0.0 eq 80
access-list 101 deny ip any 216.211.73.222 0.0.0.0 - r u
```

The advantages of packet-filtering router are simple and fast. The disadvantages are lack of authentication and the difficulty of setting up packet filter rules correctly.

Stateful packet filters are more intelligent than simple packet filters. A stateful packet filter can block pretty much all incoming traffic and still can allow return traffic for the traffic generated by inside hosts. To do that, a record of the transport layer connections that are established through them are kept. An example of connection state table is shown in Figure 10.4.

source address	source port	dest address	dest port	connect state
129.168.1.100	1030	210.9.88.29	80	established
129.168.1.102	1031	216.32.42.123	80	established
129.168.1.101	1033	173.66.32.122	25	established

Figure 10.4: Stateful Firewall State Table

Usually, an application that creates a TCP connection uses a port number less than 1024 for the remote server but a port number between 1024 and 16383 for local client. If we permit inbound network traffic on all these high-numbered TCP ports, then a vulnerability occurs. The stateful packet filter will allow incoming traffic to high-numbered ports only for those packets that fit the profile of one of the entries in the table.

(b) *Application-level Gateway*

An application-level gateway is also called a proxy server. The user contacts the gateway using a TCP/IP application and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the remote host and relays the application data between the two endpoints. If the gateway does not implement the proxy code for a specific application, the service is not supported and cannot be forwarded across the firewall. The gateway can be configured to support only specific features of

applications.

Application-level gateways tend to be more secure than packet filters because they are aware of application-level protocols and they can restrict or allow access based on these protocols. They can also look into the data portion of the packets and use that information to restrict access. The disadvantage of application-level gateway is the additional processing overhead on each connection which slows down the communications.

(c) *Circuit-level Gateway*

A circuit-level gateway does not permit an end-to-end TCP connection. The gateway sets two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outside host. The firewall intercepts TCP connections being made to a host behind it and completes the handshake on behalf of this host. The security function consists of determining which connection will be allowed. Once the two connections are established, the gateway usually will not exam the TCP segment.

A typical use of circuit-level gateway is in a situation in which the internal users are trusted. Then the gateway can be configured to support circuit-level functions for outbound connections and proxy service on inbound connections (i.e., check incoming data but not outgoings data).

10.3 Implementation of Firewall

To design a firewall, many factors need to be considered. First a security policy should be set, which depends on the requirements of the local system and the environment. Other important things need to pay attentions are ease of configuration of the firewall and the security of the firewall itself.

A common method used in firewall is designing a demilitarized zone (DMZ). A DMZ is the zone in the network that is segregated from rest of the network. A DMZ contains servers that need to be accessed from the public network, such as web server, ftp server, etc. A firewall should be designed so that even some servers in DMZ are compromised that rest part of the private network will not be compromised.

Firewall Configuration

Usually, a configuration of firewall consists of more than one systems. To

configure a firewall, a written security policy should be formed first. Then design a firewall to implement the policy. The firewall should be reviewed and updated from time to time.

In the following, we discuss three common firewall configurations. In what follows, a bastion host is a system identified by the firewall administration as a critical strong point in the network's security. Usually, a bastion host serves as an application-level or circuit-level gateway.

In a screened host firewall, single-homed bastion configuration, the firewall consists of two systems: a packet-filtering router and a bastion host. This configuration implements both packet-level and application-level filtering, allowing flexibility in defining security policy. For example, the router can be set so that only the IP packets from or to the bastion host are allowed. The bastion host performs authentication and proxy functions. This configuration also can be set to provide direct internet access. In this case, the outgoing packets can go direct to the router.

