Chapter 1
`Introduction to Cryptography

The word ***cryptography*** comes from two Greek words meaning "*secret writing*" and is the art and science of information hiding. This field is very much associated with mathematics and computer science with application in many fields like computer security, electronic commerce, telecommunication, etc.

So cryptography is a subject that should be of interest to many people, especially because we now live in the Information Age, and our secrets can be transmitted in so many ways – email, cell phone, etc. – and all these channels need to be protected [simon singh].

Secrecy and Encryption

In the ancient days, cryptography was mostly referred to as ***encryption*** – the mechanism to convert the readable ***plaintext*** into unreadable (incomprehensible) text i.e. ***ciphertext***, and ***decryption*** – the opposite process of encryption i.e. conversion of ciphertext back to the plaintext. Though the consideration of cryptography was on message confidentiality (encryption) in the past, nowadays cryptography considers the study and practices of authentication, digital signatures, integrity checking, and key management, etc.

Encryption mostly provides the secrecy of massage being transmitted over the communication network. This is called confidentiality of massage. The only sender knows the keys and can decipher the massage.

Cryptology

Cryptanalysis is the breaking of codes. Cryptanalysis encompasses all of the techniques to recover the plaintext and/or key from the ciphertext.

The combined study of cryptography and cryptanalysis is known as ***cryptology***. Though most of the time we use cryptography and cryptology in the same way.

Objective of cryptography

Encryption and Decryption

Encryption is the process of encoding a message so that its meaning is not obvious i.e. converting information from one form to some other unreadable form using some algorithm called ***cipher*** with the help of secret message called ***key***. The converting text is called is ***plaintext*** and the converted text is called ***ciphertext***.

Decryption is the reverse process, transforming an encrypted message back into its normal, original form. In decryption process also the use of key is important.

Alternatively, the terms ***encode*** and ***decode*** or ***encipher*** and ***decipher*** are used instead of ***encrypt*** and ***decrypt***. That is, we say that we encode, encrypt, or encipher the original message to hide its meaning. Then, we decode, decrypt, or decipher it to reveal the original message.

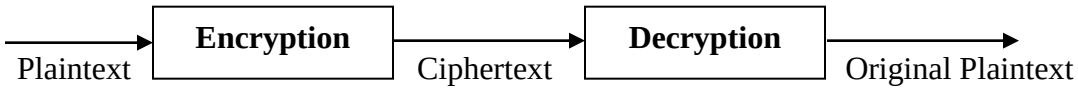


Fig: *Encryption-Decryption*

The use of encryption techniques is being used since very long period as it can be noted from the technique called ***Caesar's cipher*** used by Julius Caesar for information passing to his soldiers. Encryption techniques have also been extensively used in military purposes to conceal the information from the enemy. Nowadays to gain the confidentiality encryption is being used in many areas like communication, internet banking, digital right management, etc.

Key

A ***key*** is a parameter or a piece of information used to determine the output of cryptographic algorithm. While doing the encryption, key determines the transformation of plaintext to the cipher text and vice versa. Keys are also used in other cryptographic processes like message authentication codes and digital signatures. Most of the cryptographic systems depend upon the key and thus the secrecy of the key is very important and is one of the difficult problems in practice. Another important issue for the key is its length. Since key is the sole entity that defines the strength of the security (normally algorithm used is public) we need to select the key in a way such that attacker should take long enough to try all possibilities. To prevent the key from being guessed the choice of the key must be random.

Cipher

A ***cipher*** is an algorithm for performing encryption and decryption. The operation of cipher depends upon the special information called key. Without knowledge of the key, it should be difficult, if not nearly impossible, to decrypt the resulting cipher into readable plaintext. There are many types of encryption techniques that have advanced from history, however the distinction of encryption technique can be broadly categorized in terms of number of key used and way of converting plaintext to the ciphertext.

Cryptosystem

Cryptosystem is a 5-tuple/quintuple (E, D, M, K, C), where M set of plaintexts, K set of keys, C set of ciphertexts, E set of encryption functions e: $M \times K \rightarrow C$ and D set of decryption functions d: $C \times K \rightarrow M$.

Example: *Caesar Cipher*

M = {sequences of letters}

K = {i | i is an integer and $0 \leq i \leq 25$}

E = {E_k | $k \in K$ and for all letters m, $E_k(m) = (m + k) \bmod 26$ }

D = {D_k | $k \in K$ and for all letters c, $D_k(c) = (26 + c – k) \bmod 26$}

C = M

Cryptographic system characteristics

Cryptographic systems are characterized along three independent dimensions:

The type of operations used for transforming plaintext to ciphertext. All encryption algorithms are based on two general principles: substitution, in which each element 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.

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.

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.

Classical Cryptosystem

Historical pen and paper ciphers used in the past are sometimes known as classical ciphers.
These are the very old or quite old cryptosystem that were used in pre computer age.these crypto system are too weak now days and can be broken easily with computer.
But we even studied these cryptosystem because they illustrate basic of the concepts of cryptography.

Substitution Cipher

In substitution ciphers the letters are systematically replaced by other letters or symbols.

1. Caesar Cipher

It is the simple shift monoalphabetic classical cipher where each letter is replaced by a letter 3 position (actual Caesar cipher) ahead using the circular alphabetic ordering i.e. letter after Z is A.

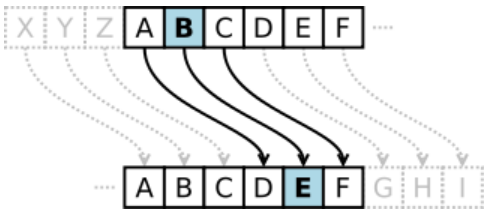


Fig: Caesar Cipher

So when we encode HELLO WORLD, the cipher text becomes KHOORZRUOG. Here we number each English alphabet starting from 0 (A) to 25 (Z). Each letter of the clear message is replaced by the letter whose number is obtained by adding the key (a number from 0 to 25) to the letter's number modulo 26. See the picture to visualize the Caesar cipher. The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1, ..., Z = 25. Encryption of a letter c by a shift k can be described mathematically as,

$$c = E_k(m) = (m + k) \mod 26$$

Decryption is performed similarly,

$$m = D_k(c) = (c + 26 - k) \mod 26$$

Similarly, consider some examples of Caesar cipher;

```
Plaintext:  meet me after the toga party
```

Ciphertext: PHHW PH DIWHU WKH WRJD SDUWB

```
Plaintext:  the quick brown fox jumps over the lazy dog
```
Ciphertext: WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ

Attacking the Cipher

Caesar Cipher is quite easily broken even with ciphertext only. One can attack the cipher text using exhaustive search by trying all possible keys until you find the right one. Exhaustive search is best suited if the key space is small and we have only 26 possible keys in Caesar cipher. Another approach of attacking the cipher is statistical analysis where we compare the ciphertext to 1-gram model of English.

Caesar's Problem

The main problem with Caesar's Cipher is that the key is too short and can be found by exhaustive search. Again statistical frequencies not concealed well i.e. they look too much like regular English letters. So the solution can be to increase the key length (can be done using multiple letters in key) so that cryptanalysis gets harder.

Transposition Cipher

In transposition ciphers the letters are systematically arranged so that the actual position of letters is gets changed making the text garble.

2. Rail-Fence Cipher

The Rail Fence Cipher is a form of transposition cipher that derives its name from the way in which it is encoded. In the rail fence cipher, the plaintext is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.

For example, using 3 "rails" and a message of 'WE ARE DISCOVERED FLEE AT ONCE', the cipherer writes out:

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

Then reads off:

```
WECRL TEERD SOEEF EAOCA IVDEN
```

Similarly, if we have 3 "rails" and a message of THIS IS THE PLAINTEXT, the cipherer writes out (we are not showing diagonal move here just write in down rail a step ahead):

T S T P I E

H I H L N X

I S E A T T

The ciphertext is T S T P I E H I H L N X I S E A T T

The problem with Rail Fence Cipher is that the rail fence cipher is not very strong; the number of practical keys is small enough that a cryptanalyst can try them all by hand. To decrypt we get the number of letters to be skipped. For this if the number of rail is n key is $\lceil total\ letters\ in\ ciphertext\ /\ n \rceil$ so in our e.g. n = 3 and key is 18/3 = 6 i.e. skip 6 letters from the letter you are reading every time to get plaintext (remember to go circular that is if count ends continue from the starting letter leaving the read letter). See below:

We

T	S	T	P	I	E	H	I	H	L	N	X	I	S	E	A	T	T
1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6

have selected letter with index 1 THI. Now choose the letter with index 2, see below

T	S	T	P	I	E	H	I	H	L	N	X	I	S	E	A	T	T
1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6

Continue like this until you read off all the characters.

3. Vigenere Cipher: Substitution Cipher (Polyalphateic)

It is like Caesar cipher, but uses a phrase for e.g. for the message THE BOY HAS THE BALL and the key VIG, encipher using Caesar cipher for each letter:

key VIGVIGVIGVIGVIGV

plain THEBOYHASTHEBALL

cipher OPKWWECIYOPKWIRG

Here, generally, we repeatedly write key above the plaintext and use the Caesar cipher for each letter in the plaintext where key for each letter being processed is taken from the repeated key letter just above it. This process is simplified by using the table as below called 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

Fig: Vigenere Tableau

Period: length of key. In example above it is 3.

Tableau: Table used to encipher and decipher. In tableau Vigènere cipher has key letters on top, plaintext letters on the left or vice versa. It is also possible to have key on top (left) plaintexts in middle and ciphertexts in left (top).

Assuming key on top and the plaintext on left, Decryption is performed by finding the position of the ciphertext letter in a column, corresponding to the key letter, of the table, and then taking the label of the row in which it appears as the plaintext letter. For example, in column V (key letter), the ciphertext letter O appears in row T, which taken as the first plaintext letter. The second letter is decrypted by looking up P in column I of the table; it appears in row H, which is taken as the plaintext letter. This process continues until we find the plaintext letters for all the ciphertext letters

4. One-Time Pad (simple XOR)

It is a variant of a Vigenère cipher with a random key at least as long as the message. Since it has very high key length it is provably unbreakable. *Joseph Mauborgne* proposed this concept. He suggested using a random key that is as long as the message, so 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. In One-time pad keys must be random, or we can attack the cipher by trying to regenerate the key approximations, such as using pseudorandom number generators to generate keys, are not random. This approach produces random output that bears no statistical relationship to the plaintext. Because the ciphertext contains no information whatsoever about the plaintext, there is simply no way to break the code.

5. 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 keyword is MONARCHY then the matrix is:

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

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. 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).

6. Hill Cipher

Another interesting multi letter 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 example, consider the plaintext "paymoremoney" and use the encryption key

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

The first three letters of the plaintext are represented by the vector

$$\begin{pmatrix}15\\ 0\\ 24\end{pmatrix} \text{ then } K.\begin{pmatrix}15\\ 0\\ 24\end{pmatrix}=\begin{pmatrix}375\\ 819\\ 487\end{pmatrix}\bmod 26=\begin{pmatrix}11\\ 13\\ 18\end{pmatrix}=\text{LNS}$$

the ciphertext for the entire plaintext is LNSHDLEWMTRW.

Hence in general the hill cipher can be expressed as

$C= E(K, P) = KP \bmod 26$

$P = D(K, P) = K^{-1}C \bmod 26 = K^{1}KP = P$

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.

Viruses, Worms and Trojan Horse

Introduction

Malicious logic is a set of instructions that cause a site's security policy to be violated.

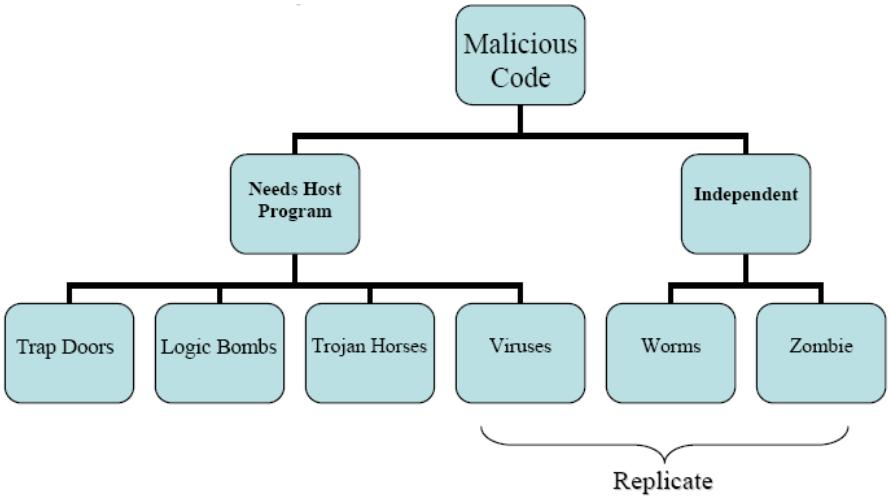
Example: The following UNIX script is named ls and is placed in a directory.

```
cp /bin/sh /tmp/.xxsh
chmod o+s,w+x /tmp/.xxsh
rm ./ls
ls $*
```

It creates a copy of the UNIX shell that is setuid to the user executing this program. This program is deleted, and then the correct ls command is executed. On most systems, it is against policy to trick someone into creating a shell that is setuid to themselves. If someone is tricked into executing this script, a violation of the (implicit) security policy occurs. This script is an example of malicious logic.

Malicious code refers to a broad category of software threats to our network and systems. Perhaps the most sophisticated types of threats to computer systems are presented by malicious codes that exploit vulnerabilities in computer systems. Any code which *modifies or destroys data, steals data , allows unauthorized access, exploits or damage a system, and does something that user did not intend to do,* is called malicious code. There are various types of malicious code we will encounter, including Viruses, Trojan horses, Logic bombs, and Worms.

A computer program is a sequence of symbols that are caused to achieve a desired functionality; the program is termed malicious when their sequences of instructions are used to intentionally cause adverse affects to the system. In the other words we can't call any "bug" as a Malicious Code. Malicious codes are also called programmed threats. The following figure provides an overall taxonomy of Malicious Code.



As presented in the above figure, threats can be divided into two categories:

- Independents: are self contained program that can be scheduled and ran by the operating system.

- Needs host program: are essentially fragments of programs that can not exist independently of some actual application program, utility or system program.

Vulnerability to Malicious code (Malware)

Various factors make a system more vulnerable to malware:

- **Homogeneity** – e.g. when all computers in a network run the same OS, if you can hack that OS, you can break into any computer running it.
- **Defects** – most systems containing errors which may be exploited by malware.
- **Unconfirmed code** – code from a floppy disk, CD-ROM or USB device may be executed without the user's agreement.
- **Over-privileged users** – some systems allow all users to modify their internal structures.
- **Over-privileged code** – most popular systems allow code executed by a user all rights of that user.

Trojan Horse

A worm is a program that can replicate itself and send copies from computers across network connections.

A *Trojan Horse* is a program with an overt (documented or known) effect and a covert (undocumented or unexpected) effect. Dan Edwards was the first to use this term.

A Trojan horse is a useful, or apparently useful, program or command procedure containing hidden code that, when invoked, performs some unwanted or harmful actions

Trojan horses are impostors—files that claim to be something desirable but, in fact, are malicious. Trojan horses contain malicious code that when triggered cause loss, or even theft, of data. For a Trojan horse to spread, we must invite these programs onto our computers; for example, by opening an email attachment or downloading and running a file from the Internet. Trojan.Vundo is a Trojan horse.

To replicate itself, a network worm uses some sort of network vehicle. Examples include the following

- **Electronic mail facility** : A worm mails a copy of itself to other systems.
- **Remote execution capability** : A worm executes a copy of itself on another system.
- **Remote login capability** : A worm logs on a remote system as a user and then uses commands to copy itself from one system to the other.

