CRYPTOGRAPHY (Effective from the academic year 2018 -2019) SEMESTER – VII			
Course Code	**18CS744**	**CIE Marks**	40
Number of Contact Hours/Week	3:0:0	**SEE Marks**	60
Total Number of Contact Hours	40	**Exam Hours**	03
CREDITS –3			

Course Learning Objectives: This course (18CS744) will enable students to:

- Define cryptography and its principles
- Explain Cryptography algorithms
- Illustrate Public and Private key cryptography
- Explain Key management, distribution and ceritification
- Explain authentication protocols
- Tell about IPSec

Module – 1	**Contact Hours**
Classical Encryption Techniques Symmetric Cipher Model, Cryptography, Cryptanalysis and Brute-Force Attack, Substitution Techniques, Caesar Cipher, Monoalphabetic Cipher, Playfair Cipher, Hill Cipher, Polyalphabetic Cipher, One Time Pad. **Block Ciphers and the data encryption standard:** Traditional block Cipher structure, stream Ciphers and block Ciphers, Motivation for the feistel Cipher structure, the feistel Cipher, The data encryption standard, DES encryption, DES decryption, A DES example, results, the avalanche effect, the strength of DES, the use of 56-Bit Keys, the nature of the DES algorithm, timing attacks, Block cipher design principles, number of rounds, design of function F, key schedule algorithm **Textbook 1: Ch. 2.1,2.2, Ch. 3** **RBT: L1, L2**	08
Module – 2	
Public-Key Cryptography and RSA: Principles of public-key cryptosystems. Public-key cryptosystems. Applications for public-key cryptosystems, requirements for public-key cryptosystems. public-key cryptanalysis. The RSA algorithm, desription of the algorithm, computational aspects, the security of RSA. **Other Public-Key Cryptosystems:** Diffie-hellman key exchange, The algorithm, key exchange protocols, man in the middle attack,Elgamal Cryptographic systems **Textbook 1: Ch. 9, Ch. 10.1,10.2** **RBT: L1, L2**	08
Module – 3	
Elliptic curve arithmetic, abelian groups, elliptic curves over real numbers, elliptic curves over Zp, elliptic curves overGF(2m), Elliptic curve cryptography, Analog of Diffie-hellman key exchange, Elliptic curve encryption/ decryption, security of Elliptic curve cryptography, Pseudorandom number generation based on an asymmetric cipher, PRNG based on RSA. **Key Management and Distribution:** Symmetric key distribution using Symmetric encryption, A key distribution scenario, Hierarchical key control, session key lifetime, a transparent key control scheme, Decentralized key control, controlling key usage, Symmetric key distribution using asymmetric encryption, simple secret key distribution, secret key distribution with confidentiality and authentication, A hybrid scheme, distribution of public keys, public announcement of public keys, publicly available directory,public key	08

authority, public keys certificates. **Textbook 1: Ch. 10.3-10.5, Ch.14.1 to 14.3** **RBT: L1, L2**	
Module – 4	
X-509 certificates. Certificates, X-509 version 3, public key infrastructure .**User Authentication:** Remote user Authentication principles, Mutual Authentication, one wayAuthentication, remote user Authentication using Symmetric encryption, Mutual Authentication, one way Authentication, Kerberos, Motivation , Kerberos version 4, Kerberos version 5, Remote user Authentication using Asymmetric encryption, Mutual Authentication, one way Authentication. **Electronic Mail Security:** Pretty good privacy, notation, operational; description, S/MIME, RFC5322, Multipurpose internet mail extensions, S/MIME functionality, S/MIME messages, S/MIME certificate processing, enhanced security services, Domain keys identified mail, internet mail architecture, E-Mail threats, DKIM strategy, DKIM functional flow. **Textbook 1: Ch. 14.4, Ch. 15.1 to 15.4, Ch.19** **RBT: L1, L2**	08
Module – 5	
IP Security: IP Security overview, applications of IPsec, benefits of IPsec, Routing applications, IPsec documents, IPsec services, transport and tunnel modes, IP Security policy, Security associations, Security associations database, Security policy database, IP traffic processing, Encapsulating Security payload, ESP format, encryption and authentication algorithms, Padding, Anti replay service **Transport and tunnel modes**, combining security associations, authentication plus confidentiality, basic combinations of security associations, internet key exchange, key determinations protocol, header and payload formats, cryptographic suits. **Textbook 1: Ch. 20.1 to 20.3** **RBT: L1, L2**	08
Course outcomes: The students should be able to:	
• Define cryptography and its principles • Explain Cryptography algorithms • Illustrate Public and Private key cryptography • Explain Key management, distribution and ceritification • Explain authentication protocols • Tell about IPSec	
Question paper pattern: • The question paper will have ten questions. • There will be 2 questions from each module. • Each question will have questions covering all the topics under a module. • The students will have to answer 5 full questions, selecting one full question from each module.	
Text Books:	
1. William Stallings: Cryptography and Network Security, Pearson 6th edition.	
Reference Books:	
1. V K Pachghare: Cryptography and Information Security, PHI 2nd Edition.	

Module-1:

Classical Encryption Techniques

Symmetric encryption, also referred to as conventional encryption or single-key encryption, was the only type of encryption in use prior to the development of public-key encryption in the 1970s. It remains by far the most widely used of the two types of encryption.

1.1 Some Basic Terminology

An original message is known as the **plaintext**, while the coded message is called the **ciphertext**. The process of converting from plaintext to ciphertext is known as **enciphering** or **encryption**; restoring the plaintext from the ciphertext is **deciphering** or **decryption**. The many schemes used for encryption constitute the area of study known as **cryptography**. Such a scheme is known as a **cryptographic system** or a **cipher**. Techniques used for deciphering a message without any knowledge of the enciphering details fall into the area of **cryptanalysis**. Cryptanalysis is what the layperson calls "breaking the code." The areas of cryptography and cryptanalysis together are called **cryptology**.

1.2 Symmetric Cipher Model

A symmetric encryption scheme has five ingredients as shown in figure 1.1

- **Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.
- **Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts. The ciphertext is an apparently random stream of data and, as it stands, is unintelligible.
- **Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

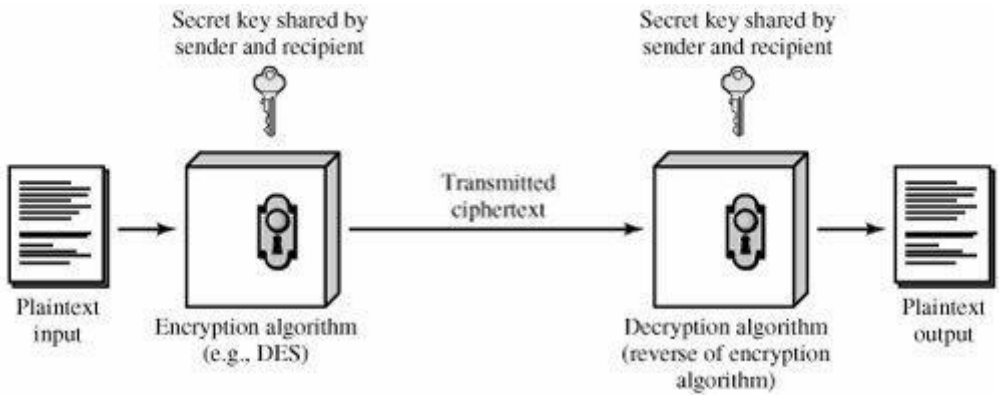


Figure 1.1: Simplified Model of Conventional Encryption

There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.
2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure 1.2. A source produces a message in plaintext, $X = [X1, X2, ..., XM]$. The M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet {0, 1} is typically used. For encryption, a key of the form $K = [K1, K2, ..., KJ]$ is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.

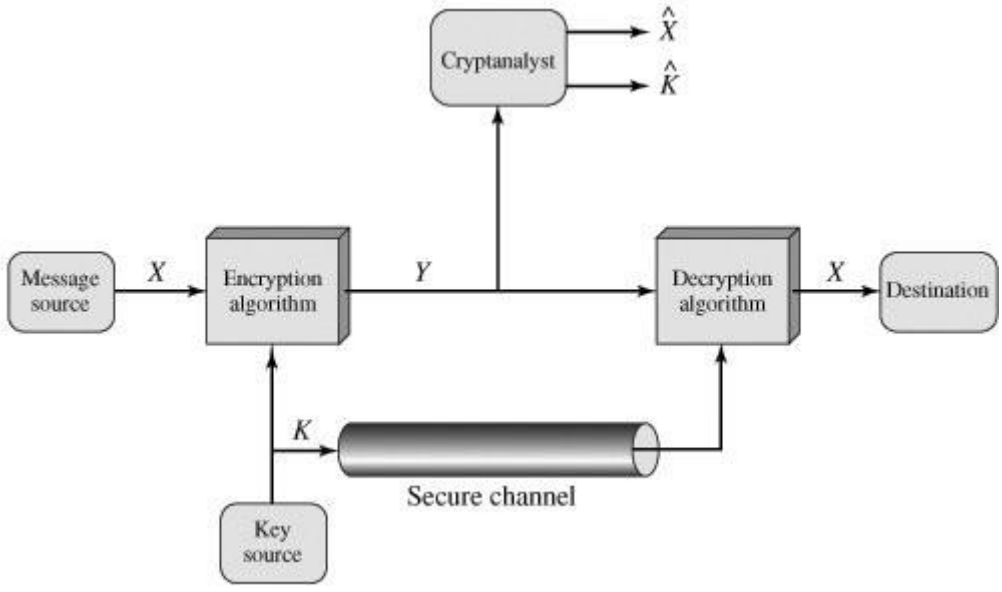


Figure 1.2. Model of Conventional Cryptosystem

With the message *X* and the encryption key *K* as input, the encryption algorithm forms the ciphertext $Y = [Y1, Y2, ..., Y_N]$.
We can write this as

$$Y = E(K, X)$$

This notation indicates that *Y* is produced by using encryption algorithm E as a function ofthe plaintext *X*, with the specific function determined by the value of the key *K*.
The intended receiver, in possession of the key, is able to invert the transformation:

$$X = D(K, Y)$$

An opponent, observing *Y* but not having access to *K* or *X*, may attempt to recover *X* or *K* or both *X* and *K*. It is assumed that the opponent knows the encryption (E) and decryption (D) algorithms. If the opponent is interested in only this particular message, then the focus of the effort is to recover *X* by generating a plaintext estimate X' . Often, however, the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover *K* by generating an estimate K'.

1.3 Cryptography

Cryptographic systems are characterized along three independent dimensions:

1. **The type of operations used for transforming plaintext to ciphertext.** All encryption algorithms are based on two general principles: substitution, in which eachelement in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (that is, that all operations are reversible). Most systems, referred to as *product systems*, involve multiple stages of substitutions and transpositions.
2. **The number of keys used.** If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.
3. **The way in which the plaintext is processed.** A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along.

1.4 Cryptanalysis

There are two general approaches to attacking a conventional encryption scheme:
Cryptanalysis: Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintextciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used.
Brute-force attack: The attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

If either type of attack succeeds in deducing the key, the effect is catastrophic: All future and past messages encrypted with that key are compromised.

Table 1.1. Types of Attacks on Encrypted Messages

Type of Attack	Known to Cryptanalyst
Ciphertext only	• Encryption algorithm • Ciphertext
Known plaintext	• Encryption algorithm • Ciphertext • One or more plaintext-ciphertext pairs formed with the secret key
Chosen plaintext	• Encryption algorithm • Ciphertext • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen ciphertext	• Encryption algorithm • Ciphertext • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen text	• Encryption algorithm • Ciphertext • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key • Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

An encryption scheme is **unconditionally secure** if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext is available. That is, no matter how much time an opponent has, it is impossible for him or her to decrypt the ciphertext, simply because the required information is not there. With the exception of a scheme known as the one-time pad, there is no encryption algorithm that is unconditionally secure. Therefore, all that the users of an encryption algorithm can strive for is an algorithm that meets one or both of the following criteria:

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

An encryption scheme is said to be **computationally secure** if either of the foregoing two criteria are met. The rub is that it is very difficult to estimate the amount of effort required to cryptanalyze ciphertext successfully.

A **brute-force attack** involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. Table 1.2 shows how much time is involved for various key spaces.

Table 1.2. Average Time Required for Exhaustive Key Search

Key size (bits)	Number of alternative keys		Time required at 1 decryption/μs		Time required at 10^6 decryption/μs
32	2^{32}	= 4.3 x 10^9	2^{31} μs	= 35.8 minutes	2.15 milliseconds
56	2^{56}	= 7.2 x 10^{16}	2^{55} μs	= 1142 years	10.01 hours
128	2^{128}	= 3.4 x 10^{38}	2^{127} μs	= 5.4 x 10^{24} years	5.4 x 10^{18} years
168	2^{168}	= 3.7 x 10^{50}	2^{167} μs	= 5.9 x 10^{36} years	5.9 x 10^{30} years
26 characters (permutation)	26!	= 4 x 10^{26}	2 x 10^{26} μs	= 6.4 x 10^{12} years	6.4 x 10^6 years

1.5 Substitution Techniques

The two basic building blocks of all encryption techniques are substitution and transposition.

A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns.

1.5.1 Caesar Cipher

The earliest known use of a substitution cipher, and the simplest, was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet.

For example,

```
plain: meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB
```

Note that the alphabet is wrapped around, so that the letter following Z is A. We can define the transformation by listing all possibilities, as follows:

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

Let us assign a numerical equivalent to each letter:

a b c d e f g h i j k l m n o p q r s t u v w x y z
0 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

Then the algorithm can be expressed as follows. For each plaintext letter p, substitute the ciphertext letter C:

$$C = E(3, p) = (p + 3) \bmod 26$$

A shift may be of any amount, so that the general Caesar algorithm is

$$C = E(k, p) = (p + k) \bmod 26$$

where k takes on a value in the range 1 to 25. The decryption algorithm is simply

$$p = D(k, C) = (C\ k) \bmod 26$$

If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis is easily performed: Simply try all the 25 possible keys.

Figure 1.3 shows the results of applying this strategy to the example ciphertext. In this case, the plaintext leaps out as occupying the third line.

KEY						
	PHHW	PH	DIWHU	WKH	WRJD	SDUWB
1	oggv	og	chvgt	vjg	vqic	rctva
2	nffu	nf	bgufs	uif	uphb	qbsuz
3	meet	me	after	the	toga	party
4	ldds	ld	zesdq	sgd	snfz	ozqsx
5	kccr	kc	ydrcp	rfc	rmey	nyprw
6	jbbq	jb	xcqbo	qeb	qldx	mxoqv
7	iaap	ia	wbpan	pda	pkcw	lwnpu
8	hzzo	hz	vaozm	ocz	ojbv	kvmot
9	gyyn	gy	uznyl	nby	niau	julns
10	fxxm	fx	tymxk	max	mhzt	itkmr
11	ewwl	ew	sxlwj	lzw	lgys	hsjlq
12	dvvk	dv	rwkvi	kyv	kfxr	grikp
13	cuuj	cu	qvjuh	jxu	jewq	fqhjo
14	btti	bt	puitg	iwt	idvp	epgin
15	assh	as	othsf	hvs	hcuo	dofhm
16	zrrg	zr	nsgre	gur	gbtn	cnegl
17	yqqf	yq	mrfqd	ftq	fasm	bmdfk
18	xppe	xp	lqepc	esp	ezrl	alcej
19	wood	wo	kpdob	dro	dyqk	zkbdi
20	vnnc	vn	jocna	cqn	cxpj	yjach
21	ummb	um	inbmz	bpm	bwoi	xizbg
22	tlla	tl	hmaly	aol	avnh	whyaf
23	skkz	sk	glzkx	znk	zumg	vgxze
24	rjjy	rj	fkyjw	ymj	ytlf	ufwyd
25	qiix	qi	ejxiv	xli	xske	tevxc

Figure 1.3. Brute-Force Cryptanalysis of Caesar Cipher

Three important characteristics of this problem enabled us to use a brute-force cryptanalysis:

1. The encryption and decryption algorithms are known.

2. There are only 25 keys to try.

3. The language of the plaintext is known and easily recognizable.

In most networking situations, we can assume that the algorithms are known. What generally makes brute-force cryptanalysis impractical is the use of an algorithm that employs a large number of keys.

For example, the triple DES algorithm, makes use of a 168-bit key, giving a key space of 2^168 or greater than 3.7 x 1050 possible keys.

The third characteristic is also significant. If the language of the plaintext is unknown, then plaintext output may not be recognizable. Furthermore, the input may be abbreviated or compressed in some fashion, again making recognition difficult.

For example, Figure 1.4 shows a portion of a text file compressed using an algorithm called ZIP. If this file is then encrypted with a simple substitution cipher (expanded to include more than just 26 alphabetic characters), then the plaintext may not be recognized when it is uncovered in the brute-force cryptanalysis.

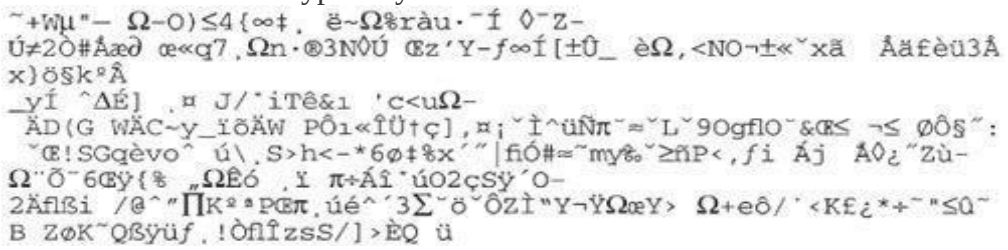


Figure 1.4: Sample of Compressed Text

1.5.2 Monoalphabetic Ciphers

With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution.

If, instead, the "cipher" line can be any permutation of the 26 alphabetic characters, then there are 26! or greater than 4 x 10^26 possible keys. This is 10 orders of magnitude greater than the key space for DES and would seem to eliminate brute-force techniques for cryptanalysis. Such an approach is referred to as a **monoalphabetic substitution cipher**, because a single cipher alphabet (mapping from plain alphabet to cipher alphabet) is used per message.

There is, another line of attack. If the cryptanalyst knows the nature of the plaintext (e.g., noncompressed English text), then the analyst can exploit the regularities of the language.The ciphertext to be solved is

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
```

As a first step, the relative frequency of the letters can be determined and compared to a standard frequency distribution for English, such as is shown in Figure 1.5. If the message were long enough, this technique alone might be sufficient, but because this is a relatively short message, we cannot expect an exact match. In any case, the relative frequencies of the letters in the ciphertext (in percentages) are as follows:

P 13.33	H 5.83	F 3.33	B 1.67	C 0.00
Z 11.67	D 5.00	W 3.33	G 1.67	K 0.00
S 8.33	E 5.00	Q 2.50	Y 1.67	L 0.00
U 8.33	V 4.17	T 2.50	I 0.83	N 0.00
O 7.50	X 4.17	A 1.67	J 0.83	R 0.00
M 6.67				

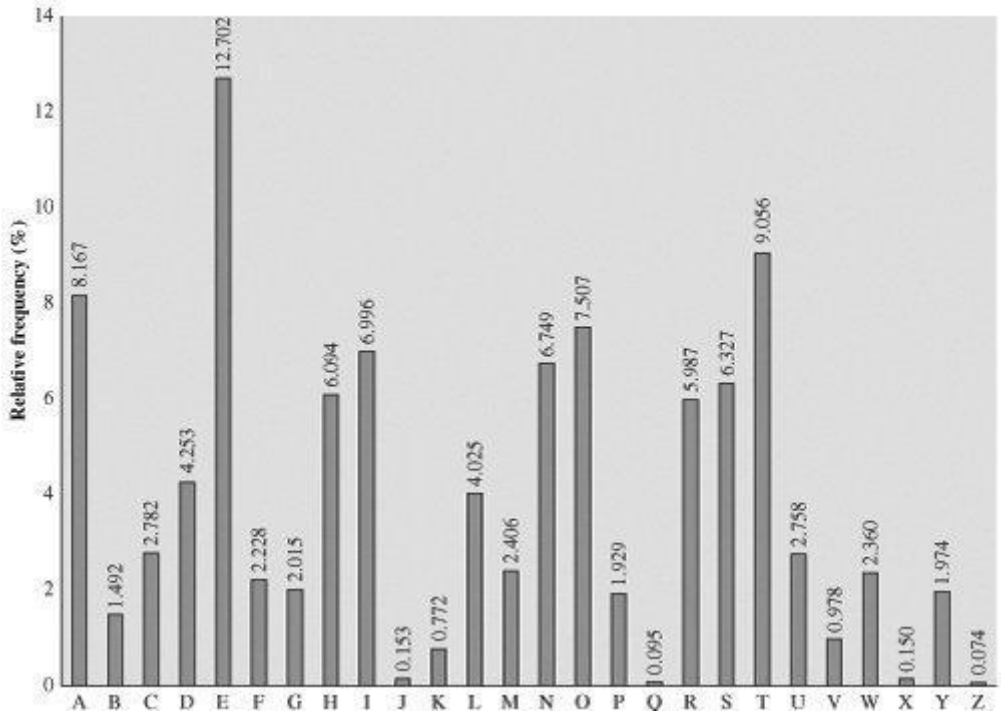


Figure 1.5. Relative Frequency of Letters in English Text

Comparing this breakdown with Figure 1.5, it seems likely that cipher letters P and Z are the equivalents of plain letters e and t, but it is not certain which is which. The letters S, U, O, M, and H are all of relatively high frequency and probably correspond to plain letters from the set {a, h, i, n, o, r, s}.The letters with the lowest frequencies (namely, A, B, G, Y, I, J) are likely included in the set {b, j, k, q, v, x, z}.

There are a number of ways to proceed at this point. We could make some tentative assignments and start to fill in the plaintext to see if it looks like a reasonable "skeleton" of a message. A more systematic approach is to look for other regularities. For example, certain words may be known to be in the text. Or we could look for repeating sequences of cipher letters and try to deduce their plaintext equivalents.

A powerful tool is to look at the frequency of two-letter combinations, known as digrams. A table similar to Figure 1.5 could be drawn up showing the relative frequency of digrams. The most common such digram is th. In our ciphertext, the most common digram is ZW, which appears three times. So we make the correspondence of Z with t and W with h. Then, by our earlier hypothesis, we can equate P with e. Now notice that the sequence ZWP appears in the ciphertext, and we can translate that sequence as "the." This is the most frequent trigram (three-letter combination) in English, which seems to indicate that we are on the right track.

Next, notice the sequence ZWSZ in the first line. We do not know that these four letters form a complete word, but if they do, it is of the form th_t. If so, S equates with a.

So far, then, we have

```
UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ
 t a        e  e te  a that e e a       a
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX
   e t   ta t ha e ee   a e  th    t  a
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ
 e  e e tat  e   the   t
```

Only four letters have been identified, but already we have quite a bit of the message. Continued analysis of frequencies plus trial and error should easily yield a solution from this point. The complete plaintext, with spaces added between words, follows:

it was disclosed yesterday that several informal but
direct contacts have been made with political
representatives of the viet cong in moscow

Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet. A countermeasure is to provide multiple substitutes, known as homophones, for a single letter. For example, the letter e could be assigned a number of different cipher symbols, such as 16, 74, 35, and 21, with each homophone used in rotation,or randomly. If the number of symbols assigned to each letter is proportional to the relative frequency of that letter, then single-letter frequency information is completely obliterated.

The great mathematician Carl Friedrich Gauss believed that he had devised an unbreakable cipher using homophones. However, even with homophones, each element of plaintext affects only one element of ciphertext, and multiple-letter patterns (e.g., digram frequencies) still survive in the ciphertext, making cryptanalysis relatively straightforward.

1.5.3 Playfair Cipher

The best-known multiple-letter encryption cipher is the Playfair, which treats digrams in the plaintext as single units and translates these units into ciphertext digrams. The Playfair algorithm is based on the use of a 5 x 5 matrix of letters constructed using a keyword. Here is an example, solved by Lord Peter Wimsey in Dorothy Sayers's *Have His Carcase*

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

In this case, the keyword is *monarchy*. The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetic order. The letters I and J count as one letter.

Plaintext is encrypted two letters at a time, according to the following rules:

1. Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.

2. Two plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last. For example, ar is encrypted as RM.

3. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.

4. Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs becomes BP and ea becomes IM (or JM, as the encipherer wishes).

The Playfair cipher is a great advance over simple monoalphabetic ciphers. For one thing, whereas there are only 26 letters, there are 26 x 26 = 676 digrams, so that identification of individual digrams is more difficult. Furthermore, the relative frequencies of individual letters exhibit a much greater range than that of digrams, making frequency analysis much more difficult. For these reasons, the Playfair cipher was for a long time considered unbreakable. It was used as the standard field system by the British Army in World War I and still enjoyed considerable use by the U.S. Army and other Allied forces during World War II.

Despite this level of confidence in its security, the Playfair cipher is relatively easy to break because it still leaves much of the structure of the plaintext language intact. A few hundred letters of ciphertext are generally sufficient.

One way of revealing the effectiveness of the Playfair and other ciphers is shown in Figure 1.6. The line labeled *plaintext* plots the frequency distribution of the more than 70,000 alphabetic characters in the *Encyclopaedia Brittanica* article on cryptology.

This is also the frequency distribution of any monoalphabetic substitution cipher. The plot was developed in the following way: The number of occurrences of each letter in the text was counted and divided by the number of occurrences of the letter e (the most frequently used letter). As a result, e has a relative frequency of 1, t of about 0.76, and so on. The points on the horizontal axis correspond to the letters in order of decreasing frequency.

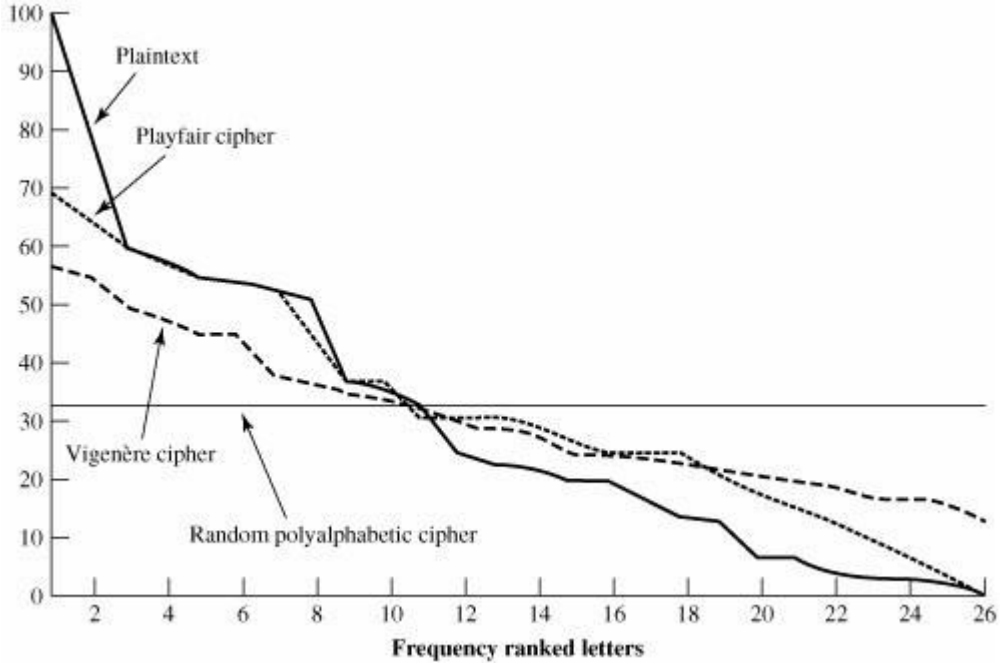


Figure 1.6. Relative Frequency of Occurrence of Letters

1.5.3 Hill Cipher

Another interesting multiletter cipher is the Hill cipher, developed by the mathematician Lester Hill in 1929. The encryption algorithm takes *m* successive plaintext letters and substitutes for them *m* ciphertext letters. The substitution is determined by *m* linear equations in which each character is assigned a numerical value (a = 0, b = 1 ... z = 25). For *m* = 3, the system can be described as follows:

$$c_1 = (k_{11}p1 + k_{12}p2 + k_{13}p3) \bmod 26$$

$$c_2 = (k_{21}p1 + k_{22}p2 + k_{23}p3) \bmod 26$$

$$c_3 = (k_{31}p1 + k_{32}p2 + k3_3p3) \bmod 26$$

This can be expressed in term of column vectors and matrices:

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \bmod 26$$

or

$$\mathbf{C} = \mathbf{KP} \bmod 26$$

where **C** and **P** are column vectors of length 3, representing the plaintext and ciphertext, and **K** is a 3 x 3 matrix, representing the encryption key. Operations are performed mod 26.

For example, consider the plaintext "paymoremoney" and use the encryption key

$$\mathbf{K} = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

The first three letters of the plaintext are represented by the vector $\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix}$. Then $\mathbf{K}\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix} = \begin{pmatrix} 375 \\ 819 \\ 486 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 13 \\ 18 \end{pmatrix}$ = LNS. Continuing in this fashion, the ciphertext for the entire plaintext is LNSHDLEWMTRW.

Decryption requires using the inverse of the matrix **K**. The inverse **K**1 of a matrix **K** is defined by the equation **KK**1 = **K**1**K** = **I**, where **I** is the matrix that is all zeros except for

ones along the main diagonal from upper left to lower right. The inverse of a matrix does not always exist, but when it does, it satisfies the preceding equation. In this case, the inverse is:

$$\mathbf{K}^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

This is demonstrated as follows:

$$\begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix} \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix} = \begin{pmatrix} 443 & 442 & 442 \\ 858 & 495 & 780 \\ 494 & 52 & 365 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

It is easily seen that if the matrix **K**1 is applied to the ciphertext, then the plaintext is recovered. To explain how the inverse of a matrix is determined, we make an exceedingly brief excursion into linear algebra.

For any square matrix (*m* x *m*) the **determinant** equals the sum of all the products that can be formed by taking exactly one element from each row and exactly one element from each column, with certain of the product terms preceded by a minus sign. For a 2 x 2 matrix

$$\begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}$$

the determinant is *k*11*k*22 *k*12*k*21. For a 3 x 3 matrix, the value of the determinant is *k*11*k*22*k*33 + *k*21*k*32*k*13 + *k*31*k*12*k*23 *k*31*k*22*k*13 *k*21*k*12*k*33 *k*11*k*32*k*23. If a square matrix **A**has a nonzero determinant, then the inverse of the matrix is computed as [**A**1]*ij* = (1)*i*+*j*(D*ij*)/ded(**A**), where (D*ij*) is the subdeterminant formed by deleting the *i*th row and the *j*th column of **A** and det(**A**) is the determinant of **A**. For our purposes, all arithmetic is done mod 26.

In general terms, the Hill system can be expressed as follows:

C = E(**K, P**) = **KP** mod 26

P = D(**K, P**) = **K**1**C** mod 26 = **K**1**KP** = **P**

As with Playfair, the strength of the Hill cipher is that it completely hides single-letter frequencies. Indeed, with Hill, the use of a larger matrix hides more frequency information. Thus a 3 x 3 Hill cipher hides not only single-letter but also two-letter frequency information. Although the Hill cipher is strong against a ciphertext-only attack, it is easily broken with a known plaintext attack. For an *m* x *m* Hill cipher, suppose we have *m* plaintext-ciphertext pairs, each of length *m*. We label the pairs

$$\mathbf{P}_j = \begin{pmatrix} p_{1j} \\ p_{2j} \\ \vdots \\ p_{mj} \end{pmatrix} \text{ and } \mathbf{C}_j = \begin{pmatrix} c_{1j} \\ c_{2j} \\ \vdots \\ c_{mj} \end{pmatrix} \text{ such that } \mathbf{C}_j = \mathbf{K}\mathbf{P}_j \text{ for } 1 \le j \le m \text{ and for some}$$

unknown key matrix **K**. Now define two *m* x *m* matrices **X** = (*Pij*) and **Y** = (*Cij*). Then we can form the matrix equation **Y** = **KX**. If **X** has an inverse, then we can determine **K** = **YX**1. If **X** is not invertible, then a new version of **X** can be formed with additional plaintext- ciphertext pairs until an invertible **X** is obtained.

Suppose that the plaintext "friday" is encrypted using a 2 x 2 Hill cipher to yield theciphertext PQCFKU. Thus, we know that

$$\mathbf{K}\begin{pmatrix}5\\17\end{pmatrix} \bmod 26 = \begin{pmatrix}15\\16\end{pmatrix}; \mathbf{K}\begin{pmatrix}8\\3\end{pmatrix} \bmod 26 = \begin{pmatrix}2\\5\end{pmatrix}; \quad \text{and} \quad \mathbf{K}\begin{pmatrix}0\\24\end{pmatrix} \bmod 26 = \begin{pmatrix}10\\20\end{pmatrix}$$

Using the first two plaintext-ciphertext pairs, we have

$$\begin{pmatrix}15 & 2\\16 & 5\end{pmatrix} = \mathbf{K}\begin{pmatrix}5 & 8\\17 & 3\end{pmatrix} \bmod 26$$

The inverse of **X** can be computed:

$$\begin{pmatrix}5 & 8\\17 & 3\end{pmatrix}^{-1} = \begin{pmatrix}9 & 2\\1 & 15\end{pmatrix}$$

So

$$\mathbf{K} = \begin{pmatrix}15 & 2\\16 & 5\end{pmatrix}\begin{pmatrix}9 & 2\\1 & 15\end{pmatrix} = \begin{pmatrix}137 & 60\\149 & 107\end{pmatrix} \bmod 26 = \begin{pmatrix}7 & 8\\19 & 3\end{pmatrix}$$

This result is verified by testing the remaining plaintext-ciphertext pair.

1.5.4 Polyalphabetic Ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is **polyalphabetic substitution cipher**.

All these techniques have the following features in common:

1. A set of related monoalphabetic substitution rules is used.

2. A key determines which particular rule is chosen for a given transformation.

The best known, and one of the simplest, such algorithm is referred to as the Vigenère cipher. In this scheme, the set of related monoalphabetic substitution rules consists of the 26 Caesar ciphers, with shifts of 0 through 25. Each cipher is denoted by a key letter, which is the ciphertext letter that substitutes for the plaintext letter a. Thus, a Caesar cipher with a shiftof 3 is denoted by the key value *d*.

To aid in understanding the scheme and to aid in its use, a matrix known as the Vigenère tableau is constructed (Table 1.3). Each of the 26 ciphers is laid out horizontally, with the key letter for each cipher to its left. A normal alphabet for the plaintext runs across the top. The process of encryption is simple: Given a key letter *x* and a plaintext letter y, the ciphertext letter is at the intersection of the row labeled *x* and the column labeled y; in this case the ciphertext is V.

Table 1.3. The Modern Vigenère Tableau

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

To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword. For example, if the keyword is *deceptive*, the message "we are discovered save yourself" is encrypted as follows:

key: *deceptivedeceptivedeceptive*

plaintext: wearediscoveredsaveyourself

ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ

Decryption is equally simple. The key letter again identifies the row. The position of the ciphertext letter in that row determines the column, and the plaintext letter is at the top of that column. The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword. Thus, the letter frequency information is obscured. However, not all knowledge of the plaintext structure is lost. For example, Figure 1.6 shows the frequency distribution for a Vigenère cipher with a keywordof length 9. An improvement is achieved over the Playfair cipher, but considerable frequency information remains.

It is instructive to sketch a method of breaking this cipher, because the method reveals some of the mathematical principles that apply in cryptanalysis.

First, suppose that the opponent believes that the ciphertext was encrypted using either monoalphabetic substitution or a Vigenère cipher. A simple test can be made to make a determination. If a monoalphabetic substitution is used, then the statistical properties of the ciphertext should be the same as that of the language of the plaintext. Thus, referring to Figure 1.5, there should be one cipher letter with a relative frequency of occurrence of about 12.7%, one with about 9.06%, and so on. If only a single message is available for analysis,we would not expect an exact match of this small sample with the statistical profile of the plaintext language. Nevertheless, if the correspondence is close, we can assume a monoalphabetic substitution.

If, on the other hand, a Vigenère cipher is suspected, then progress depends on determining the length of the keyword, as will be seen in a moment. For now, let us concentrate on how the keyword length can be determined. The important insight that leads to a solution is the following: If two identical sequences of plaintext letters occur at a distance that is an integer multiple of the keyword length, they will generate identical ciphertext sequences. In the foregoing example, two instances of the sequence "red" are separated by nine character positions. Consequently, in both cases, r is encrypted using key letter *e*, e is encrypted using key letter *p*, and d is encrypted using key letter *t*. Thus, in both cases the ciphertext sequence is VTW.

An analyst looking at only the ciphertext would detect the repeated sequences VTW at a displacement of 9 and make the assumption that the keyword is either three or nine letters in length. The appearance of VTW twice could be by chance and not reflect identical plaintext letters encrypted with identical key letters. However, if the message is long enough, there will be a number of such repeated ciphertext sequences. By looking for common factors in the displacements of the various sequences, the analyst should be able to make a good guess of the keyword length. Solution of the cipher now depends on an important insight. If the keyword length is *N*, then the cipher, in effect, consists of *N* monoalphabetic substitution ciphers. For example, with the keyword DECEPTIVE, the letters in positions 1, 10, 19, and so on are all encrypted with the same monoalphabetic cipher. Thus, we can use the known frequency characteristics of the plaintext language to attack each of the monoalphabetic ciphers separately.

The periodic nature of the keyword can be eliminated by using a nonrepeating keyword that is as long as the message itself. Vigenère proposed what is referred to as an **autokey system**, in which a keyword is concatenated with the plaintext itself to provide a running key. For our example,

```
key:        deceptivewearediscoveredsav
plaintext:  wearediscoveredsaveyourself
ciphertext:ZICVTWQNGKZEIIGASXSTSLVVWLA
```

Even this scheme is vulnerable to cryptanalysis. Because the key and the plaintext share the same frequency distribution of letters, a statistical technique can be applied. For example, e enciphered by *e*, by Figure 1.5, can be expected to occur with a frequency of (0.127)=2 0.016, whereas t enciphered by *t* would occur only about half as often. These regularities can be exploited to achieve successful cryptanalysis.

The ultimate defense against such a cryptanalysis is to choose a keyword that is as long as the plaintext and has no statistical relationship to it. Such a system was introduced by an AT&T engineer named Gilbert Vernam in 1918.

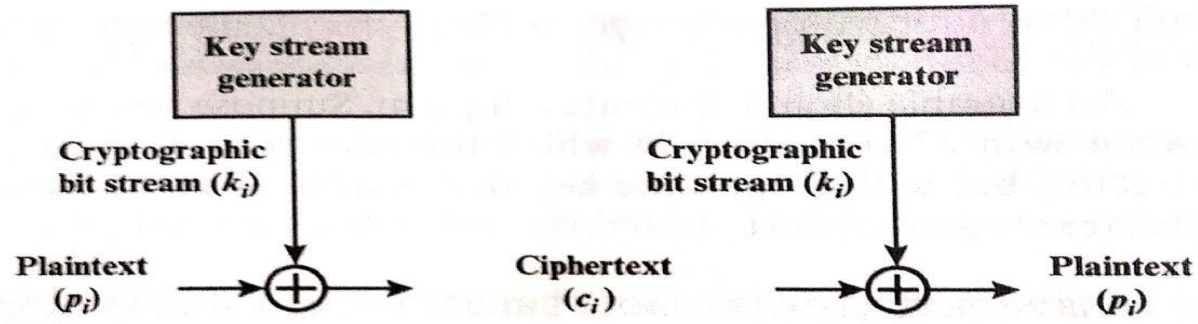


Figure 1.7: Vernam Cipher

His system works on binary data rather than letters. The system can be expressed succinctly as follows:

$$c_i = p_i \oplus k_i$$

where

p_i = ith binary digit of plaintext
k_i = ith binary digit of key
c_i = ith binary digit of ciphertext
$\oplus$ = exclusive-or (XOR) operation

Thus, the ciphertext is generated by performing the bitwise XOR of the plaintext and the key. Because of the properties of the XOR, decryption simply involves the same bitwise operation:

$$p_i = c_i \oplus k_i$$

The essence of this technique is the means of construction of the key. Vernam proposed the use of a running loop of tape that eventually repeated the key, so that in fact the system worked with a very long but repeating keyword. Although such a scheme, with a long key, presents formidable cryptanalytic difficulties, it can be broken with sufficient ciphertext, the use of known or probable plaintext sequences, or both.

1.5.6 One-Time Pad

An Army Signal Corp officer, Joseph Mauborgne, proposed an improvement to the Vernam cipher that yields the ultimate in security. Mauborgne suggested using a random key that is as long as the message, so that the key need not be repeated. In addition, the key is to be used to encrypt and decrypt a single message, and then is discarded. Each new message requires a new key of the same length as the new message. Such a scheme, known as a **one- time pad**, is unbreakable.

It produces random output that bears no statistical relationship to the plaintext. Because the ciphertext contains no information whatsoever about the plaintext, there issimply no way to break the code.

An example should illustrate our point. Suppose that we are using a Vigenère scheme with 27 characters in which the twenty-seventh character is the space character, but with a one-time key that is as long as the message. Thus, the tableau of Table 1.3 must be expanded to 27 x 27.

Consider the ciphertext

```
ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
```

We now show two different decryptions using two different keys:

```
ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key: pxlmvmsydofuyrvzwc tnlebnecvgdupahfzzlmnyih
plaintext: mr mustard with the candlestick in the hall
ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS
key: mfugpmiydgaxgoufhkllmhsqdqogtewbqfgyovuhwt
```