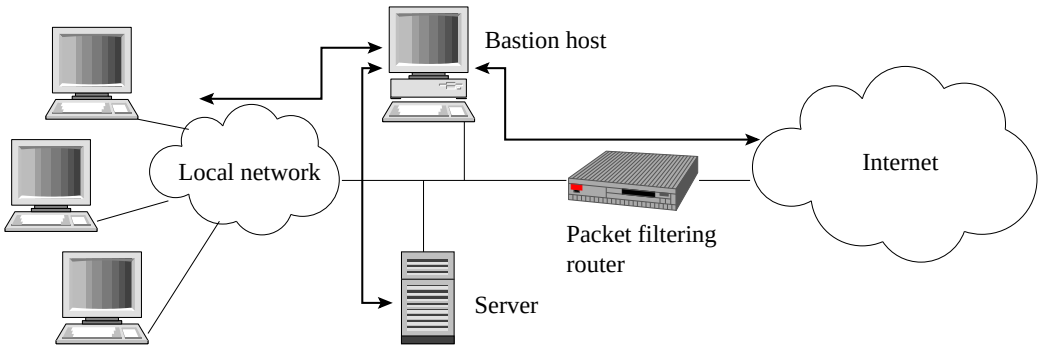


Figure 10.5: Single-homed bastion host

The screened host firewall, dual-homed bastion configuration physically separates the router and the private network hosts. In this case, even the router is compromised, the hosts are still protected by the proxy.

The third configuration is screened subnet firewall. In this configuration, two packet-filtering router are used, one between the bastion host and the internet and one between the bastion host and the internal network. This configuration creates an isolated subnetwork, which may consist of the bastion host and/or several information services and modems for dial-in capability.

A modern firewall usually provides more functions such as graphic configuration interface, allowing varying security levels to be assigned to its various

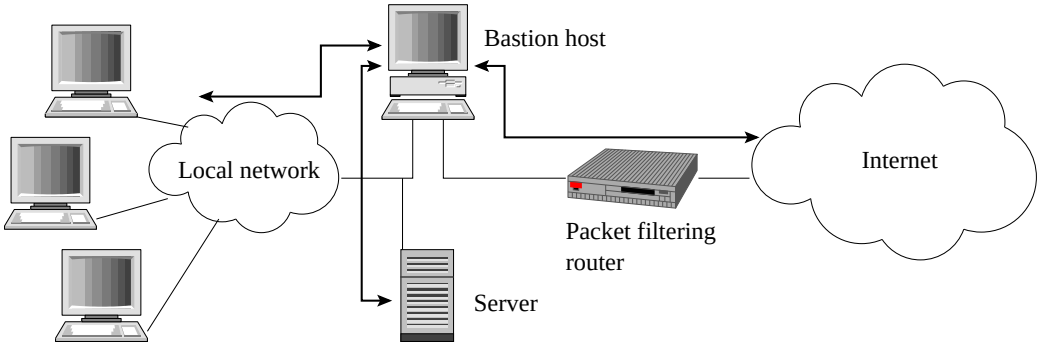


Figure 10.6: Dual-homed bastion host

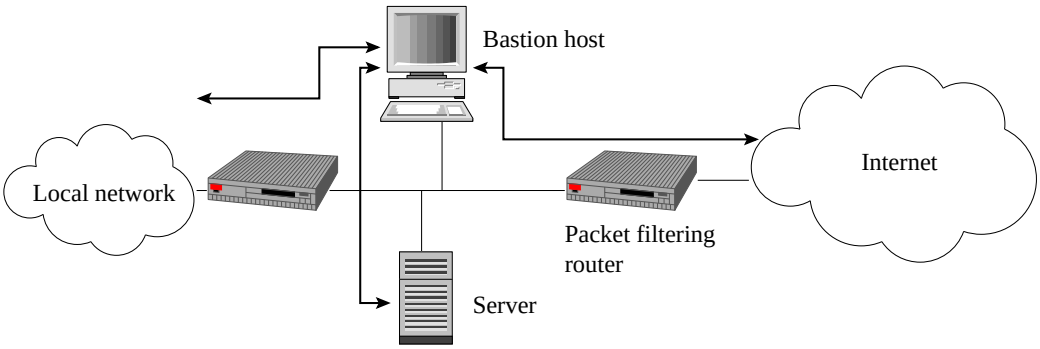


Figure 10.7: Screened-subnet firewall

interfaces, extensive logging capabilities, network address translation (NAT), etc.

Bibliography

[1] W. Stallings, Network and internetwork security, Prentice Hall.

[2] D.R. Stinson, Cryptography: theory and practice, CRC Press.

Index