When a Trojan is activated on computer, the results can vary. Some Trojans are designed to be more annoying than malicious (like changing your desktop, adding silly active desktop icons) or they can cause serious damage by deleting files and destroying information on our system. Trojans are also known to create a backdoor on our computer that gives malicious users access to our system, possibly allowing confidential or personal information to be compromised.

Example: In the example above, the overt purpose is to list the files in a directory. The covert purpose is to create a shell that is setuid to the user executing the script. Hence, this program is a Trojan horse.

Example: A program named "waterfalls.scr" serves as a simple example of a trojan horse. The author claims it is a free waterfall screensaver. When run, it instead unloads hidden programs, commands, scripts, or any number of commands without the user's knowledge or consent.

Example: The NetBus program is designed to control a Windows NT workstation remotely. Victim downloads and installs this that is usually disguised as a game program, or in other fun programs. It acts as a server, accepting and executing commands for remote administrator which includes intercepting keystrokes and mouse motions and sending them to attacker and also allows attacker to upload, download files.

Trojan horses can make copies of themselves. One of the earliest Trojan horses was a version of the game *animal*. When this game was played, it created an extra copy of itself. These copies spread, taking up much room. The program was modified to delete one copy of the earlier version and create two copies of the modified program. After a preset date, each copy of the later version deleted itself after it was played.

A *propagating Trojan horse* (also called a *replicating Trojan horse*) is a Trojan horse that creates a copy of itself.

Trojan horses are broken down in classification based on how they breach systems and the damage they cause. The seven main types of Trojan horses are:

- Remote Access Trojans
- Data Sending Trojans
- Destructive Trojans
- Proxy Trojans
- FTP Trojans
- Security software disabler Trojans
- Denial-of-service attack (DoS) Trojans

Computer Worms

A computer virus infects other programs. A variant of the virus is a program that spreads from computer to computer, producing copies of itself on each one. A *computer worm* is a program that copies itself from one computer to another. Unlike a virus, it does not need to attach itself to an existing program. Worms spread by exploiting vulnerabilities in operating systems.

A Worm uses computer networks to replicate itself. It searches for servers with security holes and copies itself there. It then begins the search and replication process again

Research into computer worms began in the mid-1970s. Schoch and Hupp developed distributed programs to do computer animations, broadcast messages, and perform other computations. These programs probed workstations. If the workstation was idle, the worm copied a segment onto the system. The segment was given data to process and communicated with the worm's controller. When any activity other than the segment's began on the workstation, the segment shut down.

Example: Internet Worm of 1988 targeted Berkeley, Sun UNIX systems entered the Internet; within hours, it had rendered several thousand computers unusable. It used virus-like attack to inject instructions into running program and run them. To recover from this the machines had to disconnect system from Internet and reboot. To prevent re-infection, several critical programs had to be patched, recompiled, and reinstalled. The only way to determine if the program had suffered other malicious side effects was to disassemble it. Fortunately, the only purpose of this virus turned out to be self-propagation.

Example: The *Father Christmas* worm was interesting because it was a form of macro worm. It was distributed in 1987 and was designed for IBM networks. It was an electronic letter instructing recipient to save it and run it as a program that drew Christmas tree, printed "Merry Christmas!" It also checked address book, list of previously received email and sent copies to each address. The worm quickly overwhelmed the IBM networks and forced the networks and systems to be shut down. This worm had the characteristics of a macro worm. It was written in a high-level job control language, which the IBM systems interpreted.

Worms with good intent

The *Nachi* family of worms, for example, tried to download and install patches from Microsoft's website to fix vulnerabilities in the host system — by exploiting those *same* vulnerabilities. In practice, although this may have made these systems more secure, it generated considerable network traffic, rebooted the machine in the course of patching it, and did its work without the consent of the computer's owner or user.

In 1982, at the *Xerox Park* research institute, a worm was created to find idle machines. It was used to distribute workloads and was not a malicious worm. So worms can be helpful.

Types of Worms:

1. Electronic mail facility: A worm mails a copy of itself to other system.

2. Remote execution capability: A worm executes a copy of itself on another system.

3. Remote login capability: A worm logs onto a remote system as a user and then uses commands to copy itself from one system to the other.

Computer Viruses

When the Trojan horse can propagate freely and insert a copy of itself into another file, it becomes a computer virus. A *computer virus* is a program that inserts itself into one or more files and then performs some (possibly null) action. Computer virus works in two phases. The first phase, in which the virus inserts itself into a file, is called the insertion phase. The second phase, in which it performs some action, is called the execution phase. The following pseudo-code fragment shows how a simple computer virus works.

```
beginvirus:
  if spread-condition then begin
     for some set of target files do begin
        if target is not infected then begin
           determine where to place virus instructions
           copy instructions from beginvirus to endvirus into target
           alter target to execute added instructions
        end;
     end;
  end;
  perform some action(s)
  goto beginning of infected program
endvirus:
```

Cryptanalytic Attacks **(asked many times in exam)**

Cryptanalysis (from the Greek *kryptós*, "hidden", and *analýein*, "to loosen" or "to untie") is the study of methods for obtaining the meaning of encrypted information, without access to the secret information which is normally required to do so. Typically, this involves finding a secret key.

Cryptanalysis usually excludes methods of attack that do not primarily target weaknesses in the actual cryptography, such as bribery, physical coercion, burglary, keystroke logging, and social engineering, although these types of attack are an important concern and are often more effective than traditional cryptanalysis.

Cryptanalysis can be performed under a number of assumptions about how much can be observed or found out about the system under attack. It it is normally assumed that the general algorithm is known; this is Kerckhoffs' principle of "the enemy knows the system". There can be many types of attacks and broadly we categorize them as attack models:

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

Bruit Force attacks

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.

Rotor machines

See text Book

Chapter2
Basic of Modern cryptography

One Way Function

A trapdoor function' is a function that is easy to compute in one direction, yet believed to be difficult to compute in the opposite direction (finding its inverse) without special information, called the "trapdoor". Trapdoor functions are widely used in cryptography.

An example of a simple mathematical trapdoor is "6895601 is the product of two prime numbers. What are those numbers?"
- A typical solution would be to try dividing 6895601 by several prime numbers until finding the answer. However, if one is told that 1931 is part of the answer, one can find the answer by entering "6895601 ÷ 1931" into any calculator.

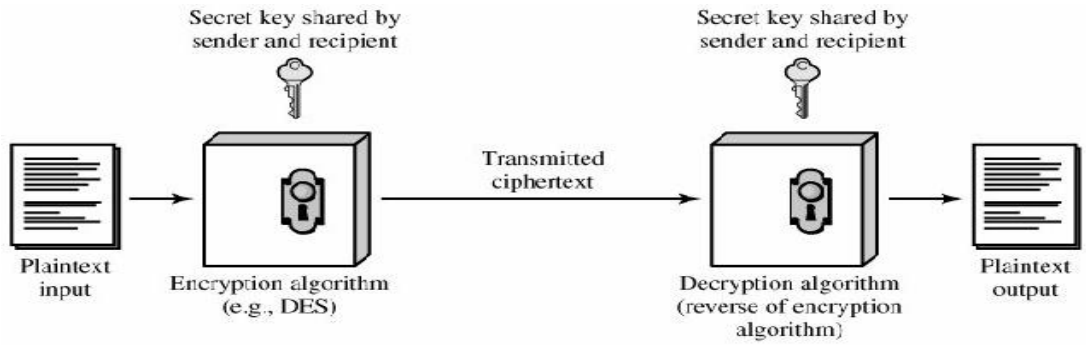
This example is not a sturdy trapdoor function--modern computers can guess all of the possible answers within a second--but this sample problem could be improved by using the product of two much larger primes.

Symmetric and Asymmetric cryptography

Symmetric Cipher Model

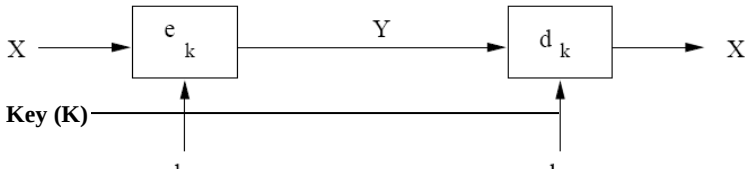
A symmetric encryption scheme has five ingredients as shown in figure:

1. Plaintext: This is the original intelligible message or data that is fed into the algorithm as input.
2. Encryption algorithm: The encryption algorithm performs various substitutions and transformations on the plaintext.
3. 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.
4. 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.
5. Decryption algorithm: This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.



Asymmetric Cipher model

Review of Symmetric Cryptography



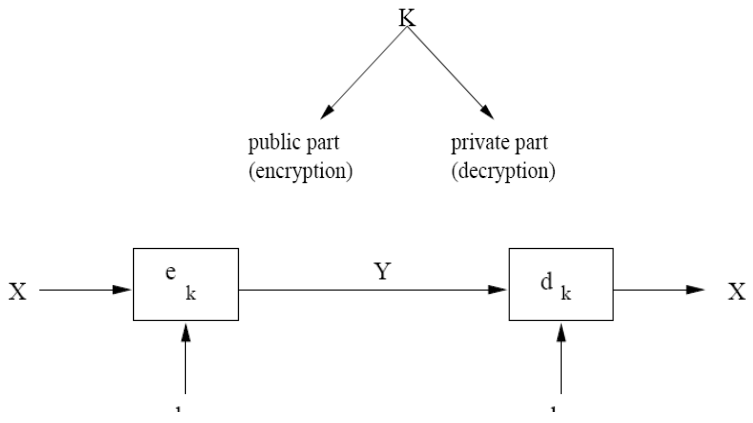
Two properties of symmetric-key schemes:
1. The algorithm requires same secret key for encryption and decryption.
2. Encryption and decryption are essentially identical (symmetric algorithms).

Analogy for symmetric key algorithms: Symmetric key schemes are like a safe box with a strong lock. Everyone with the key can deposit and retrieve messages.

Main problems with symmetric key schemes are:
1. Requires secure transmission of secret key.
2. In a network environment, each pair of users has to have a different key resulting in too many keys (n (n -1)/ 2 key pairs.

New Idea: Make a slot in the safe box so that everyone can deposit a message, but only the receiver can open the safe and look at the content of it. Idea: **Split key**.

Public Key Cryptography Protocol (Model)



Encryption Key (K_E) **Decryption Key (K_D)**

1. Alice and Bob agree on a public-key cryptosystem.
2. Bob sends Alice his public key.
3. Alice encrypts her message with Bob's public key and sends the ciphertext.
4. Bob decrypts ciphertext using his private key.

Mechanisms that can be realized with public-key algorithms

1. Key establishment protocols (e.g., Diffie-Hellman key exchange) and key transport protocols (e.g., via RSA) without prior exchange of a joint secret.
2. Digital signature algorithms (e.g., RSA, DSA)
3. Encryption

There are three families of Public-Key (PK) algorithms of practical relevance:

1. Integer factorization algorithms (RSA, ...)
2. Discrete logarithms (Diffie-Hellman, DSA, ...)
3. Elliptic curves (EC)

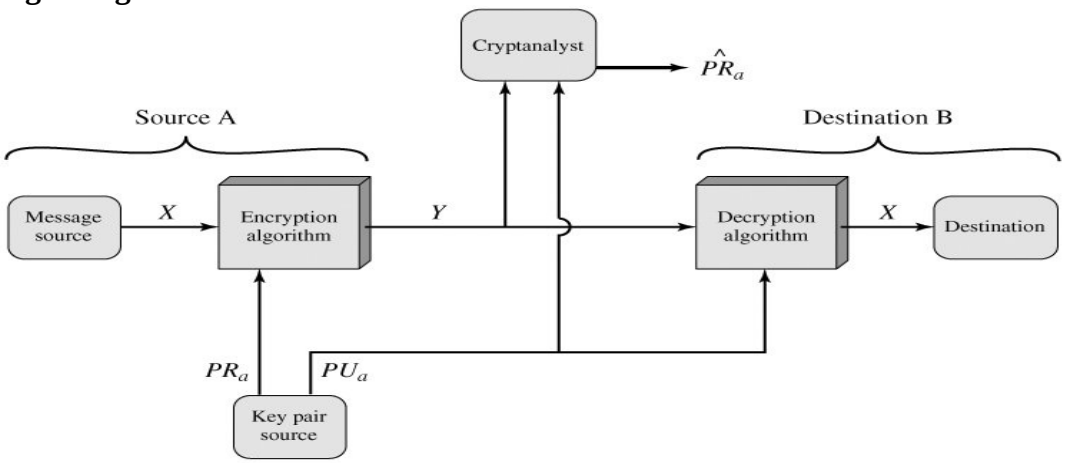
In this lecture we only consider Algorithms of family i.e. Integer Factorization Algorithm and Discrete Logarithms.

Comparison between Symmetric and asymmetric

	Secret Key (Symmetric)	**Public Key (Asymmetric)**
Number of keys	1	2
Protection of key	Must be kept secret	One key must be kept secret; the other can be freely exposed
Best uses	Cryptographic workhorse; secrecy and integrity data—single characters to blocks of data, messages, files	Key exchange, authentication
Key	Must be out-of-band	Public key can be used to distribute

	Secret Key (Symmetric)	Public Key (Asymmetric)
distribution		other keys
Speed	Fast	Slow; typically, 10,000 times slower than secret key

Digital signature



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 the message 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 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 **authenticato**r, must have the property that it is infeasible to change the document without changing the authenticator.

If the authenticator is encrypted with the sender's private key, it serves as a signature that verifies origin, content, and sequencing. This is called **Hash Authentication** and we discuss it later in detail.

Chapter 3
Conventional Encryption / Secret Key Cryptography

Stream 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 auto keyed 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. Using some of the modes of operation explained in Chapter 6, a block cipher can be used to achieve the same effect as a stream cipher.

Stream V/s Block Ciphers

Advantages of Stream Ciphers

- Speed of transformation. Because each symbol is encrypted without regard for any other plaintext symbols, each symbol can be encrypted as soon as it is read. Thus, the time to encrypt a symbol depends only on the encryption algorithm itself, not on the time it takes to receive more plaintext.

- Low error propagation. Because each symbol is separately encoded, an error in the encryption process affects only that character.

Disadvantages of Stream Ciphers

- Low diffusion. Each symbol is separately enciphered. Therefore, all the information of that symbol is contained in one symbol of the ciphertext.

- Susceptibility to malicious insertions and modifications. Because each symbol is separately enciphered, an active interceptor who has broken the code can splice together pieces of previous messages and transmit a spurious new message that may look authentic.

Advantages of Block Ciphers

- High diffusion. Information from the plain-text is diffused into several ciphertext symbols. One ciphertext block may depend on several plaintext letters.

- Immunity to insertion of symbols. Because blocks of symbols are enciphered, it is impossible to insert a single symbol into one block. The length of the block would then be incorrect, and the decipherment would quickly reveal the insertion.

Disadvantages of Block Ciphers

- Slowness of encryption. The person or machine using a block cipher must wait until an entire block of plaintext symbols has been received before starting the encryption process.

- Error propagation. An error will affect the transformation of all other characters in the same block.