```
plaintext: miss scarlet with the knife in the library
```

Suppose that a cryptanalyst had managed to find these two keys. Two plausible plaintexts are produced. How is the cryptanalyst to decide which is the correct decryption (i.e., which is the correct key)? If the actual key were produced in a truly random fashion, then the cryptanalyst cannot say that one of these two keys is more likely than the other. Thus, there is no way to decide which key is correct and therefore which plaintext is correct.

In fact, given any plaintext of equal length to the ciphertext, there is a key that produces that plaintext. Therefore, if you did an exhaustive search of all possible keys, you would end up with many legible plaintexts, with no way of knowing which was the intended plaintext.

Therefore, the code is unbreakable. The security of the one-time pad is entirely due to the randomness of the key. If the stream of characters that constitute the key is truly random, then the stream of characters that constitute the ciphertext will be truly random. Thus, there are no patterns or regularities that a cryptanalyst can use to attack the ciphertext.

In theory, we need look no further for a cipher. The one-time pad offers complete security but, in practice, has two fundamental difficulties:

1. There is the practical problem of making large quantities of random keys. Any heavily used system might require millions of random characters on a regular basis. Supplying truly random characters in this volume is a significant task.
2. Even more daunting is the problem of key distribution and protection. For every message to be sent, a key of equal length is needed by both sender and receiver. Thus, a mammoth key distribution problem exists.

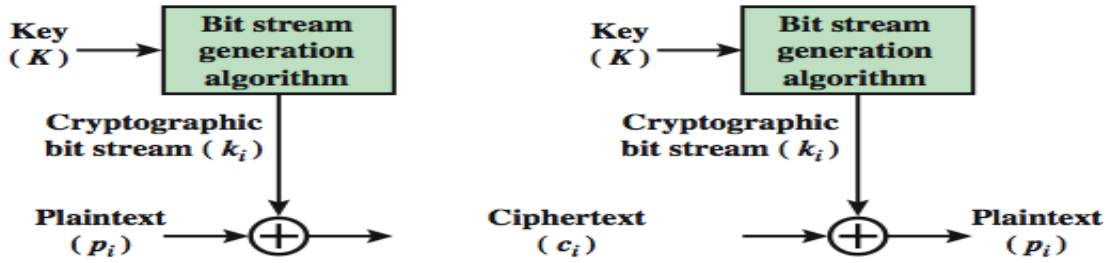
Because of these difficulties, the one-time pad is of limited utility, and is useful primarily for low bandwidth channels requiring very high security.

1.6 Block Cipher Principles

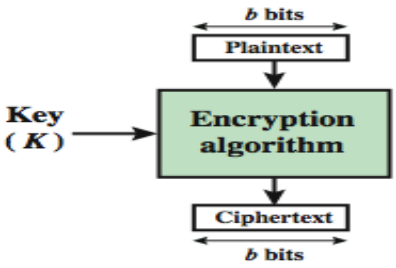
A **block cipher** is an encryption/decryption scheme in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Many block ciphers have a Feistel structure. Such a structure consists of a number of identical rounds of processing. In each round, a substitution is performed on one half of the data being processed,followed by a permutation that interchanges the two halves. The original key is expanded so that a different key is used for each round.

Stream Ciphers and Block Ciphers

A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher. A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used. Ablock cipher can be used to achieve the same effect as a stream cipher.



(a) Stream Cipher Using Algorithmic Bit Stream Generator



(b) Block Cipher

Motivation for the Feistel Cipher Structure

A block cipher operates on a plaintext block of *n* bits to produce a ciphertext block of *n* bits. There are 2^n possible different plaintext blocks and, for the encryption to be reversible (i.e., for decryption to be possible), each must produce a unique ciphertext block. Such a transformation is called reversible, or nonsingular. The following examples illustrate nonsingular and singular transformation for $n = 2$.

Reversible Mapping

Plaintext	Ciphertext
00	11
01	10
10	00
11	01

Irreversible Mapping

Plaintext	Ciphertext
00	11
01	10
10	01
11	01

In the latter case, a ciphertext of 01 could have been produced by one of two plaintext blocks. So if we limit ourselves to reversible mappings, the number of different transformations is 2*n*!.

Figure 1.8 illustrates the logic of a general substitution cipher for $n = 4$. A 4-bit input produces one of 16 possible input states, which is mapped by the substitution cipher into a unique one of 16 possible output states, each of which is represented by 4 ciphertext bits. Theencryption and decryption mappings can be defined by a tabulation, as shown in Table 1.4. This is the most general form of block cipher and can be used to define any reversible mapping between plaintext and ciphertext. Feistel refers to this as the *ideal block cipher*, because it allows for the maximum number of possible encryption mappings from the plaintext block.

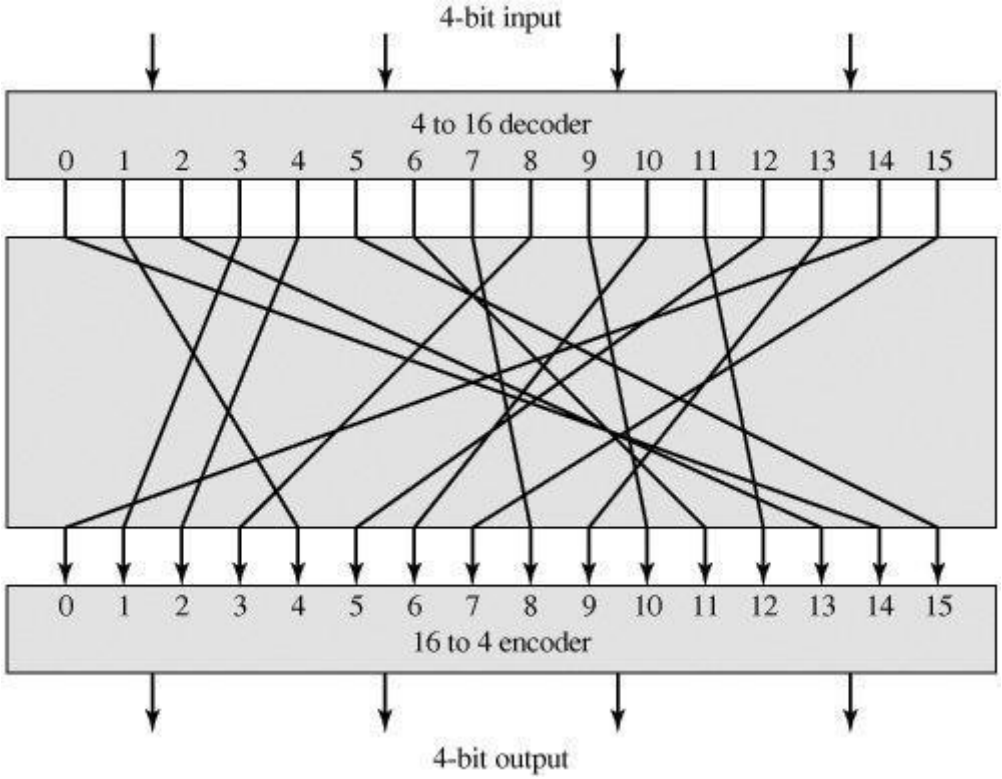


Figure 1.8: General *n*-bit-*n*-bit Block Substitution (shown with *n* = 4)

Table 1.4. Encryption and Decryption Tables for Substitution Cipher

Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010

0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101

But there is a practical problem with the ideal block cipher. If a small block size, such as n = 4, is used, then the system is equivalent to a classical substitution cipher. Such systems, as we have seen, are vulnerable to a statistical analysis of the plaintext. This weakness is not inherent in the use of a substitution cipher but rather results from the use of a small block size. If n is sufficiently large and an arbitrary reversible substitution between plaintext and ciphertext is allowed, then the statistical characteristics of the source plaintext are masked to such an extent that this type of cryptanalysis is infeasible.

An arbitrary reversible substitution cipher (the ideal block cipher) for a large block size is not practical, however, from an implementation and performance point of view. For such a transformation, the mapping itself constitutes the key.

Consider again Table 1.4, which defines one particular reversible mapping from plaintext to ciphertext for n = 4. The mapping can be defined by the entries in the second column, which show the value of the ciphertext for each plaintext block. This, in essence, is the key that determines the specific mapping from among all possible mappings. In this case, using this

straightforward method of defining the key, the required key length is (4 bits) x (16 rows) = 64 bits.

In general, for an *n*-bit ideal block cipher, the length of the key defined in this fashion is *n* x 2*n* bits. For a 64-bit block, which is a desirable length to thwart statistical attacks, the required key length is 64 x 264 = 270 1021bits.

In considering these difficulties, Feistel points out that what is needed is an approximation to the ideal block cipher system for large *n*, built up out of components that are easily realizable. But before turning to Feistel's approach, let us make one other observation. We could use the general block substitution cipher but, to make its implementation tractable, confine ourselves to a subset of the possible reversible mappings.

For example, suppose we define the mapping in terms of a set of linear equations. In the case of $n = 4$, we have

$$y_1 = k_{11}x_1 + k_{12}x_2 + k_{13}x_3 + k_{14}x_4$$
$$y_2 = k_{21}x_1 + k_{22}x_2 + k_{23}x_3 + k_{24}x_4$$
$$y_3 = k_{31}x_1 + k_{32}x_2 + k_{33}x_3 + k_{34}x_4$$
$$y_4 = k_{41}x_1 + k_{42}x_2 + k_{43}x_3 + k_{44}x_4$$

where the *xi* are the four binary digits of the plaintext block, the *yi* are the four binary digits of the ciphertext block, the *kij* are the binary coefficients, and arithmetic is mod 2. The key size is just *n*2, in this case 16 bits. The danger with this kind of formulation is that it may be vulnerable to cryptanalysis by an attacker that is aware of the structure of the algorithm.

1.7 The Feistel Cipher

Feistel proposed that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers insequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. The essence of the approach is to develop a block cipher with akey length of *k* bits and a block length of *n* bits, allowing a total of 2*k* possible transformations, rather than the 2*n*! transformations available with the ideal block cipher.

In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations. In fact, this is a practical application of a proposal by Claude Shannon to develop a product cipher that alternates *confusion* and *diffusion* functions. We look next at these concepts of diffusion and confusion and then present the Feistel cipher. But first, it is worth commenting on this remarkable fact: The Feistel cipher structure, which dates back over a quarter century and which, in turn, is based on Shannon's proposal of 1945, is the structure used by many significant symmetric block ciphers currently in use.

1.7.1 Diffusion and Confusion

The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture the two basic building blocks for any cryptographic system.

Shannon's concern was to thwart cryptanalysis based on statistical analysis. The reasoning is as follows. Assume the attacker has some knowledge of the statistical characteristics of the plaintext. For example, in a human-readable message in some language, the frequency distribution of the various letters may be known. Or there may be words or phrases likely to appear in the message (probable words). If these statistics are in any way reflected in the

ciphertext, the cryptanalyst may be able to deduce the encryption key, or part of the key, or at least a set of keys likely to contain the exact key. In what Shannon refers to as a strongly ideal cipher, all statistics of the ciphertext are independent of the particular key used.

Shannon suggests two methods for frustrating statistical cryptanalysis: diffusion and confusion. In **diffusion**, the statistical structure of the plaintext is dissipated into long-range statistics of the ciphertext. This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally this is equivalent to having each ciphertext digit be affected by many plaintext digits. An example of diffusion is to encrypt a message $M = m1, m2, m3,...$ of characters with an averaging operation:

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

adding k successive letters to get a ciphertext letter yn. One can show that the statistical structure of the plaintext has been dissipated. Thus, the letter frequencies in the ciphertextwill be more nearly equal than in the plaintext; the digram frequencies will also be more nearly equal, and so on. In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation; the effect is that bits from different positions in the original plaintext contribute to a single bit of ciphertext.

Every block cipher involves a transformation of a block of plaintext into a block of ciphertext, where the transformation depends on the key. The mechanism of diffusion seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible in order to thwart attempts to deduce the key.

On the other hand, **confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, again to thwart attempts to discover the key. Thus, even if the attacker can get some handle on the statisticsof the ciphertext, the way in which the key was used to produce that ciphertext is so complex as to make it difficult to deduce the key. This is achieved by the use of a complex substitution algorithm. In contrast, a simple linear substitution function would add little confusion.

1.7.2 Feistel Cipher Structure

Figure 1.9 depicts the structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length 2w bits and a key K. The plaintext block is divided into two halves, $L0$ and $R0$. The two halves of the data pass through n rounds of processing and then combine to produce the ciphertext block. Each round i has as inputs $Li-1$ and $Ri-1$, derived from the previous round, as well as a subkey Ki, derived from the overall K. In general, the subkeys Ki are different from K and from each other.

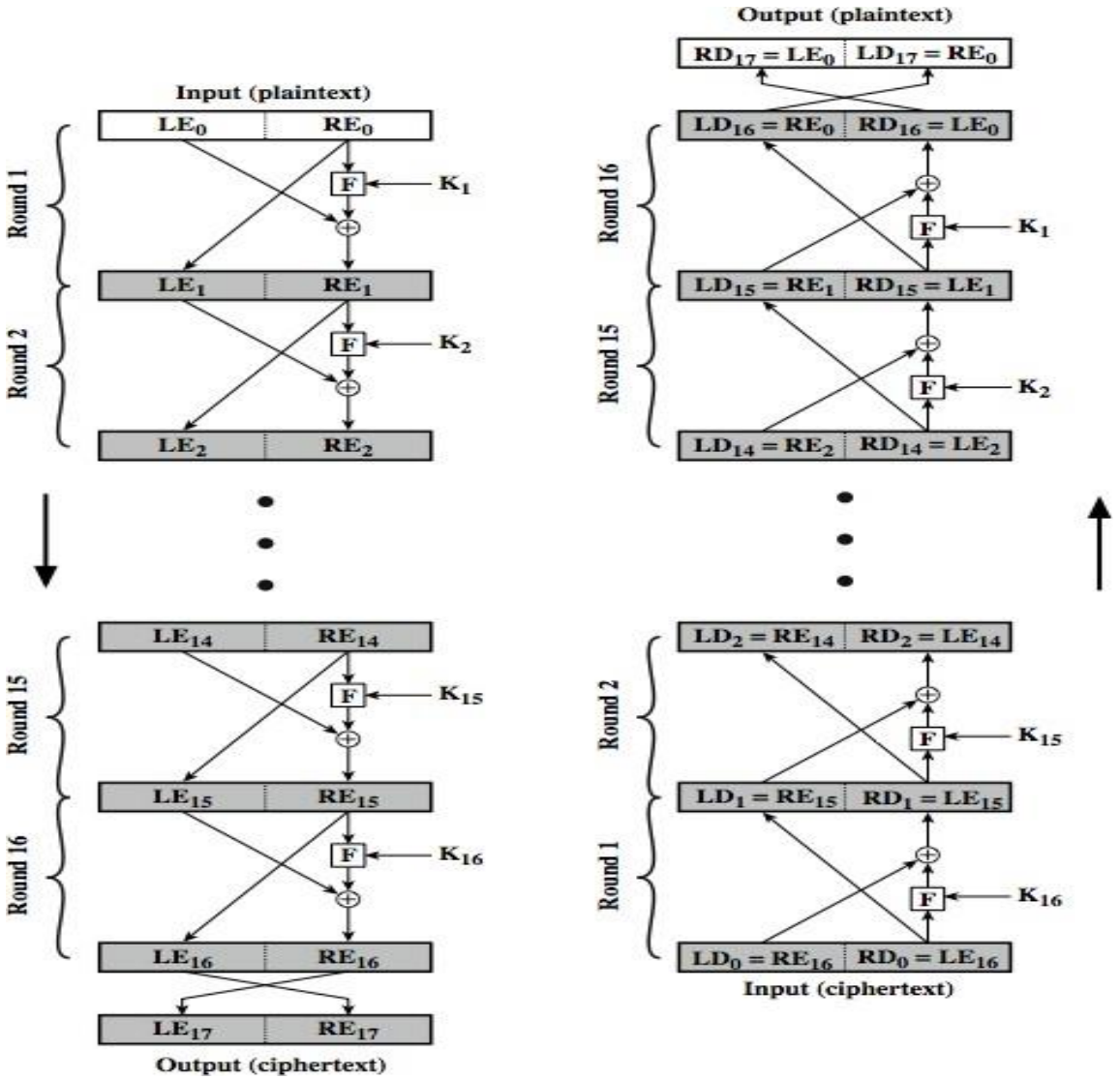


Figure 1.9 : Feistel Encryption and Decryption (16 rounds)

All rounds have the same structure. A **substitution** is performed on the left half of the data. This is done by applying a *round function* F to the right half of the data and then taking the exclusive-OR of the output of that function and the left half of the data. The round function has the same general structure for each round but is parameterized by the round subkey *Ki*. Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data. This structure is a particular form of the substitution-permutation network (SPN) proposed by Shannon.

The exact realization of a Feistel network depends on the choice of the following parameters and design features:

- **Block size:** Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion Traditionally, a block size of 64 bits has been considered a reasonable

tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.

- **Key size:** Larger key size means greater security but may decrease encryption/decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.
- **Number of rounds:** The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.
- **Subkey generation algorithm:** Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.
- **Round function:** Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a Feistel cipher:

- **Fast software encryption/decryption:** In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.
- **Ease of analysis:** Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

1.7.3 Feistel Decryption Algorithm

The process of decryption with a Feistel cipher is essentially the same as the encryption process. The rule is as follows: Use the ciphertext as input to the algorithm, but use the subkeys *Ki* in reverse order. That is, use *Kn* in the first round, *Kn-1* in the second round, and so on until *K*1 is used in the last round. This is a nice feature because it means we need not implement two different algorithms, one for encryption and one for decryption.

To see that the same algorithm with a reversed key order produces the correct result, consider Figure 1.9, which shows the encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm (the result would be the same for any number of rounds). For clarity, we use the notation LE*i* and RE*i* for data traveling through the encryption algorithm and *LDi* and RD*i* for data traveling through the decryption algorithm. The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. To put this another way, let the output of the *i*th encryption round be LE*i*||RE*i* (*Li* concatenated with *Ri*). Then the corresponding input to the (16 *i*) th decryption round is RE*i*||LE*i* or, equivalently, RD16- *i*||*LD*16-*i*.

Let us walk through Figure 1.9 to demonstrate the validity of the preceding assertions.

After the last iteration of the encryption process, the two halves of the output are swapped, so that the ciphertext is RE16||LE16. The output of that round is the ciphertext. Now take that ciphertext and use it as input to the same algorithm. The input to the first round is RE16||LE16, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process. Now we would like to show that the output of the first round of the decryption process is equal to a 32- bit swap of the input to the sixteenth round of the encryption process.

First, consider the encryption process.
We see that
LE16 = RE15
RE16 = LE15 x F(RE15, *K*16)

On the decryption side,
*LD*1 = RD0 = LE16 = RE15
RD1 = LD0 x F(RD0, *K*16)
= RE16 x F(RE15, *K*16)
= [LE15 x F(RE15, *K*16)] x F(RE15, *K*16)

The XOR has the following properties:
[*A* x *B*] x *C* = *A* x [*B* x *C*]
D x *D* = 0
E x 0 = E
Thus, we have *LD*1 = RE15 and RD1 = LE15. Therefore, the output of the first round of the decryption process is LE15||RE15, which is the 32-bit swap of the input to the sixteenth round of the encryption.
This correspondence holds all the way through the 16 iterations, as is easily shown. We can cast this process in general terms. For the *i*th iteration of the encryption algorithm,
LE*i* = RE*i-1*
RE*i* =LE*i*-1 x F(RE*i*-1, *Ki*)
Rearranging terms,
RE*i*-1 = LE*i*
LE*i*-1 = RE*i* x F(RE*i*-1, *Ki*2 = RE*i* x F(LE*i*, *Ki*)
Thus, we have described the inputs to the *i*th iteration as a function of the outputs, and these equations confirm the assignments shown in the right-hand side of Figure 1.9.
Finally, we see that the output of the last round of the decryption process is RE0||LE0. A 32-bit swap recovers the original plaintext, demonstrating the validity of the Feistel decryption process.
Note that the derivation does not require that F be a reversible function. To see this, take a limiting case in which F produces a constant output (e.g., all ones) regardless of the values of its two arguments. The equations still hold.

1.8 The Data Encryption Standard

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).

For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption. The DES enjoys widespread use. It has also been the subjectof much controversy concerning how secure the DES is. To appreciate the nature of the controversy, let us quickly review the history of the DES.

In the late 1960s, IBM set up a research project in computer cryptography led by Horst Feistel. The project concluded in 1971 with the development of an algorithm with thedesignation LUCIFER, which was sold to Lloyd's of London for use in a cash-dispensing system, also developed by IBM.

LUCIFER is a Feistel block cipher that operates on blocks of 64 bits, using a key size of 128 bits. Because of the promising results produced by the LUCIFER project, IBM embarked on an effort to develop a marketable commercial encryption product that ideally could be implemented on a single chip. The effort was headed by Walter Tuchman and Carl Meyer, and it involved not only IBM researchers but also outside consultants and technical advice from NSA. The outcome of this effort was a refined version of LUCIFER that was more resistant to cryptanalysis but that had a reduced key size of 56 bits, to fit on a single chip.

In 1973, the National Bureau of Standards (NBS) issued a request for proposals for a national cipher standard. IBM submitted the results of its Tuchman-Meyer project. This was by far the best algorithm proposed and was adopted in 1977 as the Data Encryption Standard.

Before its adoption as a standard, the proposed DES was subjected to intense criticism, which has not subsided to this day. Two areas drew the critics' fire. First, the key length in IBM's original LUCIFER algorithm was 128 bits, but that of the proposed system was only 56 bits, an enormous reduction in key size of 72 bits. Critics feared that this key length was too short to withstand brute-force attacks. The second area of concern was that the design criteria for the internal structure of DES, the S-boxes, were classified. Thus, users could not be sure that the internal structure of DES was free of any hidden weak points that would enable NSA to decipher messages without benefit of the key. Subsequent events, particularly the recent work on differential cryptanalysis, seem to indicate that DES has a very strong internal structure. Furthermore, according to IBM participants, the only changes that were made to the proposal were changes to the S-boxes, suggested by NSA, that removed vulnerabilities identified in the course of the evaluation process.

Whatever the merits of the case, DES has flourished and is widely used, especially in financial applications. In 1994, NIST reaffirmed DES for federal use for another five years; NIST recommended the use of DES for applications other than the protection of classified

information. In 1999, NIST issued a new version of its standard (FIPS PUB 46-3) that indicated that DES should only be used for legacy systems and that triple DES (which in essence involves repeating the DES algorithm three times on the plaintext using two or three different keys to produce the ciphertext) be used. We study triple DES. Because the underlying encryption and decryption algorithms are the same for DES and triple DES, it remains important to understand the DES cipher.

1.8.1 DES Encryption

The overall scheme for DES encryption is illustrated in Figure 1.10. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.

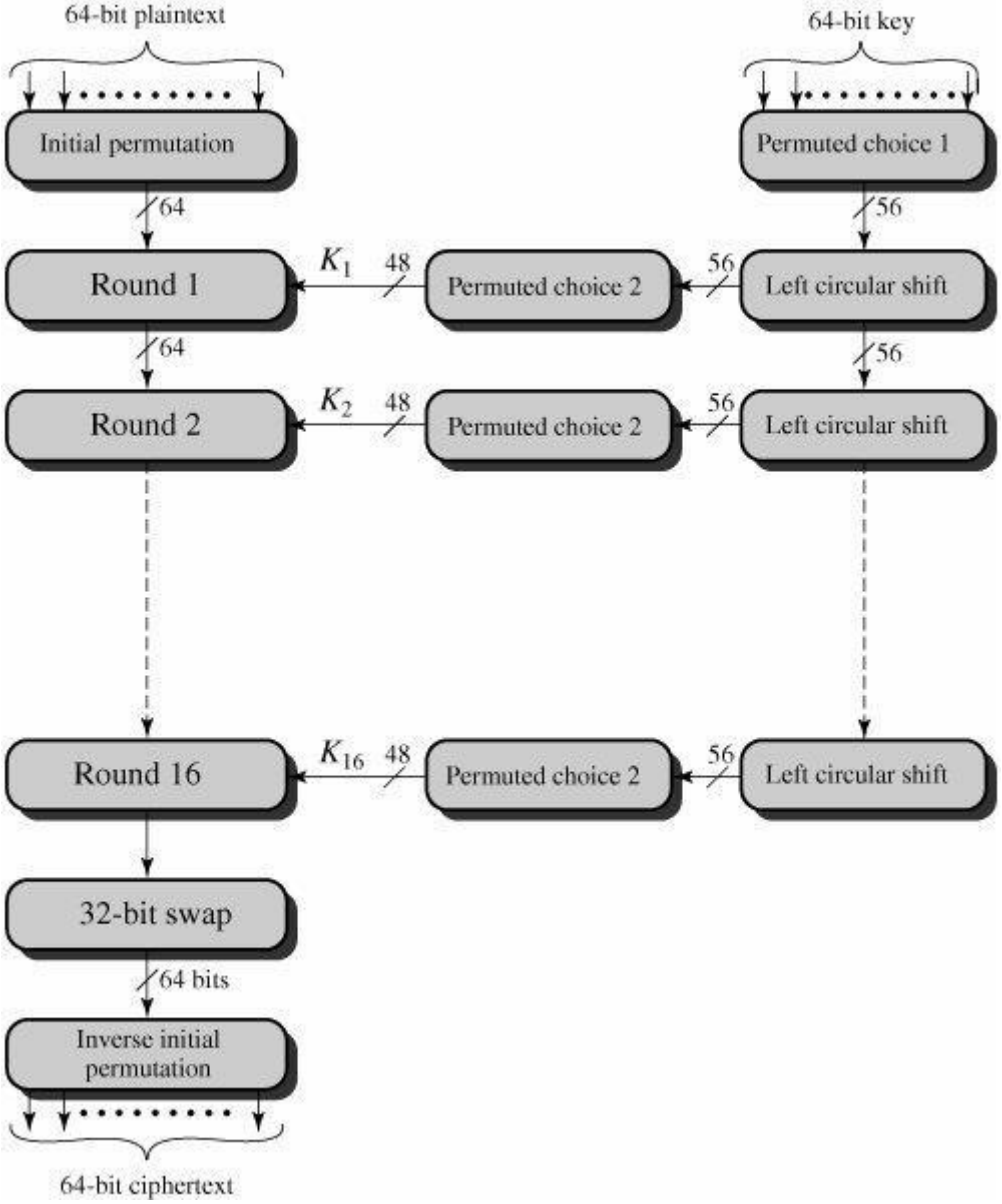


Figure 1.10: General description of DES encryption algorithm

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting

of 16 rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **preoutput**. Finally, the preoutput is passed through a permutation (IP-1) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher.

The right-hand portion of Figure 1.10 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the 16 rounds, a *subkey* (*Ki*) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits.

Initial Permutation

The initial permutation and its inverse are defined by tables, as shown in Tables 1.5a and 1.5b, respectively. The tables are to be interpreted as follows. The input to a table consists of 64 bits numbered from 1 to 64. The 64 entries in the permutation table contain a permutation of the numbers from 1 to 64. Each entry in the permutation table indicates the position of a numbered input bit in the output, which also consists of 64 bits.

Table 1.5. Permutation Tables for DES

(a) Initial Permutation (IP)							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7
(b) Inverse Initial Permutation (IP^{-1})							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27

34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

(c) Expansion Permutation (E)

	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	

(d) Permutation Function (P)

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

To see that these two permutation functions are indeed the inverse of each other, consider the following 64-bit input M:

M_1 M_2 M_3 M_4 M_5 M_6 M_7 M_8

M_9 M_{10} M_{11} M_{12} M_{13} M_{14} M_{15} M_{16}

M_{17} M_{18} M_{19} M_{20} M_{21} M_{22} M_{23} M_{24}

M_{25} M_{26} M_{27} M_{28} M_{29} M_{30} M_{31} M_{32}

M_{33} M_{34} M_{35} M_{36} M_{37} M_{38} M_{39} M_{40}

M_{41} M_{42} M_{43} M_{44} M_{45} M_{46} M_{47} M_{48}

M_{49} M_{50} M_{51} M_{52} M_{53} M_{54} M_{55} M_{56}

M_{57} M_{58} M_{59} M_{60} M_{61} M_{62} M_{63} M_{64}

where M_i is a binary digit. Then the permutation $X = \text{IP}(M)$ is as follows:

M_{58} M_{50} M_{42} M_{34} M_{26} M_{18} M_{10} M_2

M_{60} M_{52} M_{44} M_{36} M_{28} M_{20} M_{12} M_4

M_{62} M_{54} M_{46} M_{38} M_{30} M_{22} M_{14} M_6

M_{64} M_{56} M_{48} M_{40} M_{32} M_{24} M_{16} M_8

M_{57} M_{49} M_{41} M_{33} M_{25} M_{17} M_9 M_1

M_{59} M_{51} M_{43} M_{35} M_{27} M_{19} M_{11} M_3

M_{61} M_{53} M_{45} M_{37} M_{29} M_{21} M_{13} M_5

M_{63} M_{55} M_{47} M_{39} M_{31} M_{23} M_{15} M_7

If we then take the inverse permutation $Y = \text{IP}^{-1}(X) = \text{IP}^{-1}(\text{IP}(M))$, it can be seen that the original ordering of the bits is restored.

Details of Single Round

Figure 1.11 shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32- bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$Li = Ri\text{-}1$

$Ri = \text{L}i\text{-}1 \text{ x } \text{F}(Ri\text{-}1, Ki)$

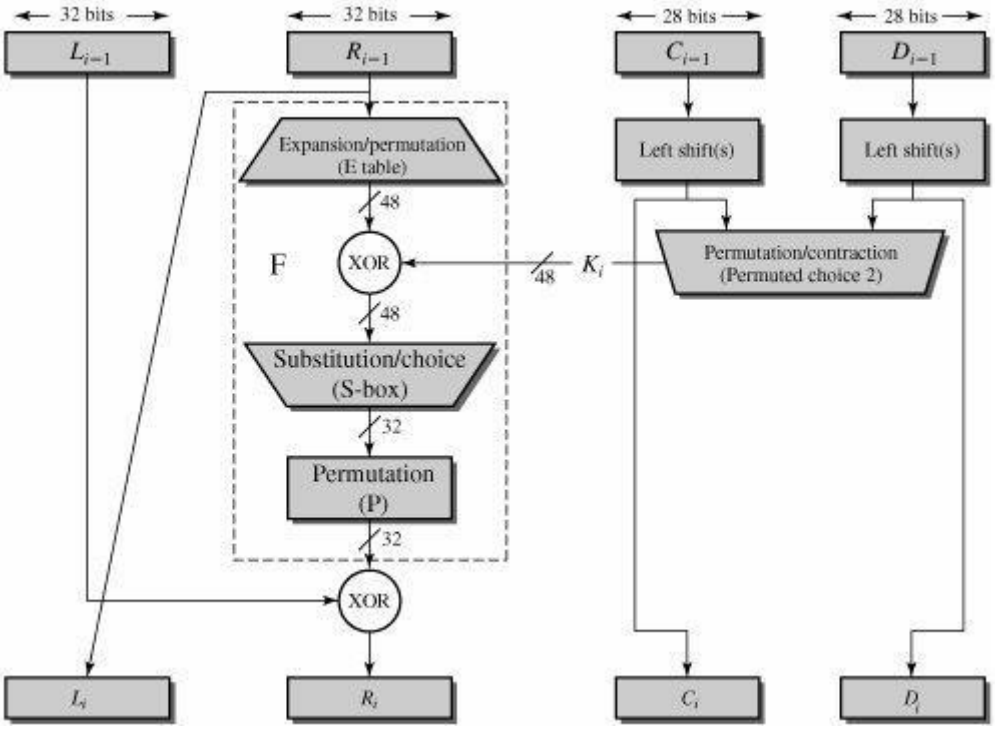


Figure 1.11: Single Round of DES Algorithm

The round key *Ki* is 48 bits. The *R* input is 32 bits. This *R* input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the *R* bits (Table 1.5c). The resulting 48 bits are XORed with *Ki*. This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined byTable 1.5d. The role of the S-boxes in the function F is illustrated in Figure 1.12. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 1.6, which is interpretedas follows: The first and last bits of the input to box S*i* form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S*i*. The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in S1 for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

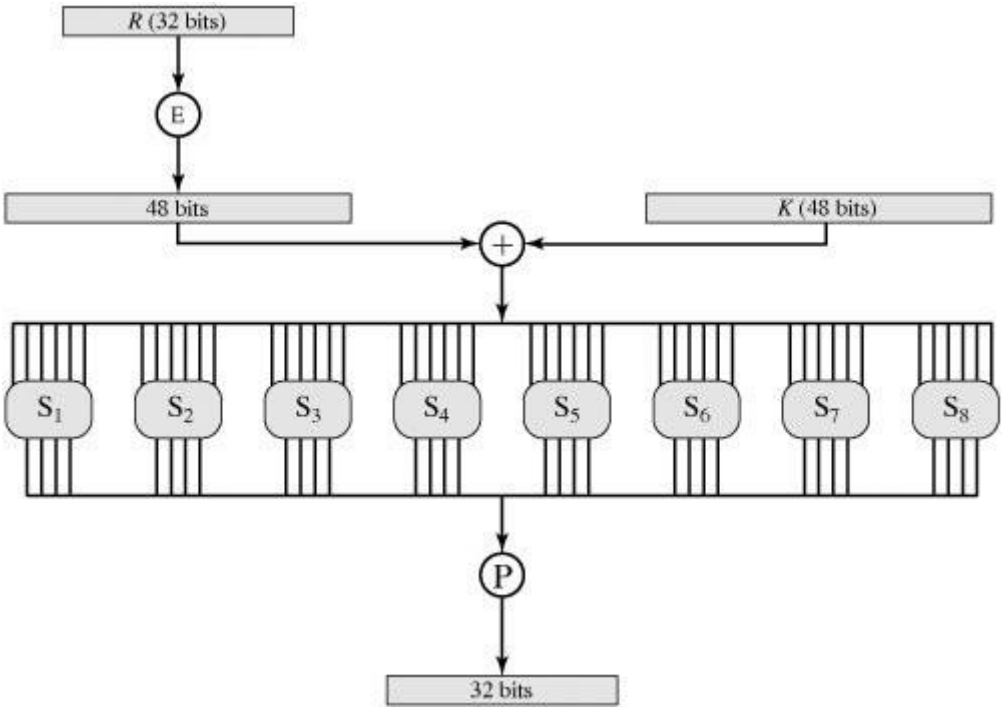


Figure 1.12: Calculation of F(R, K)

Table 1.6: Definition of DES S-Boxes

S_1	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
S_2	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
S_3	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
S_4	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
S_5	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
S_6	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
S_7	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
S_8	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

Each row of an S-box defines a general reversible substitution. The operation of the S-boxes is worth further comment. Ignore for the moment the contribution of the key (*Ki*). If you

examine the expansion table, you see that the 32 bits of input are split into groups of 4 bits, and then become groups of 6 bits by taking the outer bits from the two adjacent groups. For example, if part of the input word is

... efgh ijkl mnop ...

this becomes

... defghi hijklm lmnopq ...

The outer two bits of each group select one of four possible substitutions (one row of an S-box). Then a 4-bit output value is substituted for the particular 4-bit input (the middle four input bits). The 32-bit output from the eight S-boxes is then permuted, so that on the next round the output from each S-box immediately affects as many others as possible.

Key Generation

Returning to Figures 1.10 and 1.11, we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored, as indicated by the lack of shading in Table 1.7a. The key is first subjected to a permutation governed by a table labeled Permuted Choice One (Table 1.7b). The resulting 56-bit key is then treated as two 28-bit quantities, labeled *C*0 and *D*0. At each round, *Ci*-1 and *Di*-1 are separately subjected to a circular left shift, or rotation, of 1 or 2 bits, as governed by Table 1.7d. These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two (Table 1.7c), which produces a 48-bit output that serves as input to thefunction F(*Ri*-1, *Ki*).

Table 1.7: DES Key Schedule Calculation

(a) Input Key															
	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	
	57		58		59		60		61		62		63		64	

(b) Permuted 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		
		19		11		3		60		52		44		36		
		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		

(c) Permuted Choice Two (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	

(d) Schedule of Left Shifts

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

DES Decryption

As with any Feistel cipher, decryption uses the same algorithm as encryption, except that the application of the subkeys is reversed.

The Avalanche Effect

A desirable property of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. In particular, a change in one bit of the plaintext or one bit of the key should produce a change in many bits of the

ciphertext. If the change were small, this might provide a way to reduce the size of the plaintext or key space to be searched.

DES exhibits a strong avalanche effect. Table 1.8 shows some results taken from. In Table 1.8a, two plaintexts that differ by one bit were used:

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

with the key

0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

Table 1.8 : Avalanche effect in DES

(a) Change in Plaintext		(b) Change in Key	
Round	**Number of bits that differ**	**Round**	**Number of bits that differ**
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

The Table 1.8a shows that after just three rounds, 21 bits differ between the two blocks. On completion, the two ciphertexts differ in 34 bit positions. Table 1.8b shows a similar test in which a single plaintext is input:

01101000 10000101 00101111 01111010 00010011 01110110 11101011 10100100

with two keys that differ in only one bit position:

1110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

0110010 1111011 1101111 0011000 0011101 0000100 0110001 11011100

Again, the results show that about half of the bits in the ciphertext differ and that the avalanche effect is pronounced after just a few rounds.

1.9 The Strength of DES

Since its adoption as a federal standard, there have been lingering concerns about the level of security provided by DES. These concerns, by and large, fall into two areas: key size and the nature of the algorithm.

The Use of 56-Bit Keys

With a key length of 56 bits, there are 2^56 possible keys, which is approximately 7.2 x 10^16. Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DESencryption per microsecond would take more than a thousand years to break the cipher.

However, the assumption of one encryption per microsecond is overly conservative. As far back as 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which could perform one encryption per microsecond. This would bring the average search time down to about 10 hours. The authors estimated that the cost would be about $20 million in 1977 dollars.

DES finally and definitively proved insecure in July 1998, when the Electronic Frontier Foundation (EFF) announced that it had broken a DES encryption using a special-purpose "DES cracker" machine that was built for less than $250,000. The attack took less than three days. The EFF has published a detailed description of the machine, enabling others to build their own cracker. And, of course, hardware prices will continue to drop as speeds increase, making DES virtually worthless.

It is important to note that there is more to a key-search attack than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext. If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And ifthe message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed. The EFF approach addresses this issue as well and introduces some automated techniques that would be effective in many contexts.

The Nature of the DES Algorithm

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. This assertion is tantalizing, and over the years a number of regularities and unexpected behaviors of the S-boxes have been discovered. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.

Timing Attacks

A timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs.The Hamming weight (number of bits equal to one) of the secret key. This is a long way from knowing the actual key, but it is an intriguing first step. The authors conclude that DES appears to be fairly resistant to a successful timing attack but suggest some avenues to explore. Although this is an interesting line of attack, it so far appears unlikely that this technique will ever be successful against DES or more powerful symmetric ciphers such as triple DES and AES.

1.10 Block Cipher Design Principles

Number of Rounds

The cryptographic strength of a Feistel cipher derives from three aspects of the design: the number of rounds, the function F, and the key schedule algorithm. Let us look first at the choice of the number of rounds.

The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F. In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack. This criterion was certainly used in the design of DES. Schneier observes that for 16-round DES, a differential cryptanalysis attack is slightly less efficient than brute force: the differential cryptanalysis attack requires 2^55.1 operations, whereas brute force requires2^55. If DES had 15 or fewer rounds, differential cryptanalysis would require less effort than brute-force key search.

This criterion is attractive because it makes it easy to judge the strength of an algorithm and to compare different algorithms. In the absence of a cryptanalytic breakthrough, the strength of any algorithm that satisfies the criterion can be judged solely on key length.

Design of Function F

The heart of a Feistel block cipher is the function F. As we have seen, in DES, this function relies on the use of S-boxes. This is also the case for most other symmetric block ciphers. However, we can make some general comments about the criteria for designing F. After that, we look specifically at S-box design.

Design Criteria for F

The function F provides the element of confusion in a Feistel cipher. Thus, it must be difficult to "unscramble" the substitution performed by F. One obvious criterion is that F be nonlinear, as we discussed previously. The more nonlinear F, the more difficult any type of cryptanalysis will be. There are several measures of nonlinearity, which are beyond the scopeof this book. In rough terms, the more difficult it is to approximate F by a set of linear equations, the more nonlinear F is. Several other criteria should be considered in designing F.We would like the algorithm to have good avalanche properties. Recall that, in general, this means that a change in one bit of the input should produce a change in many bits of the output. A more stringent version of this is the **strict avalanche criterion (SAC)**, which statesthat any output bit *j* of an S-box should change with probability 1/2 when any single input bit*i* is inverted for all *i, j*. Although SAC is expressed in terms of Sboxes, a similar criterion could be applied to F as a whole. This is important when considering designs
that do not include S-boxes.

Another criterion proposed is the **bit independence criterion (BIC)**, which states that output bits *j* and *k* should change independently when any single input bit *i* is inverted, for all *i, j*, and *k*. The SAC and BIC criteria appear to strengthen the effectiveness of the confusion function.

Key Schedule Algorithm

A final area of block cipher design, and one that has received less attention than S-box design, is the key schedule algorithm. With any Feistel block cipher, the key is used to generate one subkey for each round. In general, we would like to select subkeys to maximize the difficulty of deducing individual subkeys and the difficulty of working back to the main key. No general principles for this have yet been promulgated.

Hall suggests that, at minimum, the key schedule should guarantee key/ciphertext Strict Avalanche Criterion and Bit Independence Criterion.

Module-2:

Public-Key Cryptography and RSA, Other Public-Key Cryptosystems

The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. From its earliest beginnings to modern times, virtually all cryptographic systems have been based on the elementary tools of substitution and permutation. After millennia of working with algorithms that could essentially be calculated by hand, a major advance in symmetric cryptography occurred with the development of the rotor encryption/decryption machine.

The electromechanical rotor enabled the development of fiendishly complex cipher systems. With the availability of computers, even more complex systems were devised, the most prominent of which was the Lucifer effort at IBM that culminated in the Data Encryption Standard (DES). But both rotor machines and DES, although representing significant advances, still relied on the bread-and-butter tools of substitution and permutation.

Public-key cryptography provides a radical departure from all that has gone before. For one thing, public key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

Before proceeding, we should mention several common misconceptions concerning public-key encryption. One such misconception is that public-key encryption is more secure from cryptanalysis than is symmetric encryption. In fact, the security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher. There is nothing in principle about either symmetric or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis.

Second misconception is that public-key encryption is a general-purpose technique that has made symmetric encryption obsolete. On the contrary, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned. As one of the inventors of public-key encryption has put it "the restriction of public-key cryptography to key management and signature applications is almost universally accepted."

2.1 Principles of Public-Key Cryptosystems

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption. The first problem is that of key distribution.

Key distribution under symmetric encryption requires either

(1) that two communicants already share a key, which somehow has been distributed to them; or
(2) the use of a key distribution center.

Whitfield Diffie, one of the discoverers of public-key encryption (along with Martin Hellman, both at Stanford University at the time), reasoned that this second requirement negated the very essence of cryptography: the ability to maintain total secrecy over your own communication.

As Diffie put it , "what good would it do after all to develop impenetrable cryptosystems, if their users were forced to share their keys with a KDC that could be compromised by either burglary or subpoena?"

The second problem that Diffie pondered, and one that was apparently unrelated to the first was that of "digital signatures." If the use of cryptography was to become widespread, not just in military situations but for commercial and private purposes, then electronic messages and documents would need the equivalent of signatures used in paper documents. That is, could a method be devised that would stipulate, to the satisfaction of all parties, that a digital message had been sent by a particular person? This is a somewhat broader requirement than that of authentication, and its characteristics and ramifications.

2.1.1 Public-Key Cryptosystems

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption.

These algorithms have the following important characteristic:

● It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

● Either of the two related keys can be used for encryption, with the other used for decryption.

A public-key encryption scheme has six ingredients

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

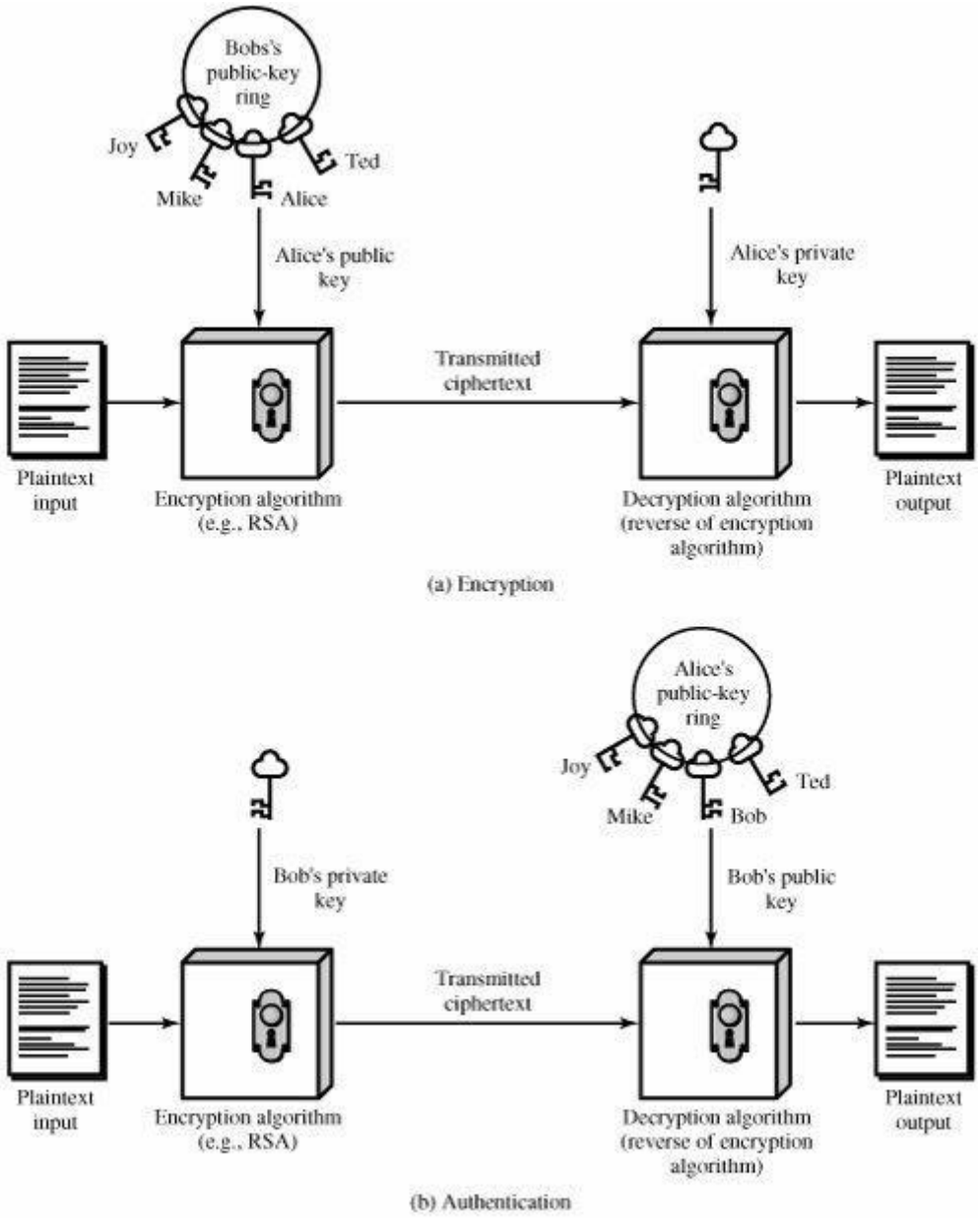


Figure 2.1: Public-Key Cryptography

The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.

2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure 2.1a suggests, each user maintains a collection of public keys obtained from others.

3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.

4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a

user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

Table 2.1 summarizes some of the important aspects of symmetric and public-key encryption. To discriminate between the two, we refer to the key used in symmetric encryption as a **secret key**. The two keys used for asymmetric encryption are referred to asthe **public key** and the **private key**. Invariably, the private key is kept secret, but it is referredto as a private key rather than a secret key to avoid confusion with symmetric encryption.

Table 2.1: Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 2.2. There is some source A that produces a message in plaintext, $X = [X1, X2, ..., XM,]$. The M elements of X are letters in some finite alphabet. The message is intended for destination B. B generates a related pair of keys: a public key, *PUb*, and a private key, *PUb*. *PUb* is known only to B, whereas *PUb* is publicly available and therefore accessible by A.

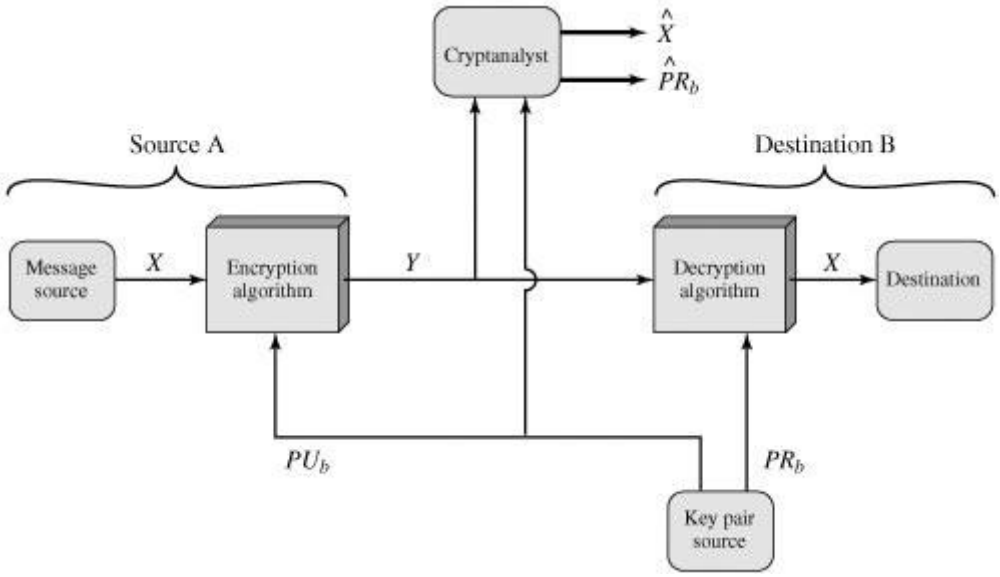


Figure 2.2: Public-Key Cryptosystem: Confidentiality

With the message *X* and the encryption key *PUb* as input, A forms the ciphertext *Y* = [*Y*1, *Y*2,..., *YN*]:

$$Y = E(PUb, X)$$

The intended receiver, in possession of the matching private key, is able to invert the transformation:

$$X = D(PRb, Y)$$

An adversary, observing *Y* and having access to *PUb* but not having access to *PRb* or *X*, must attempt to recover *X* and/or *PRb*. It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms.

If the adversary is interested only in this particular message, then the focus of effort is to recover *X*, by generating a plaintext estimate Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover *PRb* by generating an estimate .

We mentioned earlier that either of the two related keys can be used for encryption, with the other being used for decryption. This enables a rather different cryptographic scheme to be implemented. Whereas the scheme illustrated in Figure 2.2 provides confidentiality, Figures 2.1b and 2.3 show the use of public-key encryption to provide authentication:

$$Y = E(PRa, X)$$
$$Y = E(PUa, Y)$$

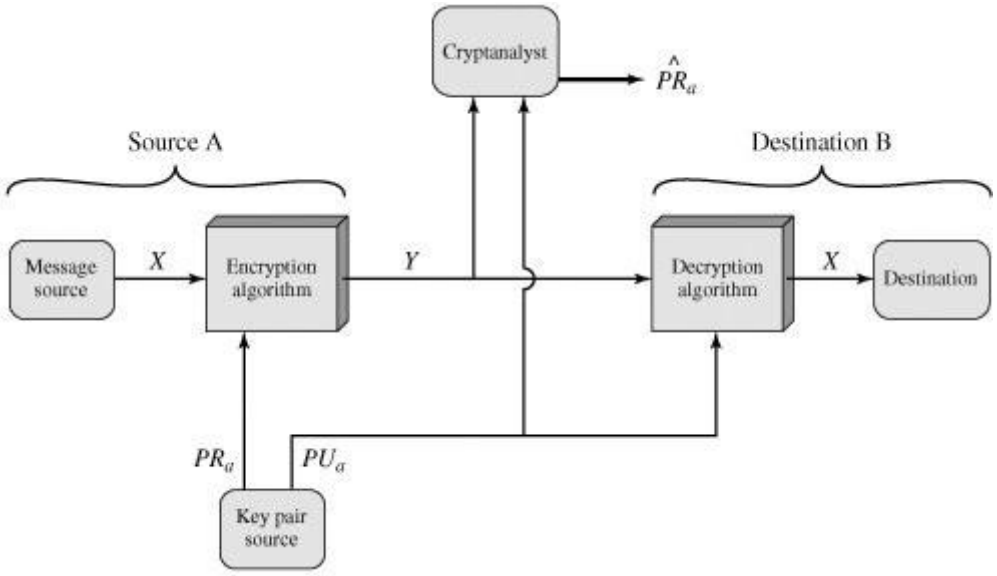


Figure 2.3: Public-Key Cryptosystem: Authentication

In this case, A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a *digital signature*. In addition, it is impossible to alter themessage without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.

In the preceding scheme, the entire message is encrypted, which, although validating both author and contents, requires a great deal of storage. Each document must be kept in plaintext to be used for practical purposes. A copy also must be stored in ciphertext so that the origin and contents can be verified in case of a dispute. A more efficient way of achieving the same results is to encrypt a small block of bits that is a function of the document. Such a block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted with thesender's private key, it serves as a signature that verifies origin, content, and sequencing.

It is important to emphasize that the encryption process depicted in Figures 2.1b and 2.3 does not provide confidentiality. That is, the message being sent is safe from alteration but not from eavesdropping. This is obvious in the case of a signature based on a portion of the message, because the rest of the message is transmitted in the clear. Even in the case of complete encryption, as shown in Figure 2.3, there is no protection of confidentiality because any observer can decrypt the message by using the sender's public key.

It is possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (Figure 2.4):

$$Z = \mathrm{E}(PUb, \mathrm{E}(PRa, X))$$
$$X = \mathrm{D}(PUa, \mathrm{E}(PRb, Z))$$

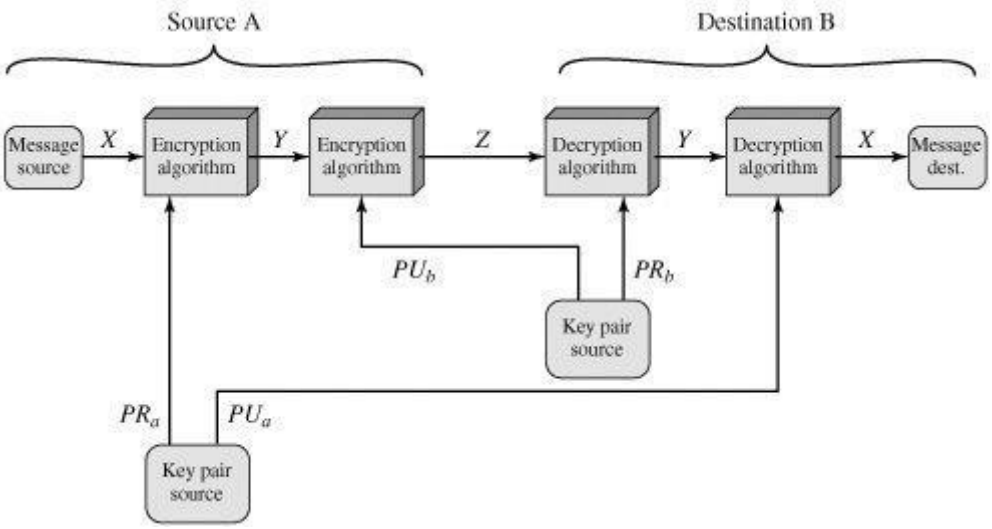


Figure 2.4: Public-Key Cryptosystem: Authentication and Secrecy

In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided. The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than twoin each communication.

2.1.2 Applications for Public-Key Cryptosystems

Before proceeding, we need to clarify one aspect of public-key cryptosystems that is otherwise likely to lead to confusion. Public-key systems are characterized by the use of a cryptographic algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender's private key or the receiver's public key, or both, to perform some type of cryptographic function. In broad terms, we can classify the use of public-key cryptosystems into three categories:

- **Encryption/decryption:** The sender encrypts a message with the recipient's public key.
- **Digital signature:** The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
- **Key exchange:** Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 2.2 indicates the applications supported by thealgorithms.

Table 2.2: Applications for Public-Key Cryptosystems

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

2.1.3 Requirements for Public-Key Cryptography

The cryptosystem illustrated in Figures 9.2 through 9.4 depends on a cryptographic algorithm based on two related keys. Diffie and Hellman postulated this system without demonstrating that such algorithms exist. However, they did lay out the conditions that such algorithms must fulfil:

1. It is computationally easy for a party B to generate a pair (public key *PUb*, private key *PRb*).

2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, *M*, to generate the corresponding ciphertext: $C = E(PUb, M)$

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message: $M = D(PRb, C) = D[PRb, E(PUb, M)]$

4. It is computationally infeasible for an adversary, knowing the public key, *PUb*, to determine the private key, *PRb*.

5. It is computationally infeasible for an adversary, knowing the public key, *PUb*, and a ciphertext, *C*, to recover the original message, *M*.

We can add a sixth requirement that, although useful, is not necessary for all public-key applications:

6. The two keys can be applied in either order:

$M = D[PUb, E(PRb, M)] = D[PRb, E(PUb, M)]$

These are formidable requirements, as evidenced by the fact that only a few algorithms (RSA, elliptic curve cryptography, Diffie-Hellman, DSS) have received widespread acceptance in the several decades since the concept of public-key cryptography was proposed.

Before elaborating on why the requirements are so formidable, let us first recast them. The requirements boil down to the need for a trap-door one-way function. A *one-way function* is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy whereas the calculation of the inverse is infeasible:

$$Y = f(X) \text{ easy}$$
$$X = f1(X) \text{ infeasible}$$

Generally, *easy* is defined to mean a problem that can be solved in polynomial time as a function of input length. Thus, if the length of the input is *n* bits, then the time to compute the function is proportional to n^a where *a* is a fixed constant. Such algorithms are said to belong to the class **P**. The term *infeasible* is a much fuzzier concept. In general, we can say a problem is infeasible if the effort to solve it grows faster than polynomial time as a function

of input size. For example, if the length of the input is *n* bits and the time to compute the function is proportional to 2*n*, the problem is considered infeasible. Unfortunately, it is difficult to determine if a particular algorithm exhibits this complexity.

Furthermore, traditional notions of computational complexity focus on the worst-case or average-case complexity of an algorithm. These measures are inadequate for cryptography, which requires that it be infeasible to invert a function for virtually all inputs, not for the worst case or even average case.

We now turn to the definition of a *trap-door one-way function*, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known. With the additional information the inverse can be calculated in polynomial time. We can summarize as follows: A trap-door one-way function is a family of invertible functions f*k*, such that

$Y = f_k(X)$ easy, if k and X are known

$X = f_k^{-1}(Y)$ easy, if k and Y are known

$X = f_k^{-1}(Y)$ infeasible, if Y is known but k is not known

Thus, the development of a practical public-key scheme depends on discovery of a suitable trap-door one-way function.

2.1.4 Public-Key Cryptanalysis

As with symmetric encryption, a public-key encryption scheme is vulnerable to a brute-force attack. The countermeasure is the same: Use large keys. However, there is a tradeoff to be considered. Public-key systems depend on the use of some sort of invertible mathematical function. The complexity of calculating these functions may not scale linearly with the number of bits in the key but grow more rapidly than that. Thus, the key size must belarge enough to make brute-force attack impractical but small enough for practical encryptionand decryption. In practice, the key sizes that have been proposed do make brute-force attack impractical but result in encryption/decryption speeds that are too slow for general-purpose use. Instead, as was mentioned earlier, public-key encryption is currently confined to key management and signature applications.

Another form of attack is to find some way to compute the private key given the public key. To date, it has not been mathematically proven that this form of attack isinfeasible for a particular public-key algorithm. Thus, any given algorithm, including the widely used RSA algorithm, is suspect. The history of cryptanalysis shows that a problemthat seems insoluble from one perspective can be found to have a solution if looked at in an entirely different way.

Finally, there is a form of attack that is peculiar to public-key systems. This is, in essence, a probable message attack. Suppose, for example, that a message were to be sent that consisted solely of a 56-bit DES key. An adversary could encrypt all possible 56-bit DES keys using the public key and could discover the encrypted key by matching the transmitted ciphertext. Thus, no matter how large the key size of the public-key scheme, the attack is reduced to a brute-force attack on a 56-bit key. This attack can be thwarted by appending some random bits to such simple messages.

2.2 The RSA Algorithm

The pioneering paper by Diffie and Hellman introduced a new approach to cryptography and, in effect, challenged cryptologists to come up with a cryptographic algorithm that met the requirements for public-key systems. One of the first of the responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978.

The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption. The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and *n*-1 for some *n*. A typical size for *n* is 1024 bits, or 309 decimal digits.That is, *n* is less than 21024. We examine RSA in this section in some detail, beginning with an explanation of the algorithm. Then we examine some of the computational and cryptanalytical implications of RSA.

2.2.1 Description of the Algorithm

The scheme developed by Rivest, Shamir, and Adleman makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number *n*. That is, the block size must be less than or equal to log2(*n*); in practice, the block size is *i* bits, where 2^*i* < *n*<=2^*i*+1. Encryption and decryption are of the following form, for some plaintext block *M* and ciphertext block *C*:

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

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{(ed)} \bmod n$$

Both sender and receiver must know the value of *n*. The sender knows the value of *e*, and only the receiver knows the value of *d*. Thus, this is a public-key encryption algorithm with a public key of $PU = \{e, n\}$ and a private key of $PU = \{d, n\}$. For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

1. It is possible to find values of *e, d, n* such that $M^{(ed)} \bmod n = M$ for all $M < n$.

2. It is relatively easy to calculate mod $M^e \bmod n$ and *Cd* for all values of $M < n$.

3. It is infeasible to determine *d* given *e* and *n*.

For now, we focus on the first requirement and consider the other questions later. We need to find a relationship of the form $M^{(ed)} \bmod n = M$

The preceding relationship holds if *e* and *d* are multiplicative inverses modulo f(*n*), where f(*n*) is the Euler totient function. It is shown that for *p, q* prime, f(*pq*) = (*p* 1)(*q* 1) The relationship between *e* and *d* can be expressed as

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

This is equivalent to saying

ed =1 mod f(*n*)

d =*e*^-1 mod f(*n*)

That is, *e* and *d* are multiplicative inverses mod f(*n*). Note that, according to the rules of modular arithmetic, this is true only if *d* (and therefore *e*) is relatively prime to f(*n*). Equivalently, gcd(f(*n*),*d*) = 1.

We are now ready to state the RSA scheme. The ingredients are the following:

p,q, two prime numbers (private, chosen)

$n = pq$ (public, calculated)
e, with gcd(f(n),e) = 1;1 < e < f(n) (public, chosen)
d= e^-1(mod f(n)) (private, calculated)
The private key consists of {d, n} and the public key consists of {e, n}. Suppose that user A has published its public key and that user B wishes to send the message M to A. Then B calculates $C = M^e \bmod n$ and transmits C. On receipt of this ciphertext, user A decrypts by calculating $M = C^d \bmod n$.

Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

Figure 2.5: The RSA Algorithm

Figure 2.5 summarizes the RSA algorithm. An example is shown in Figure 2.6. For this example, the keys were generated as follows:

1. Select two prime numbers, p = 17 and q = 11.
2. Calculate $n = pq$ = 17 x 11 = 187.
3. Calculate f(n) = (p 1)(q 1) = 16 x 10 = 160.
4. Select e such that e is relatively prime to f(n) = 160 and less than f(n) we choose e = 7.
5. Determine d such that de= 1 (mod 160) and d < 160. The correct value is d = 23, because 23 x 7 = 161 = 10 x 160 + 1; d can be calculated using the extended Euclid's algorithm.

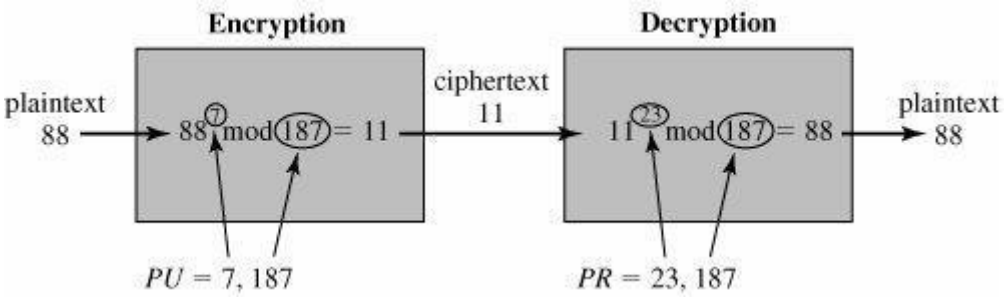


Figure 2.6: Example of RSA Algorithm

The resulting keys are public key $PU = \{7,187\}$ and private key $PR = \{23,187\}$. The example shows the use of these keys for a plaintext input of $M = 88$. For encryption, we need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows:

$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$

$88^1 \bmod 187 = 88$

$88^2 \bmod 187 = 7744 \bmod 187 = 77$

$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$

$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11$

For decryption, we calculate $M = 11^{23} \bmod 187$:

$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$

$11^1 \bmod 187 = 11$

$11^2 \bmod 187 = 121$

$11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$

$11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$

$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79{,}720{,}245 \bmod 187 = 88$

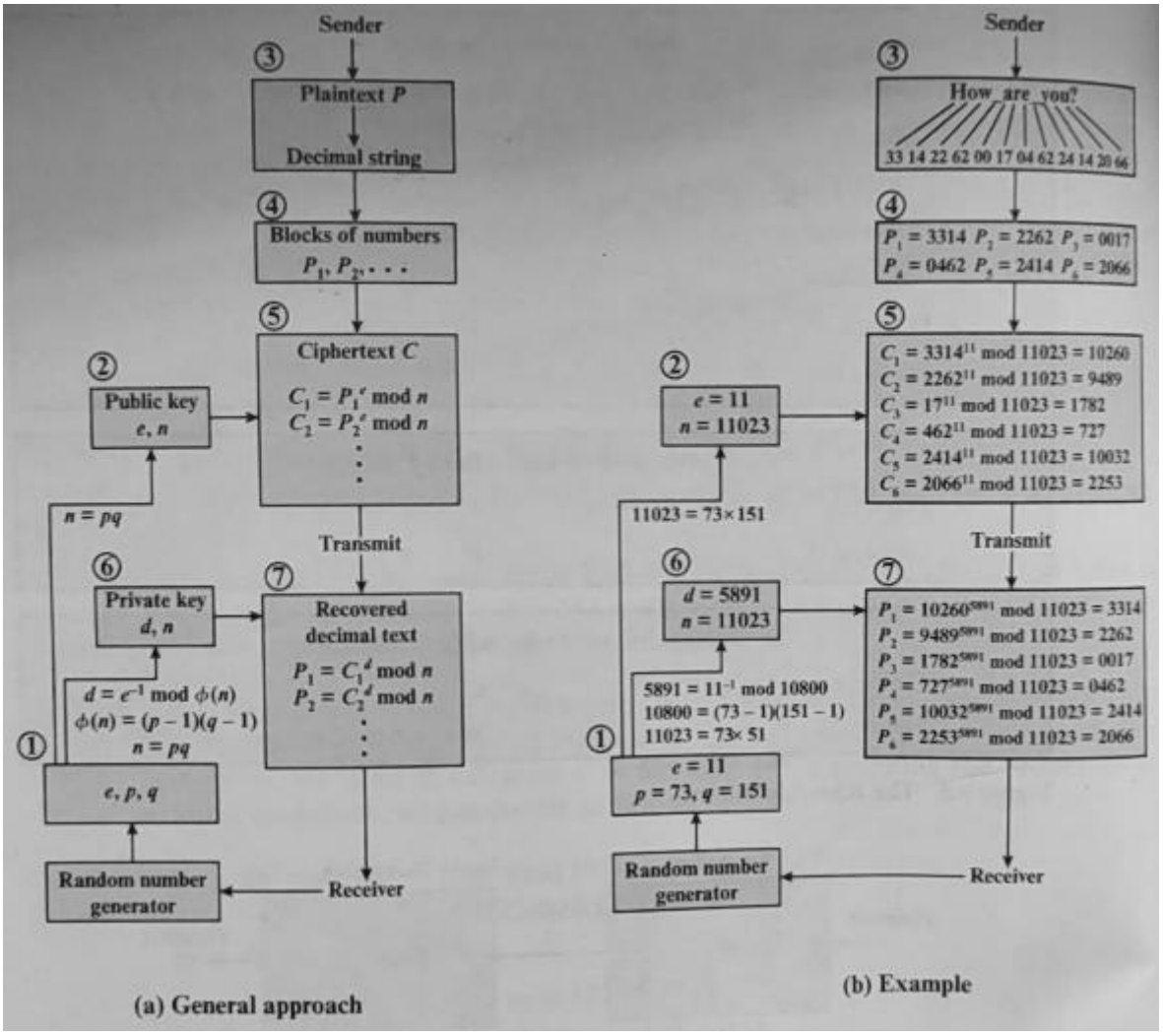


Figure 2.7: RSA Processing of Multiple Blocks

2.2.2 Computational Aspects

We now turn to the issue of the complexity of the computation required to use RSA. There are actually two issues to consider: encryption/decryption and key generation. Let us look first at the process of encryption and decryption and then consider key generation.

2.2.3 Exponentiation in Modular Arithmetic

Both encryption and decryption in RSA involve raising an integer to an integer power, mod *n*. If the exponentiation is done over the integers and then reduced modulo *n*, the intermediate values would be gargantuan. Fortunately, as the preceding example shows, we can make use of a property of modular arithmetic:

$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$

Thus, we can reduce intermediate results modulo *n*. This makes the calculation practical.

Another consideration is the efficiency of exponentiation, because with RSA we are dealing with

potentially large exponents. To see how efficiency might be increased, consider that we wish to compute x^{16}. A straightforward approach requires 15 multiplications:

$x^{16} = x \times x$

However, we can achieve the same final result with only four multiplications if we repeatedly take the square of each partial result, successively forming x^2, x^4, x^8, x^{16}. As another example, suppose we wish to calculate $x^{11} \bmod n$ for some integers x and n. Observe that $x^{11} = x^{1+2+8} = (x)(x^2)(x^8)$. In this case we compute $x \bmod n$, $x^2 \bmod n$, $x^4 \bmod n$, and $x^8 \bmod n$ and then calculate $[(x \bmod n) \times (x^2 \bmod n) \times (x^8 \bmod n)] \bmod n$.

More generally, suppose we wish to find the value ab with a and b positive integers. If we express b as a binary number $b_k b_{k1} \ldots b_0$ then we have

$$b = \sum_{b_i \neq 0} 2^i$$

Therefore,

$$a^b = a^{\left(\sum_{b_i \neq 0} 2^i\right)} = \prod_{b_i \neq 0} a^{(2^i)}$$

$$a^b \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)}\right] \bmod n = \left(\prod_{b_i \neq 0} \left[a^{(2^i)} \bmod n\right]\right) \bmod n$$

We can therefore develop the algorithm for computing $ab \bmod n$, shown in Figure 2.8. Table 2.3 shows an example of the execution of this algorithm. Note that the variable c is not needed; it is included for explanatory purposes. The final value of c is the value of the exponent.