The Feistel Cipher

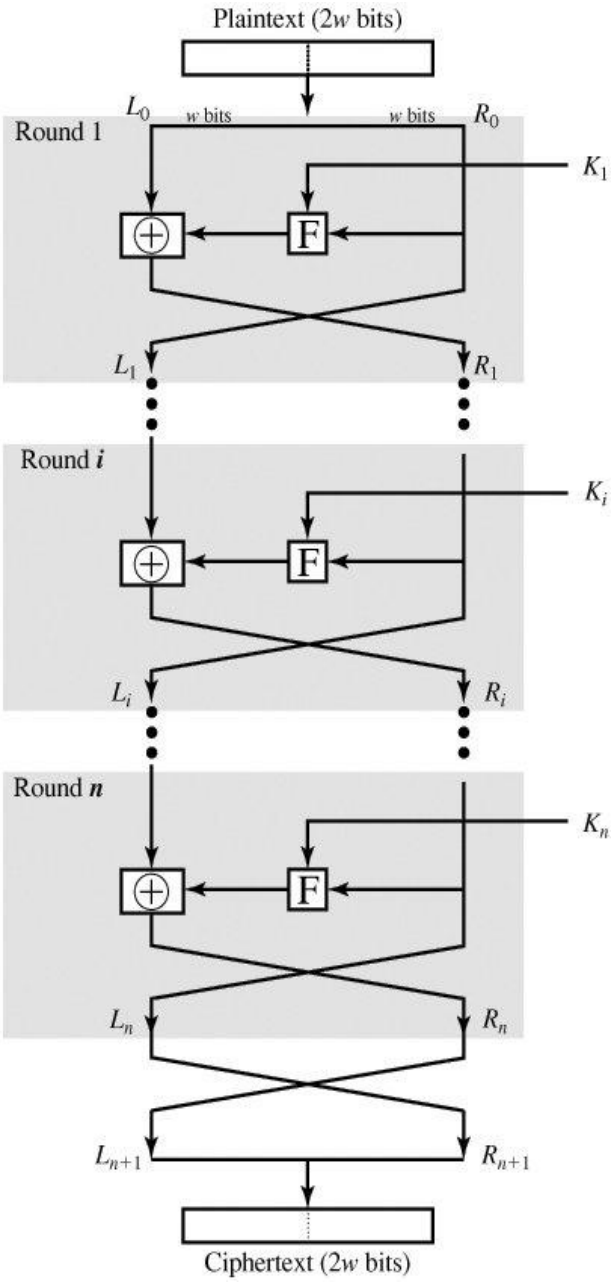
Feistel proposed that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. The essence of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

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.

Feistel Cipher Structure

Figure 3.2 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.



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 K_i. Following this substitution, a **permutation** is performed that consists of the interchange of the two halves of the data.[4] 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, Key size, Number of rounds, Subkey generation algorithm, Round function, Fast software encryption/decryption, Ease of analysis.

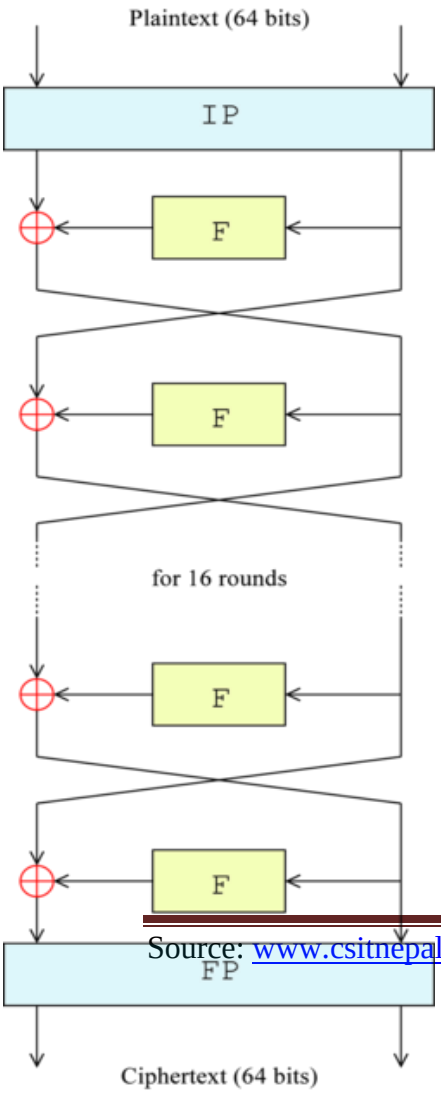
Data Encryption Standard (DES)

The *Data Encryption Standard (DES)* is a cipher (a method for encrypting information) selected as an official Federal Information Processing Standard (FIPS) for the United States in 1976, and which has subsequently enjoyed widespread use internationally. The algorithm was initially controversial, with classified design elements, a relatively short key length, and suspicions about a National Security Agency (NSA) backdoor. DES consequently came under intense academic scrutiny, and motivated the modern understanding of block ciphers and their cryptanalysis. DES

is now considered to be insecure for many applications. This is chiefly due to the 56-bit key size being too small; in January, 1999, distributed.net and the Electronic Frontier Foundation collaborated to publicly break a DES key in 22 hours and 15 minutes.

Description: DES is the block cipher - an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bitstring of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits, and it is usually quoted as such.

Structure: The algorithm's overall structure is shown in Figure below there are 16 identical stages of processing, termed rounds. There is also an initial and final permutation, termed IP and FP (see appendix for IP and FP scheme), which are inverses (IP "undoes" the action of FP, and vice versa). IP and FP have almost no cryptographic significance, but were apparently included in order to facilitate loading blocks in and out of mid-1970s hardware, as well as to make DES run slower in software.
Before the main rounds, the block is divided into two 32-bit halves and processed alternately; this criss-crossing is known as the Feistel scheme. The Feistel structure ensures that decryption and encryption are very similar processes - the only difference is that the subkeys are applied in the reverse order when decrypting. The rest of the algorithm is identical. This greatly simplifies implementation, particularly in hardware, as there is no need for separate encryption and decryption algorithms.



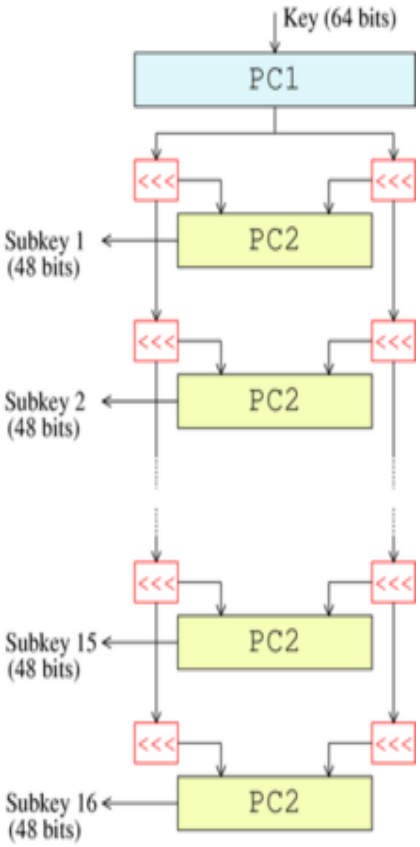
The ⊕ symbol denotes the exclusive-OR (XOR) operation. The F-function scrambles half a block together with some of the key. The output from the F-function is then combined with the other half of the block, and the halves are swapped before the next round. After the final round, the halves are not swapped; this is a feature of the Feistel structure which makes encryption and decryption similar processes.

The Feistel (F) function (Mangler Function): The F-function, depicted in Figure below, operates on half a block (32 bits) at a time and consists of four stages:
Expansion: the 32-bit half-block is expanded to 48 bits using the expansion permutation (see appendix for expansion permutation), denoted E in the diagram, by duplicating some of the bits.
Key Mixing: the result is combined with a subkey using an XOR operation. Sixteen 48-bit subkeys - one for each round - are derived from the main key using the key schedule described below.
Substitution: after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by the S-boxes, or substitution boxes. Each of the eight S-boxes replaces its six input bits with four output bits according to a non-linear transformation, provided in the form of a lookup table (see appendix for table). The S-boxes provide the core of

the security of DES - without them, the cipher would be linear, and trivially breakable.
Permutation: finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation, the P-box (see appendix for permutation P).

The alternation of substitution from the S-boxes, and permutation of bits from the P-box and E-expansion provides so-called "confusion and diffusion" respectively, a concept identified by Claude Shannon in the 1940s as a necessary condition for a secure yet practical cipher.

Key Schedule: Figure below illustrates the key schedule for encryption - the algorithm which generates the subkeys. Initially, 56 bits of the key are selected from the initial 64 by Permuted Choice 1, PC-1(see appendix for PC1) - the remaining eight bits are either discarded or used as parity check bits. The 56 bits are then divided into two 28-bit halves; each half is thereafter treated separately. In successive rounds, both halves are rotated left by one or two bits specified for each round (see appendix for key rotation schedule), and then 48 subkey bits are selected by Permuted Choice 2, PC-2(see appendix for PC2) - 24 bits from the left half, and 24 from the right. The rotations (denoted by "<<<" in the diagram) mean that a different set of bits is used in each subkey; each bit is used in approximately 14 out of the 16 subkeys.

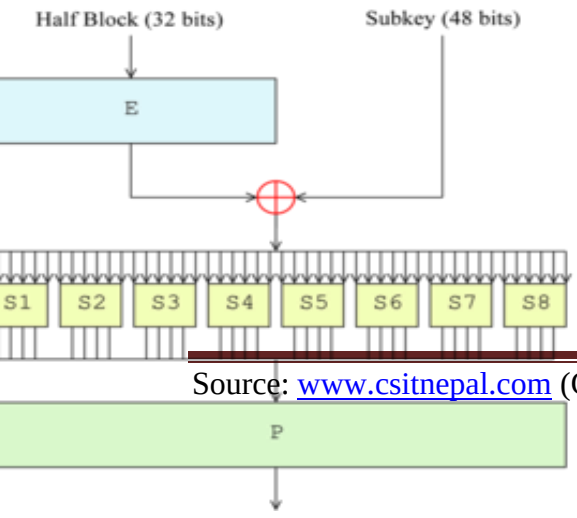


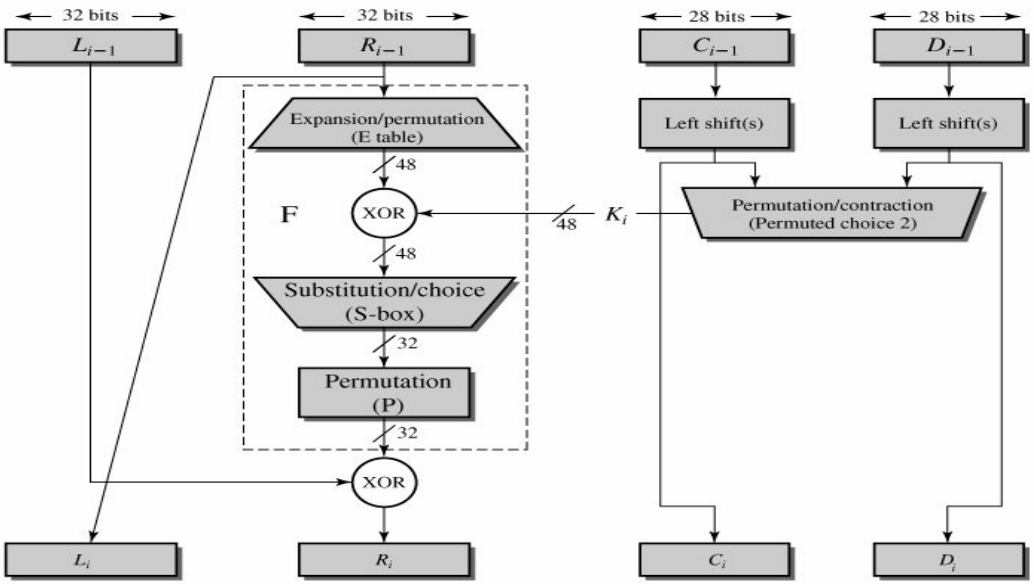
The key schedule for decryption is similar - the subkeys are in reverse order compared to encryption. Apart from that change, the process is the same as for encryption.

Weak and Semi-Weak Keys: There are sixteen DES keys that are not suggested for use. However the probability of getting such keys is very small as given by $16/2^{56}$. The sixteen weak keys are keys with all ones, all zeroes, alternating zeroes and ones, and alternating ones and zeroes for two 28 bits parts of the key generated when PC-1 is used.

Details of Single Round

Figure 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:





Note: see the table used in this algorithm in Text Book.

Security and cryptanalysis of DES:

Although more information has been published on the cryptanalysis of DES than any other block cipher, the most practical attack to date is still a brute force approach. Various minor cryptanalytic properties are known, and three theoretical attacks are possible which, while having a theoretical complexity less than a brute force attack, require an unrealistic amount of known or chosen plaintext to carry out, and are not a concern in practice. In spite of all the criticism and weaknesses of DES, there is no known example of anyone actually suffering monetary losses because of DES security limitations.

Brute force attack: For any cipher, the most basic way of attack is brute force - trying every possible key. The length of the key gives the number of possible keys, and hence the feasibility of this approach. For DES, questions were raised about the adequacy of its key size early on, even before it was adopted as a standard, and it was the small key size, rather than theoretical cryptanalysis, which dictated a need for a replacement algorithm. It is known that the NSA encouraged, if not persuaded, IBM to reduce the key size from 128 to 64 bits, and from there to 56 bits; this is often taken as an indication that the NSA thought it would be able to break keys of this length even in the mid-1970s.

2. Advanced Encryption Standard (AES)

The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. A number of AES parameters depend on the key length

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

AES Encryption Decryption Technique

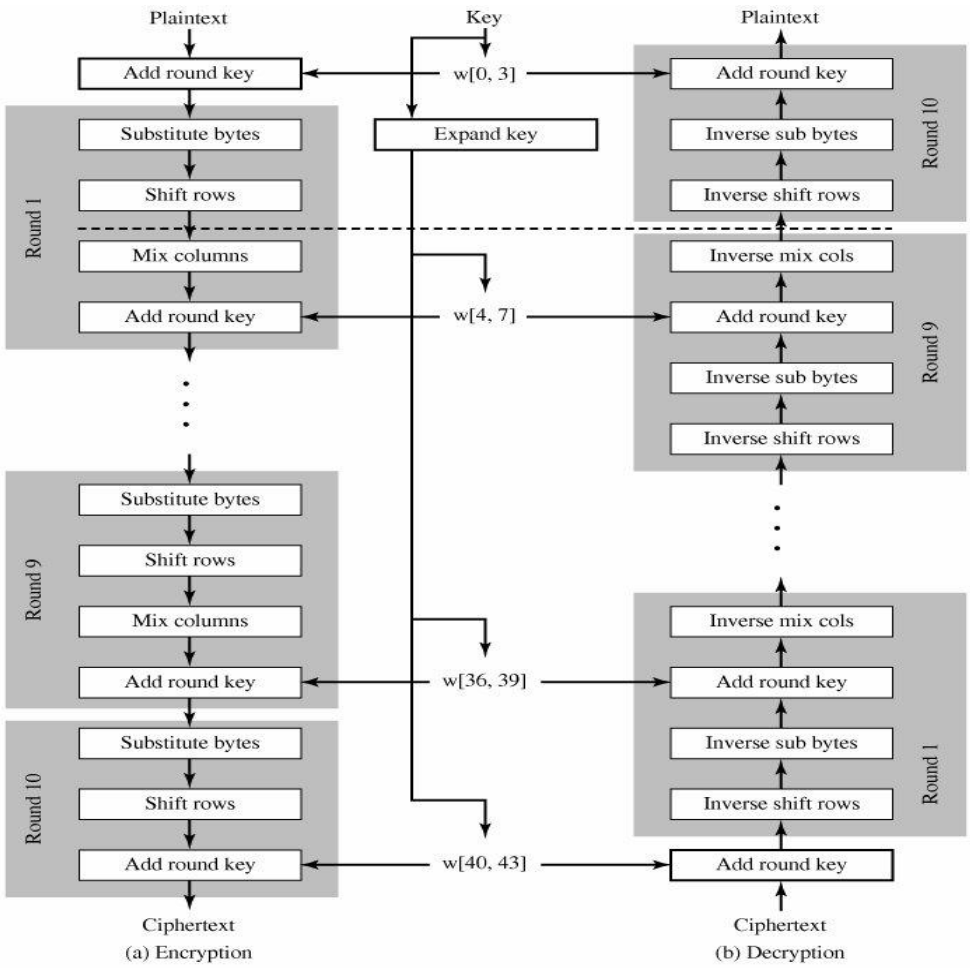
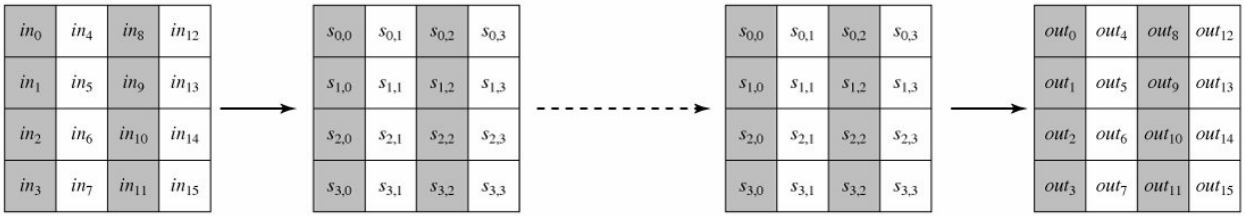


Figure 1 : The encryption/decryption Rounds on AES

This figure shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix.

AES data structures: the following matrixes are used in AES encryption and Decryptions



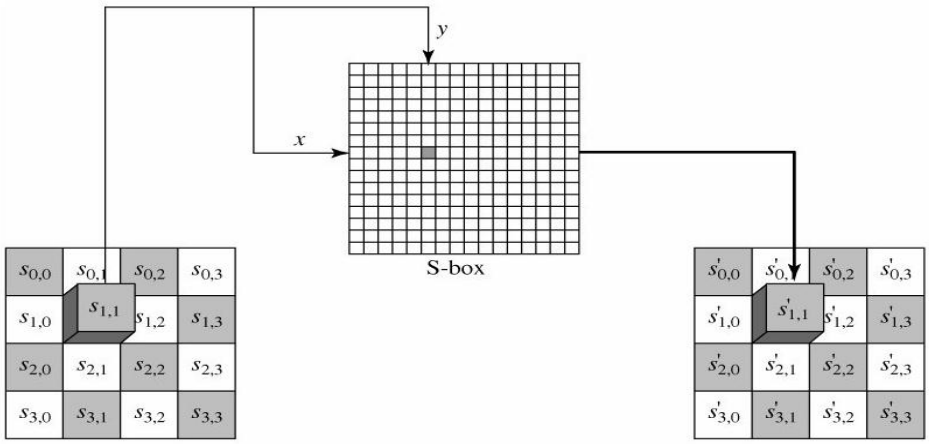
(a) Input, state array, and output

(b) Key and expanded key

The structure of AES is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages as shown in figure 1 above.

The four stages are:
Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
ShiftRows: A simple permutation
MixColumns: A substitution that makes use of arithmetic over GF(2^8)
AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key

a) Substitute Bytes Transformation
The forward substitute byte transformation, called SubBytes, is a simple table lookup (Figure 5.4a). AES defines a 16 x 16 matrix of byte values, called an S-box (see table on Text Book) that contains a permutation of all possible 256 8-bit values. Each individual byte of State is mapped into a new byte in the following way:
The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

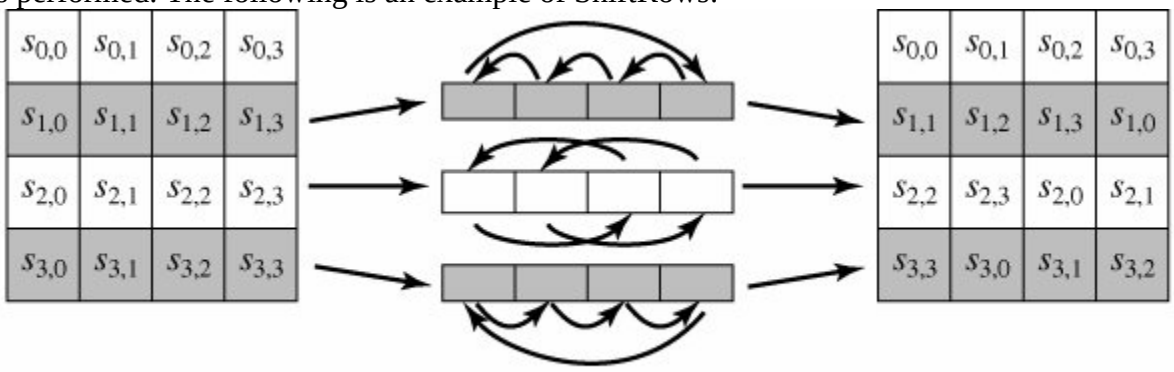


(a) Substitute byte transformation

The inverse substitute byte transformation, called InvSubBytes, makes use of the inverse S-box shown in (Text book). For example, that the input {2A} produces the output {95} and the input {95} to the S-box produces {2A}.

b) ShiftRows Transformation

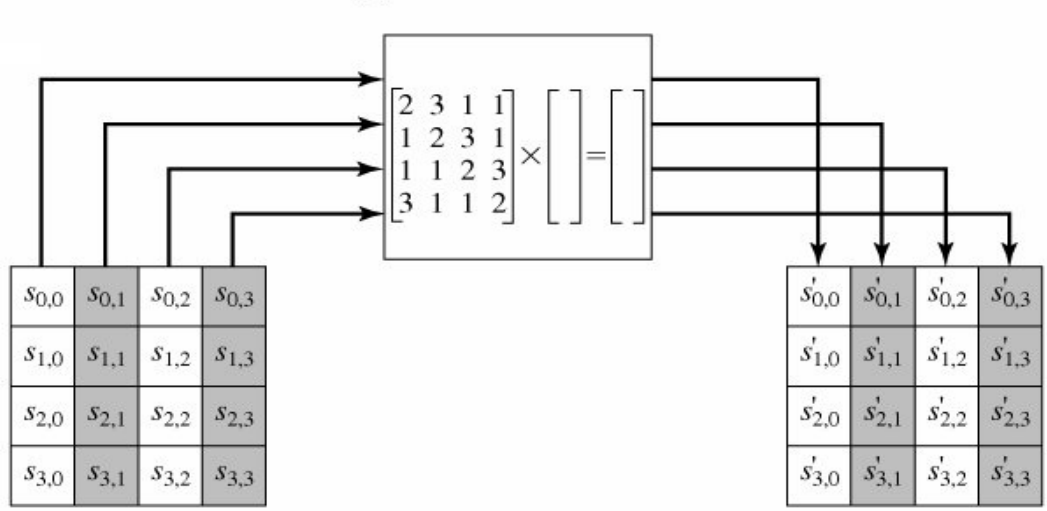
The forward shift row transformation, called ShiftRows, is depicted in Figure below. The first row of State is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of ShiftRows:



(a) Shift row transformation

c) MixColumns Transformation

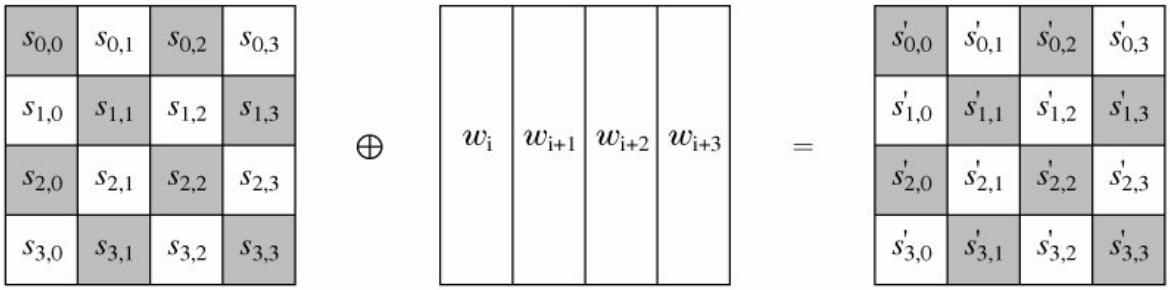
The forward mix column transformation, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column.



(b) Mix column transformation

d) AddRoundKey Transformation

In the forward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key. As shown in Figure below, the operation is viewed as a columnwise operation between the 4 bytes of a State column and one word of the round key; it can also be viewed as a byte-level operation.



(b) Add Round Key Transformation

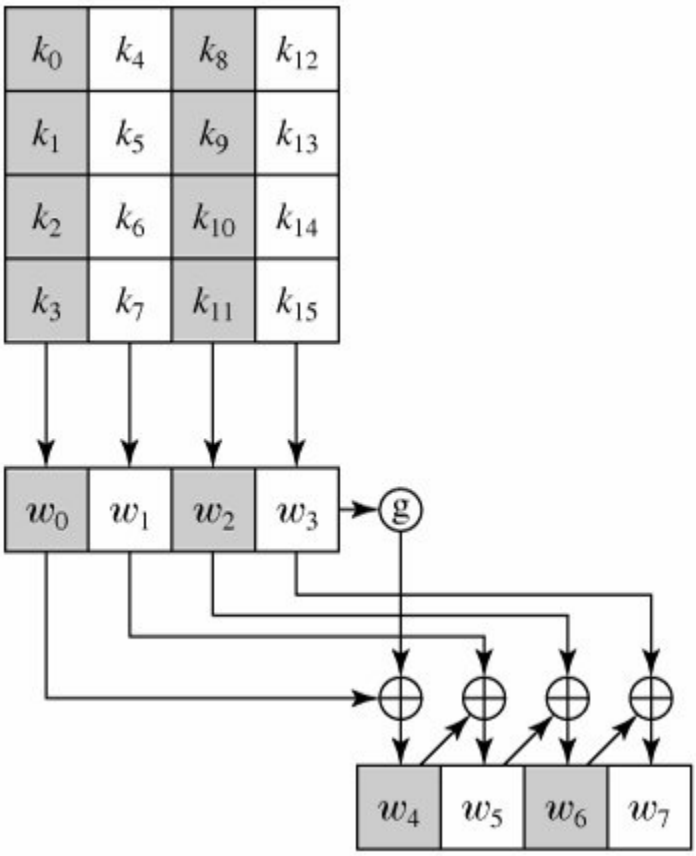
AES Key Expansion Algorithm

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The following pseudocode describes the expansion:

```
KeyExpansion (byte key[16], word w[44])
{
   word temp
   for (i = 0; i < 4; i++)
            w[i] = (key[4*i], key[4*i+1],key[4*i+2],key[4*i+3]);
   for (i = 4; i < 44; i++)
   {
    temp = w[i  1];
```

```
    if (i mod 4 = 0)
          temp = SubWord (RotWord (temp))ExOR Rcon[i/4];
    w[i] = w[i4]
   }
}
```

The key expansion technique is shown in figure below



Where g is a special function consist of following sub functions

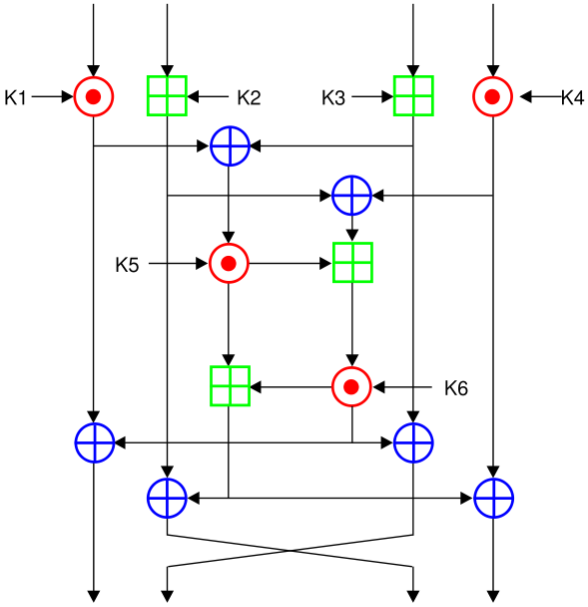
1. RotWord performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
2. SubWord performs a byte substitution on each byte of its input word, using the S-box.
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

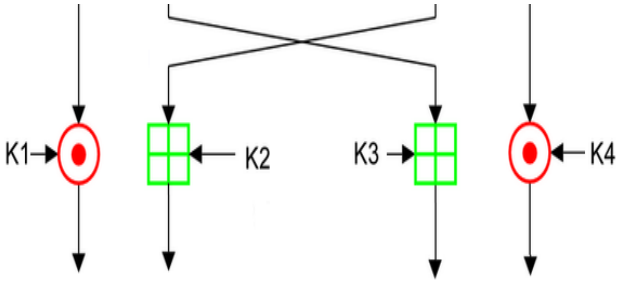
The round constant is a word in which the three rightmost bytes are always 0.(See the text book for more details)

3. International Data Encryption Algorithm (IDEA)

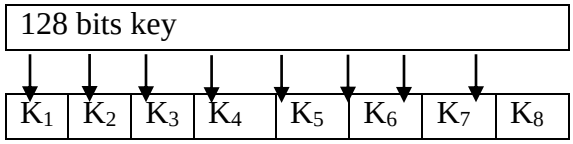
In cryptography, the IDEA is a block cipher designed by Xuejia Lai and James Massey and was first described in 1991. The algorithm was intended as a replacement for the Data Encryption Standard.

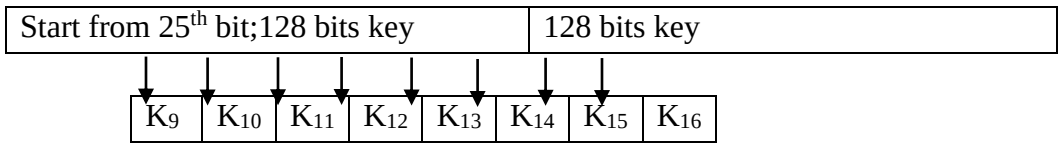
Operation: IDEA operates on 64-bit blocks using a 128-bit key, and consists of a series of eight identical rounds (see figure below) and an output round (the half-round, see figure below). The processes for encryption and decryption are similar. IDEA derives much of its security by interleaving operations from different groups - modular addition (addition mod 2^{16}, denoted by⊞), modular multiplication (multiplication mod $2^{16}+1$ denoted by ⊙, where the all-zero word (0x0000) is interpreted as 2^{16}),, and bitwise eXclusive OR (XOR) (denoted by ⊕), - which are algebraically "incompatible" in some sense. The blow figures in left shows a round (1-8) and in right shows final "half round.





Key schedule: The 128 bit key is used to generate 52 sub keys of length 16 bits. 6 keys are used for each round and the remaining 4 keys are used in final half round. The figure below shows the key generation mechanisms where we start getting the subkeys from the starting position of the 128 bits key, so in first round we get 8 subkeys. In each of the round of sub key generation 128 bits keys undergoes a 25 bit position right to generate next set of keys i.e. start with bit position 0, 25, 50 and so on until all 52 keys are derived.





Round Operations: It has been mentioned above that IDEA uses 8 full rounds and 1 half round. We now break the 8 full round and make it 16 rounds such that there are total 17 rounds where 9 odd rounds (1, 3, …., 17) are identical and 8 even rounds (2, 4, ….,16) are identical. Each odd round takes 4 subkeys and each even round takes 2 subkeys.