```
c ← 0; f ← 1
for i ← k downto 0
    do  c ← 2 × c
        f ← (f × f) mod n
    if  bi = 1
        then c ← c + 1
             f ← (f × a) mod n
return f
```

Note: The integer b is expressed as a binary number $b_k b_{k1} \ldots b_0$

Figure 2.8: Algorithm for Computing *a^b* mod *n*

Table 2.3: Result of the Fast Modular Exponentiation Algorithm for *ab* mod *n*, where *a* = 7, *b* = 560 = 1000110000, *n* = 561

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
f	7	49	157	526	160	241	298	166	67	1

2.2.4 Efficient Operation Using the Public Key

To speed up the operation of the RSA algorithm using the public key, a specific choice of *e* is usually made. The most common choice is 65537 (216 1); two other popular choices are 3 and 17. Each of these choices has only two 1 bits and so the number of multiplications required to perform exponentiation is minimized.

However, with a very small public key, such as *e* = 3, RSA becomes vulnerable to a simple attack. Suppose we have three different RSA users who all use the value *e* = 3 but have unique values of *n*, namely *n*1, *n*2, *n*3. If user A sends the same encrypted message *M* toall three users, then the three ciphertexts are *C*1 = *M*3 mod *n*1; *C*2 = *M*3 mod *n*2; *C*3 = *M*3 mod *n*3. It is likely that *n*1, *n*2, and *n*3 are pairwise relatively prime. Therefore, one can use the Chinese remainder theorem (CRT) to compute *M*3 mod (*n*1*n*2*n*3). By the rules of the RSA algorithm, *M* is less than each of the *ni* therefore *M*3 < *n*1*n*2*n*3.

Accordingly, the attacker need only compute the cube root of *M*3. This attack can be countered by adding a unique pseudorandom bit string as padding to each instance of *M* to be encrypted.

The reader may have noted that the definition of the RSA algorithm requires that during key generation the user selects a value of *e* that is relatively prime to f(*n*). Thus, for example, if a user has preselected *e* = 65537 and then generated primes *p* and *q*, it may turn out that gcd(f(*n*),*e*) is not equal to 1, Thus, the user must reject any value of *p* or *q* that is not congruent to 1 (mod 65537).

2.2.5 Efficient Operation Using the Private Key

We cannot similarly choose a small constant value of *d* for efficient operation. A small value of *d* is vulnerable to a brute-force attack and to other forms of cryptanalysis. However, there is a way to speed up computation using the CRT. We wish to compute the value *M* = *C*^*d* mod *n*. Let us define the following intermediate results:

Vp = *C*^*d* mod *p* *Vq* = *C*^*d* mod *q*

Following the CRT, define the quantities:

Xp = *q* x (*q*1 mod *p*) *Xq* = *p* x (*p*1 mod *q*)

The CRT then shows, that

M = (*VpXp* + *VqXq*) mod *n*

Further, we can simplify the calculation of *Vp* and *Vq* using Fermat's theorem, which states that *a*^p1=1(mod *p*) if *p* and *a* are relatively prime. Some thought should convince you that the following are valid:

$$V_p = C^d \bmod p = C^{d \bmod (p1)} \bmod p \quad V_q = C^d \bmod q = C^{d \bmod (q1)} \bmod q$$

The quantities *d* mod (*P*1) and *d* mod (*q*1) can be precalculated. The end result is that the calculation is approximately four times as fast as evaluating *M* = *Cd* mod *n* directly.

2.2.6 Key Generation

Before the application of the public-key cryptosystem, each participant must generate a pair of keys.

This involves the following tasks:

- Determining two prime numbers, *p* and *q*
- Selecting either *e* or *d* and calculating the other

First, consider the selection of *p* and *q*. Because the value of *n* = *pq* will be known to any potential adversary, to prevent the discovery of *p* and *q* by exhaustive methods, these primes must be chosen from a sufficiently large set (i.e., *p* and *q* must be large numbers). On the other hand, the method used for finding large primes must be reasonably efficient.

At present, there are no useful techniques that yield arbitrarily large primes, so some other means of tackling the problem is needed. The procedure that is generally used is to pick at random an odd number of the desired order of magnitude and test whether that number is prime. If not, pick successive random numbers until one is found that tests prime.

A variety of tests for primality have been developed for a description of a number of such tests). Almost invariably, the tests are probabilistic. That is, the test will merely determine that a given integer is *probably* prime. Despite this lack of certainty, these tests canbe run in such a way as to make the probability as close to 1.0 as desired. As an example, oneof the more efficient and popular algorithms, the Miller-Rabin algorithm. With this algorithm and most such algorithms, the procedure for testing whether a given integer *n* is prime is to perform some calculation that involves *n* and a randomly chosen integer *a*. If *n* "fails" the test, then *n* is not prime. If *n* "passes" the test, then *n* may be prime or nonprime. If *n* passes many such tests with many different randomly chosen values for *a*, then we can have high confidence that *n* is, in fact, prime.

In summary, the procedure for picking a prime number is as follows.

1. Pick an odd integer *n* at random (e.g., using a pseudorandom number generator).

2. Pick an integer *a* < *n* at random.

3. Perform the probabilistic primality test, such as Miller-Rabin, with *a* as a parameter. If *n* fails the test, reject the value *n* and go to step 1.

4. If *n* has passed a sufficient number of tests, accept *n*; otherwise, go to step 2.

This is a somewhat tedious procedure. However, remember that this process is performed relatively infrequently: only when a new pair (*PU*, *PR*) is needed.

2.2.7 The Security of RSA

Four possible approaches to attacking the RSA algorithm are as follows:

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks**: These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

The defense against the brute-force approach is the same for RSA as for other cryptosystems, namely, use a large key space. Thus, the larger the number of bits in *d*, the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

2.2.7.1 The Factoring Problem

We can identify three approaches to attacking RSA mathematically:

- Factor n into its two prime factors. This enables calculation of $\phi(n) = (p\ 1) \times (q\ 1)$, which, in turn, enables determination of $d \equiv e^1 \pmod{\phi(n)}$.
- Determine $\phi(n)$ directly, without first determining p and q. Again, this enables determination of $d \equiv e^1 \pmod{\phi(n)}$.
- Determine d directly, without first determining $\phi(n)$.

Table 2.4: Progress in Factorization

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	Quadratic sieve
110	365	April 1992	75	Quadratic sieve
120	398	June 1993	830	Quadratic sieve
129	428	April 1994	5000	Quadratic sieve
130	431	April 1996	1000	Generalized number field sieve
140	465	February 1999	2000	Generalized number field sieve
155	512	August 1999	8000	Generalized number field sieve
160	530	April 2003		Lattice sieve
174	576	December 2003		Lattice sieve
200	663	May 2005		Lattice sieve

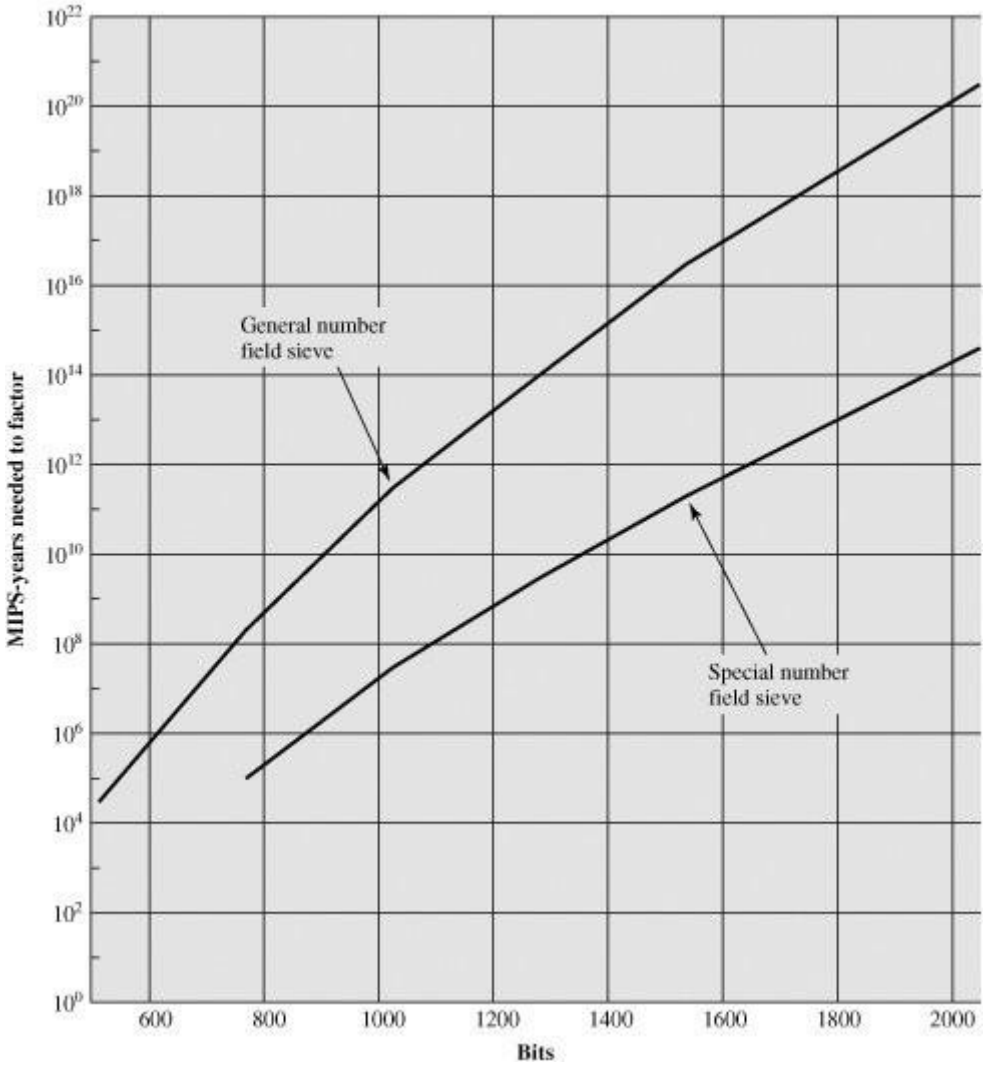


Figure 2.9: MIPS-years Needed to Factor

In addition to specifying the size of *n*, a number of other constraints have been suggested by researchers. To avoid values of *n* that may be factored more easily, the algorithm's inventors suggest the following constraints on *p* and *q*:

1. *p* and *q* should differ in length by only a few digits. Thus, for a 1024-bit key (309 decimal digits), both *p* and *q* should be on the order of magnitude of 1075 to 10100.

2. Both (*p* 1) and (*q* 1) should contain a large prime factor.

3. gcd(*p* 1, *q* 1) should be small.

In addition, it has been demonstrated that if $e < n$ and $d < n¼$, then *d* can be easily determined.

2.2.7.2 Timing Attacks

A timing attack is somewhat analogous to a burglar guessing the combination of a safe by observing how long it takes for someone to turn the dial from number to number. We can explain the attack using the modular exponentiation algorithm of Figure 2.8, but theattack can be adapted to work with any implementation that does not run in fixed time. In thisalgorithm, modular exponentiation is accomplished bit by bit, with one modular

multiplication performed at each iteration and an additional modular multiplication performed for each 1 bit.

As Kocher points out in his paper, the attack is simplest to understand in an extreme case. Suppose the target system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes much more time than an entire average modular exponentiation. The attack proceeds bit-by bit starting with the leftmost bit, *bk*. Suppose that the first *j* bits are known (to obtain the entire exponent, start with $j = 0$ and repeat the attack until the entire exponent is known). For a given ciphertext, the attacker can complete the first *j* iterations of the **for** loop. The operation of the subsequent step depends on the unknown exponent bit. If the bit is set, d<- (d x a) mod n will be executed. For a few values of a and d, the modular multiplication will be extremely slow, and the attacker knows which these are. Therefore, if the observed time to execute the decryption algorithm is always slow when this particular iteration is slow with a 1 bit, then this bit is assumed to be 1. If a number of observed execution times for the entire algorithm are fast, then this bit is assumed to be 0.

In practice, modular exponentiation implementations do not have such extreme timing variations, in which the execution time of a single iteration can exceed the mean execution time of the entire algorithm. Nevertheless, there is enough variation to make this attack practical.

Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following:

- **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
- **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher points out that if defenders don't add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.
- **Blinding:** Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

RSA Data Security incorporates a blinding feature into some of its products. The private-key operation $M = C_d \bmod n$ is implemented as follows:

1. Generate a secret random number r between 0 and n-1.

2.Compute $C' = C(r^e) \bmod n$, where e is the public exponent.

3. Compute $M' = (C')d \bmod n$ with the ordinary RSA implementation.

4. Compute $M = M'r$^-1 mod n. In this equation, $r1$ is the multiplicative inverse of r mod n;

It can be demonstrated that this is the correct result by observing that $red \bmod n = r \bmod n$.

RSA Data Security reports a 2 to 10% performance penalty for blinding.

2.2.7.3 Chosen Ciphertext Attack and Optimal Asymmetric Encryption Padding

The basic RSA algorithm is vulnerable to a chosen ciphertext attack (CCA). CCA is defined as an attack in which adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key. Thus, the adversary could

select a plaintext, encrypt it with the target's public key and then be able to get the plaintext back by having it decrypted with the private key. Clearly, this provides the adversary with no new information. Instead, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed forcryptanalysis. A simple example of a CCA against RSA takes advantage of the following property of RSA:

$$\mathrm{E}(PU, M_1) \times \mathrm{E}(PU, M_2) = \mathrm{E}(PU, [M_1 \times M_2])$$

We can decrypt $C = M^e$ using a CCA as follows.

1.

Compute $X = (C \times 2^e) \bmod n$.

2.

Submit X as a chosen ciphertext and receive back $Y = X^d \bmod n$.

But now note the following:

$$\begin{aligned} X &= (C \bmod n) \times (2^e \bmod n) \\ &= (M^e \bmod n) \times (2^e \bmod n) \\ &= (2M)^e \bmod n \end{aligned}$$

Therefore, $Y = (2M) \bmod n$ From this, we can deduce M. To overcome this simple attack, practical RSA based cryptosystems randomly pad the plaintext prior to encryption. This randomizes the ciphertext so that Equation no longer holds. However, more sophisticated CCAs are possible and a simple padding with a random value has been shown to be insufficient to provide the desired security. To counter such attacks RSA Security Inc., a leading RSA vendor and former holder of the RSA patent, recommends modifying theplaintext using a procedure known as optimal asymmetric encryption padding (OAEP). A fulldiscussion of the threats and OAEP are beyond our scope;

Here, we simply summarize the OAEP procedure. Figure 2.10 depicts OAEP encryption. As a first step the message M to be encrypted is padded. A set of optional parameters P is passed through a hash function H. The output is then padded with zeros to get the desired length in the overall data block (DB). Next, a random seed is generated and passed through another hash function, called the mask generating function (MGF). The resulting hash value is bit-by-bit XORed with DB to produce a maskedDB. The maskedDB is in turn passed through the MGF to form a hash that is XORed with the seed to produce the masked seed. The concatenation of the maskedseed and the maskedDB forms the encoded message EM. Note that the EM includes the padded message, masked by the seed, and the seed, masked by the maskedDB. The EM is then encrypted using RSA.

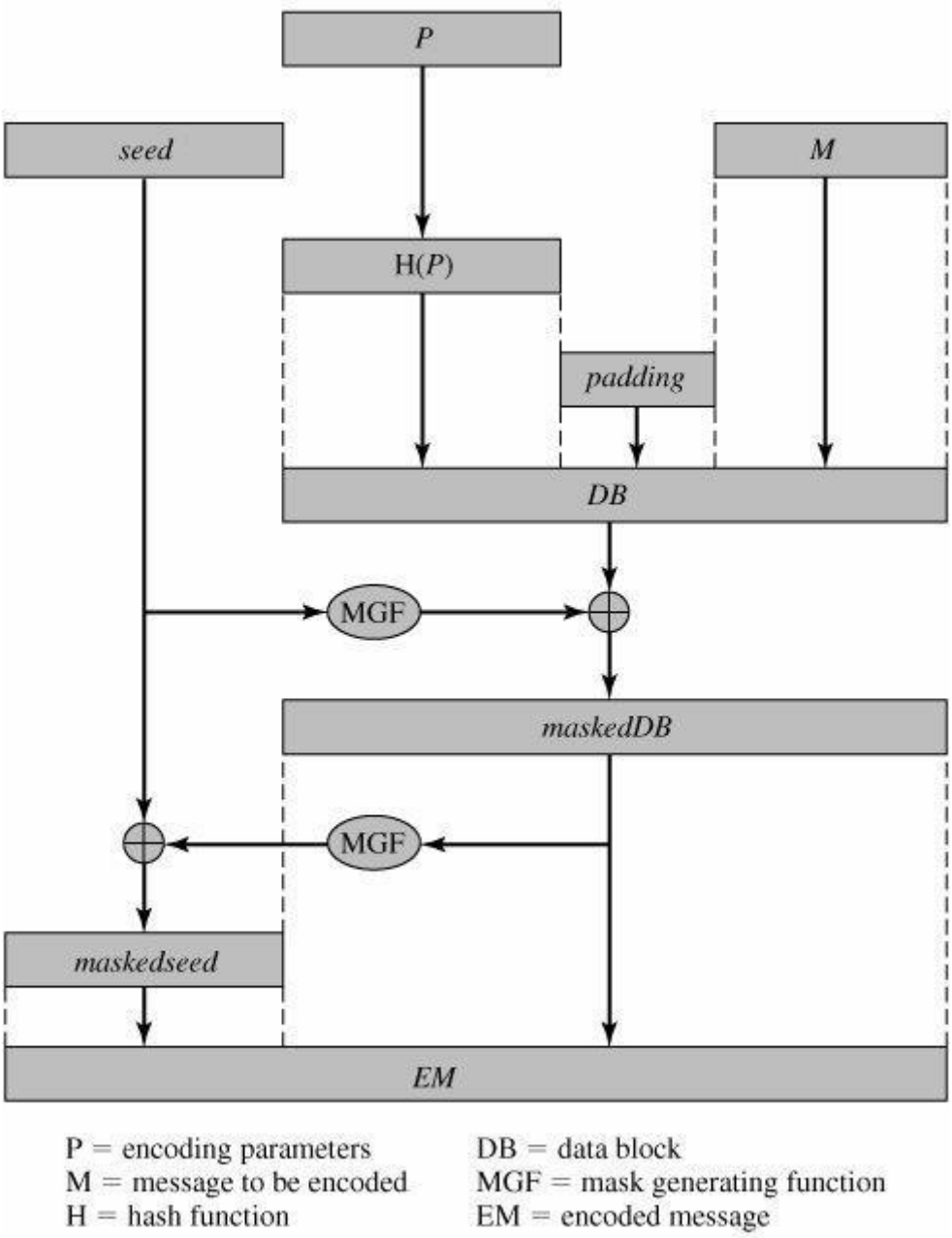


Figure 2.10: Encryption Using Optimal Assymetric Encryption Padding (OAEP)

2.3 Diffie-Hellman Key Exchange

The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography and is generally referred to as Diffie-Hellman key exchange. A number of commercial products employ this key exchange technique.

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in the following way. First, we define a primitive root of a prime number p as one whose powers modulo p generate all

the integers from 1 to *p*-1. That is, if *a* is a primitive root of the prime number *p*, then the numbers

$a \bmod p, a^2 \bmod p, \ldots, a^{p1} \bmod p$

are distinct and consist of the integers from 1 through p 1 in some permutation.

For any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

$b \equiv a^i \pmod p$ where $0 \leq i \leq (p\ 1)$

The exponent i is referred to as the discrete logarithm of b for the base a, mod p. We express this value as $\text{dlog}_{a,p}(b)$.

2.3.1 The Algorithm

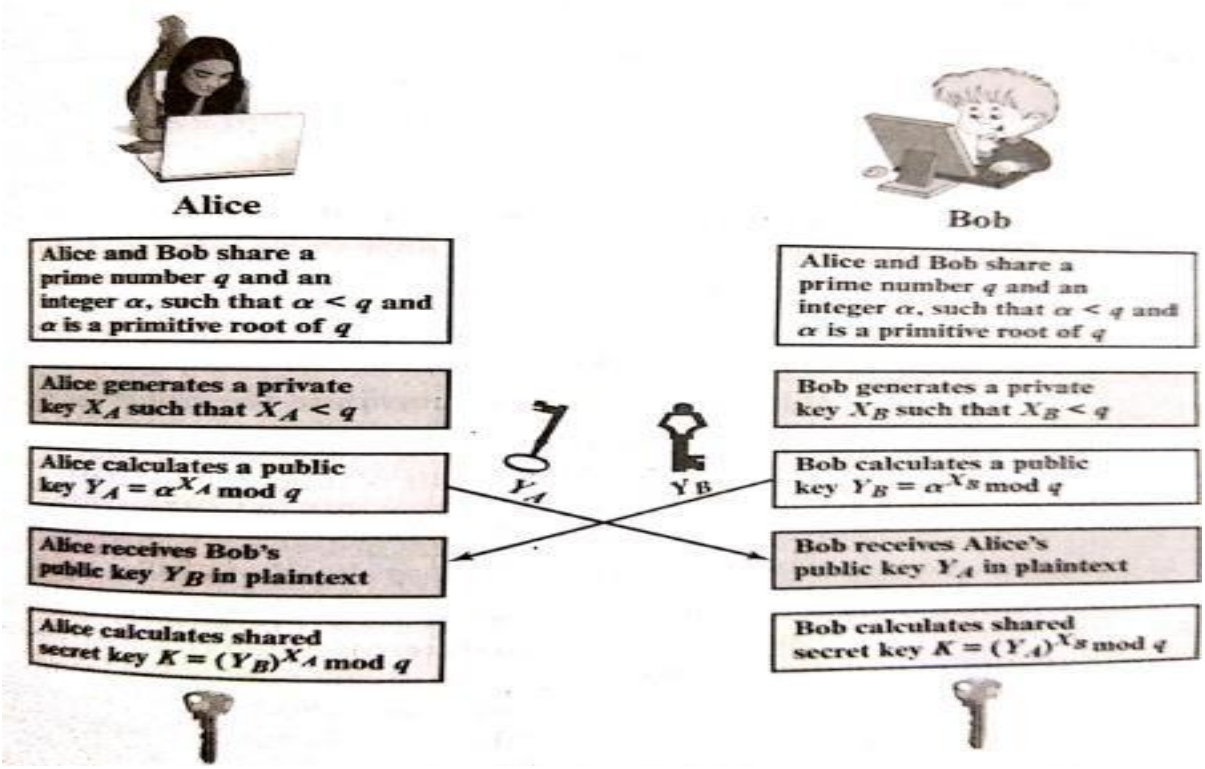


Figure 2.11: The Diffie-Hellman Key Exchange

Figure 2.11 summarizes the Diffie-Hellman key exchange algorithm. For this scheme, there are two publicly known numbers: a prime number q and an integer that is a primitive root of q. Suppose the

users A and B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{XA} \bmod q$. Similarly, user B independently selects a random integer $X_A < q$ and computes $Y_B = \alpha^{XB} \bmod q$. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{XB} \bmod q$. These two calculations produce identical results:

$K = (Y_B)^{XA} \bmod q$

$= (\alpha^{X_B} \bmod q)^{X_A} \bmod q$

$= (\alpha^{X_B})^{X_A} \bmod q$ by the rules of modular arithmetic

$= (\alpha^{X_B X_A} \bmod q$

$= (\alpha^{X_A})^{X_B} \bmod q$

$= (\alpha^{X_A} \bmod q)$

$= (\alpha^{X_A} \bmod q)^{X_B} \bmod q$

$= (Y_A)^{X_B} \bmod q$

Global Public Elements

q	prime number
α	$\alpha < q$ *and* α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Calculation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Calculation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Figure 2.12: The Diffie-Hellman Key Exchange Algorithm

The result is that the two sides have exchanged a secret value. Furthermore, because X_A and X_B are private, an adversary only has the following ingredients to work with: q, α, Y_A, and Y_B. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

$$X_B = \text{dlog}_{\alpha,q}(Y_B)$$

The adversary can then calculate the key K in the same manner as user B calculates it.

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

Here is an example. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $\alpha = 3$. A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

A computes $Y_A = 3^{97} \bmod 353 = 40$.

B computes $Y_B = 3^{233} \bmod 353 = 248$.

After they exchange public keys, each can compute the common secret key:

A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.

B computes $K = (Y_A)^{X_E} \bmod 353 = 40^{233} \bmod 353 = 160$.

We assume an attacker would have available the following information:

$q = 353$; $\alpha = 3$; $Y_A = 40$; $Y_B = 248$

In this simple example, it would be possible by brute force to determine the secret key 160. In particular, an attacker E can determine the common key by discovering a solution to the equation $3^a \bmod 353 = 40$ or the equation $3^b \bmod 353 = 248$. The brute-force approach is to calculate powers of 3 modulo 353, stopping when the result equals either 40 or 248. The desired answer is reached with the exponent value of 97, which provides $3^{97} \bmod 353 = 40$.

With larger numbers, the problem becomes impractical.

2.3.2 Key Exchange Protocols

Figure 2.13 shows a simple protocol that makes use of the Diffie-Hellman calculation.Suppose that user A wishes to set up a connection with user B and use a secret key to encrypt messages on that connection. User A can generate a one-time private key *XA*, calculate *YA*, and send that to user B. User B responds by generating a private value *XB* calculating *YB*, andsending *YB* to user A. Both users can now calculate the key. The necessary public values q and α would need to be known ahead of time. Alternatively, user A could pick values for q and α and include those in the first message.

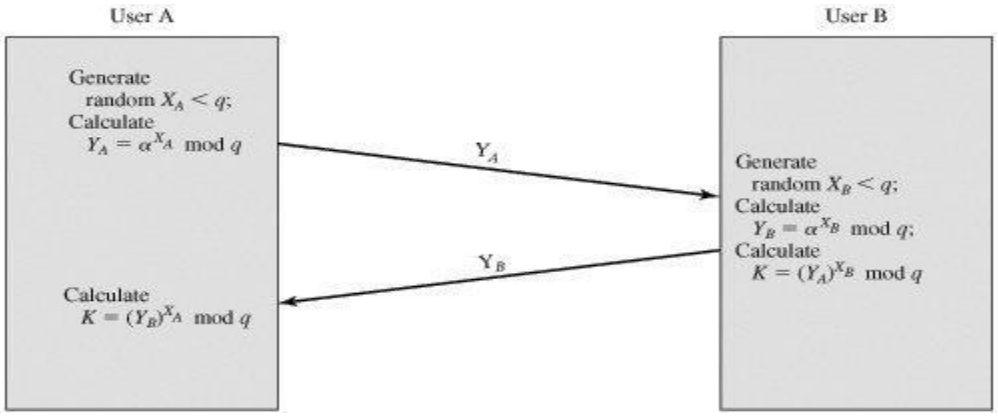


Figure 2.13: Diffie-Hellman Key Exchange

As an example of another use of the Diffie-Hellman algorithm, suppose that a group of users (e.g., all users on a LAN) each generate a long-lasting private value *Xi* (for user *i*) and calculate a public value *Yi*. These public values, together with global public values for *q* anda, are stored in some central directory. At any time, user *j* can access user *i*'s public value, calculate a secret key, and use that to send an encrypted message to user A. If the central directory is trusted, then this form of communication provides both confidentiality and a degree of authentication. Because only *i* and *j* can determine the key, no other user can read the message (confidentiality). Recipient *i* knows that only user *j* could have created a message using this key (authentication). However, the technique does not protect against replay attacks.

2.3.3 Man-in-the-Middle Attack

The protocol depicted in Figure 2.11 is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. Alice transmits Y_A to Bob.
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_E} \bmod q$.
5. Bob transmits X_A to Alice.
6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{X_{D1}} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$.

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret
key $K1$ and Alice and Darth share secret key $K2$. All future communication between Bob and Alice is
compromised in the following way:

1. Alice sends an encrypted message M: E($K2$, M).
2. Darth intercepts the encrypted message and decrypts it, to recover M.
3. Darth sends Bob E($K1$, M) or E($K1$, M'), where M' is any message. In the first case, Darth simply wants to eavesdrop on the communication without altering it. In the second case, Darth wants to modify the message going to Bob.

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

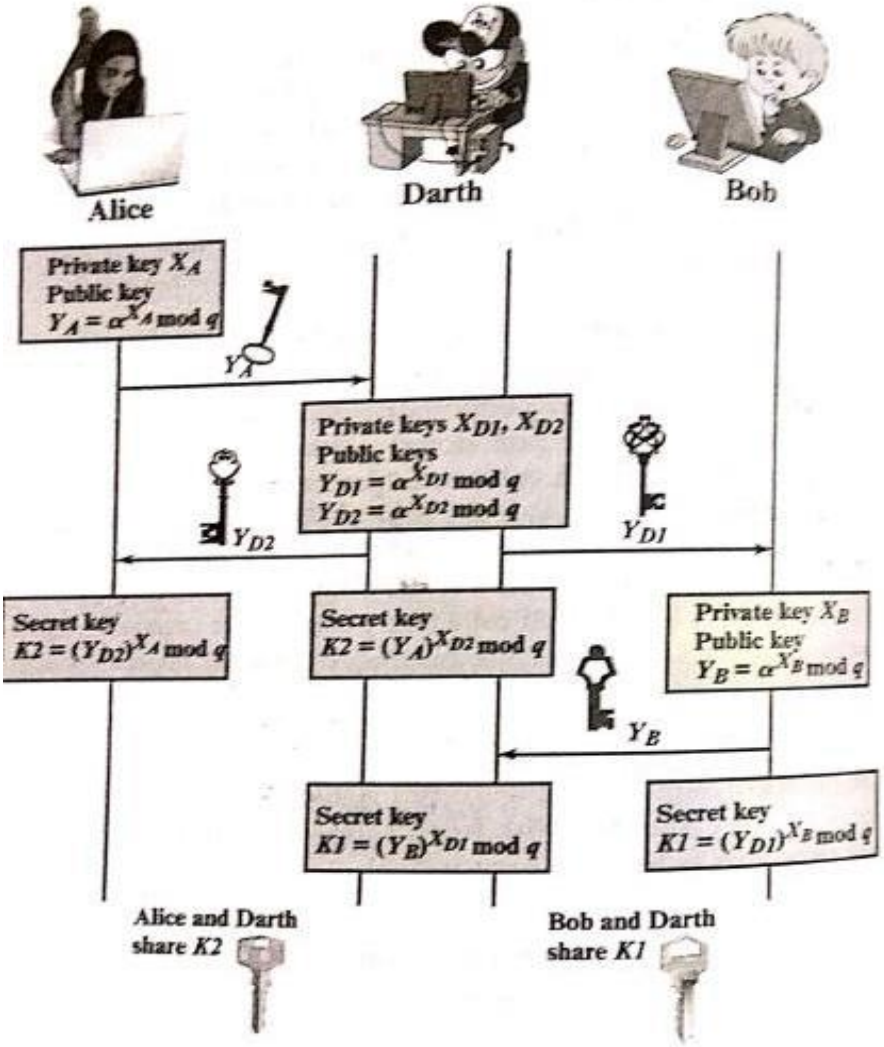


Figure 2.14: Man-in-the-Middle Attack

2.4 ElGamal Cryptographic System

In 1984, T Elgamal announced a public-key scheme based on discrete logarithms, closely related to the Diffie-Hellman technique. The Elgamal cryptosystem is used in some form in a number of standards including the digital signature standard(DSS) and the S/MIME email standard.

As with Diffie–Hellman, the global elements of Elgamal are a prime number q and α, which is a primitive root of q. User A generates a private/public key pair as follows:

1. Generate a random integer X_A, such that $1 < X_A < q - 1$.
2. Compute $Y_A = \alpha^{X_A} \bmod q$.
3. A's private key is X_A and A's public key is $\{q, \alpha, Y_A\}$.

Any user B that has access to A's public key can encrypt a message as follows:

1. Represent the message as an integer M in the range $0 \le M \le q - 1$. Longer messages are sent as a sequence of blocks, with each block being an integer less than q.
2. Choose a random integer k such that $1 \le k \le q - 1$.
3. Compute a one-time key $K = (Y_A)^k \bmod q$.
4. Encrypt M as the pair of integers (C_1, C_2) where

$$C_1 = \alpha^k \bmod q; \quad C_2 = KM \bmod q$$

User A recovers the plaintext as follows:

1. Recover the key by computing $K = (C_1)^{X_A} \bmod q$.
2. Compute $M = (C_2 K^{-1}) \bmod q$.

This steps are summarized in figure 2.15. It corresponds to figure 2.1a. Alice generates a public/private keys pair; Bob encrypts using Alice's public key; and Alice decrypts using her private key.

Let us demonstrate why the Elgamal scheme works. First, we show how K is recovered by the decryption process:

$K = (Y_A)^k \bmod q$	K is defined during the encryption process
$K = (\alpha^{X_A} \bmod q)^k \bmod q$	substitute using $Y_A = \alpha^{X_A} \bmod q$
$K = \alpha^{kX_A} \bmod q$	by the rules of modular arithmetic
$K = (C_1)^{X_A} \bmod q$	substitute using $C_1 = \alpha^k \bmod q$

Next, using K, we recover the plaintext as

$$C_2 = KM \bmod q$$
$$(C_2 K^{-1}) \bmod q = KMK^{-1} \bmod q = M \bmod q = M$$

We can restate the Elgamal process as follows, using Figure 10.3.

1. Bob generates a random integer k.
2. Bob generates a one-time key K using Alice's public-key components Y_A, q, and k.
3. Bob encrypts k using the public-key component α, yielding C_1. C_1 provides sufficient information for Alice to recover K.
4. Bob encrypts the plaintext message M using K.
5. Alice recovers K from C_1 using her private key.
6. Alice uses K^{-1} to recover the plaintext message from C_2.

Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q

Key Generation by Alice	
Select private X_A	$X_A < q - 1$
Calculate Y_A	$Y_A = \alpha^{X_A} \bmod q$
Public key	$\{q, \alpha, Y_A\}$
Private key	X_A

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < q$
Select random integer k	$k < q$
Calculate K	$K = (Y_A)^k \bmod q$
Calculate C_1	$C_1 = \alpha^k \bmod q$
Calculate C_2	$C_2 = KM \bmod q$
Ciphertext:	(C_1, C_2)

Decryption by Alice with Alice's Private Key	
Ciphertext:	(C_1, C_2)
Calculate K	$K = (C_1)^{X_A} \bmod q$
Plaintext:	$M = (C_2 K^{-1}) \bmod q$

Figure 2.15: The Elgamal Cryptosystem

Thus, K functions as a one-time key, used to encrypt and decrypt the message.

For example, let us start with the prime field GF(19); that is, $q = 19$. It has primitive roots {2, 3, 10, 13, 14, 15}, [redacted]. We choose $\alpha = 10$.

Alice generates a key pair as follows:

1. Alice chooses $X_A = 5$.
2. Then $Y_A = \alpha^{X_A} \bmod q = \alpha^5 \bmod 19 = 3$ [redacted].
3. Alice's private key is 5 and Alice's public key is $\{q, \alpha, Y_A\} = \{19, 10, 3\}$.

Suppose Bob wants to send the message with the value $M = 17$. Then:

1. Bob chooses $k = 6$
2. Then $K = (Y_A)^k \bmod q = 3^6 \bmod 19 = 729 \bmod 19 = 7$
3. So
   $C_1 = \alpha^k \bmod q = \alpha^k \bmod 19 = 11$
   $C_2 = KM \bmod q = 7 \times 17 \bmod 19 = 119 \bmod 19 = 5$
4. Bob sends the ciphertext (11, 5).

For decryption:

1. Alice calculates $K = (C_1)^{X_A} \bmod q = 11^5 \bmod 19 = 161051 \bmod 19 = 7$.
2. Then K^{-1} in GF(19) is $7^{-1} \bmod 19 = 11$.
3. Finally, $M = (C_2 K^{-1}) \bmod q = 5 \times 11 \bmod 19 = 55 \bmod 19 = 17$.

If a message must be broken up into blocks and sent as a sequence of encrypted blocks, a unique value of k should be used for each block. If k is used for more than one block, knowledge of one block M_1 of the message enables the user to compute other blocks as follows. Let

$$C_{1,1} = \alpha^k \bmod q; C_{2,1} = KM_1 \bmod q$$
$$C_{1,2} = \alpha^k \bmod q; C_{2,2} = KM_2 \bmod q$$

Then,

$$\frac{C_{2,1}}{C_{2,2}} = \frac{KM_1 \bmod q}{KM_2 \bmod q} = \frac{M_1 \bmod q}{M_2 \bmod q}$$

If M_1 is known, then M_2 is easily computed as

$$M_2 = (C_{2,1})^{-1} C_{2,2} M_1 \bmod q$$

The security of Elgamal is based on the difficulty of computing discrete logarithms. To recover A's private key, an adversary would have to compute $X_A = \text{dlog}_{\alpha,q}(Y_A)$. Alternatively, to recover the one-time key K, an adversary would have to determine the random number k, and this would require computing the discrete logarithm $k = \text{dlog}_{\alpha,q}(C_1)$. [STIN06] points out that these calculations are regarded as infeasible if p is at least 300 decimal digits and $q - 1$ has at least one "large" prime factor.

Module-3:

Elliptic Curve Cryptography, Key Management and Distribution

3.1 Elliptic Curve Arithmetic

Most of the products and standards that use public-key cryptography for encryption and digital signatures use RSA. As we have seen, the key length for secure RSA use has increasedover recent years, and this has put a heavier processing load on applications using RSA. This burden has ramifications, especially for electronic commerce sites that conduct large numbersof secure transactions. Recently, a competing system has begun to challenge RSA: elliptic curve cryptography (ECC). Already, ECC is showing up in standardization efforts, including the IEEE P1363 Standard for Public-Key Cryptography.

The principal attraction of ECC, compared to RSA, is that it appears to offer equal security for a far smaller key size, thereby reducing processing overhead. On the other hand, although the theory of ECC has been around for some time, it is only recently that products have begun to appear and that there has been sustained cryptanalytic interest in probing for weaknesses. Accordingly, the confidence level in ECC is not yet as high as that in RSA. ECC is fundamentally more difficult to explain than either RSA or Diffie-Hellman, and a full mathematical description.

3.1.1 Abelian Groups

An abelian **group** G, sometimes denoted by $\{G, \bullet\}$, is a set of elements with a binary operation, denoted by $\bullet$, that associates to each ordered pair (a, b) of elements in G anelement $(a \bullet b)$ in G, such that the following axioms are obeyed.

(A1) Closure:	If a and b belong to G, then $a \bullet b$ is also in G.
(A2) Associative:	$a \bullet (b \bullet c) = (a \bullet b) \bullet c$ for all a, b, c in G.
(A3) Identity element:	There is an element e in G such that $a \bullet e = e \bullet a = a$ for all a in G.
(A4) Inverse element:	For each a in G there is an element a' in G such that $a \bullet a' = a' \bullet a = e$.
(A5) Commutative:	$a \bullet b = b \bullet a$ for all a, b in G.

A number of public-key ciphers are based on the use of an abelian group. Forexample, Diffie-Hellman key exchange involves multiplying pairs of nonzero integers modulo a prime number q. Keys are generated by exponentiation over the group, withexponentiation defined as repeated multiplication:

$$q = \underbrace{(a \times a \times \ldots \times a)}_{k \text{ times}}$$

For example, $a^k \bmod q = \bmod q$. To attack Diffie-Hellman, the attacker must determine k given a and a^k; this is the discrete log problem.

For elliptic curve cryptography, an operation over elliptic curves, called addition, is used Multiplication is defined by repeated addition. For example,

$$a \times k = \underbrace{(a + a + \ldots + a)}_{k \text{ times}}$$

where the addition is performed over an elliptic curve. Cryptanalysis involves determining k given a and $(a \times k)$.

An elliptic curve is defined by an equation in two variables, with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field, which results in the definition of a finite abelian group. Before looking at this, we first look at elliptic curves in which the variables and coefficients are real numbers.

3.1.2 Elliptic Curves over Real Numbers

Elliptic curves are not ellipses. They are so named because they are described by cubic equations, similar to those used for calculating the circumference of an ellipse. In general, cubic equations for elliptic curves take the form

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

where *a, b, c, d,* and *e* are real numbers and *x* and *y* take on values in the real numbers. For our purpose, it is sufficient to limit ourselves to equations of the form.

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

Such equations are said to be cubic, or of degree 3, because the highest exponent they contain is a 3. Also included in the definition of an elliptic curve is a single element denoted *O* and called the *point at infinity* or the *zero point.* To plot such a curve, we need to compute

$$y = \sqrt{x^3 + ax + b}$$

For given values of *a* and *b*, the plot consists of positive and negative values of *y* for each value of *x*. Thus each curve is symmetric about $y = 0$.

Figure 3.1 shows two examples of elliptic curves. As you can see, the formula sometimes produces weird-looking curves.

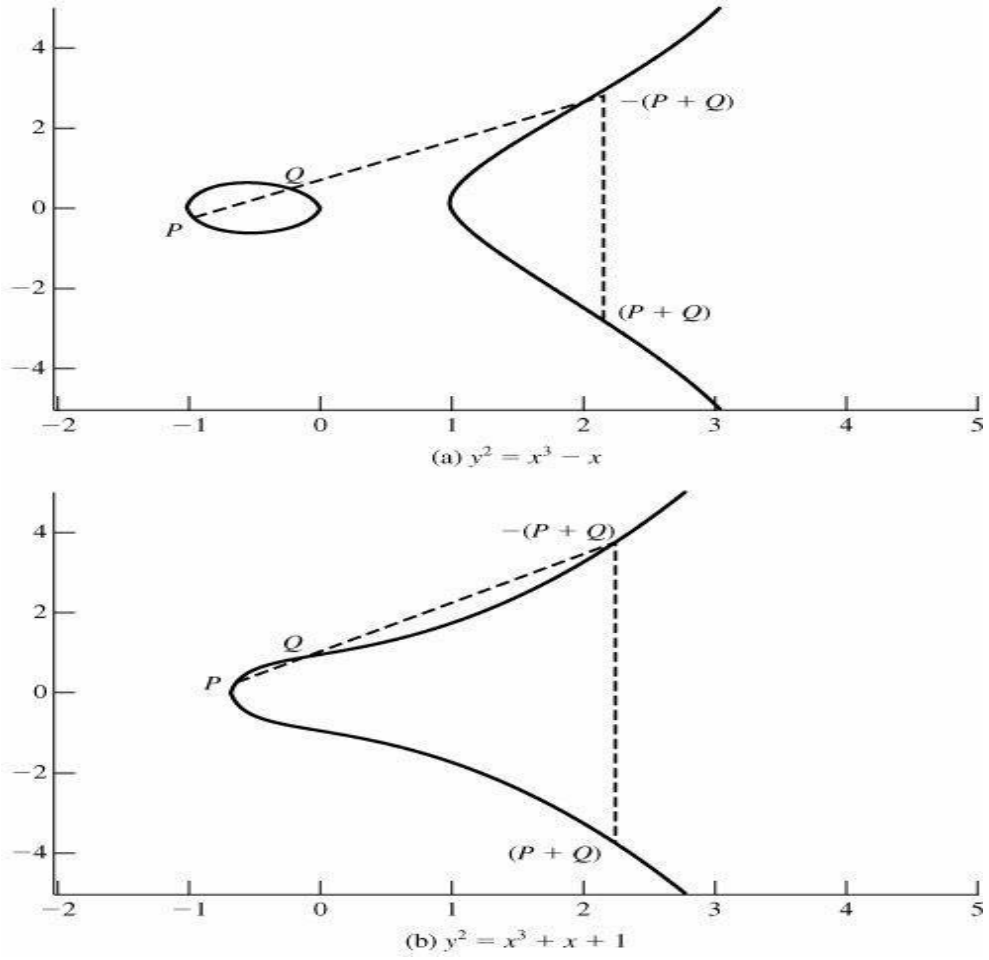


Figure 3.1: Example of Elliptic Curve

Now, consider the set of points E(a, b) consisting of all of the points (x, y) that satisfy together with the element O. Using a different value of the pair (a, b) results in a different set E(a, b). Using this terminology, the two curves in Figure 3.1 depict the sets E(1,0) and E(1, 1), respectively.

3.1.3 Geometric Description of Addition

It can be shown that a group can be defined based on the set E(a, b) for specific values of a and b in $y^2 = x^3 + ax + b$, provided the following condition is met:

$$4a^3 + 27b^2 \neq 0$$

To define the group, we must define an operation, called addition and denoted by +, for the set E(a, b), where a and b satisfy above equation. In geometric terms, the rules for addition can be stated as follows: If three points on an elliptic curve lie on a straight line,their sum is O. From this definition, we can define the rules of addition over an elliptic curve:

1. O serves as the additive identity. Thus $O + O = O$; for any point P on the elliptic curve, $P + O = P$. In what follows, we assume $P \neq O$ and $Q \neq O$.
2. The negative of a point P is the point with the same x coordinate but the negative of the y coordinate; that is, if $P = (x, y)$, then $-P = (x, -y)$. Note that these two points can be joined by a vertical line. Note that $P + (-P) = P - P = O$.