Odd Round: Odd round is simple and uses four keys where first and fourth keys are uses multiplication mod $2^{16}+1$ operation with first and fourth 16 bits fragment of 64 bits input block respectively. The second and third subkeys are subjected to addition mod 2^{16} with second and third bits fragment of 64 bits input block respectively. The output so obtained from operations with first and fourth input sub-block will be first and fourth input for next round and output from operations with second and third input sub-block will be reversed i.e. becomes third and second input for next round. If X_a, X_b, X_c, and X_d are input sub-blocks and K_a, K_b, K_c, and K_d are subkeys then we can write algebraic expression for odd round as:

$X_a = X_a \odot K_a$; $X_b = X_c \boxplus K_c$; $X_c = X_b \boxplus K_b$; $X_d = X_d \odot K_d$; (see figure above)

Even Round: Even round is bit complicated then the odd round. There are four input sub-blocks (X_a, X_b, X_c, and X_d) from the previous round and two subkeys (K_e and K_f). In even rounds first and second input sub-blocks are subjected to XOR operation to get single 16 bits output (say, Y_{in}) and third and fourth input sub-blocks are subjected to XOR operation to get single 16 bits output (say, Z_{in}). Y_{in} and Z_{in} are fed to mangler function along with K_e, and K_f to get outputs Y_{out} and Z_{out}, where Y_{out} is XOR'ed with X_a and X_b, to get first two input sub-blocks for next round and Z_{out} is XOR'ed with X_c and X_d, to get last two input sub-blocks for next round. Algebraic expressions for even round can be written as:

$Y_{in} = X_a \oplus X_b$; $Z_{in} = X_c \oplus X_d$; $Y_{out} = ((K_e \odot Y_{in}) \boxplus Z_{in}) \odot K_f$; $Z_{out} = (K_e \odot Y_{in}) \boxplus Y_{out}$;

$X_a = X_a \oplus Y_{out}$; $X_b = X_b \oplus Y_{out}$; $X_c = X_b \oplus Z_{out}$; $X_d = X_d \oplus Z_{out}$; (see figure above)

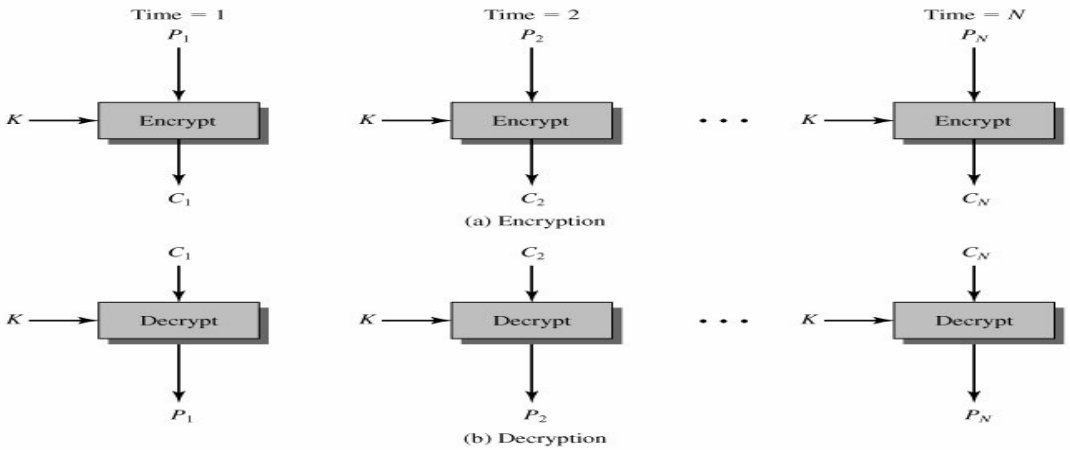
Security: The designers analyzed IDEA to measure its strength against differential cryptanalysis and concluded that it is immune under certain assumptions. No successful linear or algebraic weaknesses have been reported. Some classes of weak keys have been found but these are of little concern in practice, being so rare as to be unnecessary to avoid explicitly. As of 2004, the best attack which applies to all keys can break IDEA reduced to 5 rounds (the full IDEA cipher uses 8.5 rounds).

Modes of Operations

A mode of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream. The four modes are intended to cover virtually all the possible applications of encryption for which a block cipher could be used.

1. Electronic Codebook Mode

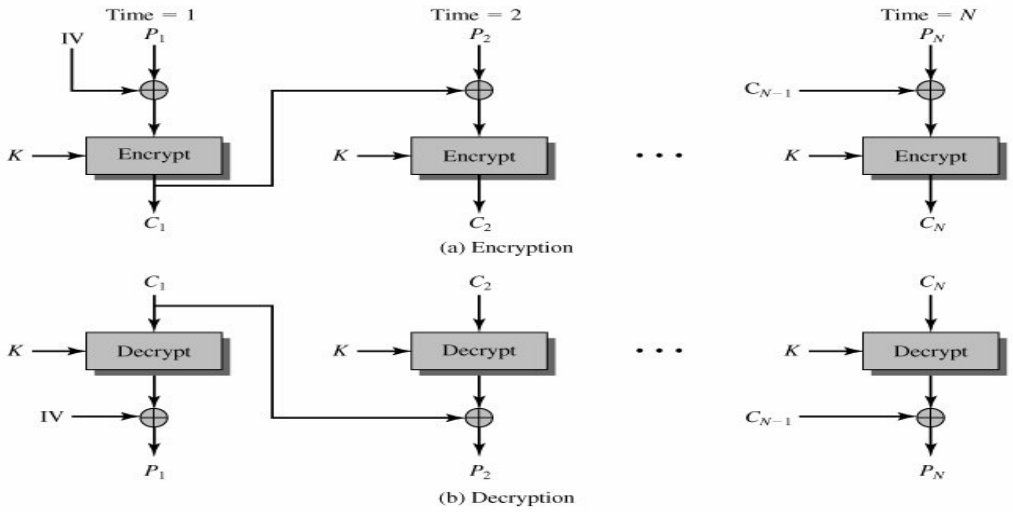
The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key The term codebook is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext. Therefore, we can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext.



The ECB method is ideal for a short amount of data, such as an encryption key. Thus, if you want to transmit a DES key securely, ECB is the appropriate mode to use. The most significant characteristic of ECB is that the same b-bit block of plaintext, if it appears more than once in the message, always produces the same ciphertext. For lengthy messages, the ECB mode may not be secure. If the message is highly structured, it may be possible for a cryptanalyst to exploit these regularities.

2. Cipher Block Chaining Mode

To overcome the security deficiencies of ECB, we would like a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. A simple way to satisfy this requirement is the cipher block chaining (CBC) mode. In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. In effect, we have chained together the processing of the sequence of plaintext blocks.



because of the chaining mechanism of CBC, it is an appropriate mode for encrypting messages of length greater than b bits.

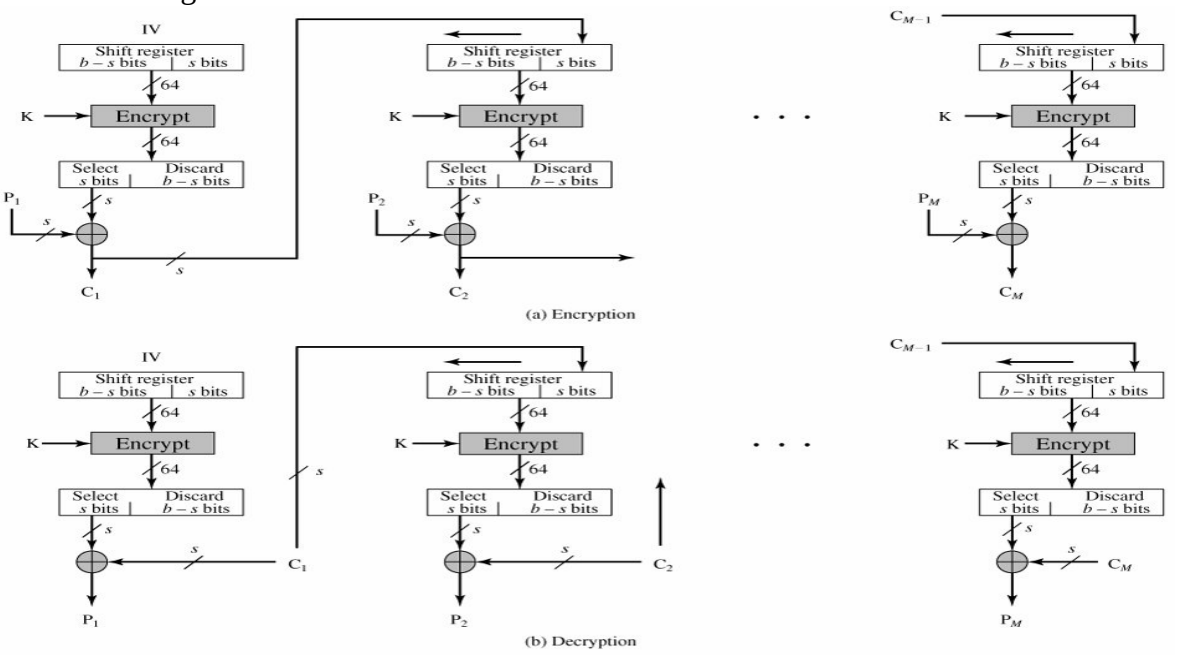
3. Cipher Feedback Mode

The DES scheme is essentially a block cipher technique that uses b-bit blocks. However, it is possible to convert DES into a stream cipher, using either the cipher feedback (CFB) or the

output feedback mode. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

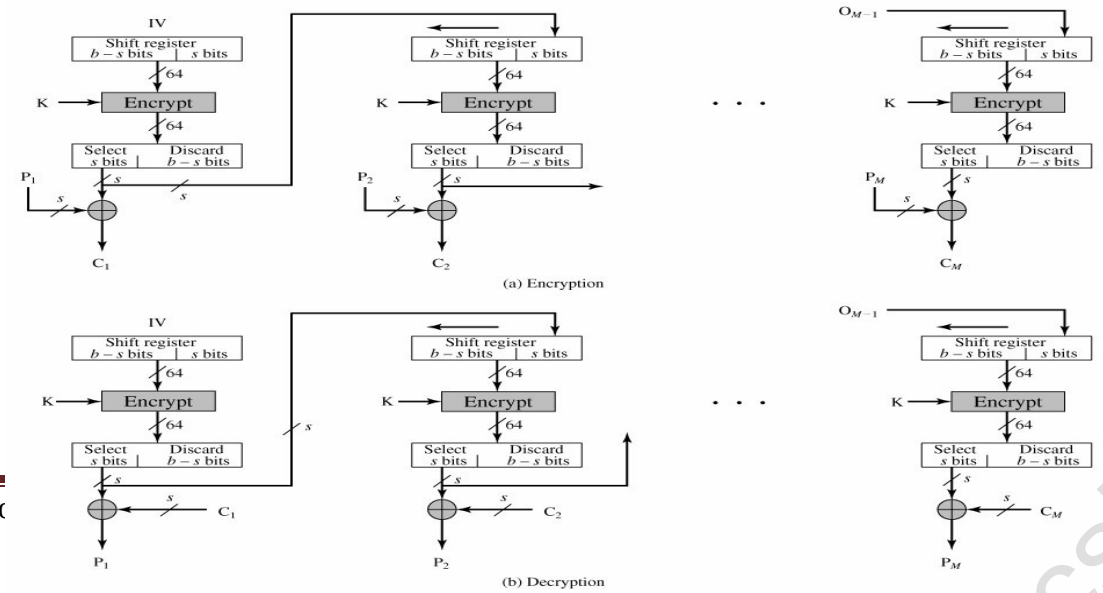
One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be encrypted to produce a cipher text output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.

In the figure, it is assumed that the unit of transmission is s bits; a common value is s = 8. As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext. In this case, rather than units of b bits, the plaintext is divided into segments of s bits.



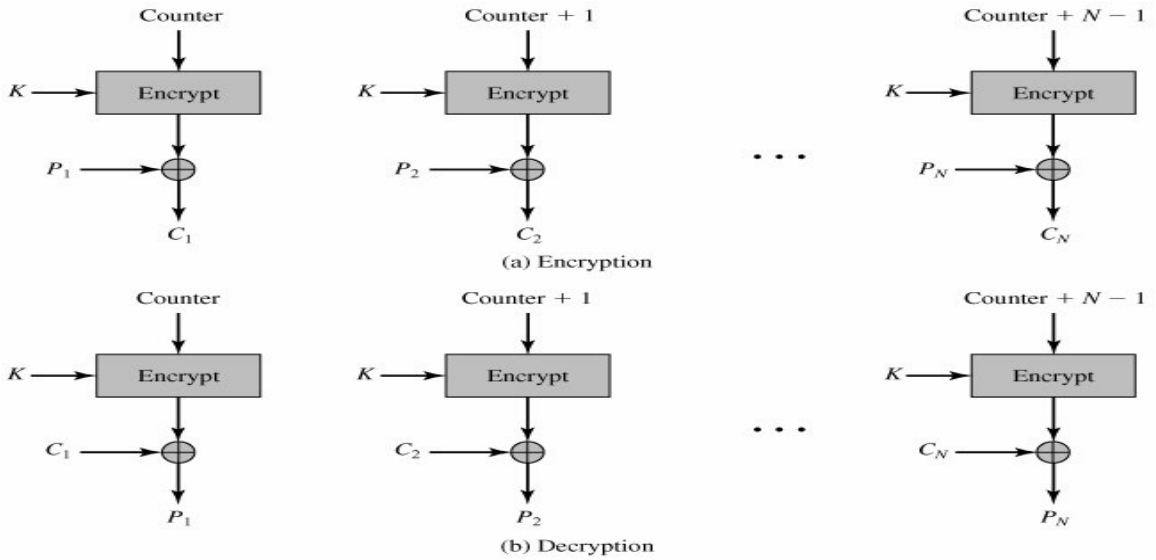
4. Output Feedback Mode

The output feedback (OFB) mode is similar in structure to that of CFB, as illustrated in figure. As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.



5. Counter mode

Figure depicts the CTR mode. A counter, equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^b where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block.



Old questions

1. How many rounds are used in AES and what does the number of rounds depend on?
2. What are the steps that go into the construction of the 16*16 S-box lookup table for AES algorithm?
3. What are the characteristics of a stream cipher?

Chapter 4
Public Key Cryptography

Basic Number theory

Groups

A **group** G, sometimes denoted by {G, ·} is a set of elements with a binary operation, denoted by ·, that associates to each ordered pair (a, b) of elements in G an element (a · b) in G, such that the following axioms are obeyed.[1]The operator · is generic and can refer to addition, multiplication, or some other mathematical operation.

(A1) Closure:	If a and b belong to G, then a · b is also in G.
(A2) Associative:	a · (b · c) = (a · b) · c for all a, b, c in G.
(A3) Identity element:	There is an element e in G such that a · e = e · 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 · a' = a' · a = e.

If a group has a finite number of elements, it is referred to as a **finite group**, and the **order** of the group is equal to the number of elements in the group. Otherwise, the group is an **infinite group**.

A group is said to be **abelian** if it satisfies the following additional condition:

(A5) Commutative: a · b = b · a for all a, b in G.

The set of integers (positive, negative, and 0) under addition is an abelian group.

Cyclic Group

We define exponentiation within a group as repeated application of the group operator, so that $a^3 = a \cdot a \cdot a$. Further, we define $a^0 = e$, the identity element; and $a^{-n} = (a')^n$. A group G is cyclic if every element of G is a power a^k (k is an integer) of a fixed element a ε G. The element a is said to generate the group G, or to be a **generator** of G. A cyclic group is always abelian, and may be finite or infinite.

The additive group of integers is an infinite cyclic group generated by the element 1. In this case, powers are interpreted additively, so that n is the nth power of 1.

Ring

A **ring** R, sometimes denoted by {R, +, x}, is a set of elements with two binary operations, called addition and multiplication, such that for all a, b, c in R the following axioms are obeyed:

(A1-A5) R is an abelian group with respect to addition; that is, R satisfies axioms A1 through A5. For the case of an additive group, we denote the identity element as 0 and the inverse of a as a.

(M1) Closure under multiplication:	If a and b belong to R, then ab is also in R.
(M2) Associativity of multiplication:	a(bc) = (ab)c for all a, b, c in R.

(M3) Distributive laws: $a(b + c) = ab + ac$ for all a, b, c in R.
$(a + b)c = ac + bc$ for all a, b, c in R.

With respect to addition and multiplication, the set of all n-square matrices over the real numbers is a ring.

Commutative Ring

A ring is said to be **commutative** if it satisfies the following additional condition:

(M4) Commutativity of multiplication: $ab = ba$ for all a, b in R.