3. To add two points P and Q with different x coordinates, draw a straight line between them and find the third point of intersection R. It is easily seen that there is a unique point R that is the point of intersection (unless the line is tangent to the curve at either P or Q, in which case we take $R = P$ or $R = Q$, respectively). To form a group structure, we need to define addition on these three points as follows: $P + Q = R$. That is, we define $P + Q$ to be the mirror image (with respect to the x axis) of the third point of intersection. Figure 3.1 illustrates this construction.
4. The geometric interpretation of the preceding item also applies to two points, P and $-P$, with the same x coordinate. The points are joined by a vertical line, which can be viewed as also intersecting the curve at the infinity point. We therefore have $P + (-P) = O$, consistent with item (2).
5. To double a point Q, draw the tangent line and find the other point of intersection S. Then $Q + Q = 2Q = -S$.

With the preceding list of rules, it can be shown that the set E(a, b) is an abelian group.

Algebraic Description of Addition

We present some results that enable calculation of additions over elliptic curves. For two distinct points $P = (xP, yP)$ and $Q = (xQ, yQ)$ that are not negatives of each other, the slope of the line l that joins them is $D = (yQ - yP)$. There is exactly one other point where l intersects the elliptic curve, and that is the negative of the sum of P and Q. After some algebraic manipulation, we can express the sum $R = P + Q$ as follows:

$$x_R = \Delta^2 - x_P - x_Q$$
$$y_R = -y_P + \Delta(x_P - x_R)$$

We also need to be able to add a point to itself: $P + P = 2P = R$. When $yP \neq 0$, the expressions are

$$x_R = \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P$$
$$y_R = \left(\frac{3x_P^2 + a}{2y_P}\right)(x_P - x_R) - y_P$$

3.1.4 Elliptic Curves over Z*p*

Elliptic curve cryptography makes use of elliptic curves in which the variables and coefficients are all restricted to elements of a finite field. Two families of elliptic curves are used in cryptographic applications: prime curves over Z*p* and binary curves over GF(2*m*). Fora **prime curve** over Z*p*,

we use a cubic equation in which the variables and coefficients all take on values in the set of integers from 0 through *p* 1 and in which calculations are performed modulo *p*. Fora **binary curve** defined over GF (2*m*), the variables and coefficients all take on values in GF(2*n*) and in calculations are performed over GF(2*n*) points out that prime curves are best for software applications, because the extended bit-fiddling operations needed by binary curves are not required; and that binary curves are best for hardware applications, where it takes remarkably few logic gates to create a powerful, fast cryptosystem.

There is no obvious geometric interpretation of elliptic curve arithmetic over finite fields. The algebraic interpretation used for elliptic curve arithmetic over real numbers does readily carry over, and this is the approach we take.

For elliptic curves over Z*p*, as with real numbers

$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$

The above equation is satisfied for $a = 1$, b = 1, $x = 9$, $y = 9$, $y = 7$, $p = 23$:

72 mod 23 = (93 + 9 + 1) mod 23

49 mod 23 = 739 mod 23

3 = 3

Now consider the set E*p* (*a*, *b*) consisting of all pairs of integers (*x*, *y*) that satisfy above equation, together with a point at infinity *O*. The coefficients *a* and *b* and the variables *x* and *y* are all elements of Z*p*.

Table 3.1: Points on the Elliptic Curve E23 (1,1)

(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

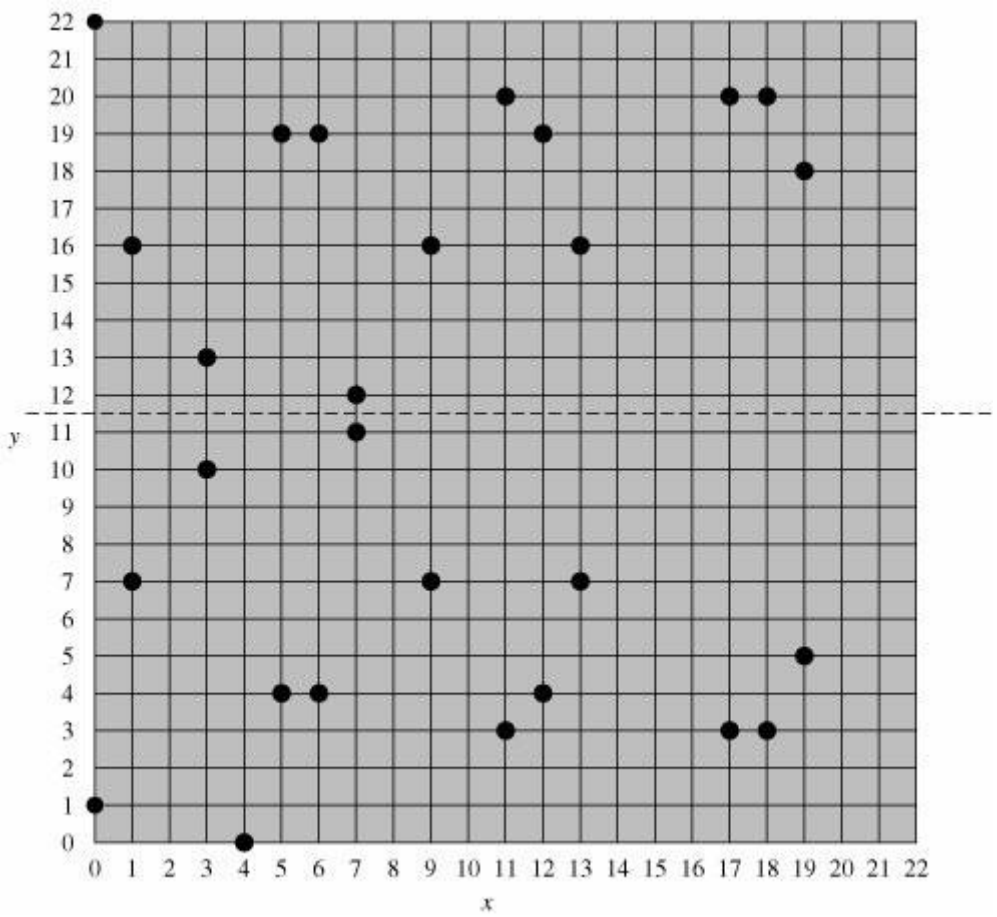


Figure 3.2: The Elliptic Curve E23(1,1)

It can be shown that a finite abelian group can be defined based on the set E*p*(*a*, *b*) provided that (*x*3 +*ax* + *b*) mod *p* has no repeated factors. This is equivalent to the condition

$$(4a^3 + 27b^2) \bmod p \neq 0 \bmod p$$

The rules for addition over E*p*(*a*, *b*) correspond to the algebraic technique described for elliptic curves
defined over real number. For all points *P*, *Q belongs* E*p*(*a*, *b*);

1. $P + O = P$.
2. If $P = (x\mathrm{P}, yP)$ then $P + (xP, yP) = O$. The point (xP, yP) is the negative of *P*, denoted as *P*. For example, in E23(1,1), for $P = (13,7)$, we have $P = (13, 7)$. But 7 mod 23 = 16. Therefore, $P = (13, 16)$, which is also in E23(1,1).
3. If $P = (xP, yQ)$ and $Q = (xQ, yQ)$ with $P \neq Q$, then $R = P + Q = (xR, yR)$ is determined by the following rules:
   $xR = (\mathrm{l}2\ xP\ xQ) \bmod p$
   $yR = (\mathrm{l}(xP\ xR)\ yP) \bmod p$
   where

$$\lambda = \begin{cases} \left(\dfrac{y_Q - y_P}{x_Q - x_P}\right) \bmod p & \text{if } P \neq Q \\ \left(\dfrac{3x_P^2 + a}{2y_P}\right) \bmod p & \text{if } P = Q \end{cases}$$

4. Multiplication is defined as repeated addition; for example, $4P = P + P + P + P$.

For example, let $P = (3,10)$ and $Q = (9,7)$ in *E*23(1,1). Then

$$\lambda = \left(\frac{7-10}{9-3}\right) \bmod 23 = \left(\frac{-3}{6}\right) \bmod 23 = \left(\frac{-1}{2}\right) \bmod 23 = 11$$

xR = (112 3 9) mod 23 = 17
yR = (11(3 17) 10) mod 23 = 164 mod 23 = 20
So $P + Q$ = (17, 20). To find 2P,

$$\lambda = \left(\frac{3(3^2)+1}{2 \times 10}\right) \bmod 23 = \left(\frac{5}{20}\right) \bmod 23 = \left(\frac{1}{4}\right) \bmod 23 = 6$$

The last step in the preceding equation involves taking the multiplicative inverse of 4 in $Z23$. This can be done using the extended Euclidean algorithm.
For determining the security of various elliptic curve ciphers, it is of some interest to know the number the number of points in a finite abelian group defined over an elliptic curve. In the case of the finite group Ep(a,b), the number of points N is bounded by

$$p + 1 - 2\sqrt{p} \leq N \leq p + 1 + 2\sqrt{p}$$

Note that the number of points in Ep(a, b) is approximately equal to the number of elements in Zp, namely p elements.

3.1.5 Elliptic Curves over GF(2^m)

A finite field GF(2m) consists of 2m elements, together with addition and multiplication operations that can be defined over polynomials. For elliptic curves over GF(2m), we use a cubic equation in which the variables and coefficients all take on values in GF(2m), for some number m, and in which calculations are performed using the rules of arithmetic in GF(2m).
It turns out that the form of cubic equation appropriate for cryptographic applications for elliptic curves is somewhat different for GF(2m) than for Zp. The form is

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

where it is understood that the variables x and y and the coefficients a and b are elements of GF(2m) of and that calculations are performed in GF(2m).
Now consider the set E_2 m(a, b) consisting of all pairs of integers (x, y) that satisfy the above equation together with a point at infinity O.
For example, let us use the finite field GF(24) with the irreducible polynomial f(x) = x4 + x + 1. This yields a generator that satisfies f(g) = 0, with a value of g4 = g + 1, or in binary 0010. We can develop the powers of g as follows:

g^0 = 0001	g^4 = 0011	g^8 = 0101	g^{12} = 1111
g^1 = 0010	g^5 = 0110	g^9 = 1010	g^{13} = 1101
g^2 = 0100	g^6 = 1100	g^{10} = 0111	g^{14} = 1001
g^3 = 1000	g^7 = 1011	g^{11} = 1110	g^{15} = 0001

For example, $g_5 = (g_4)(g) = g_2 + g$ = 0110.

Now consider the elliptic curve $y^2 + xy = x^3 + g^4x^2 + 1$. In this case $a = g^4$ and $b = g^0 = 1$. One point that satisfies this equation is (g^5, g^3):

$(g^3)^2 + (g^5)(g^3) = (g^5)^3 + (g^4)(g^5)^2 + 1$

$g^6 + g^8 = g^{15} + g^{14} + 1$

1100 + 0101 = 0001 + 1001 + 0001

1001 = 1001

Table 3.2 lists the points (other than *O*) that are part of E24(*g*4, 1). Figure 3.3 plots the points of E24(*g*4, 1).

Table 3.2: Points (other than O) on the Elliptic curve $E_{24}(g^4, 1)$

$(0, 1)$	(g^5, g^3)	(g^9, g^{13})
$(1, g^6)$	(g^5, g^{11})	(g^{10}, g)
$(1, g^{13})$	$g^6, g^8)$	(g^{10}, g^8)
(g^3, g^8)	(g^6, g^{14})	$(g^{12},0)$
(g^3, g^{13})	(g^9, g^{10})	(g^{12}, g^{12})

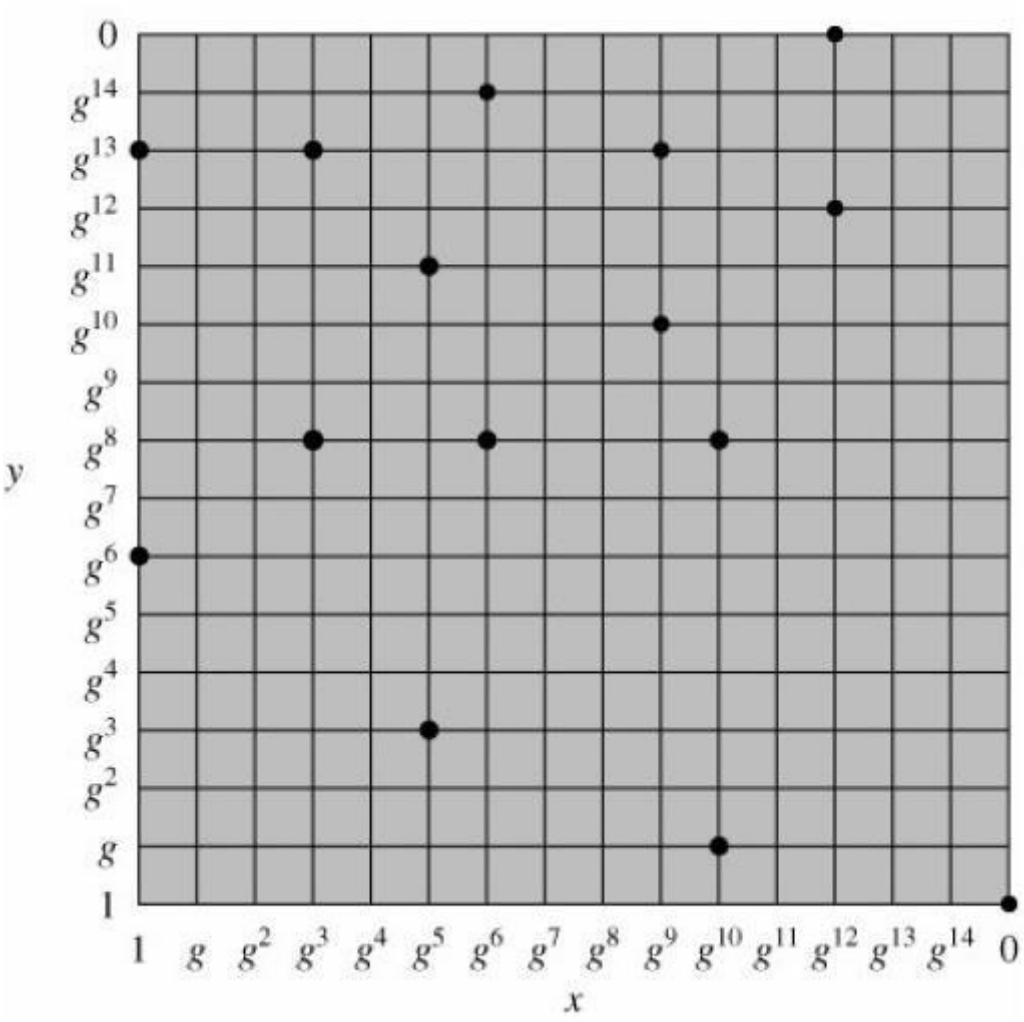


Figure 3.3: the Elliptic curve $E_{24}(g^4, 1)$

It can be shown that a finite abelian group can be defined based on the set $E_{2^m}(a, b)$, provided that $b \neq 0$. The rules for addition can be stated as follows. For all points $P, Q \in E_{2^m}(a, b)$:

1.

$P + O = P$.

2.

If $P = (x_P, y_P)$, then $P + (x_P, x_P + y_P) = O$. The point $(x_P, x_P + y_P)$ is the negative of P, denoted as P.

3.

If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ with $P \neq Q$ and $P \neq Q$, then $R = P + Q = (x_R, y_R)$ is determined by the following rules:

$x_R = \lambda^2 + \lambda + x_P + x_Q + a$

$y_R = \lambda(x_P + x_R) + x_R + y_P$

where

$$\lambda = \frac{y_Q + y_P}{x_Q + x_P}$$

4.

If $= (x_P, y_P)$ then $R = 2P = (x_R, y_R)$ is determined by the following rules:

$$x_R = \lambda^2 + \lambda + a$$
$$y_R = x_P^2 + (\lambda + 1)x_R$$

where

$$\lambda = x_P + \frac{y_P}{x_P}$$

3.2 Elliptic Curve Cryptography

The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple addition is the counterpart of modular exponentiation. To form a cryptographic system using elliptic curves, we need to find a "hard problem" corresponding to factoring the product of two primes or taking the discrete logarithm.

Consider the equation $Q = kP$ where Q, P *is belongs to* $E_p(a, b)$ and $k < p$. It is relatively easy to calculate Q given k and P, but it is relatively hard to determine k given Q and P. This is called the discrete logarithm problem for elliptic curves.
Consider the group E23 (9, 17). This is the group defined by the equation $y2 \bmod 23 = (x3 + 9x + 17) \bmod 23$. What is the discrete logarithm k of $Q = (4, 5)$ to the base $P = (16.5)$? The brute-force method is to compute multiples of P until Q is found.
Thus $P = (16, 5)$; $2P = (20, 20)$; $3P = (14, 14)$; $4P = (19, 20)$; $5P = (13, 10)$; $6P = (7, 3)$; $7P = (8, 7)$; $8P = (12, 17)$; $9P = (4, 5)$.
Because $9P = (4, 5) = Q$, the discrete logarithm $Q = (4, 5)$ to the base $P = (16, 5)$ is $k = 9$. In a real application, k would be so large as to make the brute-force approach infeasible.

3.2.1 Analog of Diffie-Hellman Key Exchange

Key exchange using elliptic curves can be done in the following manner. First pick a large integer q, which is either a prime number p or an integer of the form 2m and elliptic curve parameters a and b for

$y^2 + xy = x^3 + ax^2 + b$ **or** $y^2 \bmod p = (x^3 + ax + b) \bmod p$

This defines the elliptic group of points $E_q(a, b)$. Next, pick a *base point* $G = (x1, y1)$ in $E_p(a, b)$ whose order is a very large value n. The **order** n of a point G on an elliptic curve is the smallest positive integer n such that $nG = O$. $E_q(a, b)$ and G are parameters of the cryptosystem known to all participants.
A key exchange between users A and B can be accomplished as follows
1. A selects an integer nA less than n. This is A's private key. A then generates a public key $PA = nA \times G$; the public key is a point in $E_q(a, b)$.
2. B similarly selects a private key nB and computes a public key PB.
3. A generates the secret key $K = nA \times PB$. B generates the secret key $K = nB \times PA$.

Global Public Elements	
$E_q(a, b)$	elliptic curve with parameters a, b, and q, where q is a prime or an integer of the form 2^m
G	point on elliptic curve whose order is large value n

User A Key Generation	
Select private n_A	$n_A < n$
Calculate public P_A	$P_A = n_A \times G$

User B Key Generation	
Select private n_B	$n_A < n$
Calculate public P_B	$P_B = n_B \times G$

Calculation of Secret Key by User A
$K = n_A \times P_B$

Calculation of Secret Key by User B
$K = n_B \times P_A$

Figure 3.4. ECC Diffie-Hellman Key Exchange

The two calculations in step 3 produce the same result because $nA \times PB = nA \times (nB \times G) = nB \times (nA \times G) = nB \times PA$

To break this scheme, an attacker would need to be able to compute k given G and kG, which is assumed hard.

As an example,

[5]

take $p = 211$; Ep(0, 4), which is equivalent to the curve $y^2 = x^3 4$; and $G = (2, 2)$.

One can calculate that $240G = O$. A's private key is $nA = 121$, so A's public key is $PA = 121(2, 2) = (115, 48)$. B's private key is $nB = 203$, so B's public key is $203(2, 2) = (130, 203)$.

The shared secret key is $121(130, 203) = 203(115, 48) = (161, 69)$.

Note that the secret key is a pair of numbers. If this key is to be used as a session key for conventional encryption, then a single number must be generated. We could simply use the x coordinates or some simple function of the x coordinate.

3.2.2 Elliptic Curve Encryption/Decryption

Several approaches to encryption/decryption using elliptic curves have been analyzed in the literature. The first task in this system is to encode the plaintext message m to be sent as an x-y point Pm. It is the point Pm that will be encrypted as a ciphertext and subsequently decrypted. Note that we cannot simply encode the message as the x or y coordinate of a point,because not all such coordinates are in Eq(a, b);

Again, there are several approaches to this encoding, which we will not address here, but suffice it to say that there are relatively straightforward techniques that can be used.

As with the key exchange system, an encryption/decryption system requires a point G and an elliptic group Eq(a, b) as parameters. Each user A selects a private key nA and generates a public key $PA = nA$ x G.

To encrypt and send a message Pm to B, A chooses a random positive integer k and produces the ciphertext Cm consisting of the pair of points:

$Cm = \{kG, Pm + kPB\}$

Note that A has used B's public key PB. To decrypt the ciphertext, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point:

$Pm + kPB\ nB(kG) = Pm + k(nBG)\ nB(kG) = Pm$

A has masked the message Pm by adding kPB to it. Nobody but A knows the value of k, so even though PB is a public key, nobody can remove the mask kPB. However, A also includes a "clue," which is enough to remove the mask if one knows the private key nB. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed hard.

As an example of the encryption process (taken from [KOBL94]), take $p = 751$; Ep(1, 188), which is equivalent to the curve $y2 = x3\ x + 188$; and $G = (0, 376)$. Suppose that A wishes to send a message to B that is encoded in the elliptic point $Pm = (562, 201)$ and that Aselects the random number $k = 386$. B's public key is $PB = (201, 5)$. We have $386(0, 376) = (676, 558)$, and $(562, 201) + 386(201, 5) = (385, 328)$. Thus A sends the cipher text $\{(676, 558), (385, 328)\}$.

3.2.3 Security of Elliptic Curve Cryptography

The security of ECC depends on how difficult it is to determine k given kP and P. This is referred to as the elliptic curve logarithm problem. The fastest known technique for taking the elliptic curve logarithm is known as the Pollard rho method. Table 3.3 compares various algorithms by showing comparable key sizes in terms of computational effort for cryptanalysis. As can be seen, a considerably smaller key size can be used for ECC comparedto RSA.

Furthermore, for equal key lengths, the computational effort required for ECC and RSA is comparable. Thus, there is a computational advantage to using ECC with a shorter key length than a comparably secure RSA.

Table 3.3. Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis

Symmetric Scheme (key size in bits)	ECC-Based Scheme (size of *n* in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
92	384	7680
256	512	15360

3.3 Pseudorandom Number Generation (PRNG) based on Asymmetric Ciphers

- asymmetric encryption algorithm produce apparently random output
- hence can be used to build a pseudorandom number generator (PRNG)
- much slower than symmetric algorithms
- hence only use to generate a short pseudorandom bit sequence (eg. key)

3.3.1 PRNG based on RSA

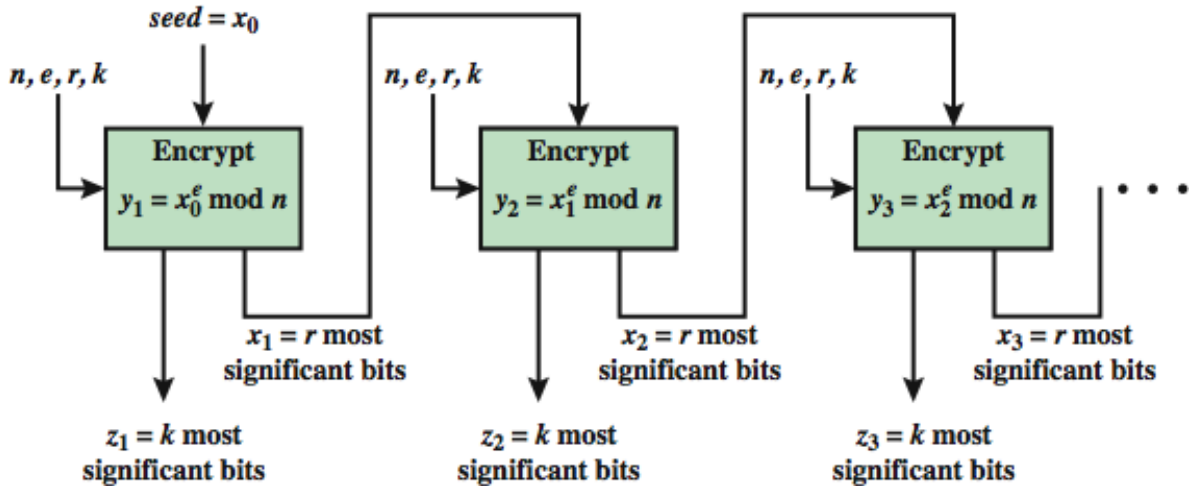


Figure 3.5: Micali-Schnorr Pseudorandom Bit Generator

We can define the PRNG as follows.

Setup Select p, q primes; $n = pq$; $\phi(n) = (p - 1)(q - 1)$. Select e such that $\gcd(e, \phi(n)) = 1$. These are the standard RSA setup selections (see Figure 9.5). In addition, let $N = [\log_2 n] + 1$ (the bitlength of n). Select r, k such that $r + k = N$.

Seed Select a random seed x_0 of bitlength r.

Generate Generate a pseudorandom sequence of length $k \times m$ using the loop

for i **from** 1 to m **do**

$$y_i = x_{i-1}^e \bmod n$$
$$x_i = r \text{ most significant bits of } y_i$$
$$z_i = k \text{ least significant bits of } y_i$$

Output The output sequence is $z_1 \| z_2 \| \ldots \| z_m$.

The parameters n, r, e, and k are selected to satisfy the following six requirements.

1. $n = pq$	n is chosen as the product of two primes to have the cryptographic strength required of RSA.
2. $1 < e < \phi(n)$; $\gcd(e, \phi(n)) = 1$	Ensures that the mapping $s \rightarrow s^e \bmod n$ is 1 to 1.
3. $re \geq 2N$	Ensures that the exponentiation requires a full modular reduction.
4. $r \geq 2 \times strength$	Protects against a cryptographic attacks.
5. k, r are multiples of 8	An implementation convenience.
6. $k \geq 8; r + k = N$	All bits are used.

3.3.2 PRNG based on ECC

- dual elliptic curve PRNG
  - NIST SP 800-9, ANSI X9.82 and ISO 18031
- some controversy on security /inefficiency
- algorithm

  for i = 1 to k do
  set $s_i = x(s_{i-1} P)$
  set $r_i = lsb_{240}(x(s_i Q))$
  end for
  return $r_1, \ldots, r_k$
- only use if just have ECC

3.4 Symmetric key distribution using symmetric encryption

- symmetric schemes require both parties to share a common secret key
- issue is how to securely distribute this key
- whilst protecting it from others
- frequent key changes can be desirable
- often secure system failure due to a break in the key distribution scheme

Given parties A and B have various **key distribution** alternatives:

1. A can select key and physically deliver to B
2. third party can select & deliver key to A & B
3. if A & B have communicated previously can use previous key to encrypt a new key
4. if A & B have secure communications with a third party C, C can relay key between A & B

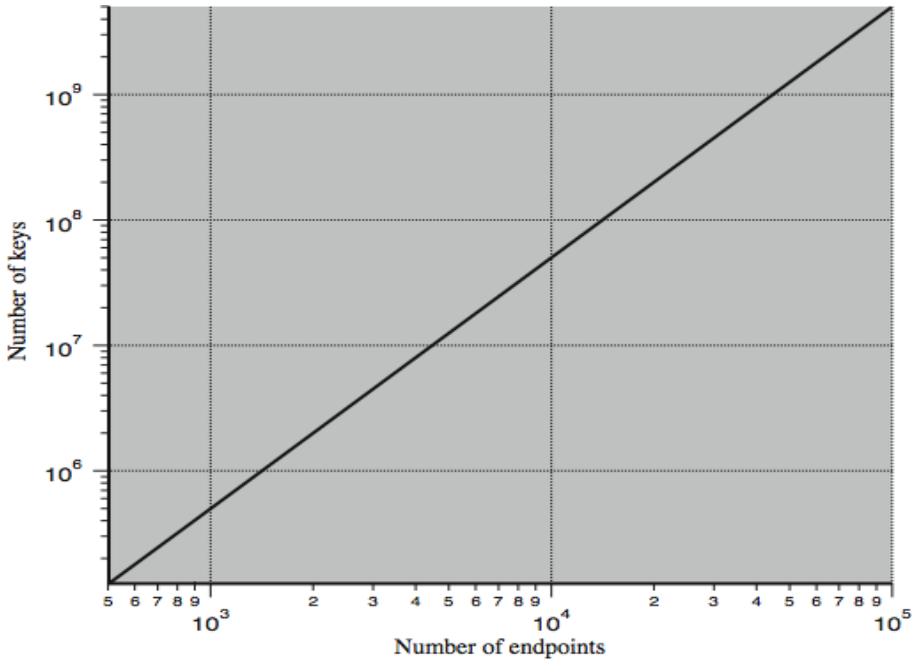


Figure 3.6: Number of keys required to support arbitrary connections between endpoints

Key Hierarchy

- typically have a hierarchy of keys
- session key

- temporary key
- used for encryption of data between users
- for one logical session then discarded

➢ master key
- used to encrypt session keys
- shared by user & key distribution center

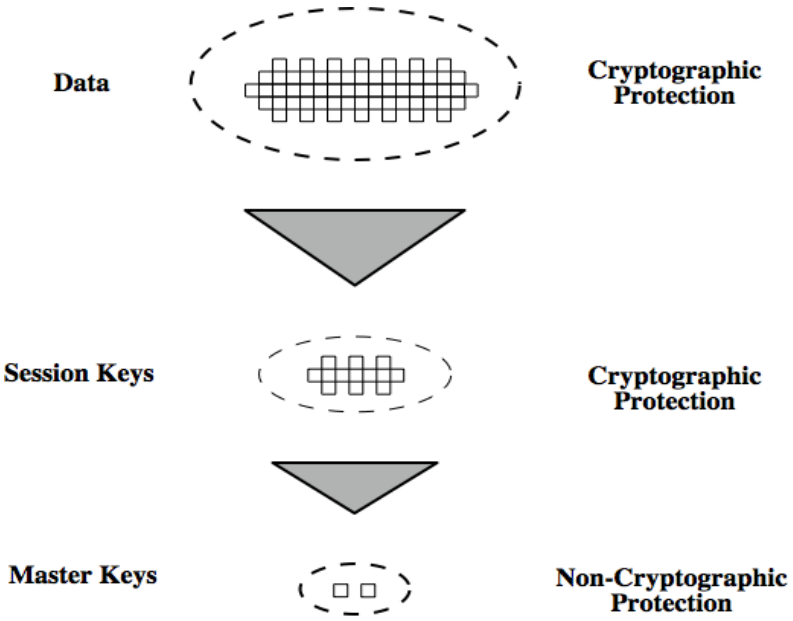


Figure 3.7: The use of a key hierarchy

3.4.1 Key Distribution Scenario

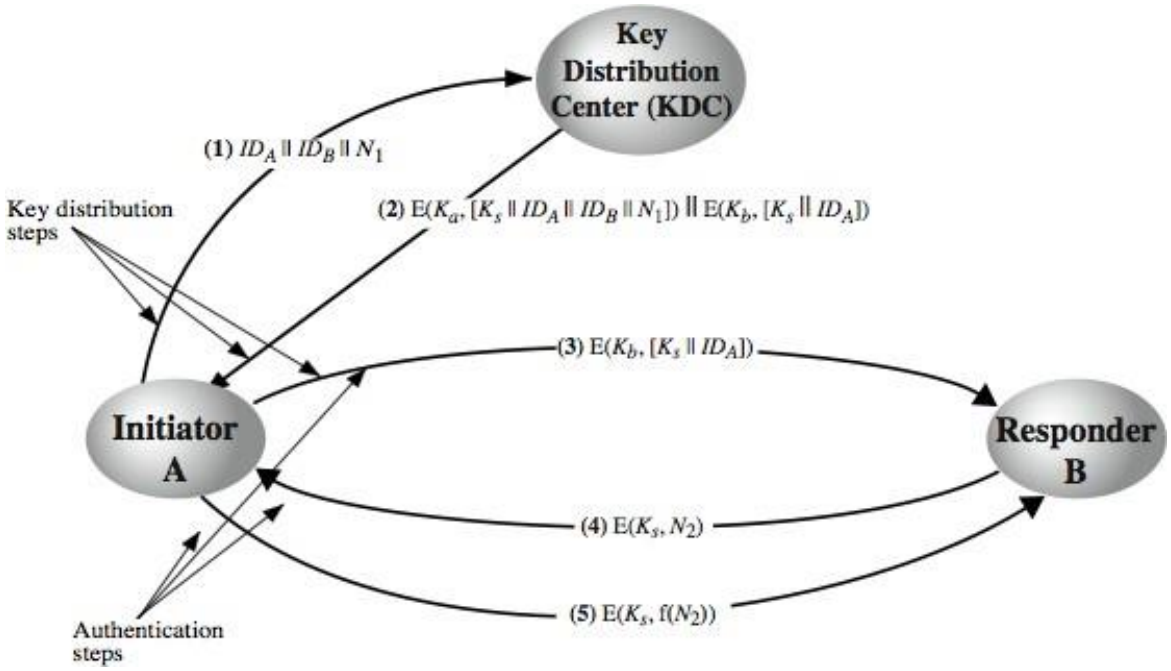


Figure 3.8: Key Distribution Scenario

3.4.2 A Transparent Key Control Scheme

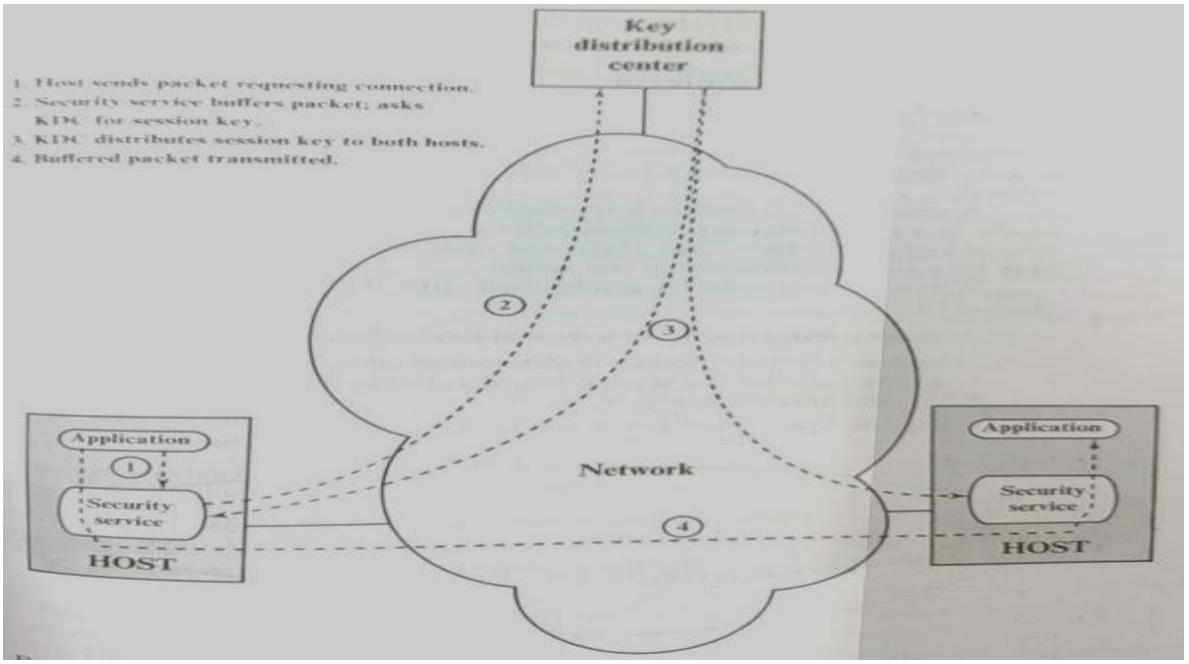


Figure 3.9: Automatic key distribution for connection-oriented protocol

3.4.3 Decentralized Key Control

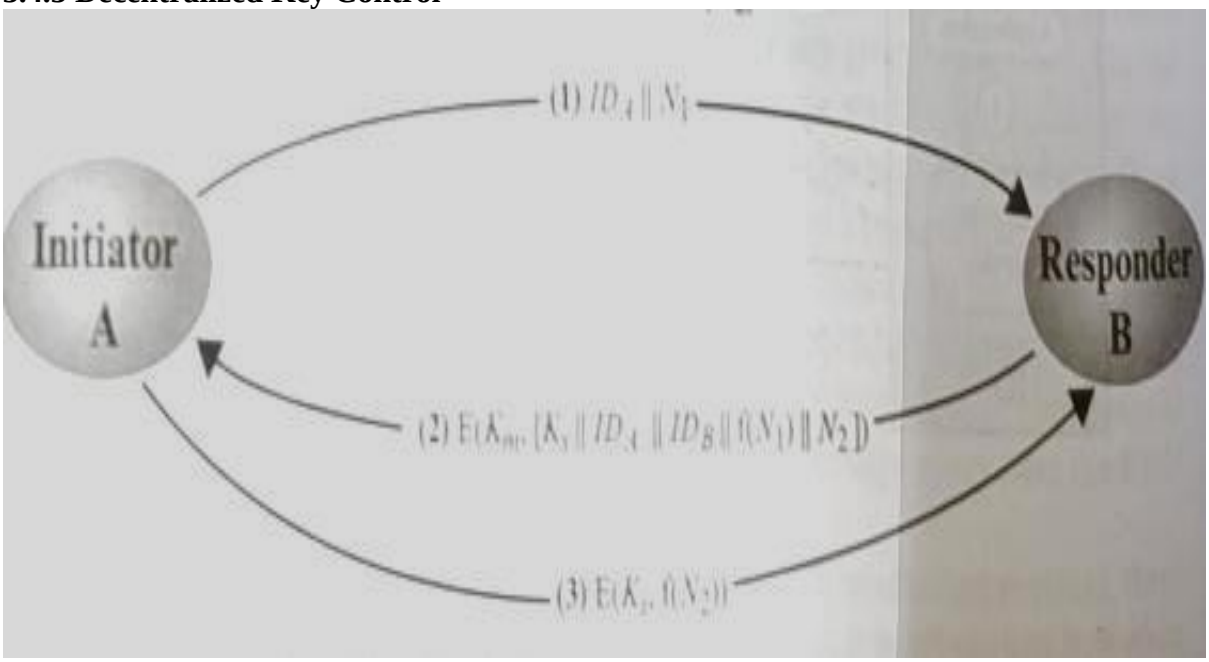


Figure 3.10: Decentralized key distribution

3.4.4 Controlling key usage

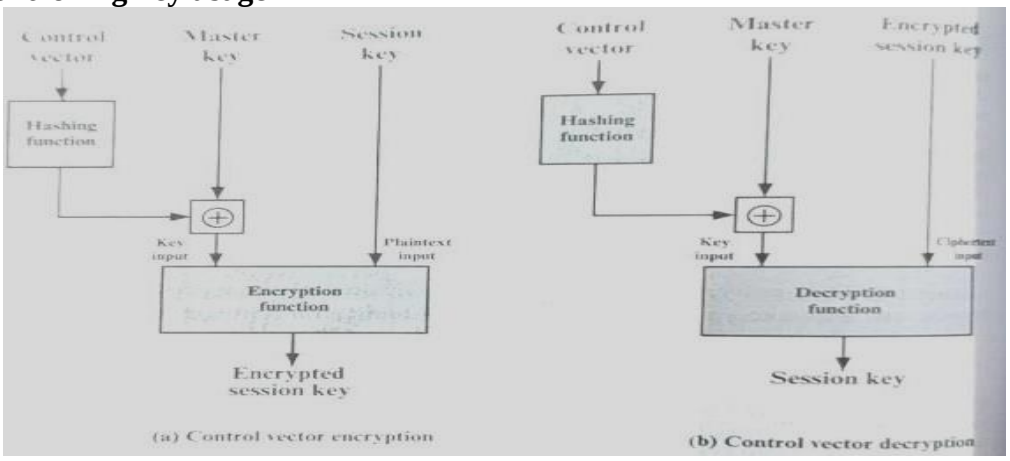


Figure 3.11: Control Vector Encryption and Decryption

Key Distribution Issues

- hierarchies of KDC's required for large networks, but must trust each other
- session key lifetimes should be limited for greater security
- use of automatic key distribution on behalf of users, but must trust system
- use of decentralized key distribution
- controlling key usage

3.5 Symmetric Key Distribution using asymmetric encryption

- public key cryptosystems are inefficient
  - so almost never use for direct data encryption
  - rather use to encrypt secret keys for distribution

3.5.1 Simple Secret Key Distribution

- Merkle proposed this very simple scheme
  - allows secure communications
  - no keys before/after exist

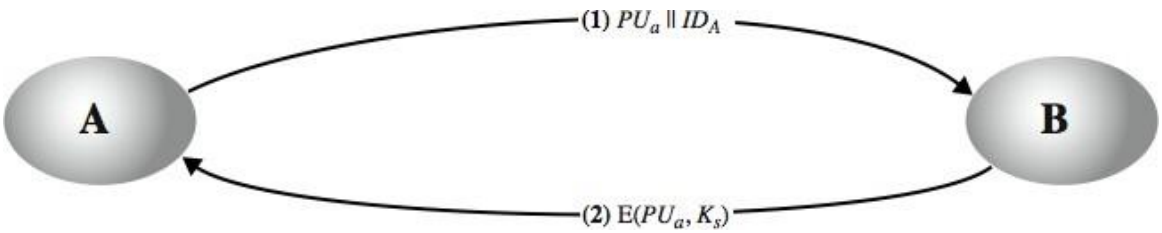


Figure 3.12: Simple use of public key encryption to establish a session key

- this very simple scheme is vulnerable to an active man-in-the-middle attack

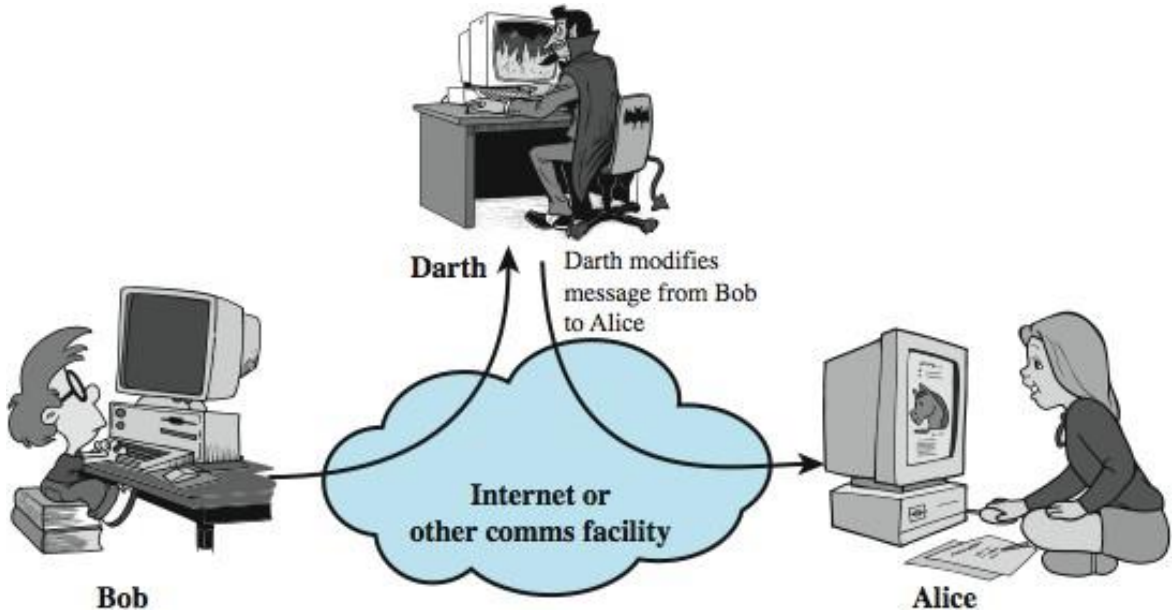


Figure 3.13: Another Man-in-the-Middle Attack

3.5.2 Secret Key Distribution with Confidentiality and Authentication

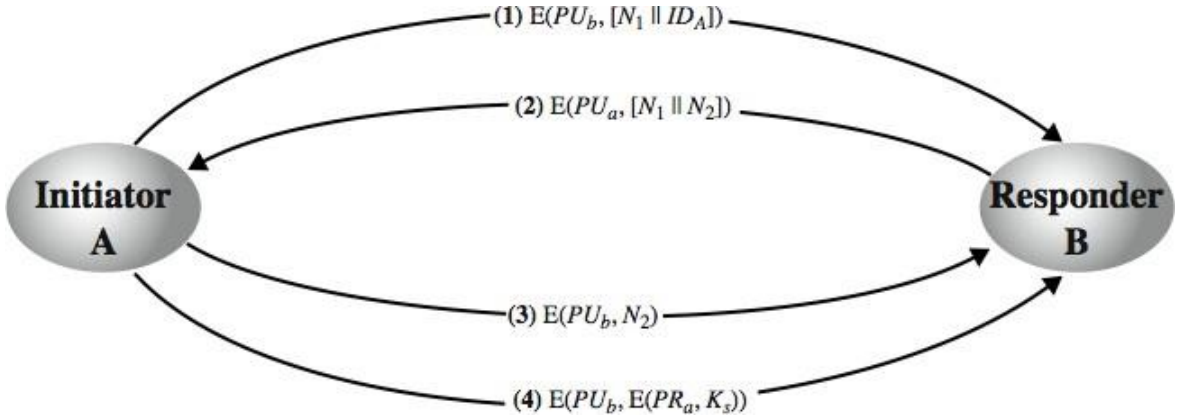


Figure 3.14: public key distribution of secret keys

3.5.3 Hybrid Key Distribution

- retain use of private-key KDC
- shares secret master key with each user
- distributes session key using master key
- public-key used to distribute master keys
  - especially useful with widely distributed users
- rationale
  - performance
  - backward compatibility

3.6 Distribution of Public Keys

- can be considered as using one of:
  - public announcement
  - publicly available directory
  - public-key authority
  - public-key certificates

3.6.1 Public Announcement

- users distribute public keys to recipients or broadcast to community at large
  - eg. append PGP keys to email messages or post to news groups or email list
- major weakness is forgery
  - anyone can create a key claiming to be someone else and broadcast it
  - until forgery is discovered can masquerade as claimed user

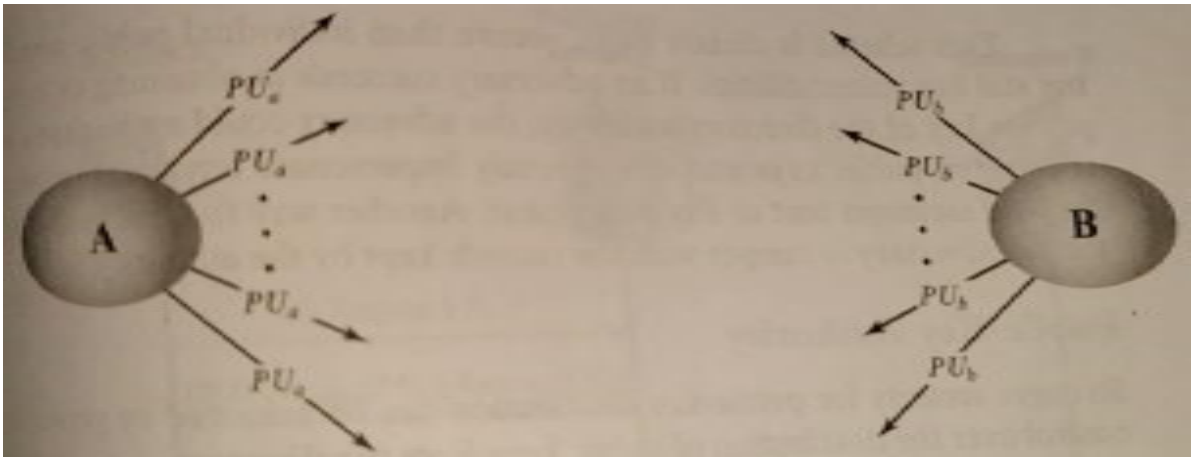


Figure 3.15: Uncontrolled public key distribution

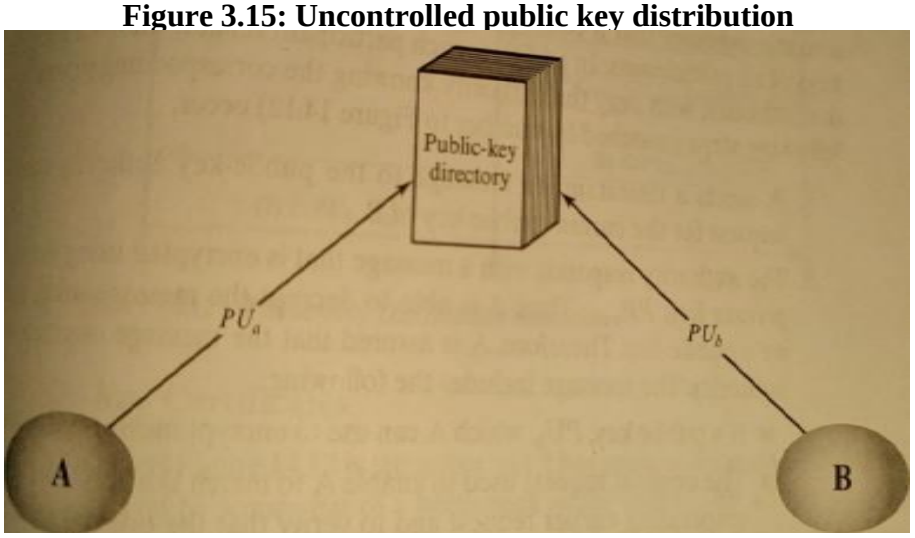


Figure 3.16: public-key publication

3.6.2 Publicly Available Directory

- can obtain greater security by registering keys with a public directory
- directory must be trusted with properties:
  - contains {name,public-key} entries
  - participants register securely with directory

- participants can replace key at any time
- directory is periodically published
- directory can be accessed electronically

➢ still vulnerable to tampering

3.6.3 Public-Key Authority

➢ improve security by tightening control over distribution of keys from directory
➢ has properties of directory
➢ and requires users to know public key for the directory
➢ then users interact with directory to obtain any desired public key securely
- does require real-time access to directory when keys are needed
- may be vulnerable to tampering

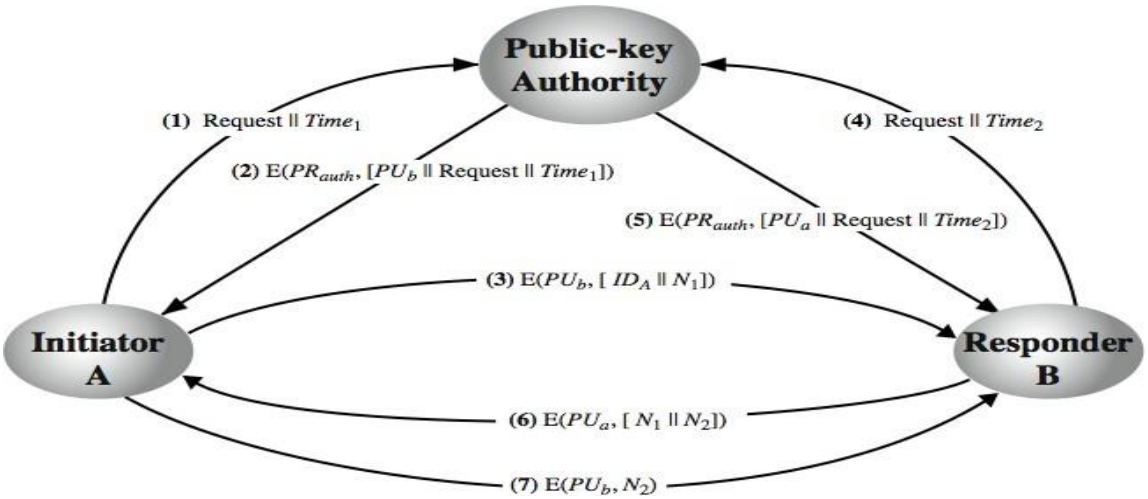


Figure 3.17: public key distribution scenario

3.6.4 Public-Key Certificates

➢ certificates allow key exchange without real-time access to public-key authority
➢ a certificate binds identity to public key
- usually with other info such as period of validity, rights of use etc

➢ with all contents signed by a trusted Public-Key or Certificate Authority (CA)
➢ can be verified by anyone who knows the public-key authorities public-key

Requirements:

1. Any participant can read a certificate to determine the name and public key of the certificate‘s owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any participant can verify the time validity of the certificate.

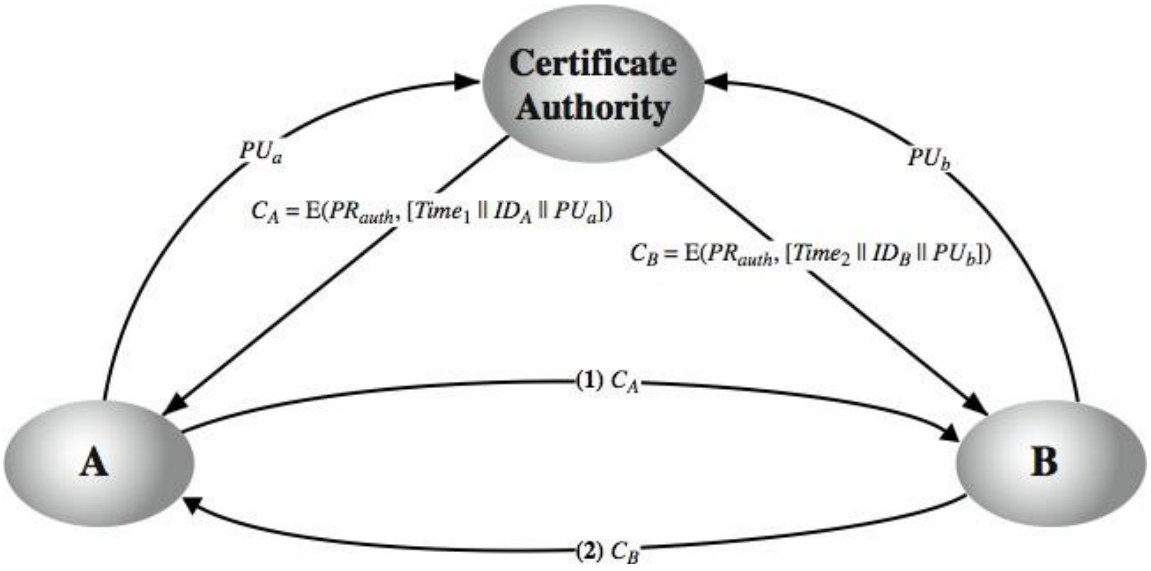


Figure 3.18: Exchange of public-key certificates

Module-4:

4.1 X.509 certificates

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users.

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts.

X.509 was initially issued in 1988. The standard was subsequently revised to address some of the security concerns documented; a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function. Again, the standard does not dictate a specific hash algorithm. The 1988 recommendation included the description of a recommended hash algorithm; this algorithm has since been shown to be insecure and was dropped from the 1993 recommendation. Figure 4.1 illustrates the generation of a public-key certificate.

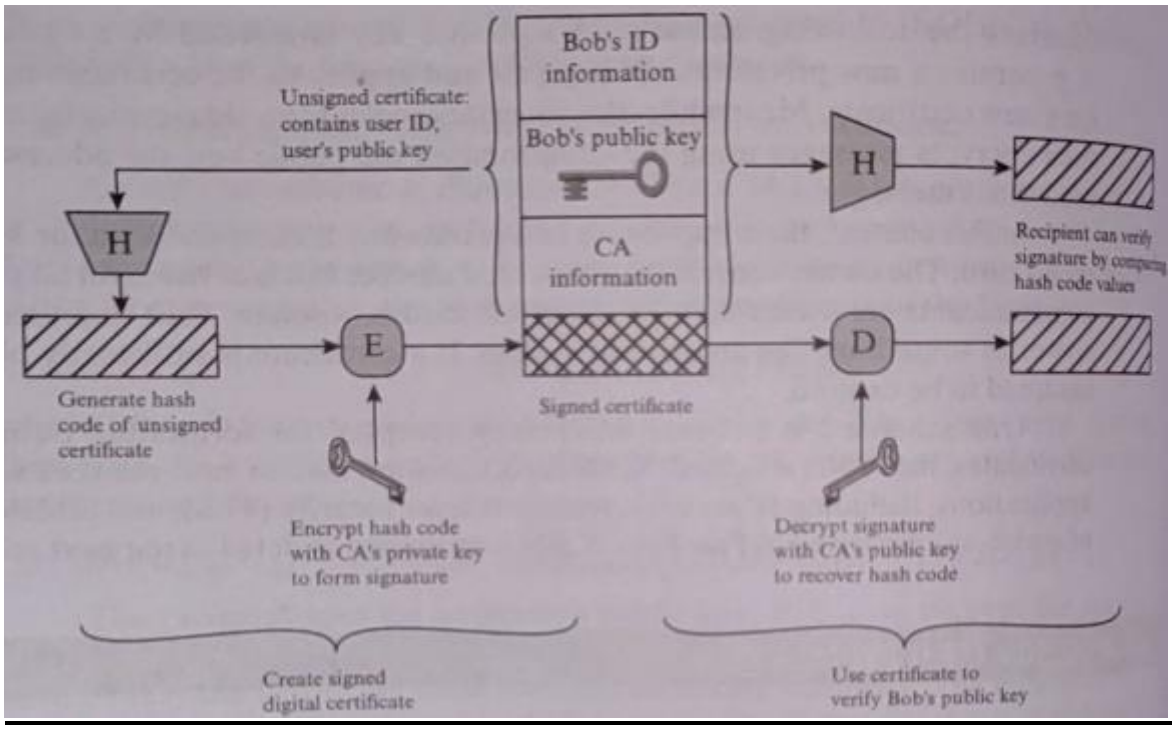


Figure 4.1: X.509 public key certification use

4.1.1 Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

Figure 14.2a shows the general format of a certificate, which includes the following elements:
● **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.
● **Serial number:** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.
● **Signature algorithm identifier:** The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.
● **Issuer name:** X.500 name of the CA that created and signed this certificate.
● **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.
● **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.
● **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
● **Issuer unique identifier:** An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
● **Subject unique identifier:** An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
● **Extensions:** A set of one or more extension fields. Extensions were added in version 3
● **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields, encrypted with the CA's private key. This field includes the signature algorithm identifier.

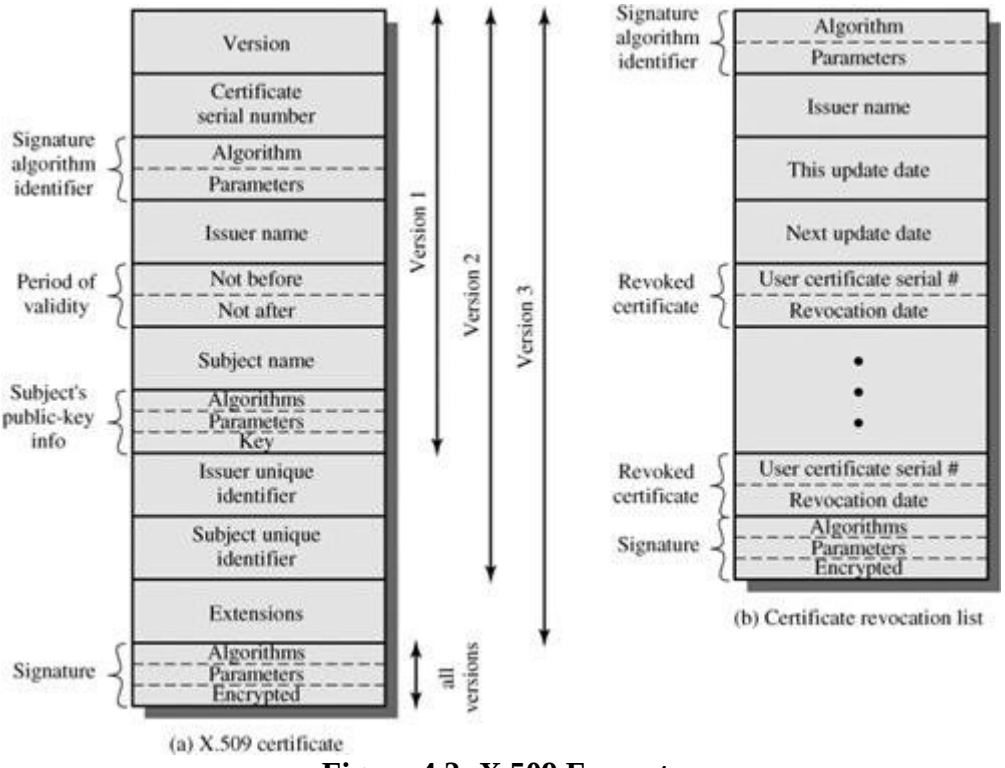


Figure 4.2: X.509 Formats

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used.
The standard uses the following notation to define a certificate:
CA<<A>> = CA {V, SN, AI, CA, TA, A, Ap}

where
Y <<X>> = the certificate of user X issued by certification authority Y
Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended
The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

4.1.2 Obtaining a User's Certificate

User certificates generated by a CA have the following characteristics:
● Any user with access to the public key of the CA can verify the user public key that was certified.
● No party other than the certification authority can modify the certificate without this being detected.
Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them. If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directoryfor access by all users. In addition, a user can transmit his or her certificate directly to other users. In either case, once B is in possession of A's certificate, B has confidence thatmessages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X1 and B has obtained a certificate from CA X2. If A does not securely know the public key of X2, then B's certificate, issued by X2, is useless to A. A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own publickeys, the following procedure will enable A to obtain B's public key:

1. A obtains, from the directory, the certificate of X2 signed by X1. Because A securely knows X1's public key, A can obtain X2's public key from its certificate and verify it by means of X1's signature on the certificate.
2. A then goes back to the directory and obtains the certificate of B signed by X2 Because A now has a trusted copy of X2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

X1<<X2>> X2 <<B>>

In the same fashion, B can obtain A's public key with the reverse chain:

X2<<X1>> X1 <<A>>

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with *N* elements would be expressed as

X1<<X2>> X2 <<X3>>... X*N*<<B>>

In this case, each pair of CAs in the chain (X*i, Xi+1*) must have created certificates for each other. All these certificates of CAs by CAs need to appear in the directory, and the user needs

to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 4.3, taken from X.509, is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

- **Forward certificates:** Certificates of X generated by other CAs
- **Reverse certificates:** Certificates generated by X that are the certificates of other CAs

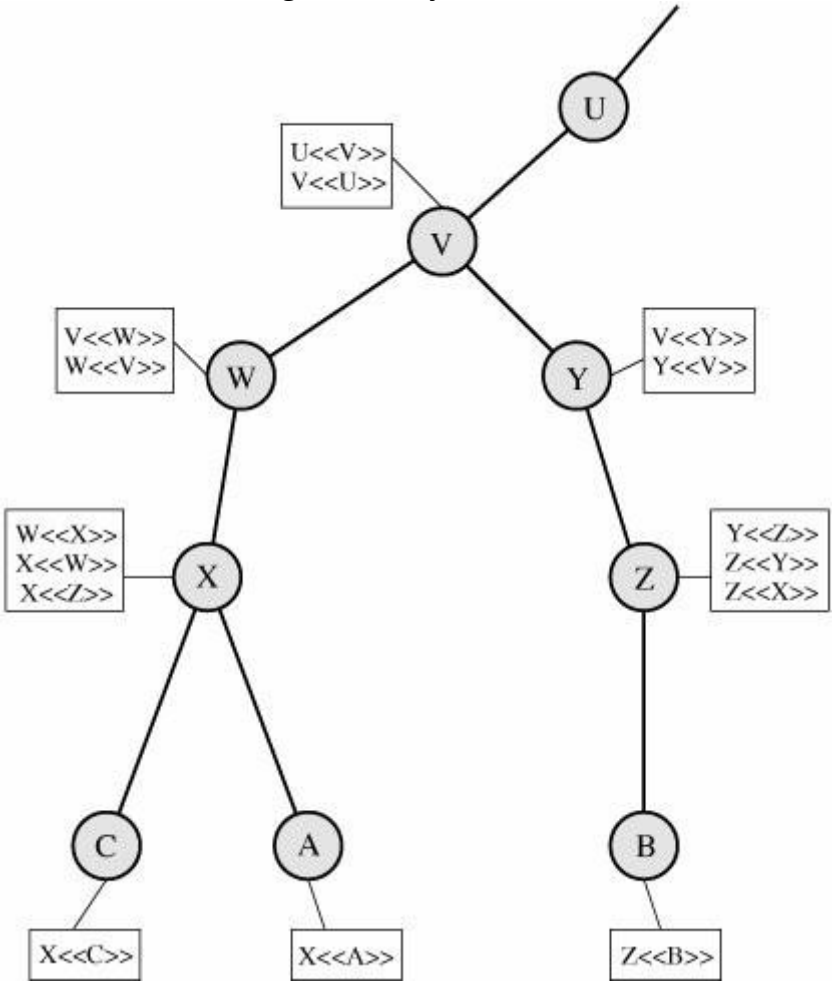


Figure 4.3:X.509 Hierarchy: A hypothetical example

In this example, user A can acquire the following certificates from the directory to establish a certification path to B:

X<<W>> W <<V>> V <<Y>> <<Z>> Z <<B>>

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

Z<<Y>> Y <<V>> V <<W>> W <<X>>X <<A>>

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.

4.1.3 Revocation of Certificates

Recall from Figure 4.2 that each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:

1. The user's private key is assumed to be compromised.
2. The user is no longer certified by this CA.
3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 4.2b) the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

4.1.4 X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. lists the following requirements not satisfied by version 2:

1. The Subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identificationdetails that may be needed by the user.
2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.
3. There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.
4. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
5. It is important to be able to identify different keys used by the same owner at different times. This feature supports key life cycle management, in particular the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. Thus, version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored. If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

4.1.5 Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy. A certificate policy is a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application withcommon security requirements. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goodswithin a given price range.

This area includes the following:

- **Authority key identifier:** Identifies the public key to be used to verify the signature on this certificate or CRL. Enables distinct keys of the same CA to be differentiated. One use of this field is to handle CA key pair updating.
- **Subject key identifier:** Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs and, correspondingly, different certificates for different purposes (e.g., digital signature and encryption key agreement).
- **Key usage:** Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used. May indicate one or more of the following: digital signature, nonrepudiation, key encryption, data encryption, key agreement, CA signature verification on certificates, CA signature verification on CRLs.
- **Private-key usage period:** Indicates the period of use of the private key corresponding to the public key. Typically, the private key is used over a different period from the validity of the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.
- **Certificate policies:** Certificates may be used in environments where multiple policies apply. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.
- **Policy mappings:** Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

4.1.6 Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, toincrease a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.

The extension fields in this area include the following:

- **Subject alternative name:** Contains one or more alternative names, using any of a variety of forms. This field is important for supporting certain applications, such as electronic mail, EDI, and IPSec, which may employ their own name forms.