Let S be the set of even integers (positive, negative, and 0) under the usual operations of addition and multiplication. S is a commutative ring. The set of all n-square matrices defined in the preceding example is not a commutative ring.

Integral Domain

we define an **integral domain**, which is a commutative ring that obeys the following axioms:

(M5) Multiplicative identity: There is an element 1 in R such that $a1 = 1a = a$ for all a in R.

(M6) No zero divisors: If a, b in R and $ab = 0$, then either $a = 0$ or $b = 0$.

Let S be the set of integers, positive, negative, and 0, under the usual operations of addition and multiplication. S is an integral domain.

Fields

A **field** F, sometimes denoted by {F, +, x}, is a set of elements with two binary operations, called addition and multiplication, such that for all a, b, c in F the following axioms are obeyed:

(A1M6) F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6.

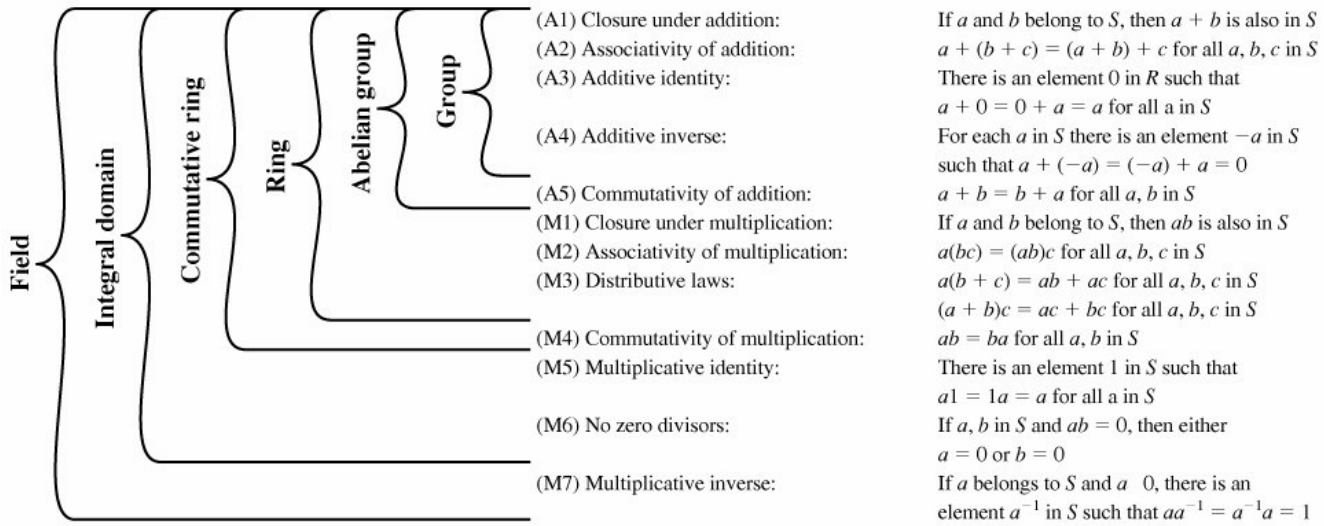
(M7) Multiplicative inverse: For each a in F, except 0, there is an element a^{-1} in F such that

$$aa^{-1} = (a^{-1})a = 1.$$

In essence, a field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set. Division is defined with the following rule: $a/b = a(b^{-1})$.

Familiar examples of fields are the ***rational numbers****, the* ***real numbers,*** *and the* ***complex numbers****.* Note that the set of **all integers** is not a field, because not every element of the set has

a multiplicative inverse; in fact, only the elements 1 and -1 have multiplicative inverses in the integers.



Modular Arithmetic

Given any positive integer n and any nonnegative integer a, if **we divide a by n,** we get an integer quotient **q** and an integer remainder **r** that obey the following relationship:

$$a=qn+r \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

If a is an integer and n is a positive integer, we define **a mod n** to be the remainder when a is divided by n. The integer n is called the **modulus**.

For example: 11 mod 7=4 and -11 mod 7 = 3

Two integers a and b are said to be **congruent modulo n**, if (a mod n) = (b mod n).

i.e. $\boldsymbol{a \equiv b \ (mod\ n)}$

Divisors

We say that a nonzero b divides a if a = mb for some m, where a, b, and m are integers. That is, b divides a if there is no remainder on division. The notation is commonly used to mean b divides a. Also, if b|a, we say that b is a divisor of a.

The following relations hold:

- If a|1, then a = ±1.
- If a|b and b|a, then a = ±b.
- Any b ≠0 divides 0.
- If b|g and b|h, then b|(mg + nh) for arbitrary integers m and n.

Properties of Congruences

Congruences have the following properties:

1. a ≡b (mod n) if n|(a-b).
2. a ≡b (mod n) implies b ≡a (mod n)..
3. a ≡b (mod n) and b ≡c (mod n) imply a ≡c (mod n).

To demonstrate the first point, if n|(a b), then (a b) = kn for some k. So we can write a = b + kn. Therefore, (a mod n) = (reminder when b + kn is divided by n) = (reminder when b is divided by n) = (b mod n)

Modular Arithmetic Operations

Modular arithmetic exhibits the following properties:

1. [(a mod n) + (b mod n)] mod n = (a + b) mod n
2. [(a mod n) - (b mod n)] mod n = (a- b) mod n
3. [(a mod n) x (b mod n)] mod n = (a x b) mod n

Examples

11 mod 8 = 3; 15 mod 8 = 7
[(11 mod 8) + (15 mod 8)] mod 8 = 10 mod 8 = 2 **(11 + 15) mod 8 = 26 mod 8 = 2**
[(11 mod 8) (15 mod 8)] mod 8 = 4 mod 8 = 4 **(11 15) mod 8 = 4 mod 8 = 4**
[(11 mod 8) x (15 mod 8)] mod 8 = 21 mod 8 = 5 **(11 x 15) mod 8 = 165 mod 8 = 5**

Exponentiation is performed by repeated multiplication, as in ordinary arithmetic.

To find 11^7 mod 13, we can proceed as follows:
$11^2 = 121 \equiv 4 \pmod{13}$
$11^4 = (11^2)^2 \equiv 4^2 \equiv 3 \pmod{13}$
$11^7 \equiv 11 \times 4 \times 3 \equiv 132 \equiv 2 \pmod{13}$

Arithmetic Modulo 8

+	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

(a) Addition modulo 8

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

(b) Multiplication modulo 8

w	$-w$	w^{-1}
0	0	—
1	7	1
2	6	—
3	5	3
4	4	—
5	3	5
6	2	—
7	1	7

(c) Additive and multiplicative inverses modulo 8

Here –w is additive inverse of w and w^{-1} is the multiplicative inverse of w.

Define the set Z_n as the set of nonnegative integers less than n

Z_n={0,1,2,3,…….n-1}.

This is referred to as the *set of residues*, or residue classes modulo n. To be more precise, each integer in Z_n represents a residue class. We can label the residue classes modulo n as *[0],* [1], [2],...,[n 1], where

[r] = {a: a is an integer, a≡ r (mod n)}

The residue classes *modulo 4* are

	[0] = { ..., 16, 12, 8, 4, 0, 4, 8, 12, 16,... }
	[1] = { ..., 15, 11, 7, 3, 1, 5, 9, 13, 17,... }
	[2] = { ..., 14, 10, 6, 2, 2, 6, 10, 14, 18,... }
	[3] = { ..., 13, 9, 5, 1, 3, 7, 11, 15, 19,... }

If we perform modular arithmetic within Z_n, the properties shown in Table (below) hold for integers in Z_n. Thus, *Z_n is a commutative ring with a multiplicative identity element.*

Commutative laws	(w + x) mod n = (x + w) mod n (w x x) mod n = (x x w) mod n
Associative laws	[(w + x) + y] mod n = [w + (x + y)] mod n [(w x x) x y] mod n = [w x (x x y)] mod n

Distributive laws	[w + (x + y)] mod n = [(w x x) + (w x y)] mod n [w + (x x y)] mod n = [(w + x) x (w + y)] mod n
Identities	(0 + w) mod n = w mod n (1 + w) mod n = w mod n
Additive inverse (-w)	For each w Z_n, there exists a z such that w + z ≡0 mod n

Proof that Z_8 is a ring

Finite Fields of Order p

For a given prime, p, the finite field of order p, GF(p) is defined as the set Z_p of integers {0, 1,..., p- 1}, together with the arithmetic operations modulo p. (GF stands for Galois field)

Arithmetic in GF(7)

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

(a) Addition modulo 7

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

(b) Multiplication modulo 7

w	$-w$	w^{-1}
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

(c) Additive and multiplicative inverses modulo 7

Finding the Multiplicative Inverse in GF(p)

It is easy to find the multiplicative inverse of an element in GF(p) for small values of p. You simply construct a multiplication table, such as shown in Table above, and the desired result can be read directly. However, for large values of p, this approach is not practical.

If gcd(m, b) = 1, then b has a multiplicative inverse modulo m. That is, for positive integer b < m, there exists a b1 < m such that bb1 = 1 mod m. The Euclidean algorithm can be extended so that, in addition to finding gcd(m, b), if the gcd is 1, the algorithm returns the multiplicative inverse of b.

```
EXTENDED EUCLID(m, b)
1. (A1, A2, A3) ←(1, 0, m); (B1, B2, B3) ←(0, 1, b)
2. if B3 = 0 return A3 = gcd(m, b); no inverse
3. if B3 = 1 return B3 = gcd(m, b); B2 = b^1 mod m
```

4. $Q=\left\lfloor \frac{A3}{B3} \right\rfloor$

```
5. (T1, T2, T3) ← (A1  QB1, A2  QB2, A3  QB3)
6. (A1, A2, A3) ← (B1, B2, B3)
7. (B1, B2, B3) ← (T1, T2, T3)
8. goto 2
```

Exercise: trace the above algorithms for finding multiplicative inverse of 550 in GF(1759)

Prime Numbers

Prime Numbers
An integer p > 1 is a prime number if and only if its only divisors are ± 1 and ±p. Examples are 7 , 13…

Any integer a > 1 can be factored in a unique way as:
$A=p1^{a1}.p2^{a2}.....pt^{at}$ where p1 < p2 < ... < pt are prime numbers and where each is a positive integer. This is known as *the fundamental theorem of arithmetic*. Examples are

91 = 7 x 13 (factorization)
3600 = 24 x 32 x 52
11011 = 7 x 112 x 13

Fermat's theorem
Fermat's theorem states the following: If p is prime and a is a positive integer not divisible by p, then

$a^{p-1}=1 \pmod p$

(here a and p are relatively prime)

Example

a = 7, p = 19
$7^2 = 49 \equiv 11 \pmod{19}$
$7^4 = 7^2 \times 7^2 \equiv 11 \times 11 = 121 \equiv 7 \pmod{19}$
$7^8 \equiv 49 \equiv 11 \pmod{19}$
$7^{16} = 121 = 7 \pmod{19}$

$$a^{p1} = 7^{18} = 7^{16} \times 7^2 = 7 \times 11 = 1 (\text{mod } 19)$$

Alternative form of fermat theorem is $a^p = a (\text{mod } p)$

Euler's Totient Function

Before presenting Euler's theorem, we need to introduce an important quantity in number theory, referred to as Euler's totient function and written $\phi(n)$, defined as the number of positive integers less than n and relatively prime to n. By convention, $\phi(1) = 1$.

n	$\phi(n)$
1	1
3	2
13	12
14	6
15	8
19	18
20	8

If n is prime number then $\phi(n)=n-1$.

Euler's theorem

Euler's theorem states that for every a and n that are relatively prime:

$$a^{\emptyset(n)} = 1 (mod\ n)$$

Examples

a = 3; n = 10; $\phi(10) = 4$	$a^{\phi(n)} = 3^4 = 81 \equiv 1 (\text{mod } 10) = 1 (\text{mod n})$
a = 2; n = 11; $\phi(11) = 10$	$a^{\phi(n)} = 2^{10} = 1024 \equiv 1 (\text{mod } 11) = 1 (\text{mod n})$

An alternative form of Euler's theorem is

$$a^{\emptyset(n)+1} = a (mod\ n)$$

Testing for Primality

For many cryptographic algorithms, it is necessary to select one or more very large prime numbers at random. Thus we are faced with the task of determining whether a given large number is prime. There is no simple yet efficient means of accomplishing this task.

Miller-Rabin Algorithm

The algorithm due to Miller and Rabin is typically used to test a large number for primality. Before explaining the algorithm, we need some background.

First, any positive odd integer n>= 3 can be expressed as follows:

n 1 = 2^kq with k > 0, q odd

Two Properties of Prime Numbers

The first property is stated as follows:

If p is prime and a is a positive integer less than p, then a^2 mod p = 1 if and only if either a mod p = 1 or a mod p= 1 mode p = p-1. By the rules of modular arithmetic (a mode p) (a mode p) = a^2 mod p. Thus if either a mode p = 1 or a mod p = 1, then a^2 mod p = 1. Conversely, if a^2 mod p = 1, then (a mod p)^2 = 1, which is true only for a mod p = 1 or a mod p = 1.

The second property is stated as follows:

Let p be a prime number greater than 2. We can then write p 1 = 2^kq, with k > 0 q odd. Let a be any integer in the range 1 < a < p 1. Then one of the two following conditions is true:

1. a^q is congruent to 1 modulo p. That is, a^q mod p = 1, or equivalently, $a^q \equiv 1$ (mod p).
2. One of the numbers a^q, a^{2q}, a^{4q},...,$a^{2^{k-1}}$q is congruent to 1 modulo p. That is, there is some =number j in the range (1 <=j <<k) such that a^{2j-1q} mod p = 1 mod p = p-1, or equivalently, $a^{2j-1q} \equiv 1$ (mod p).

Details of Miller Rubin algorithm

These considerations lead to the conclusion that if n is prime, then either the first element in the list of residues, or remainders, (aq, a^2q,..., , $a^{2^{k-1}}$ q, , a^{2^k}q) modulo n equals 1, or some element in the list equals (n-1); otherwise n is composite (i.e., not a prime). On the other hand, if the condition is met, that does not necessarily mean that n is prime.

For example, if n = 2047 = 23 x 89, then n 1 = 2 x 1023. Computing, 21023 mod 2047 = 1, so that 2047 meets the condition but is not prime.

We can use the preceding property to devise a test for primality. The procedure TEST takes a candidate integer n as input and returns the result `composite` if n is definitely not a prime, and the result `inconclusive` if n may or may not be a prime.

```
TEST (n)
```

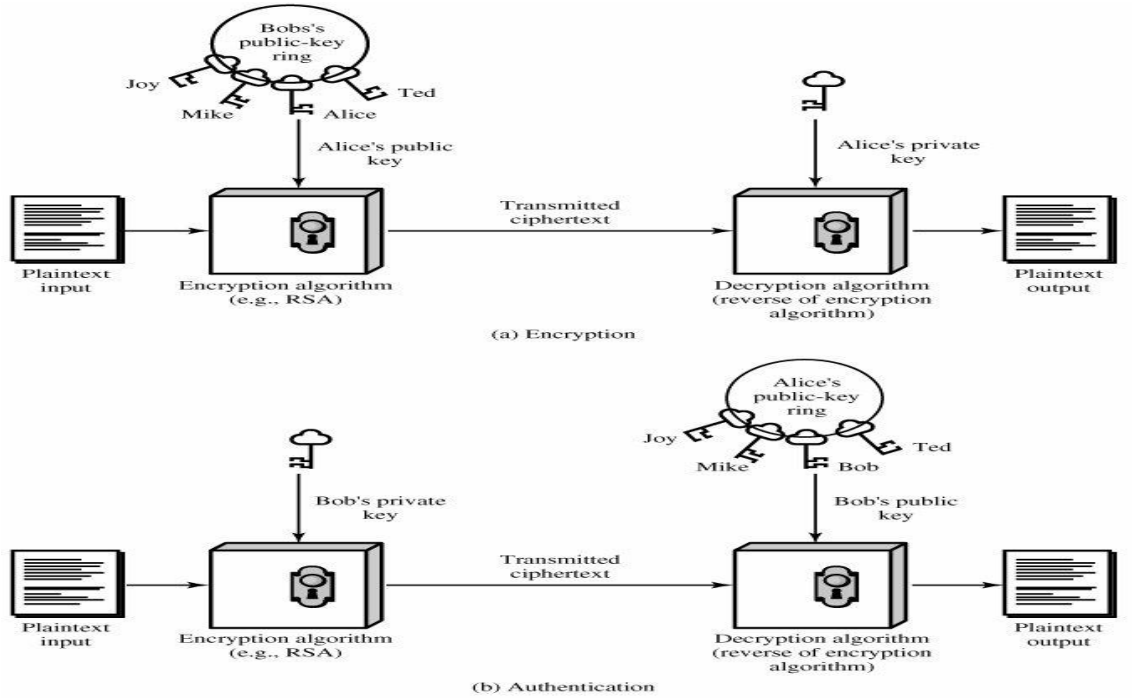
```
1.  Find integers k, q, with k > 0, q odd, so that (n-1
    = 2^k q);
2.  Select a random integer a, 1 < a < n-1;
3.  if a^q mod n = 1 then return("inconclusive");
4.  for j = 0 to k  1 do
5.     if a^(2^j q) mod n ≡ n  1 then return("inconclusive");
6.  return("composite");
```

Public key Cryptography: RSA

1. Public-Key Cryptosystems

A public-key encryption scheme has six ingredients:

1. **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
2. **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
3. **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.
4. **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.
5. **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.



1.1 Public-Key Cryptosystem for Secrecy

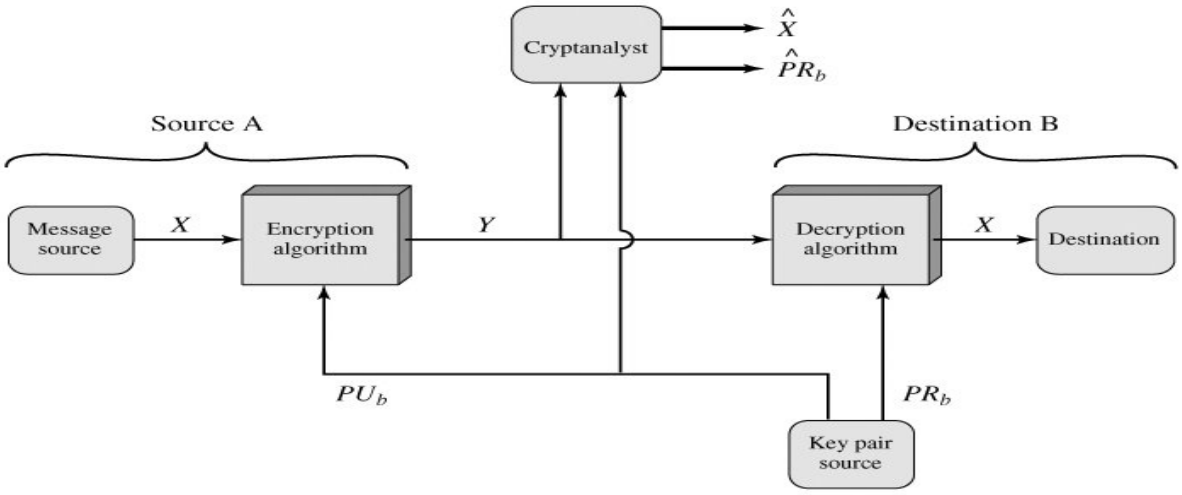
There is some source A that produces a message in plaintext, $X = [X_1, X_2, ..., X_M,]$. The M elements of X are letters in some finite alphabet. The message is intended for destination B. B generates a related pair of keys: a public key, PU_b, and a private key, PU_b. PU_b is known only to B, whereas PU_b is publicly available and therefore accessible by A. this is shown in figure next page.

With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y_1, Y_2, ..., Y_N]$:

$Y = E(PU_b, X)$

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

$X = D(PR_b, Y)$

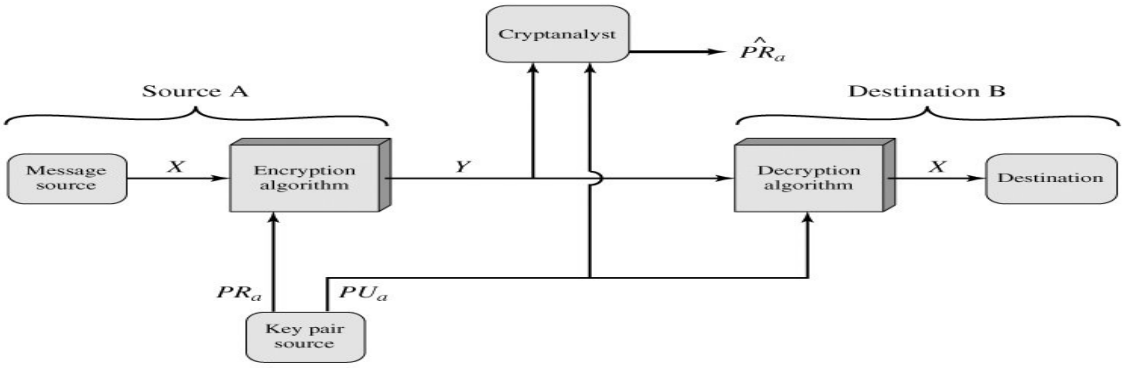


1.2 Public-Key Cryptosystem: Authentication

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 the message without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity. Figure show the use of public-key encryption to provide authentication:

$Y = E(PR_a, X)$

$Y = E(PU_a, Y)$



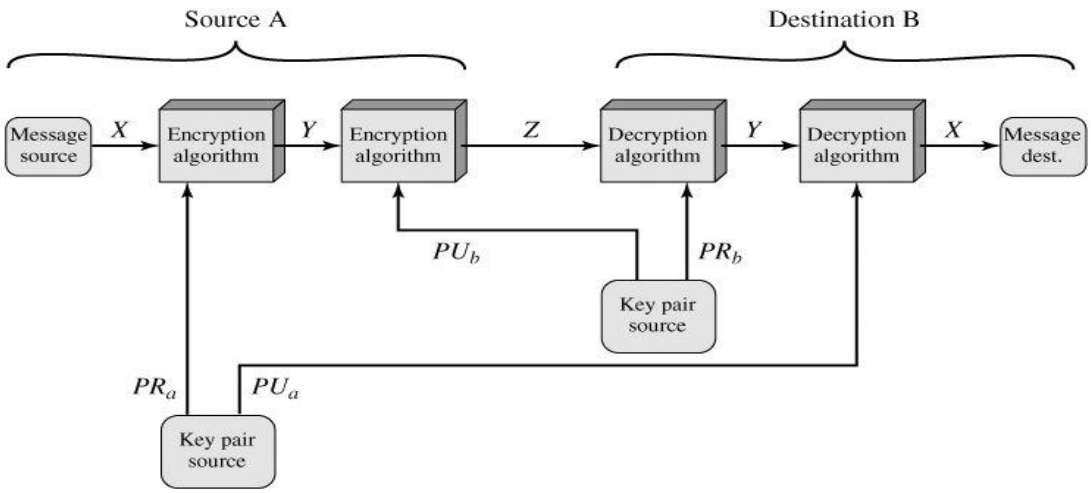
1.3 Public-Key Cryptosystem: Authentication and Secrecy

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme:

$Z = E(PU_b, E(PR_a, X))$

$X = D(PU_a, E(PR_b, Z))$

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 two in each communication.



Requirements for Public-Key Cryptography

1. It is computationally easy for a party B to generate a pair (public key PU_b, private key PR_b).
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:

   $C = E(PU_b, M)$

3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message:

   $M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$

4. It is computationally infeasible for an adversary, knowing the public key, PU_b, to determine the private key, PR_b.
5. It is computationally infeasible for an adversary, knowing the public key, PU_b, 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:

$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$

There are three families of Public-Key (PK) algorithms of practical relevance:
1. Integer factorization algorithms (RSA, ...)
2. Discrete logarithms (Diffie-Hellman, DSA, ...)
3. Elliptic curves (EC)

In this lecture we only consider Algorithms of family i.e. Integer Factorization Algorithm and Discrete Logarithms.

1. The RSA Algorithm

RSA is an algorithm for public-key cryptography. It was the first algorithm known to be suitable for signing as well as encryption, and one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys and the use of up-to-date implementations.

Operation: RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key.

RSA Key Generation:

1. Choose two distinct large random prime numbers p and q
2. Compute n = pq, n is used as the modulus for both the public and private keys
3. Compute the totient: $\varphi(n) = (p - 1)(q - 1)$.
4. Choose an integer e such that $1 < e < \varphi(n)$, and e and $\varphi(n)$ share no factors other than 1 i.e. e and $\varphi(n)$ are relatively prime)
5. e is released as the public key exponent
6. Compute d to satisfy the congruence relation $ed \equiv 1 \bmod \varphi(n)$; i.e. $de = 1 + k\varphi(n)$ for some integer k.
7. d is kept as the private key exponent

Notes on the above steps: Step 1: Numbers can be probabilistically tested for primality.

Step 4: A popular choice for the public exponents is $e = 2^{16} + 1 = 65537$. Some applications choose smaller values such as e = 3, 5, 17 or 257 instead. This is done to make encryption and signature verification faster.

Steps 4 and 5 can be performed with the extended Euclidean algorithm;

Encrypting Messages

Alice transmits her public key (n, e) to Bob and keeps the private key secret. Bob send message M to Alice by turning M into a number m < n by using a reversible protocol called a padding scheme. He then computes the ciphertext c as: $c = m^e$ **mod** n. Bob then transmits c to Alice.

Decrypting Messages

Alice can recover m from c by using her private key exponent d by the following computation: $m = c^d$ **mod** n. Given m, she can recover the original message M.

Example: Consider, p = 61 and q = 53 now, compute n = pq = 61 * 53 = 3233

Compute the totient $\varphi(n) = (p - 1)(q - 1) = (61-1)(53-1) = 3120$

Choose e > 1 relatively prime to 3120; e = 17

Compute d such that $ed \equiv 1 \bmod \varphi(n)$ e.g., by computing the modular multiplicative inverse of e modulo $\varphi(n)$: d = 2753 since 17 * 2753 = 46801 = 1 + 15 * 3120.

The public key is (n = 3233, e = 17).

For a padded message m the encryption function is:

$$c = m^e \textbf{ mod } n = m^{17} \textbf{ mod } 3233.$$

The private key is (n = 3233, d = 2753). The decryption function is:

$$m = c^d \textbf{ mod } n = c^{2753} \textbf{ mod } 3233.$$

For example, to encrypt m = 123, we calculate

$c = 123^{17}$ **mod** 3233 = 855 to decrypt c = 855, we calculate $m = 855^{2753}$ **mod** 3233 = 123

Both of these calculations can be computed efficiently using the square-and-multiply algorithm for modular exponentiation.

One More Example:

Consider primes p=11, q=3. Now, compute n = pq = 11.3 = 33 and

totient $\varphi(n) = (p-1)(q-1) = 10.2 = 20$ **.**

Choose e=3; Check gcd(e, φ(n)) = gcd(3, 20) = 1 (i.e. 3 and 20 have no common factors except 1),

Compute d such that ed ≡ 1 (mod φ(n))
i.e. find a value for d such that φ(n) divides (ed-1)
i.e. find d such that 20 divides 3d-1.
Simple testing (d = 1, 2, ...) gives d = 7
Check: ed-1 = 3.7 - 1 = 20, which is divisible by φ(n)

The notation 'a ≡ b (mod n)' means a and b have the same remainder when divided by n, or, equivalently,

Public key = (n, e) = (33, 3)
Private key = (n, d) = (33, 7).

This is actually the smallest possible value for the modulus n for which the RSA algorithm works. Now say we want to encrypt the message m = 7,
$c = m^e \bmod n = 7^3 \bmod 33 = 343 \bmod 33 = 13$.
Hence the ciphertext c = 13. To check decryption we compute $m' = c^d \bmod n = 13^7 \bmod 33 = 7$.

2. Diffie-Hellman Key Exchange

Diffie-Hellman (D-H) key exchange is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. Other names for Diffie-Hellman Key Exhange are *Diffie-Hellman Key Agreement, Diffie-Hellman Key Establishment, Diffie-Hellman Key Negotiation, Exponential Key Exchange.*

Description: The simplest and original implementation of the protocol uses the multiplicative group of integers modulo p, where p is a prime and g is primitive root of p.

Steps:

1. Generate the global public elements p and g, where p is a prime number and g < p is a primitive root of p.
2. User A selects a random integer number X_A<p, and computes $Y_A = g^{X_A} \bmod p$.
3. User B independently selects a random integer X_B<p, and computes $Y_B = g^{X_B} \bmod p$.
4. Each side keeps the X value private and makes the Y value available publicly to the other side.
5. User A generates secret key as $K = (Y_B)^{X_A} \bmod p$.

6. User B generates secret key as $K = (Y_A)^{X_B} \bmod p$

Why the key from both side same:

From user A, $K = (Y_B)^{X_A} \bmod p = (g^{X_B} \bmod p)^{X_A} \bmod p = (g^{X_B})^{X_A} \bmod p = g^{X_B X_A} \bmod p$

From user B, $K = (Y_A)^{X_B} \bmod p = (g^{X_A} \bmod p)^{X_B} \bmod p = (g^{X_A})^{X_B} \bmod p = g^{X_B X_A} \bmod p$

See above both the results are same.

Example: Alice and Bob agree to use a prime number p=23 and base g=5.

Alice chooses a secret integer $X_A = 6$, then sends Bob ($Y_A = g^{X_A} \bmod p$): $5^6 \bmod 23 = 8$.

Bob chooses a secret integer $X_B = 15$, then sends Alice ($Y_B = g^{X_B} \bmod p$): $5^{15} \bmod 23 = 19$.

Alice computes $(Y_B)^{X_A} \bmod p$: $19^6 \bmod 23 = 2$ and Bob computes $(Y_A)^{X_B} \bmod p$: $8^{15} \bmod 23 = 2$.

Once Alice and Bob compute the shared secret they can use it as an encryption key, known only to them, for sending messages across the same open communications channel. Of course, much larger values of X_A, X_B, and p would be needed to make this example secure, since it is easy to try all the possible values of $g^{X_A X_B} \bmod 23$ (there will be, at most, 22 such values, even if X_A, X_B are large). If p were a prime of at least 300 digits, and X_A, X_B were at least 100 digits long, then even the best algorithms known today could not find a given only g, p, and $g^{X_A} \bmod p$, even using all of mankind's computing power. The problem is known as the discrete logarithm problem. *Note that g need not be large at all, and in practice is usually either 2 or 5.*

Chapter 5
Digital Signatures

Digital Signatures

A digital signature or digital signature scheme is a type of asymmetric cryptography used to simulate the security properties of a handwritten signature on paper. Digital signature schemes normally give two algorithms, one for signing which involves the user's secret or private key, and one for verifying signatures which involves the user's public key. The output of the signature process is called the "digital signature."

A signature provides authentication of a "message". Messages may be anything, from electronic mail to a contract, or even a message sent in a more complicated cryptographic protocol. Digital signatures are used to create public key infrastructure (PKI) schemes in which a user's public key (whether for public-key encryption, digital signatures, or any other purpose) is tied to a user by a digital identity certificate issued by a certificate authority. PKI schemes attempt to unbreakably bind user information (name, address, phone number, etc.) to a public key, so that public keys can be used as a form of identification.