- **Issuer alternative name:** Contains one or more alternative names, using any of a variety of forms.
- **Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

4.1.7 Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The constraints may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

The extension fields in this area include the following:

● **Basic constraints:** Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.
● **Name constraints:** Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.
● **Policy constraints:** Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

4.2 Public-Key Infrastructure

RFC 2822 (*Internet Security Glossary*) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys. The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet. This section describes the PKIX model.

Figure 4.4 shows the interrelationship among the key elements of the PKIX model. These elements are
● **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities typically consume and/or support PKI-related services.
● **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more Registration Authorities.
● **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the End Entity registration process, but can assist in a number of other areas as well.
● **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
● **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by End Entities.

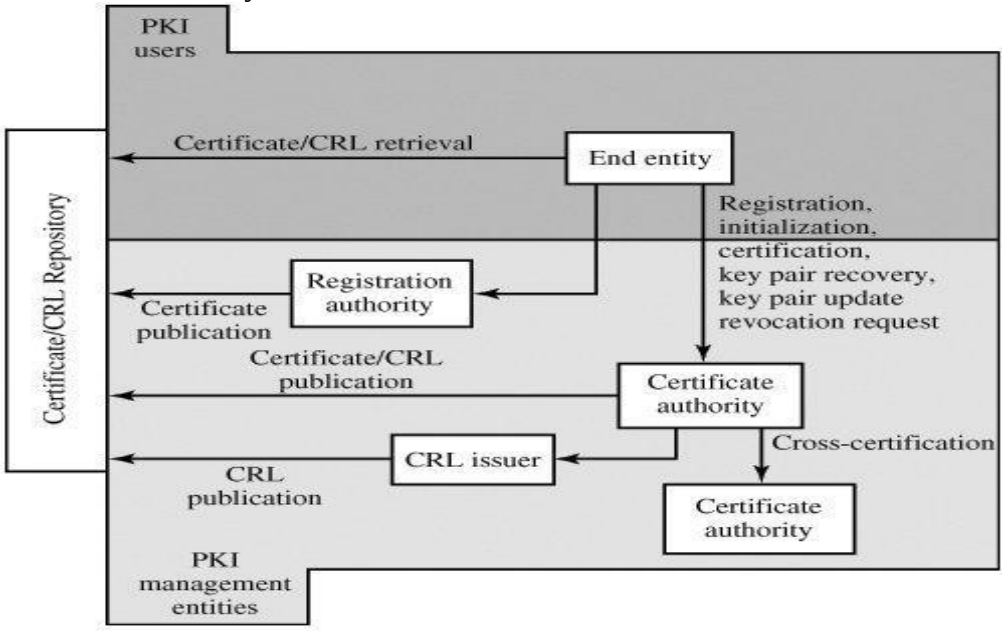


Figure 4.4: PKIX architecture model

4.2.1 PKIX Management Functions

PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 4.4 and include the following:

- **Registration:** This is the process whereby a user first makes itself known to a CA (directly, or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some offline or online procedure for mutual authentication. Typically, the end entity is issued oneor more shared secret keys used for subsequent authentication.
- **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.
- **Certification:** This is the process in which a CA issues a certificate for a user's public key, and returns that certificate to the user's client system and/or posts that certificate in a repository.
- **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible, otherwise it will not be possible to recover the encrypted data. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the End Entity's certificate).
- **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

4.2.2 PKIX Management Protocols

The PKIX working group has defines two alternative management protocols between PKIX entities that support the management functions listed in the preceding subsection. RFC 2510 defines the certificate management protocols (CMP). Within CMP, each of the management functions is explicitly identified by specific protocol exchanges. CMP is designed to be a flexible protocol able to accommodate a variety of technical, operational, and business models.

RFC 2797 defines certificate management messages over CMS (CMC), where CMS refers to RFC 2630, cryptographic message syntax. CMC is built on earlier work and is intended to leverage existing implementations. Although all of the PKIX functions are supported, the functions do not all map into specific protocol exchanges.

4.3 Remote user-authentication principles

User authentication is the fundamental security building block and the primary line of defense. User authentication is the basic for more types of access control & user accountability. RFC 4949 defines user authentication as the process of verifying an identity claimed by or for a system entity

This process consists of two steps:

- Identification step – Presenting an identifier to the security system.
- verification – presenting or generating authentication information that corroboratesthe binding between the entity and the identifier

4.3.1 The NIST Model for electronic user authentication

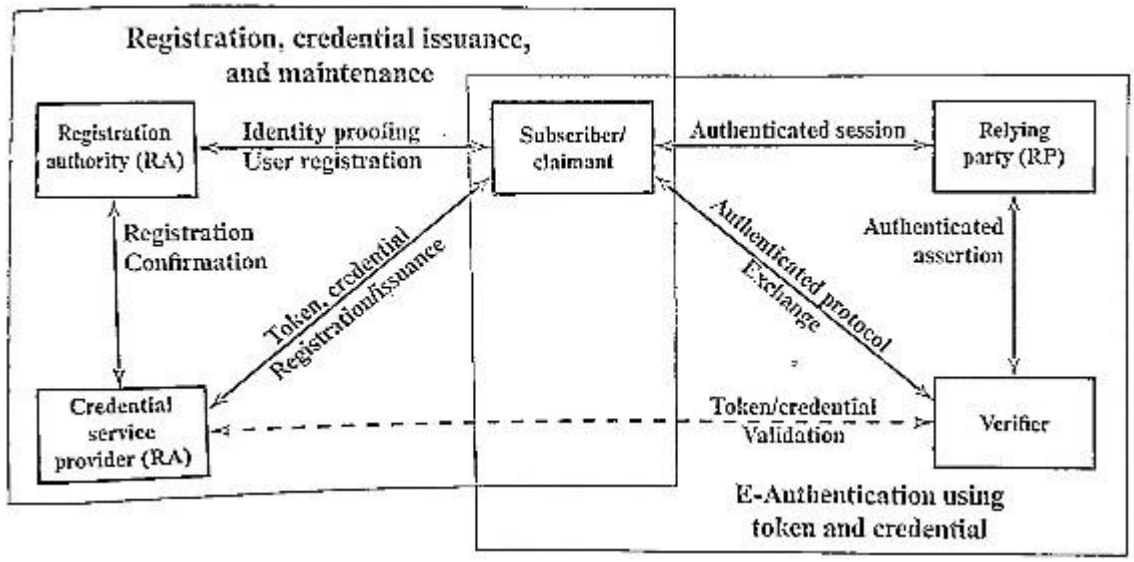


Figure 4.5: the NIST SP 800-63-2 E-authentication architectural model

4.3.2 Means of User Authentication

Four means of authenticating user's identity

- based one something the individual
  - something the individual knows - e.g. password, PIN
  - something the individual possesses - e.g. key, token, smartcard
  - something the individual is (static biometrics) - e.g. fingerprint, retina
  - something the individual does (dynamic biometrics) - e.g. voice, sign

can use alone or combined, all can provide user authentication but all have issues.

4.3.3 Mutual Authentication

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

lists the following examples of replay attacks:

- **Simple replay:** The opponent simply copies a message and replays it later.
- **Repetition that can be logged:** An opponent can replay a time stamped message within the valid time window.
- **Repetition that cannot be detected:** This situation could arise because the originalmessage could have been suppressed and thus did not arrive at its destination; only the replay message arrives.
- **Backward replay without modification:** This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

- **Timestamps:** Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.
- **Challenge/response:** Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

It can be argued that the timestamp approach should not be used for connection oriented applications because of the inherent difficulties with this technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to cope with network errors, and secure, to cope with hostile attacks.

Second, the opportunity for a successful attack will arise if there is a temporary loss of synchronization resulting from a fault in the clock mechanism of one of the parties. Finally, because of the variable and unpredictable nature of network delays, distributed clocks cannot be expected to maintain precise synchronization. Therefore, any timestamp-based procedure must allow for a window of time sufficiently large to accommodate network delays yet sufficiently small to minimize the opportunity for attack.

On the other hand, the challenge-response approach is unsuitable for a connectionless type of application because it requires the overhead of a handshake before any connectionless transmission, effectively negating the chief characteristic of a connectionless transaction. For such applications, reliance on some sort of secure time server and a consistent attempt by each party to keep its clocks in synchronization may be the best approach.

4.3.4 One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the e-mail message must be in the clear, so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400. However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism.

Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

4.4 Remote User-authentication using symmetric encryption

A two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution. This approach is quite common.

Key distribution scenario illustrates a proposal initially put forth by Needham and Schroeder for secret key distribution using a KDC that includes authentication features. The protocol can be summarized as follows:

1. A ⟶ KDC: $ID_A||ID_B||N_1$
2. KDC ⟶ A: $E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$
3. A ⟶ B: $E(K_b, [K_s||ID_A])$
4. A ⟶ A: $E(K_s, N_2)$
5. A ⟶ B: $E(K_s, f(N_2))$

Secret keys *Ka* and *Kb* are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key *Ks* to A and B. A securely acquires a new session key in step 2. The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of *Ks*, and step 5 assures B of A's knowledge of *Ks* and assures B that this is a fresh message because of the use of the nonce *N*2. The purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B. Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. X can impersonate A and trick B into using the old key by simply replaying step
3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay. If X can intercept the handshake message, step 4, then it can impersonate A's response, step 5. From this point on, X can send bogus messages to B that appear to B to come from A using an authenticated session key.

Denning proposes to overcome this weakness by a modification to the Needham/ Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys, *Ka* and *Kb* are secure, and it consists of the following steps:

1. A ⟶ KDC: $ID_A||ID_B$
2. KDC ⟶ A: $E(K_a, [K_s||ID_B||T||E(K_b, [K_s||ID_A||T])])$
3. A ⟶ B: $E(K_b, [K_s||ID_A||T])$
4. B ⟶ A: $E(K_s, N_1)$
5. A ⟶ B: $E(K_s, f(N_1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that

$$|\text{Clock } T| < \Delta t_1 + \Delta t_2$$

where Δt_1 is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and Δt_2 is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp *T* is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.

A final point: Steps 4 and 5 were not included in the original presentation but were added Later. These steps confirm the receipt of the session key at B.

The Denning protocol seems to provide an increased degree of security compared to the Needham/ Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism.

The problem occurs when a sender's clock is ahead of the intended recipient's clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks**.

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities. An attempt is made to respond to the concerns about suppress-replay attacks and at the same time fix the problems in the Needham/Schroeder protocol. Subsequently, an inconsistency in this latter protocol wasnoted and an improved strategy

The protocol is as follows:

1. $A \longrightarrow B: \quad ID_A||N_a$

2. $B \longrightarrow KDC: \quad ID_B||N_b||E(K_b, [ID_A||N_a||T_b])$

3. $KDC \longrightarrow A: \quad E(K_a, [ID_B||N_a||K_s||T_b])||E(K_b,[ID_A||K_s||T_b])||N_b$

4. $A \longrightarrow B: \quad E(K_b, [ID_A||K_s||T_b])||E(K_s, N_b)$

Let us follow this exchange step by step.

1. A initiates the authentication exchange by generating a nonce, *Na*, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.

2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, *Nb* This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct

the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.

3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent authentications, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message (*IDB*) and that this is a timely message and not a replay (*Na*) and it provides A with a session key (*Ks*) and the time limit on its use (*Tb*).

4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt E(*Ks, Nb*) to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates thatthe message came from A and is not a replay.

This protocol provides an effective, secure means for A and B to establish a session with a secure session key. Furthermore, the protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly. Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit establishedby the protocol, A desires a new session with B.

The following protocol ensues:

1. $A \longrightarrow B: \quad E(K_b, [ID_A||K_s||T_b])||N'_a$

2. $B \longrightarrow A: \quad N'_b||E(K_s, N'_a)$

3. $A \longrightarrow B: \quad E(K_s, N'_b)$

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces *N'a* and *N'b* assure each party that there is no replay attack. In all the foregoing, the time specified in *Tb* is a time relative to B's clock. Thus, this timestamp does not require synchronized clocks because B checks only self-generated timestamps.

4.4.1 One-Way Authentication

Using symmetric encryption, the decentralized key distribution scenario illustrated is impractical. This scheme requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message.

With some refinement, the KDC strategy illustrated is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content *M*,the sequence is as follows:

1. $A \longrightarrow KDC: \quad ID_A||ID_B||N_1$

2. $KDC \longrightarrow A: \quad E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$

3. $A \longrightarrow B: \quad E(K_b, [K_s||ID_A])||E(K_s, M)$

This approach guarantees that only the intended recipient of a message will be able to read it. It also provides a level of authentication that the sender is A. As specified, the protocol does not protect against replays. Some measure of defense could be provided by including a

timestamp with the message. However, because of the potential delays in the e-mail process, such timestamps may have limited usefulness.

4.5 KERBEROS

Kerberos is an authentication service developed as part of Project Athena at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services.

In particular, the following three threats exist:

1. A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
2. A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
3. A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users.

Unlike most other authentication schemes, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

Two versions of Kerberos are in common use. Version 4 implementations still exist. Version 5 corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 1510).

4.5.1 Motivation

If a set of users is provided with dedicated personal computers that have no network connections, then a user's resources and files can be protected by physically securing each personal computer. When these users instead are served by a centralized time-sharing system, the time-sharing operating system must provide the security. The operating system canenforce access control policies based on user identity and use the logon procedure to identify users.

Today, neither of these scenarios is typical. More common is a distributed architecture consisting of dedicated user workstations (clients) and distributed or centralized servers.
In this environment, three approaches to security can be envisioned:

1. Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
2. Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
3. Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

In a small, closed environment, in which all systems are owned and operated by a single organization, the first or perhaps the second strategy may suffice. But in a more open environment, in which network connections to other machines are supported, the third approach is needed to protect user information and resources housed at the server. Kerberos

supports this third approach. Kerberos assumes a distributed client/server architecture and employs one or more Kerberos servers to provide an authentication service.

The first published report on Kerberos listed the following requirements:

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.
- **Transparent:** Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.
- **Scalable:** The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder. It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.

4.5.2 Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant of Project Athena and build up to the full protocol by looking first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

After examining the protocol, we look at some other aspects of version 4.

4.5.3 A Simple Authentication Dialogue

In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers mustbe able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other securemanner. Consider the following hypothetical dialogue:

(1) C ⟶ AS: $ID_C||P_C||ID_V$

(2) AS ⟶ C: $Ticket$

(3) C ⟶ V: $ID_C||Ticket$

$Ticket = E(K_v, [ID_C||AD_C||ID_V])$

where

C = client

AS = authentication server

V =server

ID_C = identifier of user on C

ID_V = identifier of V

P_C = password of user on C

AD_C = network address of C

K_v = secret encryption key shared by AS and V

In this scenario, the user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a ticket that contains the user's ID and network address and the server's ID. This ticketis encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C.

Because the ticket is encrypted, it cannot be altered by C or by an opponent. With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match, the server considers the user authenticated and grants the requested service.

Each of the ingredients of message (3) is significant. The ticket is encrypted to prevent alteration or forgery. The server's ID (*IDV*) is included in the ticket so that the server

can verify that it has decrypted the ticket properly. *IDC* is included in the ticket to indicate that this ticket has been issued on behalf of C. Finally, *ADC* serves to counter the following threat. An opponent could capture the ticket transmitted in message (2), then use the name *IDC* and transmit a message of form (3) from another workstation.

The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation. To prevent this attack, the AS includes in the ticket the network address from which the original request came. Now the ticket is valid only if it is transmitted from the same workstation that initially requested the ticket.

4.5.4 A More Secure Authentication Dialogue

Although the foregoing scenario solves some of the problems of authentication in an open network environment, problems remain. Two in particular stand out. First, we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable.

For a single logon session, the workstation can store the mail server ticket after it is receive and use it on behalf of the user for multiple accesses to the mail server. However, under this scheme it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on, thefirst instance of each access would require a new ticket and hence require the user to enter the password.

The second problem is that the earlier scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any service accessible to the victim.

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the ticket-granting server (TGS). The new but still hypothetical scenario is as follows:

Once per user logon session:

(1) C → AS: $ID_C || ID_{tgs}$

(2) AS → C: $E(K_C, Ticket_{tgs})$

Once per type of service:

(3) C → TGS: $ID_C || ID_V || Ticket_{tgs}$

(4) TGS → C: $Ticket_V$

Once per service session:

(5) C → V: $ID_C || Ticket_V$

$Ticket_{tgs} = E(K_{tgs}, [ID_C || AD_C || ID_{tgs} || TS_1 || Lifetime_1])$

$Ticket_V = E(K_V, [ID_C || AD_C || ID_V || TS_2 || Lifetime_2])$

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket (*Tickettgs*) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and usesit to authenticate its user to a server each time a particular service is requested.

Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID and password to the AS, together with the TGS ID, indicating a request to use the TGS service.

2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password. When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user, and the ID of the TGS. This corresponds to the first scenario. The idea is that the client can use this ticket to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable.

However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the login ticket and waits until the user has logged off his or her workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of thevictim. The opponent would be able to reuse the ticket to spoof the TGS.

To counter this, the ticket includes a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid (e.g., eight hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note that the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password. This assures that the ticket can be recovered only by the correct user, providing the authentication.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.

4. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again,the ticket contains a timestamp and lifetime.

If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (*Kv*) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5:

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

4.5.5 The Version 4 Authentication Dialogue

Although the foregoing scenario enhances security compared to the first attempt, two additional problems remain. The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password.

If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. An opponent could eavesdrop on the network and capture a copy of the ticket- granting ticket and then wait for the legitimate user to log out. Then the opponent could forgethe legitimate user's network address and send the message of step (3) to the TGS. Thiswould give the opponent unlimited access to the resources and files available to the legitimateuser.

Similarly, if an opponent captures a service-granting ticket and uses it before itexpires, the opponent has access to the corresponding service. Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able toprove that the person using a ticket is the same person to whom that ticket was issued.

The second problem is that there may be a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage theconfiguration so that messages to a server were directed to another location. The false server would then be in a position to act as a real server and capture any information from the user and deny the true service to the user.

We examine these problems in turn and refer to Table 4.1, which shows the actual Kerberos protocol.

Table 4.1: Summary of Kerberos Version 4 Message Exchanges

(1) C → AS	$ID_c \| \| ID_{tgs} \| \| TS_1$
(2) AS → C	$E(K_c, [K_{c,tgs} \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2 \| \| Ticket_{tgs}])$
	$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| \| ID_c \| \| AD_c \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2])$
(a) Authentication Service Exchange to obtain ticket-granting ticket	
(3) C → TGS	$ID_v \| \| Ticket_{tgs} \| \| Authenticator_c$
(4) TGS → C	$E(K_{c,tgs}, [K_{c,v} \| \| ID_v \| \| TS_4 \| \| Ticket_v])$
	$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| \| ID_C \| \| AD_C \| \| ID_{tgs} \| \| TS_2 \| \| Lifetime_2])$ $Ticket_v = E(K_v, [K_{c,v} \| \| ID_C \| \| AD_C \| \| ID_v \| \| TS_4 \| \| Lifetime_4])$ $Authenticator_c = E(K_{c,tgs}, [ID_C \| \| AD_C \| \| TS_3])$
(b) Ticket-Granting Service Exchange to obtain service-granting ticket	
(5) C → V	$Ticket_v \| \| Authenticator_c$
(6) V → C	$E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)
	$Ticket_v = E(K_v, [K_{c,v} \| \| ID_c \| \| AD_c \| \| ID_v \| \| TS_4 \| \| Lifetime_4])$ $Authenticator_c = E(K_{c,v}, [ID_c \| \| AD_C \| \| TS_5])$
(c) Client/Server Authentication Exchange to obtain service	

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. The threat is that an opponent will steal the ticket and use it before it expires. To get around this problem, let us have the AS provide both the client and the TGS with a secret piece of information in a secure manner. Then the client can prove its identity to the TGS by revealing the secret information, again in a secure manner. An efficient way of accomplishing this is to use an encryption key as the secure information; this is referred to as a session key in Kerberos.

Table 4.1a shows the technique for distributing the session key. As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password (*Kc*) that contains the ticket. The encrypted message also contains a copy of the session key, *Kc,tgs*, where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with *Kc*, only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service (message (3) in Table 4.1b). In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key *Kc,tgs*. In effect, the ticket says, "Anyone who uses *Kc,tgs* must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner.

In effect, the authenticator says, "At time *TS*3, I hereby use *Kc,tgs*." Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only onceand has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.

The reply from the TGS, in message (4), follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself includes the same session key. C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply as shown in message (6) of Table 4.1. The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

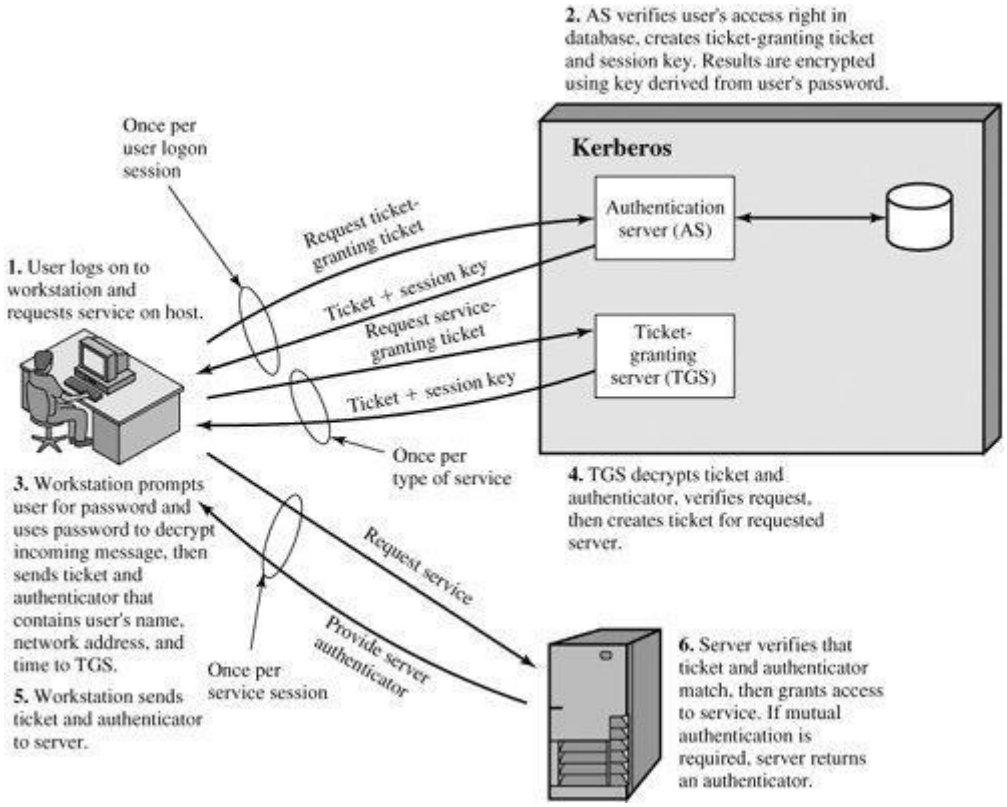


Figure 4.6: Overview of Kerberos

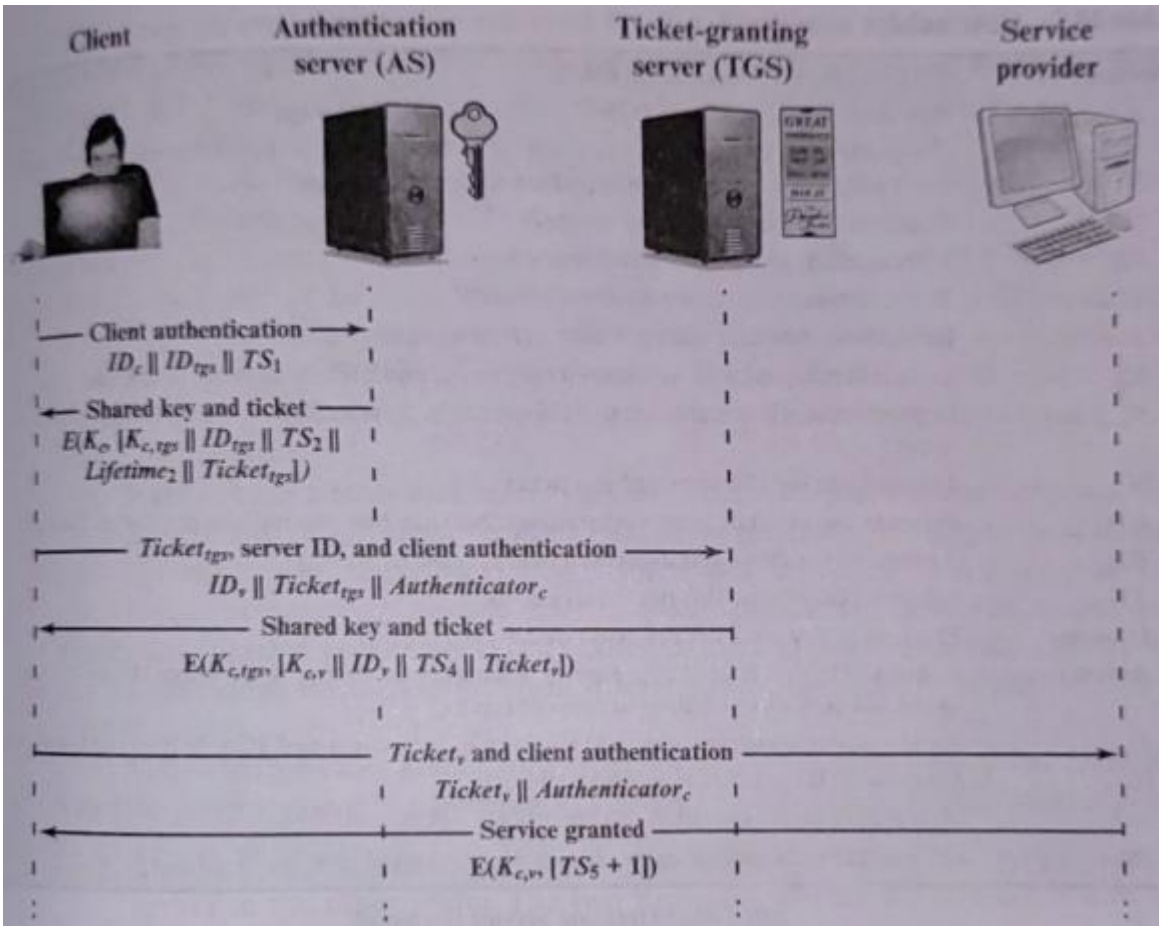


Figure 4.7: Kerberos Exchanges

Table 4.2 summarizes the justification for each of the elements in the Kerberos protocol.

Table 4.2: Rationale for the elements of the Kerberos Version 4 protocol.

Message (1)	Client requests ticket-granting ticket
ID_C	Tells AS identity of user from this client
ID_{tgs}	Tells AS that user requests access to TGS
TS_1	Allows AS to verify that client's clock is synchronized with that of AS
Message (2)	AS returns ticket-granting ticket
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2)
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key
ID_{tgs}	Confirms that this ticket is for the TGS
TS_2	Informs client of time this ticket was issued
$Lifetime_2$	Informs client of the lifetime of this ticket
$Ticket_{tgs}$	Ticket to be used by client to access TGS
	(a) Authentication Service Exchange
Message (3)	Client requests service-granting ticket
ID_V	Tells TGS that user requests access to server V
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS
$Authenticator_c$	Generated by client to validate ticket
Message (4)	TGS returns service-granting ticket
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4)
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key
ID_v	Confirms that this ticket is for server V
TS_4	Informs client of time this ticket was issued
$Ticket_v$	Ticket to be used by client to access server V
$Ticket_{tgs}$	Reusable so that user does not have to reenter password
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket

ID_C	Indicates the rightful owner of this ticket
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_{tgs}	Assures server that it has decrypted ticket properly
TS_2	Informs TGS of time this ticket was issued
$Lifetime_2$	Prevents replay after ticket has expired
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tamperig
ID_C	Must match ID in ticket to authenticate ticket
AD_C	Must match address in ticket to authenticate ticket
TS_3	Informs TGS of time this authenticator was generated
	(b) Ticket-Granting Service Exchange
Message (5)	Client requests service
$Ticket_v$	Assures server that this user has been authenticated by AS
$Authenticator_c$	Generated by client to validate ticket
Message (6)	Optional authentication of server to client
$K_{c,v}$	Assures C that this message is from V
$TS_5 + 1$	Assures C that this is not a replay of an old reply

$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering
$K_{C,V}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket
ID_C	Indicates the rightful owner of this ticket
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_V	Assures server that it has decrypted ticket properly
TS_4	Informs server of time this ticket was issued
$Lifetime_4$	Prevents replay after ticket has expired
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering
ID_C	Must match ID in ticket to authenticate ticket
AD_c	Must match address in ticket to authenticate ticket
TS_5	Informs server of time this authenticator was generated
	(c) Client/Server Authentication Exchange

4.5.6 Kerberos Realms and Multiple Kerberi

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **Kerberos realm**. The concept of *realm* can be explained as follows. A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room. A read-only copy of the Kerberos databasemight also reside on other Kerberos computer systems. However, all changes to the database must be made on the master computer system.

Changing or accessing the contents of a Kerberos database requires the Kerberos master password. A related concept is that of a **Kerberos principal**, which is a service or user that is known to the Kerberos system. Each Kerberos principal is identified by its principal name. Principal names consist of three parts: a service or user name, an instance name, and a realm name

Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical, or does not conform to administrative policy, to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (Figure 4.): A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

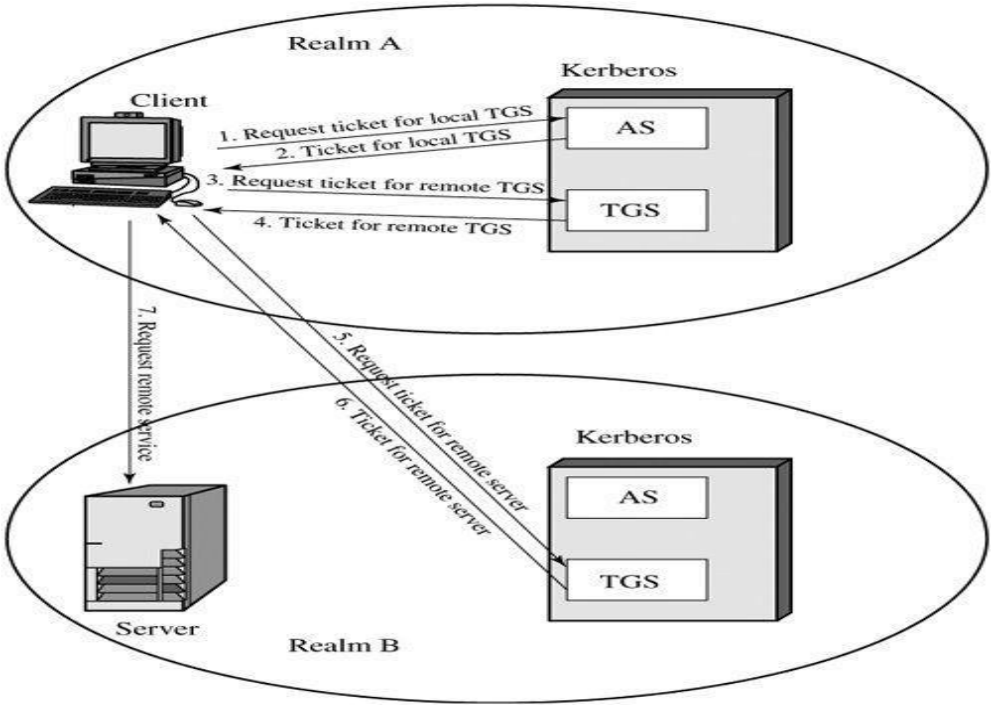


Figure 4.8: Request for service in another realm

The details of the exchanges illustrated in Figure 4.8 are as follows (compare Table 4.1):

(1) C ⟶ AS: $ID_c||ID_{tgs}||TS_1$

(2) AS ⟶ C: $E(K_c, [K_{c,tgs}||ID_{tgs}||TS_2||Lifetime_2||Ticket_{tgs}])$

(3) C ⟶ TGS: $ID_{tgsrem}||Ticket_{tgs}||Authenticator_c$

(4) TGS ⟶ C: $E(K_{c,tgs}, [K_{c,tgsrem}||ID_{tgsrem}||TS_4||Ticket_{tgsrem}])$

(5) C ⟶ TGS_{rem}: $ID_{vrem}||Ticket_{tgsrem}||Authenticator_c$

(6) TGS_{rem} ⟶ C: $E(K_{c,tgsrem}, [K_{c,vrem}||ID_{vrem}||TS_6||Ticket_{vrem}])$

(7) C ⟶ V_{rem}: $Ticket_{vrem}||Authenticator_c$

The ticket presented to the remote server (*Vrem*) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are *N* realms, then there must be *N*(*N* 1)/2 secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

4.5.7 Kerberos Version 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4. To begin, we provide an overview of the changes from version 4 to version 5 and then look at the version 5 protocol.

Differences between Versions 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. Let us briefly summarize the improvements in each area.

Kerberos Version 4 was developed for use within the Project Athena environment and, accordingly, did not fully address the need to be of general purpose. This led to the following **environmental shortcomings:**

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption type identifier so that anyencryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.
2. **Internet protocol dependence:** Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.
3. **Message byte ordering:** In version 4, the sender of a message employs a byteordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.
4. **Ticket lifetime:** Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is 28 x 5 = 1280 minutes, or a little over 21 hours. This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout

execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

5. **Authentication forwarding:** Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.
6. **Interrealm authentication:** In version 4, interoperability among *N* realms requires on the order of *N*2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Apart from these environmental limitations, there are **technical deficiencies** in the version 4 protocol itself. Most of these deficiencies were documented, and version 5 attempts toaddress these. The deficiencies are the following:

1. **Double encryption:** Note in Table 4.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice, once with the secret key of the target server and thenagain with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.
2. **PCBC encryption:** Encryption in version 4 makes use of a nonstandard mode of DES known as propagating cipher block chaining (PCBC). It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode tobe used for encryption. In particular, a checksum or hash code is attached to the message prior to encryption using CBC.
3. **Session keys:** Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the sessionkey may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new subsession key.
4. **Password attacks:** Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password. An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. This is the same type of password attack, with the same kinds of countermeasures being applicable. Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

4.5.8 The Version 5 Authentication Dialogue

Table 4.3 summarizes the basic version 5 dialogue. This is best explained by comparison with version 4 (Table 4.1).

Table 4.3: Summary of Kerberos Version 5 Message Exchanges

(1) C → AS	Options||ID_c||$Realm_c$||ID_{tgs}||$Times$||$Nonce_1$
(2) AS → C	$Realm_c$||ID_C||$Ticket_{tgs}$||E(K_c, [$K_{c,tgs}$||$Times$||$Nonce_1$||$Realm_{tgs}$||ID_{tgs}])
	$Ticket_{tgs}$ = E(K_{tgs}, [$Flags$||$K_{c,tgs}$||$Realm_c$||ID_c||AD_c||$Times$])
(a) Authentication Service Exchange to obtain ticket-granting ticket	
(3) C → TGS	Options||ID_v||$Times$||||$Nonce_2$||$Ticket_{tgs}$||$Authenticator_c$
(4) TGS → C	$Realm_c$||ID_c||$Ticket_v$||E($K_{c,tgs}$, [$K_{c,v}$||$Times$||$Nonce_2$||$Realm_v$||ID_v]) $Ticket_{tgs}$ = E(K_{tgs}, [$Flags$||$K_{C,tgs}$||$Realm_c$||ID_C||AD_C||$Times$])
	$Ticket_v$ = E(K_v, [Flags||$K_{c,v}$||$Realm_c$||ID_C||AD_c||Times]) $Authenticator_c$ = E($K_{c,tgs}$, [ID_C||$Realm_c$||TS_1])
(b) Ticket-Granting Service Exchange to obtain service-granting ticket	
(5) C → V	Options||$Ticket_v$||$Authenticator_c$
(6) V → C	$E_{K_{c,v}}$[TS_2||Subkey||Seq#]
	$Ticket_v$ = E(K_v, [Flags||$K_{c,v}$||$Realm_c$||ID_C||AD_C||Times]) $Authenticator_c$ = E($K_{c,v}$,[ID_C||$Realm_c$||TS_2||$Subkey$||$Seq\#$])
(c) Client/Server Authentication Exchange to obtain service	

First, consider the **authentication service exchange**. Message (1) is a client request for a ticket granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

- **Realm:** Indicates realm of user
- **Options:** Used to request that certain flags be set in the returned ticket
- **Times:** Used by the client to request the following time settings in the ticket:
  - from: the desired start time for the requested ticket
  - till: the requested expiration time for the requested ticket
  - rtime: requested renew-till time
- **Nonce**: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options.

These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the **ticket-granting service exchange** for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2), returning a ticket plus information needed by the client, the latter encrypted with the session key now shared by the client and the TGS.

Finally, for the **client/server authentication exchange**, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields as follows:

- **Subkey**: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket (Kc,v) is used.
- **Sequence number**: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). Thismessage includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the formatof messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys. The subkey field, if present, overrides the subkey field, if present, in message (5). The optional sequence number field specifies the starting sequence number to be used by the client.

4.5.9 Ticket Flags

The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4. Table 4.4 summarizes the flags that may be included in a ticket.

Table 4.4. Kerberos Version 5 Flags

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

The INITIAL flag indicates that this ticket was issued by the AS, not by the TGS. When a client requests a service-granting ticket from the TGS, it presents a ticket-granting ticket obtained from the AS. In version 4, this was the only way to obtain a service-granting ticket. Version 5 provides the additional capability that the client can get a service-granting ticket directly from the AS. The utility of this is as follows: A server, such as a password-changing server, may wish to know that the client's password was recently tested.

The PRE-AUTHENT flag, if set, indicates that when the AS received the initial request [message (1)], it authenticated the client before issuing a ticket. The exact form ofthis preauthentication is left unspecified. As an example, the MIT implementation of version5 has encrypted timestamp preauthentication, enabled by default. When a user wants to get a ticket, it has to send to the AS a preauthentication block containing a random confounder, a version number, and a timestamp, encrypted in the client's password-based key. The AS decrypts the block and will not send a ticket-granting ticket back unless the timestamp in the preauthentication block is within the allowable time skew (time interval to account for clock drift and network delays). Another possibility is the use of a smart card that generates continually changing passwords that are included in the preauthenticated messages.

The passwords generated by the card can be based on a user's password but be transformed by the card so that, in effect, arbitrary passwords are used. This prevents an attack based on easily guessed passwords. If a smart card or similar device was used, this is indicated by the HW-AUTHENT flag. When a ticket has a long lifetime, there is the potential for it to be stolen and used by an opponent for a considerable period. If a short lifetime is used to lessen the threat, then overhead is involved in acquiring new tickets. In the case of a ticket-granting ticket, the client would either have to store the user's secret key, which is clearly risky, or repeatedly ask the user for a password.

A compromise scheme is the use of renewable tickets. A ticket with theRENEWABLE flag set includes two expiration times: one for this specific ticket and one thatis the latest permissible value for an expiration time. A client can have the ticket renewed by presenting it to the TGS with a requested new expiration time. If the new time is within the limit of the latest permissible value, the TGS can issue a new ticket with a new session time and a later specific expiration time. The advantage of this mechanism is that the TGS may refuse to renew a ticket reported as stolen. A client may request that the AS provide a ticket- granting ticket with the MAY-POSTDATE flag set. The client can then use this ticket to request a ticket that is flagged as POSTDATED and INVALID from the TGS. Subsequently, the client may submit the postdated ticket for validation.

This scheme can be useful for running a long batch job on a server that requires a ticket periodically. The client can obtain a number of tickets for this session at once, with spread-out time values. All but the first ticket are initially invalid. When the execution reaches a point in time when a new ticket is required, the client can get the appropriate ticket validated. With this approach, the client does not have to repeatedly use its ticketgranting ticket to obtain a service-granting ticket.

In version 5 it is possible for a server to act as a proxy on behalf of a client, in effect adopting the credentials and privileges of the client to request a service from another server. If a client wishes to use this mechanism, it requests a ticket-granting ticket with the PROXIABLE flag set. When this ticket is presented to the TGS, the TGS is permitted to issue a service-granting ticket with a different network address; this latter ticket will have its PROXY flag set. An application receiving such a ticket may accept it or require additional authentication to provide an audit trail.

The proxy concept is a limited case of the more powerful forwarding procedure. If a ticket is set with the FORWARDABLE flag, a TGS can issue to the requestor a ticket- granting ticket with a different network address and the FORWARDED flag set. This ticket

can then be presented to a remote TGS. This capability allows a client to gain access to a server on another realm without requiring that each Kerberos maintain a secret key with Kerberos servers in every other realm. For example, realms could be structured hierarchically. Then a client could walk up the tree to a common node and then back down to reach a target realm. Each step of the walk would involve forwarding a ticket-granting ticket to the next TGS in the path.

4.6 Remote User-authentication using asymmetric encryption

Mutual authentication

we presented one approach to the use of public-key encryption for the purpose of session key distribution. This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

A protocol using timestamps is provided:

1. $A \longrightarrow AS: \quad ID_A||ID_B$
2. $AS \longrightarrow A: \quad E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$
3. $A \longrightarrow B: \quad E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$
   $||E(PU_b, E(PR_a, [K_s||T]))$

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposureby the AS. The timestamps protect against replays of compromised keys.

This protocol is compact but, as before, requires synchronization of clocks. Another approach, proposed by Woo and Lam, makes use of nonces.

The protocol consists of the following steps:

1. $A \longrightarrow KDC: \quad ID_A||ID_B$
2. $KDC \longrightarrow A: \quad E(PR_{auth}, [ID_B||PU_b])$
3. $A \longrightarrow B: \quad E(PU_b, [N_a||ID_A])$
4. $B \longrightarrow KDC: \quad ID_A||ID_B||E(PU_{auth}, N_a)$
5. $KDC \longrightarrow B: \quad E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_B]))$
6. $B \longrightarrow A: \quad E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_B)||N_b]))$
7. $A \longrightarrow B: \quad E(K_s, N_b)$

In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A

informs B of its desire to communicate and sends a nonce *Na* (step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information {*Na, Ks, IDB*}. This information basically says that *Ks* is a secret key generated by the KDC on behalf of B and tied to *Na*; the binding of *Ks* and *Na* will assure A that *Ks* is fresh. This triple is encrypted, using the KDC's private key, to allow B to verify that the tripleis in fact from the KDC. It is also encrypted using B's public key, so that no other entity may use the triple in an attempt to establish a fraudulent connection with A. In step 6, the triple {*Na, Ks, IDB*}, still encrypted with the KDC's private key, is relayed to A, together with a nonce *Nb* generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key *Ks* and uses it to encrypt *Nb* and return it to B. This last message assures B of A's knowledge of the session key.

This seems to be a secure protocol that takes into account the various attacks. However, the authors themselves spotted a flaw and submitted a revised version of the algorithm

1. A ⟶ KDC: $ID_A||ID_B$
2. KDC ⟶ A: $E(PR_{auth}, [ID_B||PU_b])$
3. A ⟶ B: $E(PU_b, [N_a||ID_A])$
4. B ⟶ KDC: $ID_A||ID_B||E(PU_{auth}, N_a)$
5. KDC ⟶ B: $E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_A||ID_B])$
6. B ⟶ A: $E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_A||ID_B)||N_b]))$
7. A ⟶ B: $E(K_s, N_b)$

The identifier of A, *IDA*, is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key *Ks* to the identities of the two parties that will be engaged in the session.

This inclusion of *IDA* accounts for the fact that the nonce value *Na* is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair {*IDA, Na*} that uniquely identifies the connection request of A.

In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

4.6.1 One-Way Authentication

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality, authentication, or both. These approaches require that either the sender know the recipient's public key (confidentiality) or the recipient know the sender's public key (authentication) or both (confidentiality plus authentication). In addition, the public-key algorithm must beapplied once or twice to what may be a long message.

If confidentiality is the primary concern, then the following may be more efficient:

$$A \longrightarrow B: E(PU_b, K_s)||E(K_s, M)$$

In this case, the message is encrypted with a one-time secret key. A also encrypts this one- time key with B's public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B's public key.
If authentication is the primary concern, then a digital signature may suffice, as was illustrated

$$A \longrightarrow B: M||E(PR_a, H(M))$$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the e-mail system. Eventually, the message will get delivered to Alice's mailbox. But suppose that Max has heard of Bob's idea and gains access to the mail queue before delivery. He finds Bob's message, strips off his signature, appends his, and requeues the message to be delivered to Alice. Max gets credit for Bob's idea.

To counter such a scheme, both the message and signature can be encrypted with the recipient's public key:

$$A \longrightarrow B: E(PU_b, [M||E(PR_a, H(M))])$$

The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate. Now we have

$$A \longrightarrow B: M||E(PR_a, H(M))||E(PR_{as}, [T||ID_A||PU_a])$$

In addition to the message, A sends B the signature, encrypted with A's private key, and A's certificate, encrypted with the private key of the authentication server. The recipient of the message first uses the certificate to obtain the sender's public key and verify that it is authentic and then uses the public key to verify the message itself. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted with a one-time secret key; the secret key is also transmitted,encrypted with B's public key.

4.7 Electronic Mail Security

4.7.1 Pretty Good Privacy

PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, Zimmermann has done the following:
1. Selected the best available cryptographic algorithms as building blocks
2. Integrated these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands
3. Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line)

4. Entered into an agreement with a company (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth:

1. It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.
2. It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES forsymmetric encryption; and SHA-1 for hash coding.
3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
4. It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of "the establishment," this makes PGP attractive.
5. PGP is now on an Internet standards track (RFC 3156). Nevertheless, PGP still has an aura of an antiestablishment endeavor.

We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys are created and stored. Then, we address the vital issue of public key management.

4.7.2 Notation

Most of the notation used in this chapter has been used before, but a few terms are new. It is perhaps best to summarize those at the beginning. The following symbols are used:

K_s =session key used in symmetric encryption scheme

PR_a =private key of user A, used in public-key encryption scheme

PU_a =public key of user A, used in public-key encryption scheme

EP = public-key encryption

DP = public-key decryption

EC = symmetric encryption

DC = symmetric decryption

H = hash function

|| = concatenation

Z = compression using ZIP algorithm

R64 = conversion to radix 64 ASCII format

The PGP documentation often uses the term *secret key* to refer to a key paired with a public key in a public-key encryption scheme. As was mentioned earlier, this practice risks confusion with a secret key used for symmetric encryption. Hence, we will use the term *private key* instead.

4.7.3 Operational Description

The actual operation of PGP, as opposed to the management of keys, consists of five services: authentication, confidentiality, compression, e-mail compatibility, and segmentation (Table 4.5). We examine each of these in turn.

Table 4.5: Summary of PGP Services

Function	Algorithms	Used Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key and included with the message.
Compression	ZIP	A message may be compressed, for storage or transmission, using ZIP.
Email compatibility	Radix 64 conversion	To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix 64 conversion.
Segmentation		To accommodate maximum message size limitations, PGP performs segmentation and reassembly.

Authentication

Figure 4.a illustrates the digital signature service provided by PGP. This is the digital signature scheme. The sequence is as follows:

1. The sender creates a message.
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4. The receiver uses RSA with the sender's public key to decrypt and recover the hash code.
5. The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

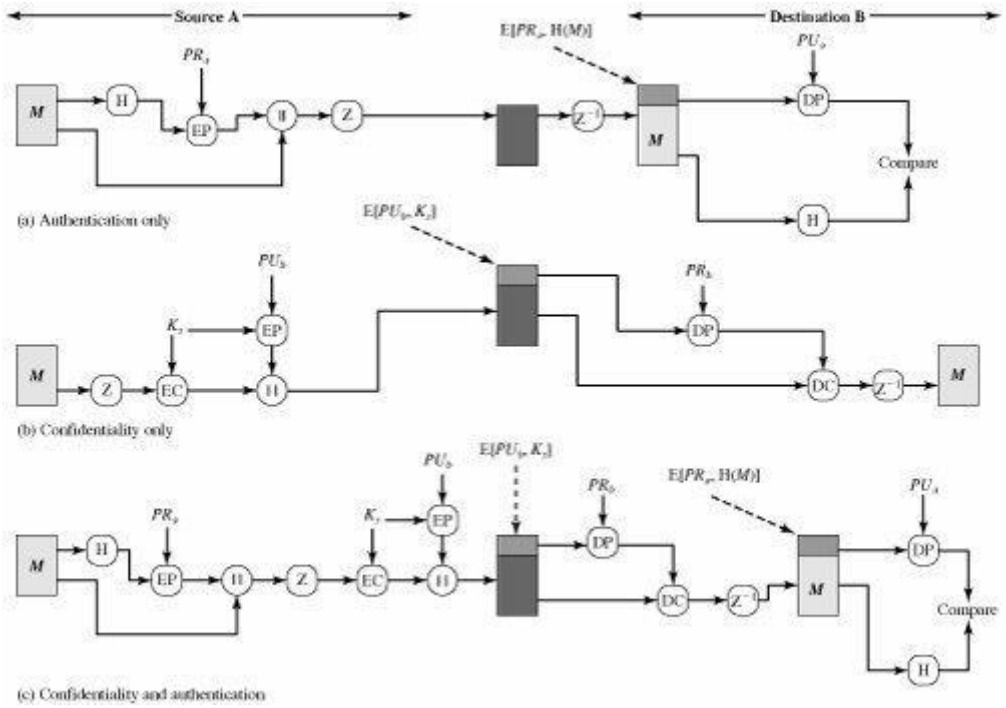


Figure 4. : PGP Cryptographic Functions

Confidentiality

Another basic service provided by PGP is confidentiality, which is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the symmetric encryption algorithm CAST-128 may be used. Alternatively, IDEA or 3DES may be used. The 64-bit cipher feedback (CFB) mode is used. which can be described as follows:

1. The sender generates a message and a random 128-bit number to be used as a session key for this message only.
2. The message is encrypted, using CAST-128 (or IDEA or 3DES) with the session key.
3. The session key is encrypted with RSA, using the recipient's public key, and is prepended to the message.
4. The receiver uses RSA with its private key to decrypt and recover the session key.
5. The session key is used to decrypt the message.

Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

The placement of the compression algorithm, indicated by Z for compression and Z-1 for decompression is critical:

1.The signature is generated before compression for two reasons:

a. It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.

b. Even if one were willing to generate dynamically a recompressed message for verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and, as a result, produce different compressed forms. However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version. Applying the hash function and

signature after compression would constrain all PGP implementations to the same version of the compression algorithm.

2.Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext,cryptanalysis is more difficult.

E-mail Compatibility

When PGP is used, at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key). Thus, part or all of the resulting block consists of a stream of arbitrary 8-bit octets. However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters.

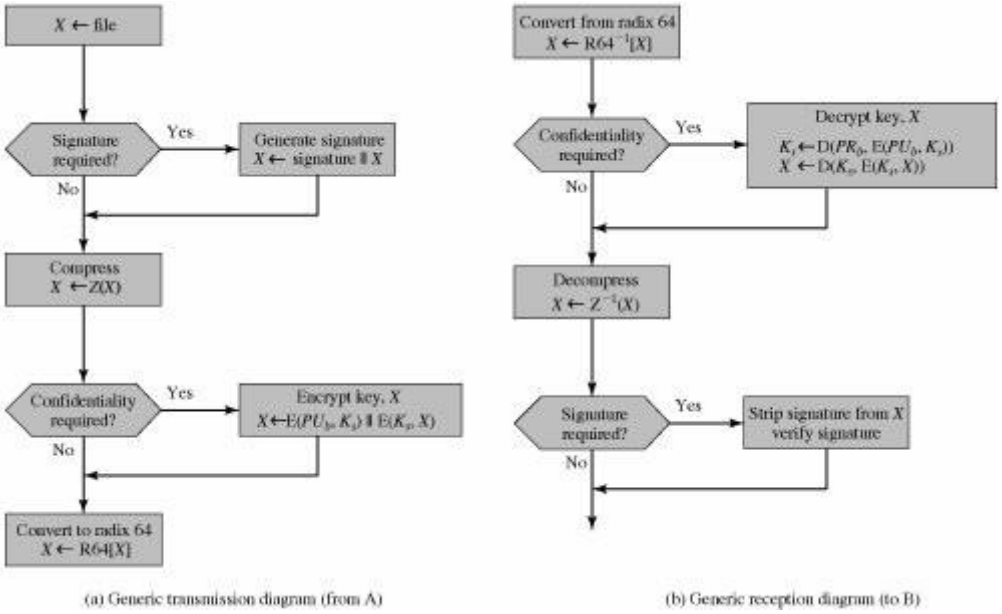


Figure 4. : Transmission and Reception of PGP Messages

4.8 S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet email format standard, based on technology from RSA Data Security. Although both PGP and S/MIME are on an IETF standards track, 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 e-mail security for many users. S/MIME is definedin a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851.

To understand S/MIME, we need first to have a general understanding of the underlying e-mail format that it uses, namely MIME. But to understand the significance of MIME, we need to go back to the traditional e-mail format standard, RFC 822, which is still in common use. Accordingly, this section first provides an introduction to these two earlier standards and then moves on to a discussion of S/MIME.

Multipurpose Internet Mail Extensions

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other

mail transfer protocol and RFC 822 for electronic mail. [RODR02] lists the following limitations of the SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/UUdecode scheme. However, none of these is a standard or even a de facto standard.
2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:
   - Deletion, addition, or reordering of carriage return and linefeed
   - Truncating or wrapping lines longer than 76 characters
   - Removal of trailing white space (tab and space characters)
   - Padding of lines in a message to the same length
   - Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

Overview

The MIME specification includes the following elements:

1. Five new message header fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

The five header fields defined in MIME are as follows:

● **MIME-Version:** Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.

● **Content-Type:** Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.

● **Content-Transfer-Encoding:** Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.

● **Content-ID:** Used to identify MIME entities uniquely in multiple contexts.

● **Content-Description:** A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

Any or all of these fields may appear in a normal RFC 822 header. A compliant implementation must support the MIME-Version, Content-Type, and Content-Transfer- Encoding fields; the Content-ID and Content-Description fields are optional and may be ignored by the recipient implementation.

MIME Content Types

The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Below Table lists the content types specified in RFC 2046. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.

Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/ rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript.
	octet-stream	General binary data consisting of 8-bit bytes.

MIME Transfer Encodings

The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.

base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability. We then look in more detail at this capability by examining message formats and message preparation.

Functions

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.
● **Signed data:** A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.
● **Clear-signed data:** As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the 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.

Cryptographic Algorithms

Cryptographic Algorithms Used in S/MIME

Function	Requirement
Create a message digest to be used in forming a digital signature. Encrypt message digest to form digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility. Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.
Encrypt message for transmission with one-time session key.	Sending and receiving agents MUST support encryption with triple DES Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.
Create a message authentication code	Receiving agents MUST support HMAC with SHA-1. Receiving agents SHOULD support HMAC with SHA-1.

S/MIME Messages

S/MIME makes use of a number of new MIME content types, which are shown in Table. All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

S/MIME Content Types

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs 7-mime	signedData	A signed S/MIME entity.
	pkcs 7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs 7-mime	degenerate signedData	An entity containing only public- key certificates.
	pkcs 7-mime	CompressedData	A compressed S/MIME entity
	pkcs 7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

S/MIME Certificate Processing

S/MIME uses public-key certificates that conform to version 3 of X.509. The key management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust. As with the PGP model, S/MIME managers and/or users must configure each client with a list of trusted keys and with certificaterevocation lists. That is, the responsibility is local for maintaining the certificates needed to verify incoming signatures and to encrypt outgoing messages. On the other hand, the certificates are signed by certification authorities.

User Agent Role

An S/MIME user has several key-management functions to perform:

● **Key generation:** The user of some related administrative utility (e.g., one associated with LAN management) MUST be capable of generating separate Diffie-Hellman and DSS key pairs and SHOULD be capable of generating RSA key pairs. Each key pair MUST be generated from a good source of nondeterministic random input and be protected in a secure fashion. A user agent SHOULD generate RSA key pairs with a length in the range of 768 to 1024 bits and MUST NOT generate a length of less than 512 bits.

● **Registration:** A user's public key must be registered with a certification authority in order to receive an X.509 public-key certificate.

● **Certificate storage and retrieval:** A user requires access to a local list of certificates in order to verify incoming signatures and to encrypt outgoing messages. Such a list could be maintained by the user or by some local administrative entity on behalf of a number of users.

VeriSign Certificates

There are several companies that provide certification authority (CA) services. For example, Nortel has designed an enterprise CA solution and can provide S/MIME support within an organization. There are a number of Internet-based CAs, including VeriSign, GTE, and the U.S. Postal Service. Of these, the most widely used is the VeriSign CA service, a brief description of which we now provide. VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications. VeriSign issues X.509 certificates with the product name VeriSign Digital ID. As of early 1998, over 35,000commercial Web sites were using VeriSign Server Digital IDs, and over a million consumer Digital IDs had been issued to users of Netscape and Microsoft browsers.

The information contained in a Digital ID depends on the type of Digital ID and its use. At a minimum, each Digital ID contains

● Owner's public key

- Owner's name or alias
- Expiration date of the Digital ID
- Serial number of the Digital ID
- Name of the certification authority that issued the Digital ID
- Digital signature of the certification authority that issued the Digital ID

Digital IDs can also contain other user-supplied information, including

- Address
- E-mail address
- Basic registration information (country, zip code, age, and gender)

VeriSign provides three levels, or classes, of security for public-key certificates, as summarized in Table . A user requests a certificate online at VeriSign's Web site or other participating Web sites. Class 1 and Class 2 requests are processed on line, and in most cases take only a few seconds to approve.

Briefly, the following procedures are used:

- For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.
- For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with a consumer database in addition to performing all of the checking associated with a Class 1 Digital ID. Finally, confirmation is sent to the specified postal address alerting the user that a Digital ID has been issued in his or her name.
- For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

VeriSign Public-Key Certificate Classes

	Summary of Confirmation of Identity	IA Private Key Protection	Certificate Applicant and Subscriber Private Key Protection	Applications implemented or contemplated by Users
Class 1	Automated unambiguous name and e-mail address search	PCA: trustworthy hardware; CA: trust-worthy software or trustworthy hardware	Encryption software (PIN protected) recommended but not required	Web-browsing and certain e-mail usage
Class 2	Same as Class 1, plus automated enrollment information check plus automated address check	PCA and CA: trustworthy hardware	Encryption software (PIN protected) required	Individual and intra and inter-company E-mail, online subscriptions, password replacement, and software validation
Class 3	Same as Class 1, plus personal presence and ID documents plus Class 2 automated ID check for individuals; business records (or filings) for organizations	PCA and CA: trustworthy hardware	Encryption software (PIN protected) required; hardware token recommended but not required	E-banking, corp, database access, personal banking, membership-based online services, content integrity services, e-commerce server, software validation; authentication of LRAAs; and strong encryption for certain servers

IA Issuing Authority

CA Certification Authority

PCA VeriSign public primary certification authority

PIN Personal Identification Number

LRAA Local Registration Authority Administrator

Enhanced Security Services

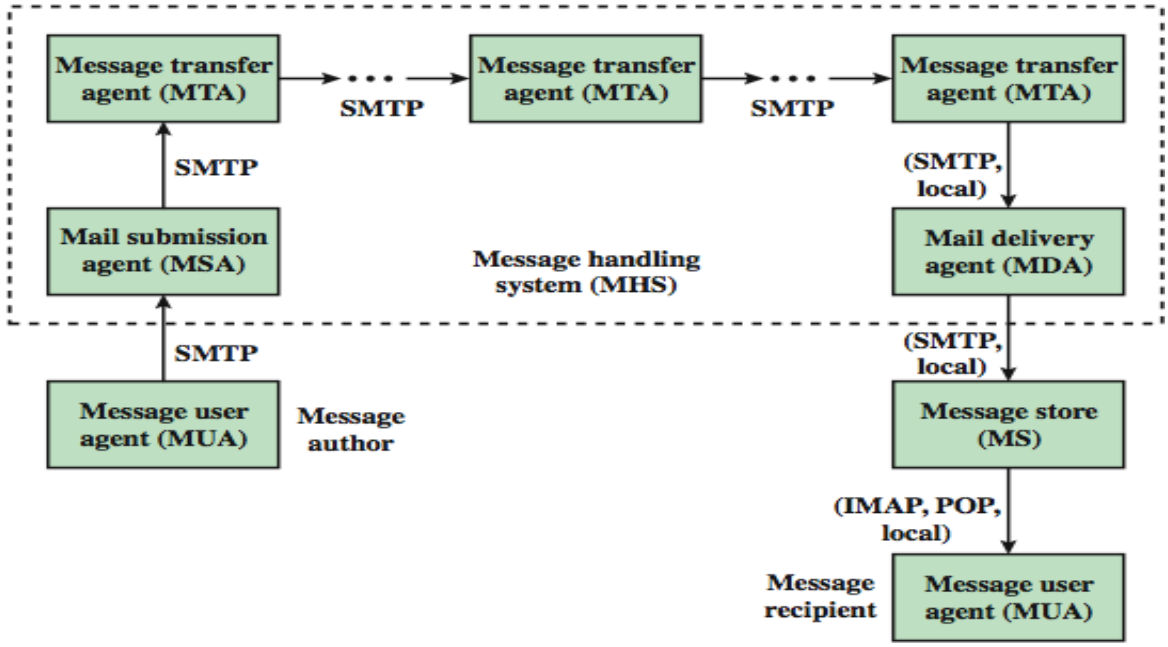
As of this writing, three enhanced security services have been proposed in an Internet draft. The details of these may change, and additional services may be added. The three services are as follows:

- **Signed receipts:** A signed receipt may be requested in a SignedData object. Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message. In essence, the recipient signs the entire original message plus original (sender's) signature and appends the new signature to form a new S/MIME message.
- **Security labels:** A security label may be included in the authenticated attributes of a SignedData object. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. The labels may be used for access control, by indicating which users are permitted access to an object. Other uses include priority (secret, confidential, restricted, and so on) or role based, describing which kind of people can see the information (e.g., patient's health-care team, medical billing agents, etc.).
- **Secure mailing lists:** When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA, with encryption performed using the MLA's public key.

Domain Keys Identified Mail

- a specification for cryptographically signing email messages
- so signing domain claims responsibility
- recipients / agents can verify signature
- proposed Internet Standard RFC 4871
- has been widely adopted

Internet Mail Architecture

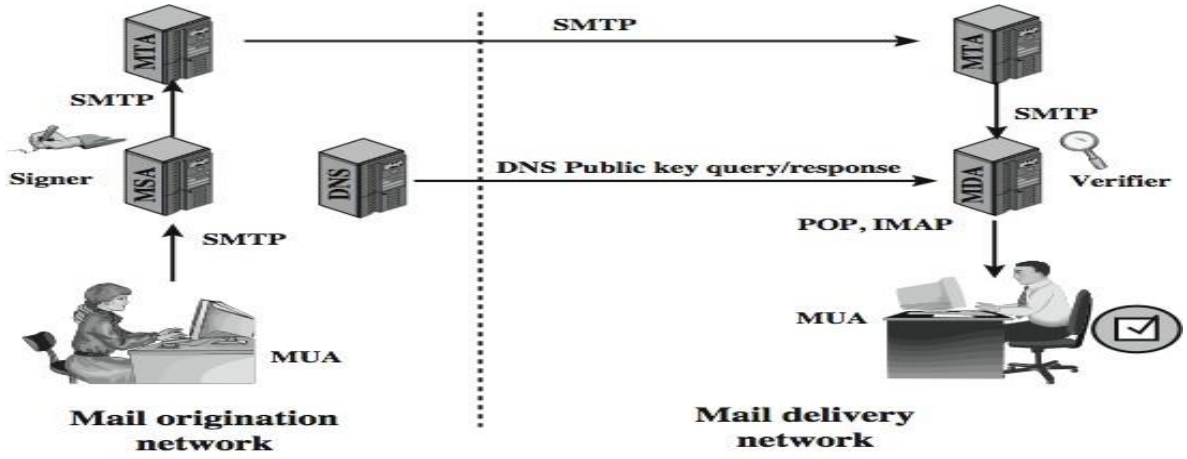


Email Threats

- describes the problem space in terms of:
  - range: low end, spammers, fraudsters
  - capabilities in terms of where submitted, signed, volume, routing naming etc
  - outside located attackers

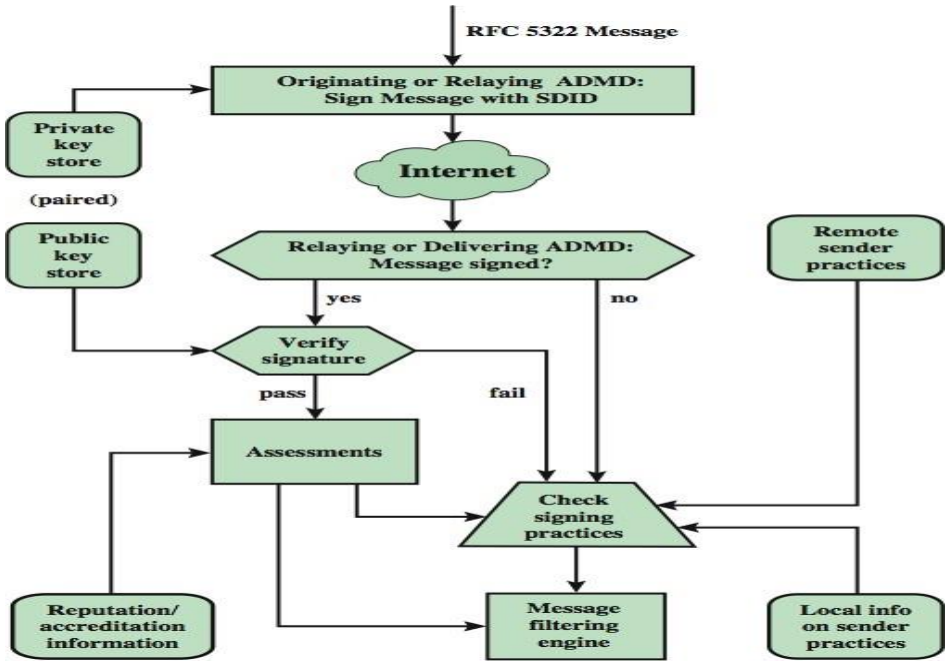
DKIM Strategy

- transparent to user
  - MSA sign
  - MDA verify
- for pragmatic reasons



DNS = domain name system
MDA = mail delivery agent
MSA = mail submission agent
MTA = message transfer agent
MUA = message user agent

DCIM Functional Flow



Module-5:

IP SECURITY

Key Points

- IP security (IPSec) is a capability that can be added to either current version of the Internet Protocol (IPv4 or IPv6), by means of additional headers.
- IPSec encompasses three functional areas: authentication, confidentiality, and key management.
- Authentication makes use of the HMAC message authentication code. Authentication can be applied to the entire original IP packet (tunnel mode) or to all of the packet except for the IP header (transport mode).
- Confidentiality is provided by an encryption format known as encapsulating security payload. Both tunnel and transport modes can be accommodated.
- IPSec defines a number of techniques for key management.

The Internet community has developed application-specific security mechanisms in a number of application areas, including electronic mail (S/MIME, PGP), client/server (Kerberos), Web access (Secure Sockets Layer), and others. However, users have some security concerns that cut across protocol layers.
For example, an enterprise can run a secure, private TCP/IP network by disallowing links to untrusted sites, encrypting packets that leave the premises, and authenticating packets that enter the premises.

By implementing security at the IP level, an organization can ensure secure networking not only for applications that have security mechanisms but also for the many security-ignorant applications.

IP-level security encompasses **three functional areas: authentication, confidentiality, and key management**. The authentication mechanism assures that a received packet was, in fact, transmitted by the party identified as the source in the packet header. In addition, this mechanism assures that the packet has not been altered in transit. The confidentiality facility enables communicating nodes to encrypt messages to prevent eavesdropping by third parties. The key management facility is concerned with the secure exchange of keys.

5.1 IP Security Overview

In 1994, the Internet Architecture Board(IAB) issued a report titled " Security in the Internet Architecture"(RFC 1636). The report identified key areas for security mechanisms. Among these were the need to secure the network infrastructure from unauthorized monitoring and control of network traffic and the need to secure end-user-to-end-user traffic using authentication and encryption mechanism.

To provide security, the IAB included authentication and encryption as necessary features in the next generation IP, which has been issued as IPv6. Fortunately, these security capabilities were designed to be usable both with the current IPv4 and the future IPv6. This means that vendors can begin offering these features now, and many vendors do now have some IPSec capability in their products.

5.1.1 Applications of IPSec

IPSec provides the capability to secure communications across a LAN, across private and public WANs, and across the Internet. Examples of its use include the following:

- **Secure branch office connectivity over the Internet:** A company can build a secure virtual private network over the Internet or over a public WAN. This enables a business to rely heavily on the Internet and reduce its need for private networks, saving costs and network management overhead.
- **Secure remote access over the Internet:** An end user whose system is equipped with IP security protocols can make a local call to an Internet service provider (ISP) and gain secure access to a company network. This reduces the cost of toll charges for traveling employees and telecommuters.
- **Establishing extranet and intranet connectivity with partners:** IPSec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism.
- **Enhancing electronic commerce security:** Even though some Web and electronic commerce applications have built-in security protocols, the use of IPSec enhances that security.

The principal feature of IPSec that enables it to support these varied applications is that it can encrypt and/or authenticate *all* traffic at the IP level. Thus, all distributed applications, including remote logon, client/server, e-mail, file transfer, Web access, and so on, can be secured.

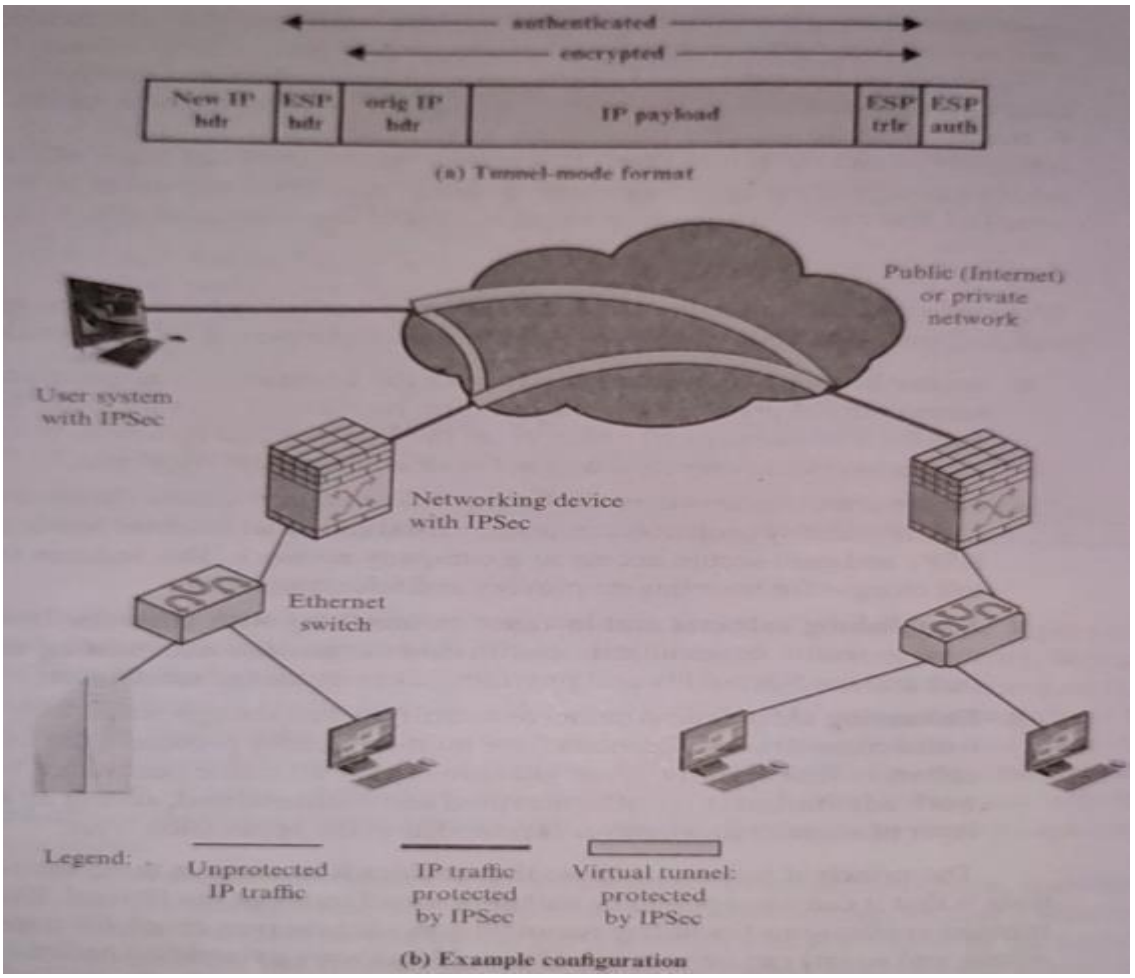


Figure 5.1: An IPSec VPN Scenario

Figure 5.1a shows a simplified packet format for an IPSec option known as tunnel mode, described subsequently. Tunnel mode makes use of an IPSec function, a combined authentication/encryption function called Encapsulating Security Payload (ESP), and a key exchange function. For VPNs, both authentication and encryption are generally desired, because it is important both to

1. Assure that unauthorized users do not penetrate the VPN.
2. Assure that eavesdroppers on the Internet cannot read messages sent over the VPN.

Figure 5.1b is a typical scenario of IPSec usage. An organization maintains LANs at dispersed locations. Nonsecure IP traffic is conducted on each LAN. For traffic offsite, through some sort of private or public WAN, IPSec protocols are used. These protocols operate in networking devices, such as a router or firewall, that connect each LAN to the outside world. The IPSec networking device will typically encrypt and compress all traffic going into the WAN, and decrypt and decompress traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN. Secure transmission isalso possible with individual users who dial into the WAN. Such user workstations must implement the IPSec protocols to provide security.

5.1.2 Benefits of IPSec

- When IPSec is implemented in a firewall or router, it provides strong security that can be applied to all traffic crossing the perimeter. Traffic within a company or workgroup does not incur the overhead of security-related processing.
- IPSec in a firewall is resistant to bypass if all traffic from the outside must use IP, and the firewall is the only means of entrance from the Internet into the organization.
- IPSec is below the transport layer (TCP, UDP) and so is transparent to applications. There is no need to change software on a user or server system when IPSec is implemented in the firewall or router. Even if IPSec is implemented in end systems, upper-layer software, including applications, is not affected.
- IPSec can be transparent to end users. There is no need to train users on security mechanisms, issue keying material on a per-user basis, or revoke keying material when users leave the organization.
- IPSec can provide security for individual users if needed. This is useful for offsite workers and for setting up a secure virtual subnetwork within an organization for sensitive applications.

5.1.3 Routing Applications

In addition to supporting end users and protecting premises systems and networks, IPSec can play a vital role in the routing architecture required for internetworking. IPSec can assure that

● A router advertisement (a new router advertises its presence) comes from an authorized router

● A neighbor advertisement (a router seeks to establish or maintain a neighbor relationship with a router in another routing domain) comes from an authorized router.

● A redirect message comes from the router to which the initial packet was sent.

● A routing update is not forged.

Without such security measures, an opponent can disrupt communications or divert some traffic. Routing protocols such as OSPF should be run on top of security associations between routers that are defined by IPSec.

5.1.4 IPSec Documents

The IPSec specification consists of numerous documents. The most important of these, issued in November of 1998, are RFCs 2401, 2402, 2406, and 2408:

- RFC 2401: An overview of a security architecture
- RFC 2402: Description of a packet authentication extension to IPv4 and IPv6
- RFC 2406: Description of a packet encryption extension to IPv4 and IPv6
- RFC 2408: Specification of key management capabilities

Support for these features is mandatory for IPv6 and optional for IPv4. In both cases, the security features are implemented as extension headers that follow the main IP header. The extension header for authentication is known as the Authentication header; that for encryption is known as the Encapsulating Security Payload (ESP) header.

In addition to these four RFCs, a number of additional drafts have been published by the IP Security Protocol Working Group set up by the IETF. The documents are divided into seven groups, as depicted in Figure 5.2 (RFC 2401):

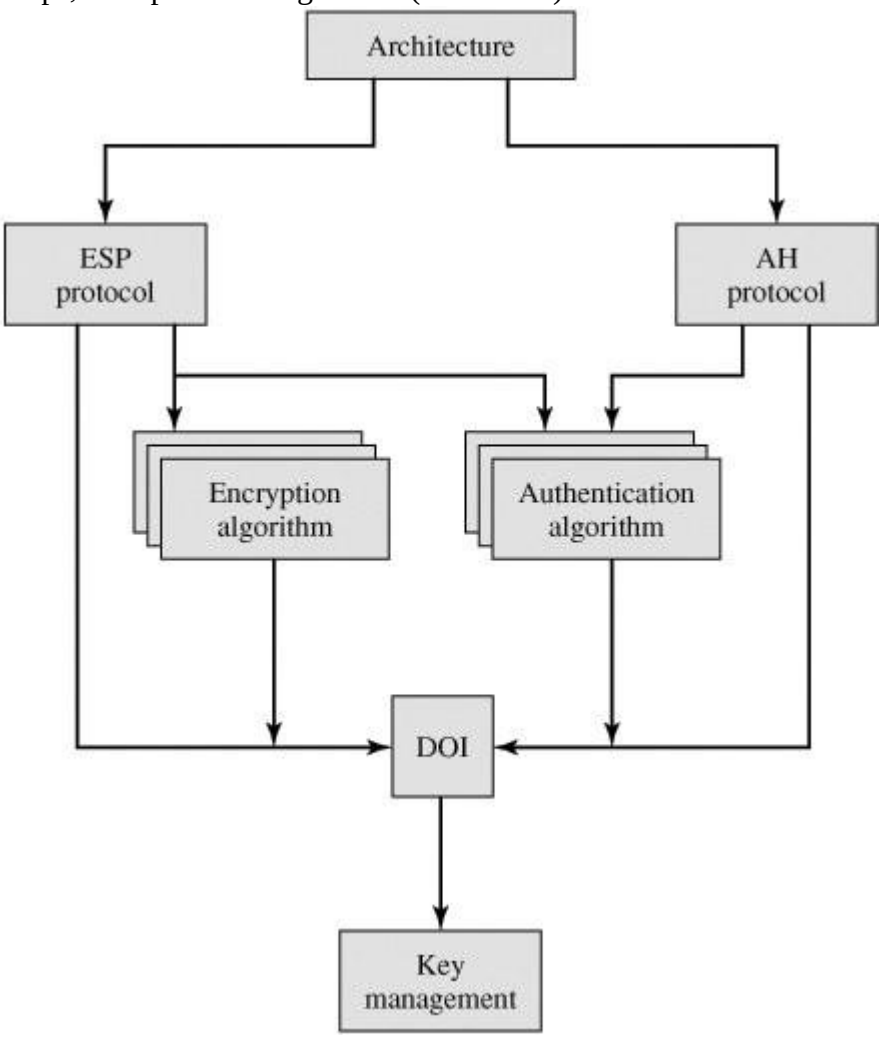


Figure 5.2: IPSec Document Overview

- **Architecture:** Covers the general concepts, security requirements, definitions, and mechanisms defining IPSec technology.
- **Encapsulating Security Payload (ESP):** Covers the packet format and general issues related to the use of the ESP for packet encryption and, optionally, authentication.
- **Authentication Header (AH):** Covers the packet format and general issues related to the use of AH for packet authentication.

- **Encryption Algorithm:** A set of documents that describe how various encryption algorithms are used for ESP.
- **Authentication Algorithm:** A set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP.
- **Key Management:** Documents that describe key management schemes.
- **Domain of Interpretation (DOI):** Contains values needed for the other documents to relate to each other. These include identifiers for approved encryption and authentication algorithms, as well as operational parameters such as key lifetime.
- **Internet Key Exchange(IKE):** this is a collection of documents describing the key management schemes for use with IPSec. The main specification is RFC 7296, Internet Key Exchange(IKEv2) protocol, but there are a number of related RFCs.
- **Cryptography algorithms:** this categories encompasses a large set of documents that define and describe cryptographic algorithms for encryption, message authentication, pseudorandom function(PRFs), and cryptographic key exchange.
- **Other:** there are a variety of other IPSec related RFCs including those dealing with security policy and management information base (MIB) content.

5.1.5 IPSec Services

IPSec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. Two protocols are used to provide security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined encryption/authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP). The services are

- Access control
- Connectionless integrity
- Data origin authentication
- Rejection of replayed packets (a form of partial sequence integrity)
- Confidentiality (encryption)
- Limited traffic flow confidentiality

Table 5.1 shows which services are provided by the AH and ESP protocols. For ESP, there are two cases: with and without the authentication option. Both AH and ESP are vehicles for access control, based on the distribution of cryptographic keys and the management of traffic flows relative to these security protocols.

Table 5.1: IPSec Services

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✔	✔	✔
Connectionless integrity	✔		✔
Data origin authentication	✔		✔
Rejection of replayed packets	✔	✔	✔
Confidentiality		✔	✔
Limited traffic flow confidentiality		✔	✔

5.1.6 Transport and Tunnel Modes

Both AH and ESP support two modes of use: transport and tunnel mode. The operation of these two modes is best understood in the context of a description of AH and ESP, respectively. Here we provide a brief overview.

Transport Mode

Transport mode provides protection primarily for upper-layer protocols. That is, transport mode protection extends to the payload of an IP packet. Examples include a TCP or UDP segment or an ICMP packet, all of which operate directly above IP in a host protocol stack. Typically, transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations). When a host runs AH or ESP over IPv4, the payload is the data that normally follow the IP header. For IPv6, the payload is the data that normally follow both the IP header and any IPv6 extensions headers that are present, with the possible exception of the destination options header, which may be included in the protection.

ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header. AH in transport mode authenticates the IP payload and selected portions of the IP header.

Tunnel Mode

Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of new "outer" IP packet with a new outer IP header. The entire original, or inner, packet travels through a "tunnel" from one point of an IP network to another; no routers along the way are able to examine the inner IP header. Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security. Tunnel mode is used when one or both ends of an SA are a security gateway, suchas a firewall or router that implements IPSec. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementingIPSec. The unprotected packets generated by such hosts are tunneled through external networks by tunnel mode SAs set up by the IPSec software in the firewall or secure router at the boundary of the local network.

Here is an example of how tunnel mode IPSec operates. Host A on a networkgenerates an IP packet with the destination address of host B on another network. This packetis routed from the originating host to a firewall or secure router at the boundary of A's network. The firewall filters all outgoing packets to determine the need for IPSec processing. If this packet from A to B requires IPSec, the firewall performs IPSec processing and encapsulates the packet with an outer IP header. The source IP address of this outer IP packet is this firewall, and the destination address may be a firewall that forms the boundary to B's local network. This packet is now routed to B's firewall, with intermediate routers examining only the outer IP header. At B's firewall, the outer IP header is stripped off, and the inner packet is delivered to B.

ESP in tunnel mode encrypts and optionally authenticates the entire inner IP packet, including the inner IP header. AH in tunnel mode authenticates the entire inner IP packet and selected portions of the outer IP header.

Table 5.2: summarizes transport and tunnel mode functionality.

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

5.2 IP Security Policy

Fundamental to the operation of IPSec is the concept of a security policy applied to each IP packet that transmit from a source to a destination. IPsec policy is determined primarily by the interaction of two databases, the security association database (SAD) and the security policy database (SPD).

5.2.1 Security Associations

A key concept that appears in both the authentication and confidentiality mechanisms for IP is the security association (SA). An association is a one-way relationship between a sender and a receiver that affords security services to the traffic carried on it. If a peer relationship is needed, for two-way secure exchange, then two security associations are required. Security services are afforded to an SA for the use of AH or ESP, but not both.

A security association is uniquely identified by three parameters:

- **Security Parameters Index (SPI):** A bit string assigned to this SA and having local significance only. The SPI 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:** Currently, only unicast addresses are allowed; this is the address of the destination endpoint of the SA, which may be an end user system or a network system such as a firewall or router.
- **Security Protocol Identifier:** This indicates whether the association is an AH or ESP security association.

Hence, in any IP packet, the security association is uniquely identified by the Destination Address in the IPv4 or IPv6 header and the SPI in the enclosed extension header (AH or ESP).

Figure 5.3 illustrates the relevant relationships.

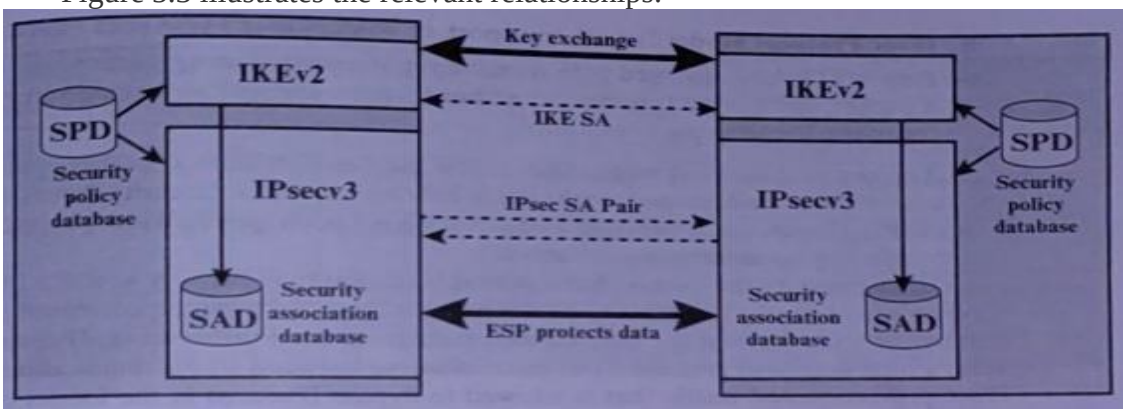


Figure 5.3: IPSec Architecture

5.2.2 Security Association Database

In each IPSec implementation, there is a nominal Security Association Database that defines the parameters associated with each SA. A security association is normally defined by the following parameters in an SAD entry:

- **Security Parameter Index:** A 32 bit value selected by the receiving end of an SA to uniquely identify the SA. In an SAD entry for an outbound SA, the SPI is used to construct the packet's AH or ESP header. In an SAD entry for an inbound SA, the SPI is used to map traffic to the appropriate SA.
- **Sequence Number Counter:** A 32-bit value used to generate the Sequence Number field in AH or ESP headers.
- **Sequence Counter Overflow:** A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA.
- **Anti-Replay Window:** Used to determine whether an inbound AH or ESP packet is a replay.
- **AH Information:** Authentication algorithm, keys, key lifetimes, and related parameters being used with AH.
- **ESP Information:** Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP.
- **Lifetime of This Security Association:** A time interval or byte count after which an SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur.
- **IPsec Protocol Mode:** Tunnel, transport, or wildcard
- **Path MTU:** any observed path maximum transmission unit and aging variables.

The key management mechanism that is used to distribute keys is coupled to the authentication and privacy mechanisms only by way of the Security Parameters Index. Hence, authentication and privacy have been specified independent of any specific keymanagement mechanism.

IPSec provides the user with considerable flexibility in the way in which IPSec services are applied to IP traffic.SAs can be combined in a number of ways to yield the desired user configuration. Furthermore, IPSec provides a high degree of granularity in discriminating between traffic that is afforded IPSec protection and traffic that is allowed to bypass IPSec, in the former case relating IP traffic to specific SAs.

5.2.3 Security Policy Database

The means by which IP traffic is related to specific SAs (or no SA in the case of traffic allowed to bypass IPSec) is the nominal Security Policy Database (SPD). In its simplest form,an SPD contains entries, each of which defines a subset of IP traffic and points to an SA for that traffic. In more complex environments, there may be multiple entries that potentially relate to a single SA or multiple SAs associated with a single SPD entry. The reader is referred to the relevant IPSec documents for a full discussion.

Each SPD entry is defined by a set of IP and upper-layer protocol field values, called *selectors*. In effect, these selectors are used to filter outgoing traffic in order to map it into a particular SA. Outbound processing obeys the following general sequence for each IP packet:

1. Compare the values of the appropriate fields in the packet (the selector fields) against the SPD to find a matching SPD entry, which will point to zero or more SAs.
2. Determine the SA if any for this packet and its associated SPI.
3. Do the required IPSec processing (i.e., AH or ESP processing).

The following selectors determine an SPD entry:
Remote IP Address: This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address. The latter two are required to support more than one destination system sharing the same SA (e.g., behind a firewall).
Local IP Address: This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address. The latter two are required to support more than one source system sharing the same SA (e.g., behind a firewall).
Next Layer Protocol: The IP protocol header includes a field that designates the protocol operating over IP. This is an individual protocol number, ANY, or for IPv6 only, OPAQUE. If AH or ESP is used, then this IP protocol header immediately precedes the AH or ESP header in the packet.
Name: a user identifies from the operating system. This is not a field in the IP or upper-layer headers but is available if IPsec is running on the same operating system as the user.
Local and Remote Ports: These may be individual TCP or UDP port values, an enumerated list of ports, or a wildcard port.

5.2.4 IP traffic processing

IPsec is executed on a packet-by-packet basis. When IPSec is implemented, each outbound IP packet is processed by the IPsec logic before transmission, and each inbound packet isprocessed by the IPsec logic after reception and before passing the packet contents on to the next higher layer. We look at the logic of these two situations in turn.

Outbound packets: Figure 5.4 highlights the main elements of IPsec processing for outbound traffic. A block of data from a higher layer, such as TCP, is passed down to the IP layer and an IP packet is formed, consisting of an IP header and an IP body. Then the following steps occur:

1. IPsec searches the SPD for a match to this packet.
2. If no match is found, then the packet is discarded and an error message is generated.
3. If a match is found, further processing is determined by the first matching entry in the SPD. If the policy for this packet is DISCARD, then the packet is discarded. If the policy is BYPASS, then there is no further IPsec processing; the packet is forwarded to the network for transmission.
4. If the policy is PROTECH, then a search is made of the SAD for a matching entry. If no entry is found, then IKE is invoked to create an SA with the appropriate keys and an entry is made in the SA.
5. The matching entry in the SAD determines the processing for this packet. Either encryption, authentication, or both can be performed, and either transport or tunnel mode can be used. The packet is then forwarded to the network for transmission.

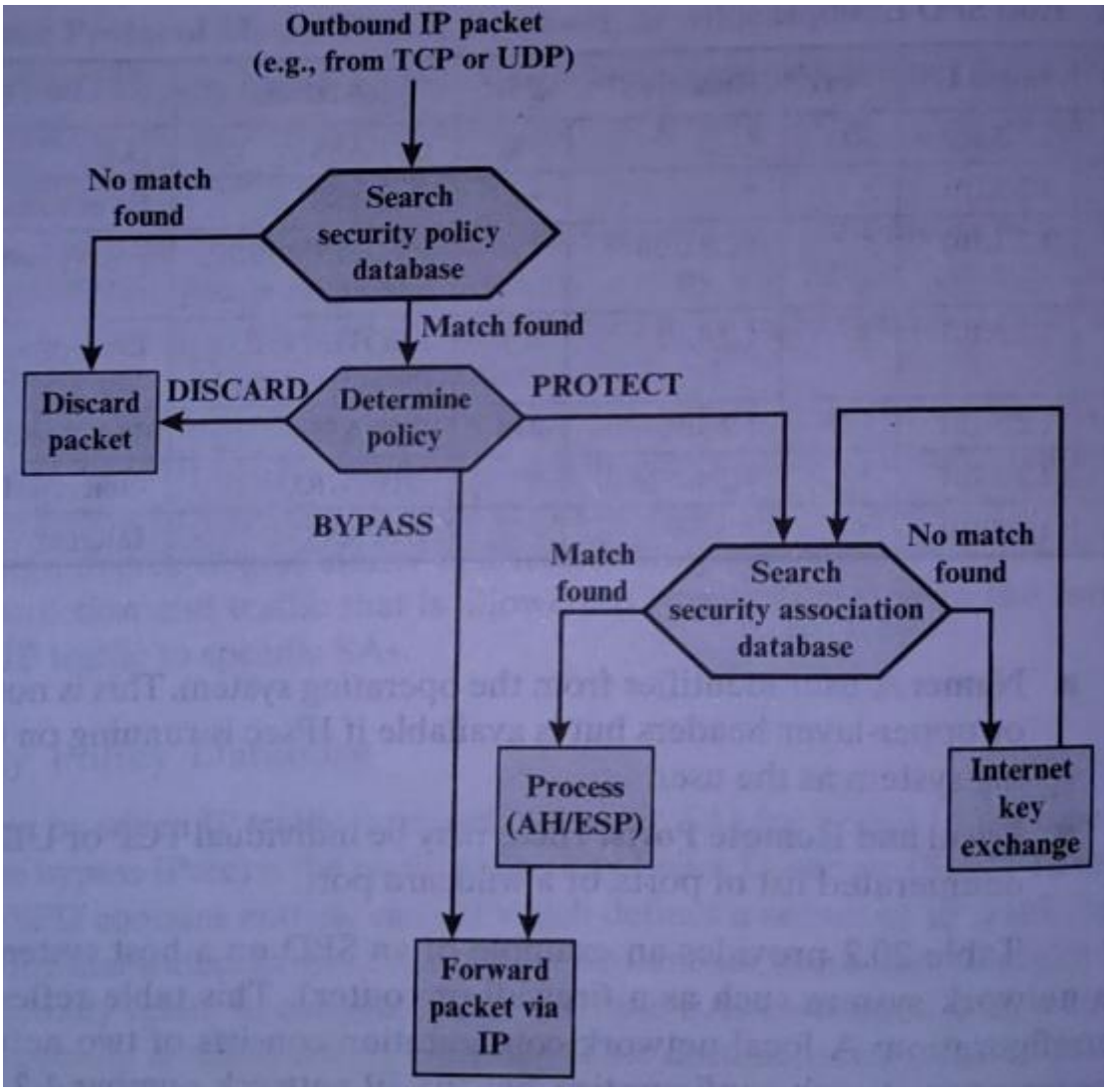


Figure 5.4: Processing Model for Outbound Packets

Inbound packets: Figure 5.5 highlights the main elements of IPsec processing for inbound traffic. An incoming IP packet triggers the IPsec processing. The following steps occur:

1. IPsec determines whether this is an unsecured IP packet or one that has ESP or AH headers/trailers, by examining the IP protocol field (IPv4) or Next header field (IPv6).
2. If the packet is unsecured, IPsec searches the SPD for a match to this packet. If the first matching entry has a policy of BYPASS, the IP header is processed and stripped off and the packet body is delivered to the next higher layer, such as TCP. If the first matching entry has a policy of PROTECT or DISCARD, or if there is no matching entry, the packet is discarded.
3. For a secured packet, IPsec searches the SAD. If no match is found, the packet is discarded. Otherwise, IPsec applies the appropriate ESP or AH processing. Then, the IP header is processed and stripped off and the packet body is delivered to the next higher layer, such as TCP.

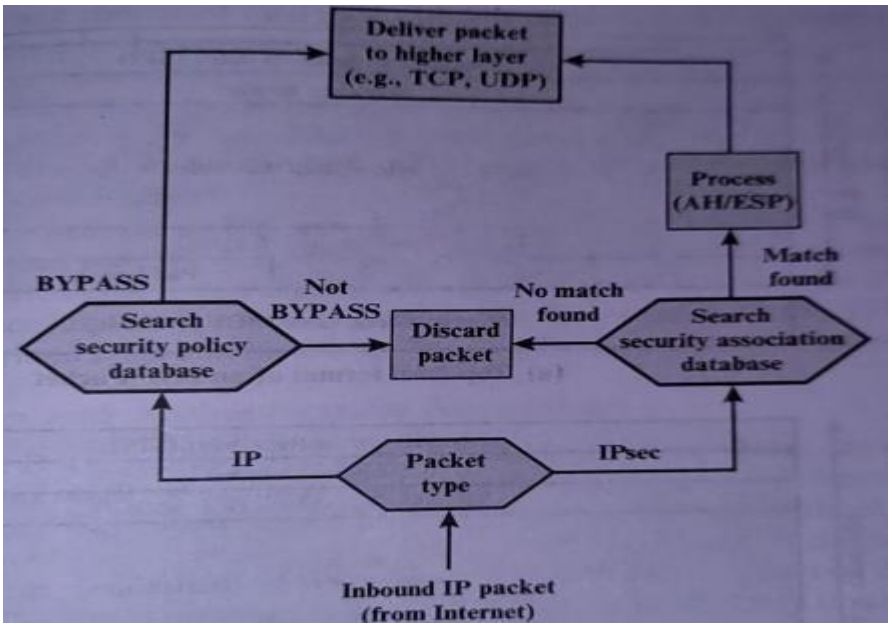
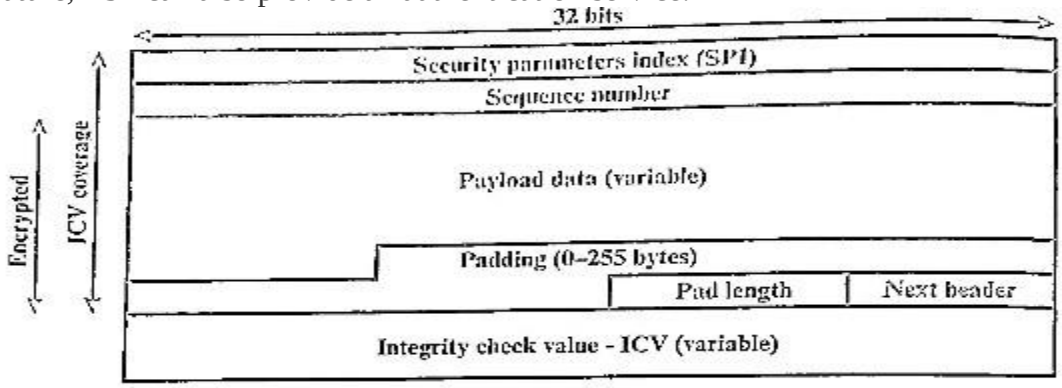


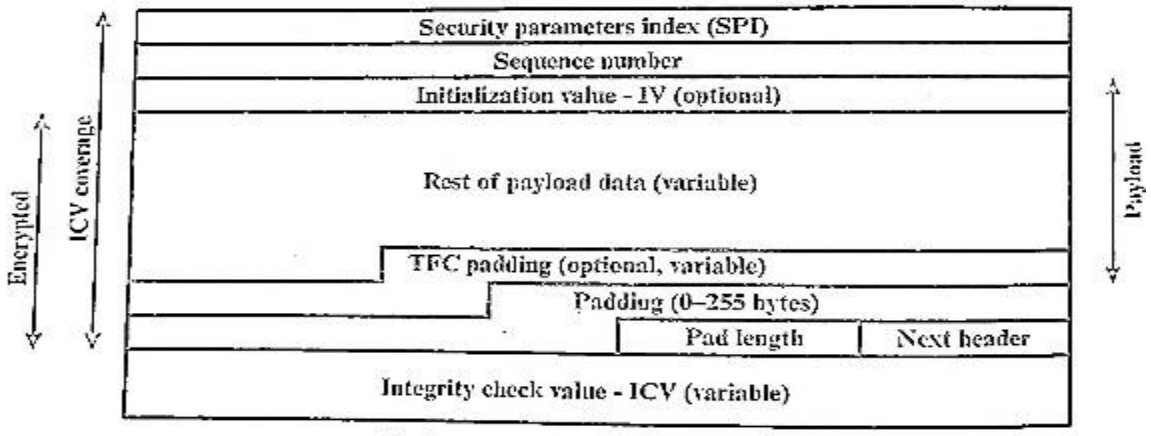
Figure 5.5: Processing Model for inbound packets

5.3 Encapsulating Security Payload

The Encapsulating Security Payload provides confidentiality services, including confidentiality of message contents and limited traffic flow confidentiality. As an optional feature, ESP can also provide an authentication service.



(a) Top-level format of an ESP Packet



(b) Substructure of payload data

Figure 5.6: ESP Packet format

5.3.1 ESP Format

Figure 5.6a shows the top-level format of an ESP packet. It contains the following fields.

Security Parameters Index (32 bits): Identifies a security association.

Sequence Number (32 bits): A monotonically increasing counter value; this provides an anti-replay function, as discussed for AH.

Payload Data (variable): This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.

Padding (0255 bytes): The purpose of this field is discussed later.

Pad Length (8 bits): Indicates the number of pad bytes immediately preceding this field.

Next Header (8 bits): Identifies the type of data contained in the payload data field by identifying the first header in that payload (for example, an extension header in IPv6, or an upper-layer protocol such as TCP).

Integrity Check Value (variable): A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value computed over the ESP packet minus the Authentication Data field.

Two additional fields may be present in the payload as in figure 5.6b. an initialization value (IV) or nonce is present if this is required by the encryption or authentication encryption algorithm used for ESP. if tunnel mode is being used, then the IPsec implementation may add traffic flow confidentiality (TFC) padding after payload data and before the padding field.

5.3.2 Encryption and Authentication Algorithms

The Payload Data, Padding, Pad Length, and Next Header fields are encrypted by the ESP service. If the algorithm used to encrypt the payload requires cryptographic synchronization data, such as an initialization vector (IV), then these data may be carried explicitly at the beginning of the Payload Data field. If included, an IV is usually not encrypted, although it is often referred to as being part of the ciphertext.

The ICV field is optional. It is present only if the integrity service is selected and is provided by either a separate integrity algorithm or a encryption mode algorithm that uses an ICV. The ICV is computed after the encryption is performed. This order of processing facilitates rapid detection and rejection of replayed or bogus packets by the receiver prior to decrypting the packets, hence potentially reducing the impact of denial of service (DoS) attacks. It also allows for the possibility of parallel processing of packets at the receiver that is decryption can take place in parallel with integrity checking. Note that because the ICV is not protected by encryption, a keyed integrity algorithm must be employed to compute the ICV.

5.3.3 Padding

The Padding field serves several purposes:

- If an encryption algorithm requires the plaintext to be a multiple of some number of bytes (e.g., the multiple of a single block for a block cipher), the Padding field is used to expand the plaintext (consisting of the Payload Data, Padding, Pad Length, and Next Header fields) to the required length.
- The ESP format requires that the Pad Length and Next Header fields be right aligned within a 32- bit word. Equivalently, the ciphertext must be an integer multiple of 32 bits. The Padding field is used to assure this alignment.
- Additional padding may be added to provide partial traffic flow confidentiality by concealing the actual length of the payload.

5.3.4 Anti-Replay Service

A replay attack is one in which an attacker obtains a copy of an authenticated packet and later transmits it to the intended destination. The receipt of duplicate, authenticated IP packets may disrupt service in some way or may have some other undesired consequence. The Sequence Number field is designed to thwart such attacks. First, we discuss sequence number generation by the sender, and then we look at how it is processed by the recipient.

When a new SA is established, the **sender** initializes a sequence number counter to 0. Each time that a packet is sent on this SA, the sender increments the counter and places the value in the Sequence Number field. Thus, the first value to be used is 1. If anti-replay is enabled (the default), the sender must not allow the sequence number to cycle past $2^{32} - 1$ back to zero. Otherwise, there would be multiple valid packets with the same sequence number. If the limit of $2^{32} - 1$ is reached, the sender should terminate this SA and negotiate a new SA with a new key.

Because IP is a connectionless, unreliable service, the protocol does not guarantee that packets will be delivered in order and does not guarantee that all packets will be delivered. Therefore, the IPSec authentication document dictates that the **receiver** should implement a window of size *W*, with a default of $W = 64$. The right edge of the window represents the highest sequence number, *N*, so far received for a valid packet. For any packet with a sequence number in the range from $N - W + 1$ to *N* that has been correctly received (i.e., properly authenticated), the corresponding slot in the window is marked (Figure 5.7). Inbound processing proceeds as follows when a packet is received:

1. If the received packet falls within the window and is new, the MAC is checked. If the packet is authenticated, the corresponding slot in the window is marked.
2. If the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked.
3. If the received packet is to the left of the window, or if authentication fails, the packet is discarded; this is an auditable event.

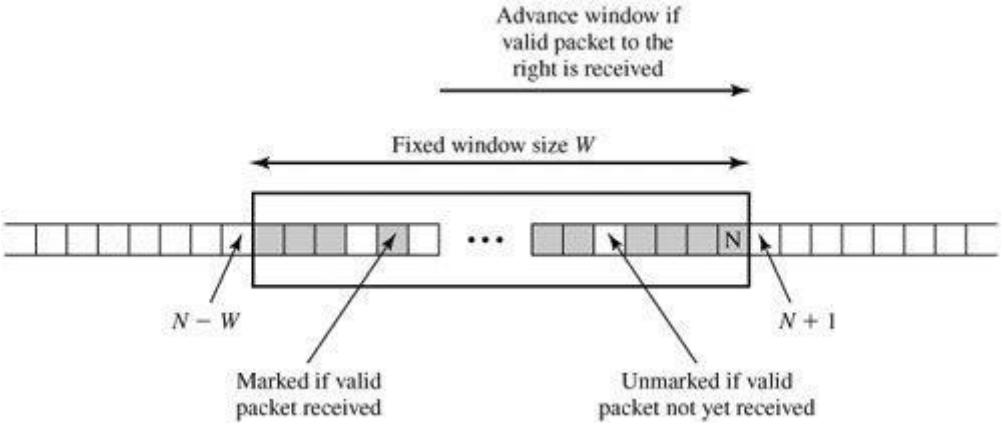


Figure 5.7: Anti-replay mechanism

5.3.5 Transport and Tunnel Modes

Figure 5.8 shows two ways in which the IPSec ESP service can be used. In the upper part of the figure, encryption (and optionally authentication) is provided directly between two hosts. Figure 5.8b shows how tunnel mode operation can be used to set up a *virtual private network*. In this example, an organization has four private networks interconnected across the Internet. Hosts on the internal networks use the Internet for transport of data but do not interact with other Internet-based hosts. By terminating the tunnels at the security gateway to each internal network, the configuration allows the hosts to avoid implementing the security capability.

The former technique is support by a transport mode SA, while the latter technique uses a tunnel mode SA.

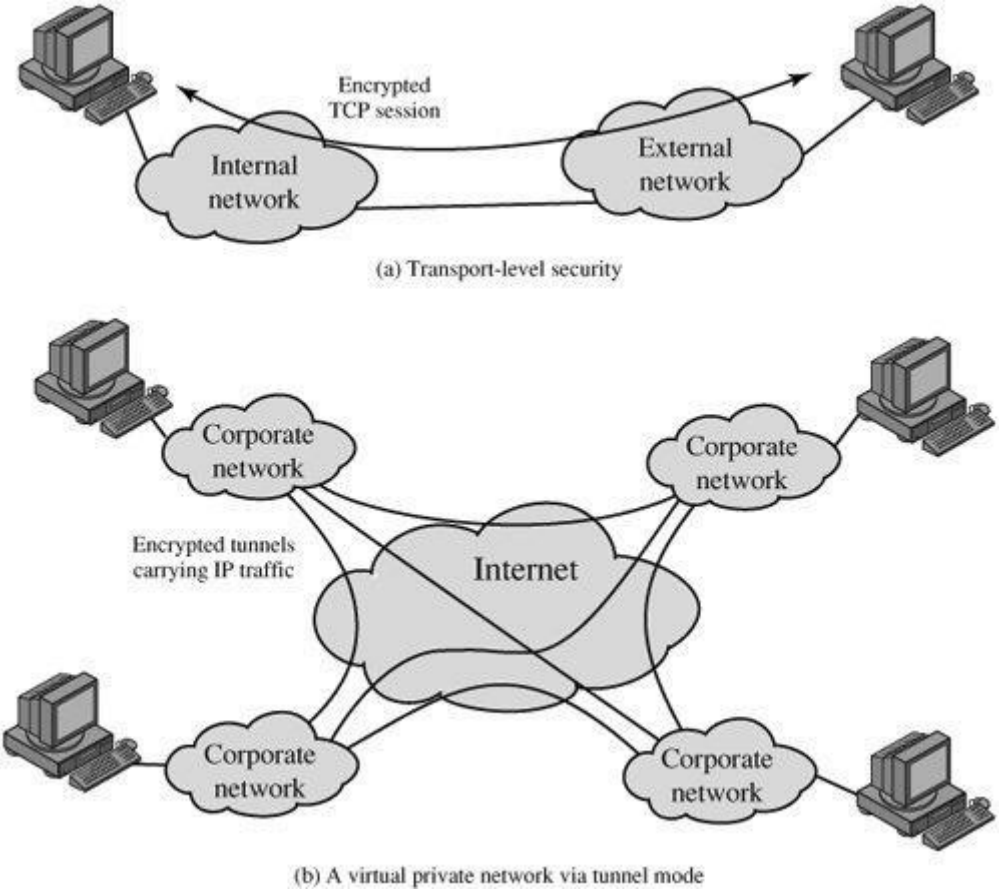


Figure 5.8: Transport-Mode versus Tunnel-Mode Encryption

Transport Mode ESP

Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (e.g., a TCP segment), as shown in Figure 5.9a. For this mode using IPv4, the ESP header is inserted into the IP packet immediately prior to the transport-layer header (e.g., TCP, UDP, ICMP) and an ESP trailer (Padding, Pad Length, and Next Header fields) is placed after theIP packet; if authentication is selected, the ESP Authentication Data field is added after the ESP trailer. The entire transport-level segment plus the ESP trailer are encrypted. Authentication covers all of the ciphertext plus the ESP header.

In the context of IPv6, ESP is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the ESP header appears after the IPv6 base header and the hop-by hop, routing, and fragment extension headers. The destination options extension header could appear before or after the ESP header, depending on the semantics desired. For IPv6, encryption covers the entire transport-level segment plus the ESP trailer plus the destination options extension header if it occurs after the ESP header. Again, authentication covers the ciphertext plus the ESP header.

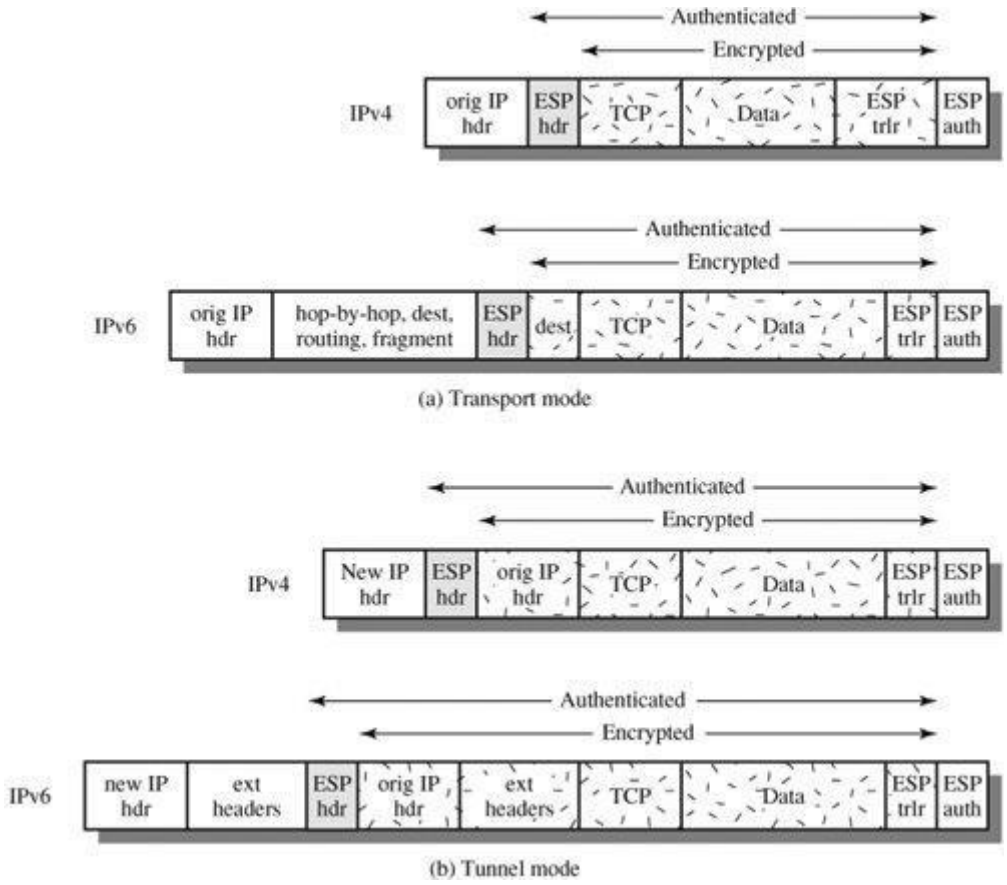


Figure 5.9: Scope of ESP encryption and authentication

Transport mode operation may be summarized as follows:

1. At the source, the block of data consisting of the ESP trailer plus the entire transport-layer segment is encrypted and the plaintext of this block is replaced with its ciphertext to form the IP packet for transmission. Authentication is added if this option is selected.
2. The packet is then routed to the destination. Each intermediate router needs to examine and process the IP header plus any plaintext IP extension headers but does not need to examine the ciphertext.
3. The destination node examines and processes the IP header plus any plaintext IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext transport-layer segment.

Transport mode operation provides confidentiality for any application that uses it, thusavoiding the need to implement confidentiality in every individual application. This mode of operation is also reasonably efficient, adding little to the total length of the IP packet. One drawback to this mode is that it is possible to do traffic analysis on the transmitted packets.

Tunnel Mode ESP

Tunnel mode ESP is used to encrypt an entire IP packet (Figure 5.9b). For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted. This method can be used to counter traffic analysis.

Because the IP header contains the destination address and possibly source routing directives and hopby- hop option information, it is not possible simply to transmit theencrypted IP packet prefixed by the ESP header. Intermediate routers would be unable to process such a packet. Therefore, it is necessary to encapsulate the entire block (ESP header

plus ciphertext plus Authentication Data, if present) with a new IP header that will contain sufficient information for routing but not for traffic analysis.

Whereas the transport mode is suitable for protecting connections between hosts that support the ESP feature, the tunnel mode is useful in a configuration that includes a firewall or other sort of security gateway that protects a trusted network from external networks. In this latter case, encryption occurs only between an external host and the security gateway or between two security gateways. This relieves hosts on the internal network of the processing burden of encryption and simplifies the key distribution task by reducing the number of needed keys. Further, it thwarts traffic analysis based on ultimate destination.

Consider a case in which an external host wishes to communicate with a host on an internal network protected by a firewall, and in which ESP is implemented in the external host and the firewalls. The following steps occur for transfer of a transport-layer segment from the external host to the internal host:

1. The source prepares an inner IP packet with a destination address of the target internal host. This packet is prefixed by an ESP header; then the packet and ESP trailer are encrypted and Authentication Data may be added. The resulting block is encapsulated with a new IP header (base header plus optional extensions such as routing and hop-by-hop options for IPv6) whose destination address is the firewall; this forms theouter IP packet.
2. The outer packet is routed to the destination firewall. Each intermediate router needs to examine and process the outer IP header plus any outer IP extension headers but does not need to examine the ciphertext.
3. The destination firewall examines and processes the outer IP header plus any outer IP extension headers. Then, on the basis of the SPI in the ESP header, the destination node decrypts the remainder of the packet to recover the plaintext inner IP packet.This packet is then transmitted in the internal network.
4. The inner packet is routed through zero or more routers in the internal network to the destination host.

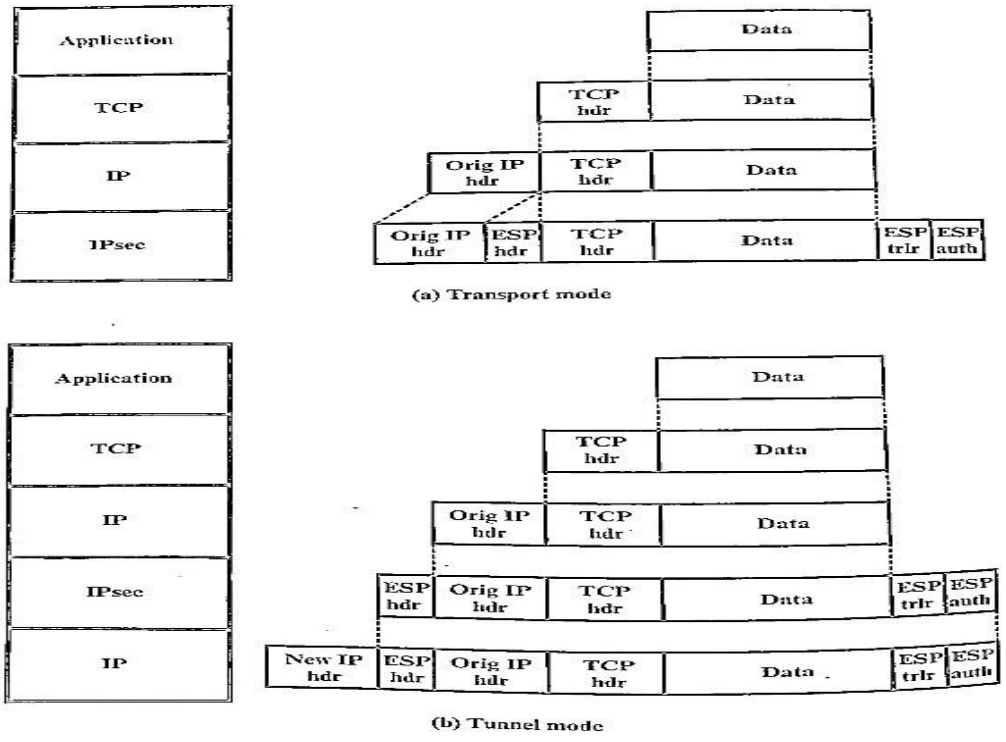


Figure 5.10: Protocol Operation for ESP

5.4 Combining Security Associations

An individual SA can implement either the AH or ESP protocol but not both. Sometimes a particular traffic flow will call for the services provided by both AH and ESP. Further, a particular traffic flow may require IPSec services between hosts and, for that same flow, separate services between security gateways, such as firewalls. In all of these cases, multiple SAs must be employed for the same traffic flow to achieve the desired IPSec services. The term *security association bundle* refers to a sequence of SAs through which traffic must be processed to provide a desired set of IPSec services. The SAs in a bundle may terminate at different endpoints or at the same endpoints.

Security associations may be combined into bundles in two ways:

- **Transport adjacency:** Refers to applying more than one security protocol to the same IP packet, without invoking tunneling. This approach to combining AH and ESP allows for only one level of combination; further nesting yields no added benefit since the processing is performed at one IPsec instance: the (ultimate) destination.
- **Iterated tunneling:** Refers to the application of multiple layers of security protocols effected through IP tunneling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path.

The two approaches can be combined, for example, by having a transport SA between hosts travel part of the way through a tunnel SA between security gateways. One interesting issue that arises when considering SA bundles is the order in which authentication and encryption may be applied between a given pair of endpoints and the ways of doing so. We examine that issue next. Then we look at combinations of SAs that involve at least one tunnel.

5.4.1 Authentication Plus Confidentiality

Encryption and authentication can be combined in order to transmit an IP packet that has both confidentiality and authentication between hosts. We look at several approaches.

ESP with Authentication Option

This approach is illustrated in Figure 5.9. In this approach, the user first applies ESP to the data to be protected and then appends the authentication data field. There are actually two subcases:

- **Transport mode ESP:** Authentication and encryption apply to the IP payload delivered to the host, but the IP header is not protected.
- **Tunnel mode ESP:** Authentication applies to the entire IP packet delivered to the outer IP destination address (e.g., a firewall), and authentication is performed at that destination. The entire inner IP packet is protected by the privacy mechanism, for delivery to the inner IP destination.

For both cases, authentication applies to the ciphertext rather than the plaintext.

Transport Adjacency

Another way to apply authentication after encryption is to use two bundled transport SAs, with the inner being an ESP SA and the outer being an AH SA. In this case ESP is used without its authentication option. Because the inner SA is a transport SA, encryption is applied to the IP payload. The resulting packet consists of an IP header (and possibly IPv6 header extensions) followed by an ESP. AH is then applied in transport mode, so thatauthentication covers the ESP plus the original IP header (and extensions) except for mutable fields. The advantage of this approach over simply using a single ESP SA with the ESP authentication option is that the authentication covers more fields, including the source and destination IP addresses. The disadvantage is the overhead of two SAs versus one SA.

Transport-Tunnel Bundle

The use of authentication prior to encryption might be preferable for several reasons. First, because the authentication data are protected by encryption, it is impossible for anyone to intercept the message and alter the authentication data without detection. Second, it may be desirable to store the authentication information with the message at the destination for later reference. It is more convenient to do this if the authentication information applies to the unencrypted message; otherwise the message would have to be reencrypted to verify the authentication information.

One approach to applying authentication before encryption between two hosts is to use a bundle consisting of an inner AH transport SA and an outer ESP tunnel SA. In this case, authentication is applied to the IP payload plus the IP header (and extensions) except for mutable fields. The resulting IP packet is then processed in tunnel mode by ESP; the result is that the entire, authenticated inner packet is encrypted and a new outer IP header (and extensions) is added.

5.4.2 Basic Combinations of Security Associations

The IPSec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPSec hosts (e.g., workstation, server) or security gateways (e.g. firewall, router). These are illustrated in Figure 5.11. The lower part of each case in the figure represents the physical connectivity of the elements; the upper part represents logical connectivity via one or more nested SAs. Each SA can be either AH or ESP. For host-to-host SAs, the mode may be either transport or tunnel; otherwise it must be tunnel mode.

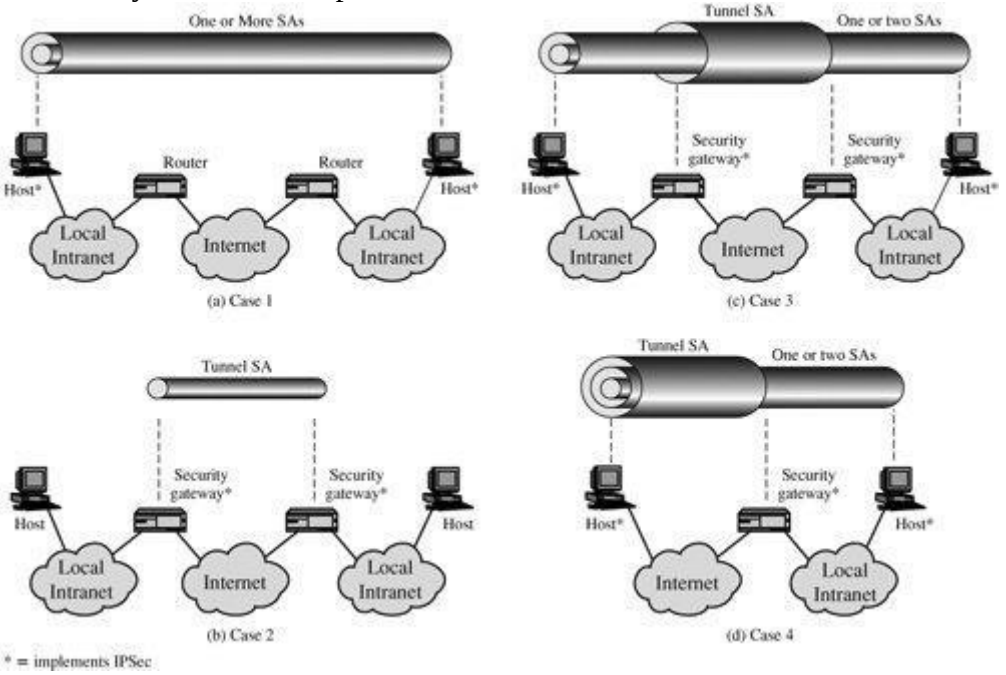


Figure 5.11: Basic Combinations of Security Associations

In **Case 1**, all security is provided between end systems that implement IPSec. For any two end systems to communicate via an SA, they must share the appropriate secret keys. Among the possible combinations:

a. AH in transport mode
b. ESP in transport mode
c. ESP followed by AH in transport mode (an ESP SA inside an AH SA)
d. Any one of a, b, or c inside an AH or ESP in tunnel mode

For **Case 2**, security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPSec. This case illustrates simple virtual private network support. The security architecture document specifies that only a single tunnel SA is needed for this case. The tunnel could support AH, ESP, or ESP with the authentication option. Nested tunnels are not required because the IPSec services apply to the entire inner packet.

Case 3 builds on Case 2 by adding end-to-end security. The same combinations discussed for cases 1 and 2 are allowed here. The gateway-to-gateway tunnel provides either authentication or confidentiality or both for all traffic between end systems. When the gateway-to-gateway tunnel is ESP, it also provides a limited form of traffic confidentiality. Individual hosts can implement any additional IPSec services required for given applications or given users by means of end-to-end SAs.

Case 4 provides support for a remote host that uses the Internet to reach an organization's firewall and then to gain access to some server or workstation behind the firewall. Onlytunnel mode is required between the remote host and the firewall. As in Case 1, one or two SAs may be used between the remote host and the local host.

5.5 INTERNET KEY EXCHANGE

The key management portion of IPSec involves the determination and distribution of secret keys. A typical requirement is four keys for communication between two applications: transmit and receive pairs for both AH and ESP. The IPSec Architecture document mandates support for two types of key management:

- **Manual:** A system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments.
- **Automated:** An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration. The default automated key management protocol for IPSec is referred to as ISAKMP/Oakley and consists of the following elements:
- **Oakley Key Determination Protocol**: Oakley is a key exchange protocol based on the Diffie-Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats.
- **Internet Security Association and Key Management Protocol (ISAKMP):** ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

ISAKMP by itself does not dictate a specific key exchange algorithm; rather, ISAKMP consists of a set of message types that enable the use of a variety of key exchange algorithms. Oakley is the specific key exchange algorithm mandated for use with the initial version of ISAKMP.

5.5.1 Key Determination Protocol

IKE key determination is a refinement of the Diffie-Hellman key exchange algorithm. Recall that Diffie-Hellman involves the following interaction between users A and B. There is prior agreement on two global parameters: q, a large prime number; and a a primitive root of q. A selects a random integer X_A as its private key, and transmits to B its public key $Y_A = a^{XA} \bmod q$. Similarly, B selects a random integer XB as its private key and transmits to A its public key $YB = a^{XB} \bmod q$. Each side can now compute the secret session key:

$$K = (Y_B)^{X_A} \bmod q = (Y_A)^{X_B} \bmod q = \alpha^{X_A X_B} \bmod q$$

The Diffie-Hellman algorithm has two attractive features:

- Secret keys are created only when needed. There is no need to store secret keys for a long period of time, exposing them to increased vulnerability.

● The exchange requires no preexisting infrastructure other than an agreement on the global parameters.

However, there are a number of weaknesses to Diffie-Hellman:

● It does not provide any information about the identities of the parties.

● It is subject to a man-in-the-middle attack, in which a third party C impersonates B while communicating with A and impersonates A while communicating with B. Both A and B end up negotiating a key with C, which can then listen to and pass on traffic. The man-in-the-middle attack proceeds as follows:

1. B sends his public key *YB* in a message addressed to A .

2. The enemy (E) intercepts this message. E saves B's public key and sends a message to A that has B's User ID but E's public key *YE*. This message is sent in such a way that it appears as though it was sent from B's host system. A receives E's message and stores E's public key with B's User ID. Similarly, E sends a message to B with E's public key, purporting to come from A.

3. B computes a secret key *K1* based on B's private key and *YE*. A computes a secret key *K2* based on A's private key and *YE*. E computes *K1* using E's secret key *XE* and *YB* and computer *K2* using *YE* and *YB*.

4. From now on E is able to relay messages from A to B and from B to A, appropriately changing their encipherment en route in such a way that neither A nor B will know that they share their communication with E.

● It is computationally intensive. As a result, it is vulnerable to a clogging attack, in which an opponent requests a high number of keys. The victim spends considerable computing resources doing useless modular exponentiation rather than real work.

IKE key determination is designed to retain the advantages of Diffie-Hellman while countering its weaknesses.

Features of IKE key determination

The IKE key determination algorithm is characterized by five important features:

1. It employs a mechanism known as cookies to thwart clogging attacks.
2. It enables the two parties to negotiate a *group*; this, in essence, specifies the global parameters of the Diffie-Hellman key exchange.
3. It uses nonces to ensure against replay attacks.
4. It enables the exchange of Diffie-Hellman public key values.
5. It authenticates the Diffie-Hellman exchange to thwart man-in-the-middle attacks.

We have already discussed Diffie-Hellman. Let us look the remainder of these elements in turn. First, consider the problem of clogging attacks. In this attack, an opponent forges the source address of a legitimate user and sends a public Diffie-Hellman key to the victim. The victim then performs a modular exponentiation to compute the secret key. Repeated messagesof this type can *clog* the victim's system with useless work. The **cookie exchange** requires that each side send a pseudorandom number, the cookie, in the initial message, which the other side acknowledges. This acknowledgment must be repeated in the first message of the Diffie-Hellman key exchange. If the source address was forged, the opponent gets no answer.Thus, an opponent can only force a user to generate acknowledgments and not to perform the Diffie-Hellman calculation.

IKE mandates that cookie generation satisfy three basic requirements:

1. The cookie must depend on the specific parties. This prevents an attacker fromobtaining a cookie using a real IP address and UDP port and then using it to swamp the victim with requests from randomly chosen IP addresses or ports.
2. It must not be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity. This implies that the issuing entity will use local

secret information in the generation and subsequent verification of a cookie. It must not be possible to deduce this secret information from any particular cookie. The pointof this requirement is that the issuing entity need not save copies of its cookies, whichare then more vulnerable to discovery, but can verify an incoming cookie acknowledgment when it needs to.

3. The cookie generation and verification methods must be fast to thwart attacks intended to sabotage processor resources.

The recommended method for creating the cookie is to perform a fast hash (e.g., MD5) over the IP Source and Destination addresses, the UDP Source and Destination ports, and a locally generated secret value.

Oakley (IKE key determination) supports the use of different **groups** for the Diffie-Hellman key exchange. Each group includes the definition of the two global parameters and the identity of the algorithm. The current specification includes the following groups:

- Modular exponentiation with a 768-bit modulus

  $q = 2^{768} - 2^{704} - 1 + 2^{64} \times ([2^{638} \times \pi] + 149686)$

  $\alpha = 2$
- Modular exponentiation with a 1024-bit modulus

  $q = 2^{1024} - 2^{960} - 1 + 2^{64} \times ([2^{894} \times \pi] + 129093)$

  $\alpha = 2$
- Modular exponentiation with a 1536-bit modulus

  Parameters to be determined
- Elliptic curve group over 2^{155}

  Generator (hexadecimal):X = 7B, Y = 1C8

  Elliptic curve parameters (hexadecimal):A = 0, Y = 7338F
- Elliptic curve group over 2^{185}

  Generator (hexadecimal):X = 18, Y = D

  Elliptic curve parameters (hexadecimal):A = 0, Y = 1EE9

The first three groups are the classic Diffie-Hellman algorithm using modular exponentiation. The last two groups use the elliptic curve analog to Diffie-Hellman.

Oakley employs **nonces** to ensure against replay attacks. Each nonce is a locally generated pseudorandom number. Nonces appear in responses and are encrypted during certain portions of the exchange to secure their use.

Three different **authentication** methods can be used with Oakley:

● **Digital signatures:** The exchange is authenticated by signing a mutually obtainable hash; each party encrypts the hash with its private key. The hash is generated over important parameters, such as user IDs and nonces.

● **Public-key encryption**: The exchange is authenticated by encrypting parameters such as IDs and nonces with the sender's private key.
● **Symmetric-key encryption:** A key derived by some out-of-band mechanism can be used to authenticate the exchange by symmetric encryption of exchange parameters.

IKEv2 Exchanges

The Oakley specification (IKEv2 protocol) includes a number of examples of exchanges that are allowable under the protocol. To give a flavor of Oakley, we present one example, called aggressive key exchange in the specification, so called because only three messages are exchanged.

Figure 5.12 shows the aggressive key exchange protocol. In the first step, the initiator (I) transmits a cookie, the group to be used, and I's public Diffie-Hellman key for this exchange. I also indicates the offered public-key encryption, hash, and authentication algorithms to be used in this exchange. Also included in this message are the identifiers of I and the responder (R) and I's nonce for this exchange. Finally, I appends a signature using I's private key that signs the two identifiers, the nonce, the group, the Diffie-Hellman public key, and the offered algorithms.

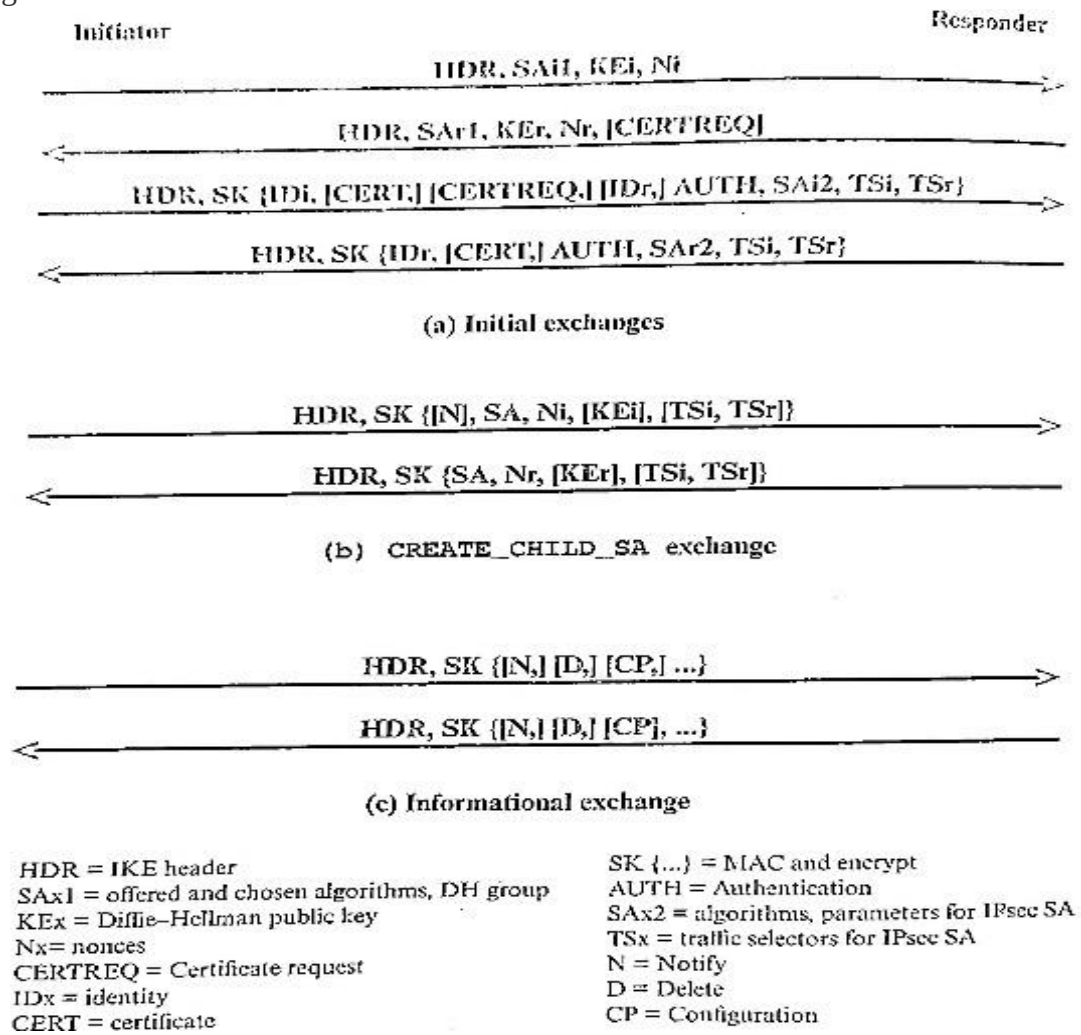


Figure 5.12: IKEv2 Exchanges

When R receives the message, R verifies the signature using I's public signing key. R acknowledges the message by echoing back I's cookie, identifier, and nonce, as well as the group. R also includes in the message a cookie, R's Diffie-Hellman public key, the selected algorithms (which must be among the offered algorithms), R's identifier, and R's nonce for

this exchange. Finally, R appends a signature using R's private key that signs the two identifiers, the two nonces, the group, the two Diffie-Hellman public keys, and the selected algorithms.

When I receives the second message, I verifies the signature using R's public key. The nonce values in the message assure that this is not a replay of an old message. To complete the exchange, I must send a message back to R to verify that I has received R's public key.

Header and payload formats

IKE defines procedures and packet formats to establish, negotiate, modify, and delete security associations. As part of SA establishment, ISAKMP defines payloads for exchanging key generation and authentication data. These payload formats provide a consistent framework independent of the specific key exchange protocol, encryption algorithm, and authentication mechanism.

IKE Header Format

An ISAKMP (IKE) message consists of an ISAKMP header followed by one or more payloads. All of this is carried in a transport protocol. The specification dictates that implementations must support the use of UDP for the transport protocol.

Figure 5.13a shows the header format for an ISAKMP(IKE) message. It consists of the following fields:

- **Initiator SPI (64 bits):** Cookie of entity that initiated SA establishment, SA notification, or SA deletion.
- **Responder SPI (64 bits):** Cookie of responding entity; null in first message from initiator.
- **Next Payload (8 bits):** Indicates the type of the first payload in the message;
- **Major Version (4 bits):** Indicates major version of ISAKMP in use.
- **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. Two bits so far defined: The Encryption bit is set if all payloads following the header are encrypted using the encryption algorithm for this SA. The Commit bit is used to ensure that encrypted material is not received prior to completion of SA establishment.
- **Message ID (32 bits):** Unique ID for this message.
- **Length (32 bits):** Length of total message (header plus all payloads) in octets.

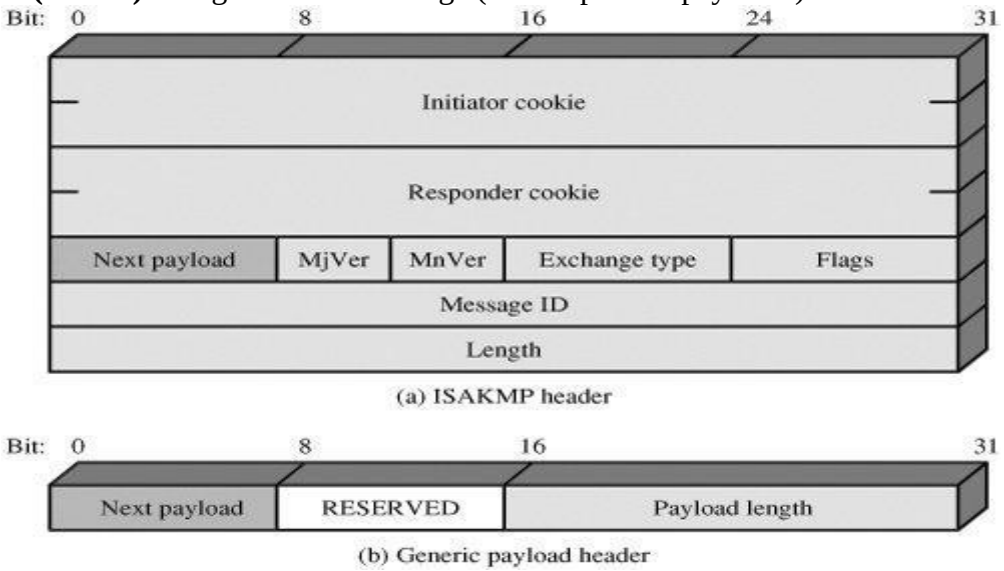


Figure 5.13 IKE formats

IKE Payload Types

All IKE (ISAKMP) payloads begin with the same generic payload header shown in Figure 5.13b. The Next Payload field has a value of 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.

Table 5.3 summarizes the payload types defined for IKE (ISAKMP), and lists the fields, or parameters, that are part of each payload. The **SA payload** is used to begin the establishment of an SA. In this payload, the Domain of Interpretation parameter identifies theDOI under which negotiation is taking place. The IPSec DOI is one example, but ISAKMP can be used in other contexts. The Situation parameter defines the security policy for this negotiation; in essence, the levels of security required for encryption and confidentiality are specified (e.g., sensitivity level, security compartment).

The **Proposal payload** contains information used during SA negotiation. The payload indicates the protocol for this SA (ESP or AH) for which services and mechanisms are being negotiated. The payload also includes the sending entity's SPI and the number of transforms. Each transform is contained in a transform payload. The use of multiple transform payloads enables the initiator to offer several possibilities, of which the responder must choose one or reject the offer.

The **Transform payload** defines a security transform to be used to secure the communications channel for the designated protocol. The Transform # parameter serves to identify this particular payload so that the responder may use it to indicate acceptance of this transform. The Transform-ID and Attributes fields identify a specific transform (e.g., 3DES for ESP, HMAC-SHA-1-96 for AH) with its associated attributes (e.g., hash length).

Table 5.3 IKE payload types

Type	Parameters
Security Association	Proposals
Key Exchange	DH Group #, Key Exchange Data
Identification	ID Type, ID Data
Certificate	Cert Encoding, Certificate Data
Certificate Request	Cert Encoding, Certification Authority
Authentication	Auth Method, Authentication Data
Nonce	Nonce Data
Notify	Protocol-ID, SPI Size, Notify Message Type, SPI, Notification Data
Delete	Protocol-ID, SPI Size, # of SPIs, SPI (one or more)
Vendor ID	Vendor ID
Traffic Selector	Number of TSs, Traffic Selectors
Encrypted	IV, Encrypted IKE payloads, Padding, Pad Length, ICV
Configuration	CFG Type, Configuration Attributes
Extensible Authentication Protocol	EAP Message

The **Key Exchange payload** can be used for a variety of key exchange techniques, including Oakley, Diffie-Hellman, and the RSA-based key exchange used by PGP. The Key Exchange data field contains the data required to generate a session key and is dependent on the key exchange algorithm used.

The **Identification payload** is used to determine the identity of communicating peers and may be used for determining authenticity of information. Typically the ID Data field will contain an IPv4 or IPv6 address.

The **Certificate payload** transfers a public-key certificate. The Certificate Encoding field indicates the type of certificate or certificate-related information, which may include the following:

- PKCS #7 wrapped X.509 certificate
- PGP certificate
- DNS signed key
- X.509 certificate signature
- X.509 certificate key exchange
- Kerberos tokens
- Certificate Revocation List (CRL)
- Authority Revocation List (ARL)
- SPKI certificate

At any point in an ISAKMP exchange, the sender may include a **Certificate Request**payload to request the certificate of the other communicating entity. The payload may list more than one certificate type that is acceptable and more than one certificate authority that isacceptable.

The **Hash payload** contains data generated by a hash function over some part of the message and/or ISAKMP state. This payload may be used to verify the integrity of the data in a message or to authenticate negotiating entities.

The **Signature payload** contains data generated by a digital signature function over some part of the message and/or ISAKMP state. This payload is used to verify the integrity of the data in a message and may be used for nonrepudiation services.

The **Nonce payload** contains random data used to guarantee liveness during an exchange and protect against replay attacks.

The **Notification payload** contains either error or status information associated with this SA or this SA negotiation.

Error Messages	Status Messages
Unsupported Critical Payload	Initial Contact
Invalid IKE SPI	Set Window Size
Invalid Major Version	Additional TS Possible
Invalid Syntax	IPCOMP Supported
Invalid Payload Type	NAT Detection Source IP
Invalid Message ID	NAT Detection Destination IP
Invalid SPI	Cookie
	Use Transport Mode

Error Messages	Status Messages
No Proposal Chosen	HTTP Cert Lookup Supported
Invalid KE Payload	Rekey SA
Authentication Failed	ESP TFC Padding Not Supported
Single Pair Required	Non First Fragments Also
No Additional SAS	
Internal Address Failure	
Failed CP Required	
TS Unacceptable	
Invalid Selectors	

Responder-Lifetime: Communicates the SA lifetime chosen by the responder.

- **Replay-Status:** Used for positive confirmation of the responder's election of whether or not the responder will perform anti-replay detection.
- **Initial-Contact:** Informs the other side that this is the first SA being established with the remote system. The receiver of this notification might then delete any existing SA's it has for the sending system under the assumption that the sending system has rebooted and no longer has access to those SAs.

The **Delete payload** indicates one or more SAs that the sender has deleted from its database and that therefore are no longer valid.

5.6 Cryptographic Suits

	VPN-A	VPN-B
ESP encryption	3DES-CBC	AES-CBC (128-bit key)
ESP integrity	HMAC-SHA1-96	AES-XCBC-MAC-96
IKE encryption	3DES-CBC	AES-CBC (128-bit key)
IKE PRF	HMAC-SHA1	AES-XCBC-PRF-128
IKE Integrity	HMAC-SHA1-96	AES-XCBC-MAC-96
IKE DH group	1024-bit MODP	2048-bit MODP

(a) Virtual private networks (RFC 4308)

	GCM-128	GCM-256	GMAC-128	GMAC-256
ESP encryption/ Integrity	AES-GCM (128-bit key)	AES-GCM (256-bit key)	Null	Null
ESP integrity	Null	Null	AES-GMAC (128-bit key)	AES-GMAC (256-bit key)
IKE encryption	AES-CBC (128-bit key)	AES-CBC (256-bit key)	AES-CBC (128-bit key)	AES-CBC (256-bit key)
IKE PRF	HMAC-SHA-256	HMAC-SHA-384	HMAC-SHA-256	HMAC-SHA-384
IKE Integrity	HMAC-SHA-256-128	HMAC-SHA-384-192	HMAC-SHA-256-128	HMAC-SHA-384-192
IKE DH group	256-bit random ECP	384-bit random ECP	256-bit random ECP	384-bit random ECP

(b) NSA Suite B (RFC 6379)

Figure 5.14: Cryptographic suits for IPSec

Three types of secret key algorithms are used:

- **Encryption:** For encryption, the cipher block chaining (CBC) mode is used.
- **Message authentication:** For message authentication, VPN-A relies on HMAC with SHA-1 with the output truncated to 96 bits. VPN-B relies on a variant of CMAC with the output truncated to 96 bits.
- **Pseudorandom function:** IKEv2 generates pseudorandom bits by repeated use of the MAC used for message authentication.

Three categories of secret key algorithms are listed:

- **Encryption:** For ESP, authenticated encryption is provided using the GCM mode with either 128-bit or 256-bit AES keys. For IKE encryption, CBC is used, as it was for the VPN suites.
- **Message authentication:** For ESP, if only authentication is required, then GMAC is used. As discussed in Chapter 12, GMAC is simply the authentication portion of GMC. For IKE, message authentication is provided using HMAC with one of the SHA-3 hash functions.
- **Pseudorandom function:** As with the VPN suites, IKEv2 in these suites generates pseudorandom bits by repeated use of the MAC used for message authentication.

For the Diffie-Hellman algorithm, the use of elliptic curve groups modulo a prime is specified. For authentication, elliptic curve digital signatures are listed. The original IKEv2 documents used RSA based digital signatures. Equivalent or greater strength can be achieved using ECC with fewer key bits.