A digital signature scheme typically consists of three algorithms:

A key generation algorithm G, that randomly produces a "key pair" (PK, SK) for the signer. PK is the verifying key, which is to be public, and SK is the signing key, to be kept private.

A signing algorithm S, that, on input of a message m and a signing key SK, produces a signature σ.

A signature verifying algorithm V, which on input a message m, a verifying key PK, and a signature σ, either accepts or rejects.

Required Properties of Digital Signature Schemes

Two main properties are required. First, signatures computed honestly should always verify. That is, V should accept (m, PK, S (m, SK)) where SK is the secret key related to PK, for any message m. Secondly, it should be hard for any adversary, knowing only PK, to create valid signature(s)

Generally hashes are used in digital signature scheme due to the following reasons:

For efficiency: The signature will be much shorter and thus save time since hashing is generally much faster than signing in practice.

For compatibility: Messages are typically bit strings, but some signature schemes operate on other domains (such as, in the RSA, numbers modulo a number N). A hash function can be used to convert an arbitrary input into the proper format.

For integrity: Without the hash function, the text to be signed may split in many blocks for the signature scheme to act on them. However, the receiver of the signed blocks is not able to recognize if all the blocks are not present and not in the appropriate order.

Benefits of digital signatures

- **Authentication:** Digital signatures can be used to authenticate the source of messages. When ownership of a digital signature secret key is bound to a specific user, a valid signature shows that the message was sent by that user. For example, suppose a bank's branch office sends instructions to the central office requesting a change in the balance of an account. If the central office is not convinced that such a message is truly sent from an authorized source, acting on such a request could be a grave mistake.

- **Integrity:** In many cases, the sender and receiver of a message may have a need for trust that the message has not been altered during transmission. Although encryption hides the contents of a message, it may be possible to change an encrypted message
without understanding it. However, if a message is digitally signed, any change in the message will invalidate the signature. Furthermore, there is no efficient way to modify a message and its signature to produce a new message with a valid signature, because this is still considered to be computationally infeasible by most cryptographic hash functions.

Drawbacks of digital signatures

-**Trusted Time Stamping:** Digital signature algorithms and protocols do not inherently provide certainty about the date and time at which the underlying document was signed. The signer might, or might not, have included a time stamp with the signature, or the document itself might

have a date mentioned on it, but a later reader cannot be certain the signer did not, for instance, backdate the date or time of the signature.

- Non-repudiation: The word repudiation refers to any act of disclaiming responsibility for a message. A message's recipient may insist the sender attach a signature in order to make later repudiation more difficult, since the recipient can show the signed message to a third party (eg, a court) to reinforce a claim as to its signatories and integrity. However, loss of control over a user's private key will mean that all digital signatures using that key, and so ostensibly 'from' that user, are suspect. Nonetheless, a user cannot repudiate a signed message without repudiating their signature key. It is aggravated by the fact there is no trusted time stamp, so new documents (after the key compromise) cannot be separated from old ones, further complicating signature key invalidation. Certificate Authorities usually maintain a public repository of public-key so the association user-key is certified and signatures cannot be repudiated. Expired certificates are normally removed from the directory. It is a matter for the security policy and the responsibility of the authority to keep old certificates for a period of time if a non-repudiation of data service is provided.

Digital Signatures Schemes

- RSA

Basic RSA signatures are computed as follows:

- To generate RSA signature keys, one simply generates an RSA key pair. See RSA encryption fro Key generation mechanism.
- To sign a message m, the signer computes $\sigma = m^d \bmod n$. To verify, the receiver checks that $\sigma^e = m \bmod n$. where (e, n) is the public key and (d, n) is the private key.

This scheme is not very secure. To prevent attacks, one can first get a message digest of the message m and then apply the RSA algorithm described above to the result.

Signing Messages: Suppose Alice wishes to send a signed message (m) to Bob. She can use her own private key (d, n) to do so. She produces a hash value of the message (h(m)), find $(h(m))^d \bmod n$ (as she does when decrypting a message), and attaches it as a "signature" to the message. When Bob receives the signed message, he uses the same hash algorithm in conjunction with Alice's public key (e, n). He raises the $((h(m))^d \bmod n)^e \bmod n$ (as he does when encrypting a message), and compares the resulting hash value with the message's actual hash value. If the two

agree, he knows that the author of the message was in possession of Alice's secret key, and that the message has not been tampered with since.

- ElGamal Signature Scheme

The ElGamal signature scheme is based on the difficulty of computing discrete logarithms. The ElGamal signature algorithm is rarely used in practice. A variant developed at NSA and known as the Digital Signature Algorithm is much more widely used. The ElGamal signature scheme allows that a verifier can confirm the authenticity of a message m sent by the signer sent to him over an insecure channel.

Description: This scheme requires the following parameters and procedures:
A long term public/private keys where public key is (g, p, T) and private key S such that $g^S \bmod p = T$.
New Different public/private key pair for each message being signed. If message is m, choose random number S_m and find $g^{Sm} \bmod p = T_m$.

Signing Process:

Take the well known message digest (hash) function, say H. Using this hash function calculate message digest d_m of $m|T_m$ i.e. find $d_m = H(m|T_m)$.

Calculate signature $X = S_m + d_m S \bmod (p-1)$.

Signer sends message m along with X and T_m. since the receiver knows m and T_m,

Verifying Process:

Calculate d_m using obtained m and T_m.

Check whether $g^X = T_m T^{dm} \bmod p$. If this is true the signature is valid, else not valid. This is true since $g^X = g^{Sm + dmS} = g^{Sm} g^{dmS} = T_m T^{dm} \bmod p$.

Security: A third party can forge signatures either by finding the signer's secret key x or by finding collisions in the hash function. Both problems are believed to be difficult. The signer must be careful to choose a different k uniformly at random for each signature and to be certain that k, or even partial information about k, is not leaked. Otherwise, an attacker may be able to deduce the secret key x with reduced difficulty, perhaps enough to allow a practical attack. In

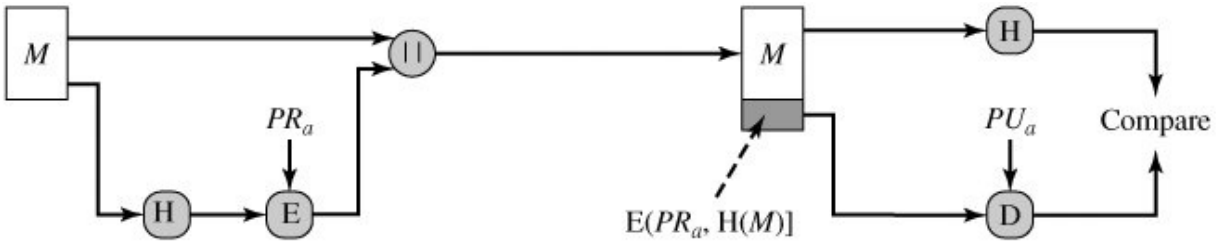
particular, if two messages are sent using the same value of k and the same key, then an attacker can compute x directly.

Digital Signature Standard

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) described in later chapter and presents a new digital signature technique, the Digital Signature Algorithm (DSA). The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. There was a further minor revision in 1996. In 2000, an expanded version of the standard was issued as FIPS 186-2. This latest version also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography. In this section, we discuss the original DSS algorithm.

Two Approaches to Digital Signatures (Revisited)

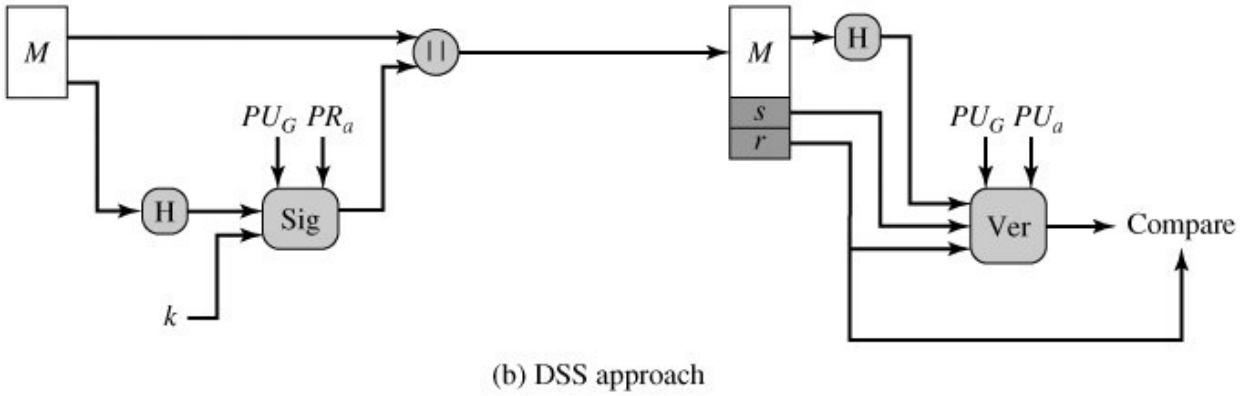
In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature. Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.



(a) RSA approach

The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (PR_a)and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PU_G).The result is a signature consisting of two components, labeled s and r. At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key (PU_a), which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component r if the signature is

valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.



(b) DSS approach

The Digital Signature Algorithm

Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$ for 512 L 1024 and L a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits

q prime divisor of (p 1), where $2^{159} < q < 2^{160}$; i.e., bit length of 160 bits

g = $h^{(p-1)/q}$ mod p, where h is any integer with 1 < h < (p 1) such that $h^{(p-1)/q}$ mod p > 1

User's Private Key

x random or pseudorandom integer with 0 < x < q

User's Public Key

y = g^x mod p

User's Per-Message Secret Number

k = random or pseudorandom integer with 0 < k < q

Signing

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

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

Signature = (r, s)

Verifying

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

u1 = [H(M')w] mod q

u2 =(r')w mod q

v = [(g^{u1} y^{u2}) mod p] mod q

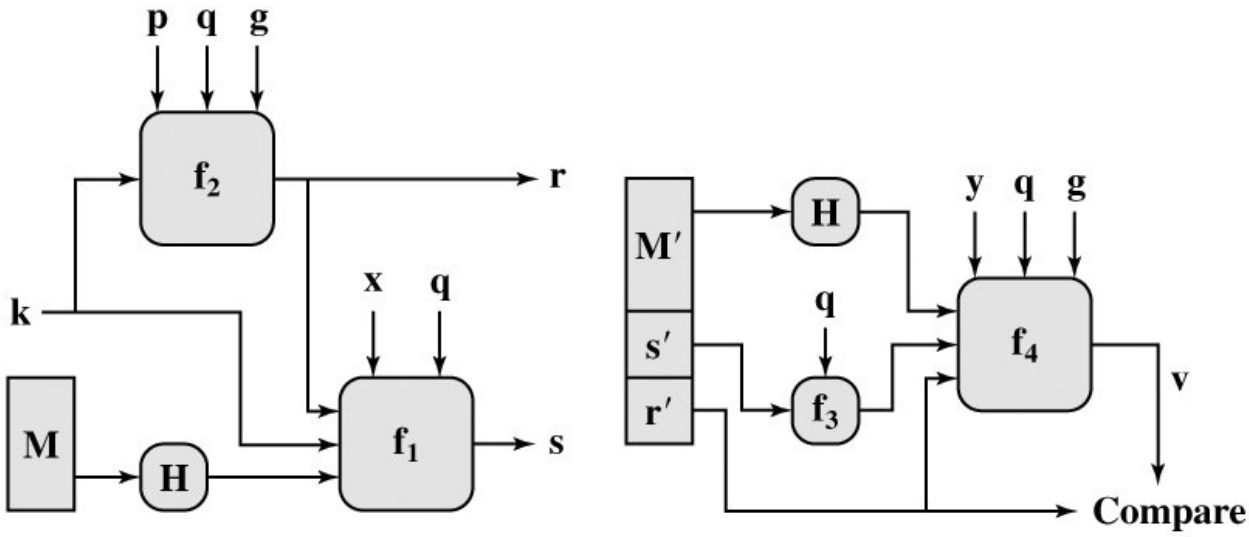
TEST: v = r'

M = message to be signed

H(M) = hash of M using SHA-1

M', r', s' = received versions of M, r, s

The following figure depicts the functions of signing and verifying.



$s = f_1(H(M), k, x, r, q) = (k^{-1} (H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

(a) Signing

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M')w) \bmod q} y^{r'w \bmod q}) \bmod p) \bmod q$

(b) Verifying

Chapter 6
Hashing and Message Digests

Hash Function

A hash value h is generated by a function H of the form

h = H(M)

where M is a variable-length message and H(M) is the fixed-length hash value. The hash value is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by recomputing the hash value. Because the hash function itself is not considered to be secret, some means is required to protect the hash value

Requirements for a Hash Function

The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be useful for message authentication, a hash function H must have the following properties:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that H(x) = h. This is sometimes referred to in the literature as **the one-way property.**
5. For any given block x, it is computationally infeasible to find y ≠x such that H(y) = H(x). This is sometimes referred to as **weak collision resistance**.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y). This is sometimes referred to as **strong collision resistance**.

Cryptographic Hash Function

A cryptographic hash function is a transformation that takes an input and returns a fixed-size string, which is called the hash value. The hash value is a concise representation of the longer message or document from which it was computed. The message digest is a sort of "digital fingerprint" of the larger document. Cryptographic hash functions are used to do message integrity checks and digital signatures in various information security applications, such as authentication and message integrity.

Hash functions are an important type of cryptographic algorithms and are widely used in cryptography such as digital signature, data authentication, e-cash and many other applications. Hash functions are at work in the millions of transactions that take place on the internet every day. The purpose of the use of hash functions in many cryptographic protocols is to ensure their security as well as improve their efficiency. The most widely used hash functions are dedicated hash functions such as MD5 and SHA-1.

A cryptographic hash function should behave as much as possible like a random function while still being deterministic and efficiently computable. A cryptographic hash function is considered insecure if either of the following is computationally feasible:

Finding a (previously unseen) message that matches a given digest

Finding "collisions", wherein two different messages have the same message digest.

An attacker who can do either of these things might, for example, use them to substitute an unauthorized message for an authorized one.

Ideally, it should not even be feasible to find two messages whose digests are substantially similar; nor would one want an attacker to be able to learn anything useful about a message given only its digest. Of course the attacker learns at least one piece of information, the digest itself, which for instance gives the attacker the ability to recognize the same message should it occur again.

A hash function must be able to process an arbitrary-length message into a fixed-length output. This can be achieved by breaking the input up into a series of equal-sized blocks, and operating on them in sequence using a one-way compression function. The compression function can either be specially designed for hashing or be built from a block cipher.

There is no formal definition which captures all of the properties considered desirable for a cryptographic hash function. These properties below are generally considered prerequisites: **Preimage resistant:** given h it should be hard to find any m such that h = hash(m). **Second preimage resistant:** given an input m_1, it should be hard to find another input, m_2 (not equal to m_1) such that hash(m_1) = hash(m_2).

Applications

Message Integrity Verification: Determining whether any changes have been made to a message (or a file), for example, can be accomplished by comparing message digests calculated before, and after, transmission (or any other event).

Password Verification: Passwords are usually not stored in cleartext, for obvious reasons, but instead in digest form. To authenticate a user, the password presented by the user is hashed and compared with the stored hash. This is sometimes referred to as one-way encryption.

Digital Signatures: while generating digital signatures, the message digest is created and it is encrypted with the private key so that the signing process becomes faster.

Massage Digest Algorithms

1. MD 4 (Message Digest 4)

MD 4 Algorithm is a cryptographic hash function developed by Ronald Rivest in 1990. It produces 128 bits (four 32 bits words) message digest and is optimized for 32-bit computers.

Operations

Step 1: MD4 Message Padding

Original Message	1000……..000	Original length in bits 64 bits

The message to be processed by MD4 computation must be multiple of 512 bits (16 32-bit words). The original message is padded by adding 1 followed by required number of 0s so that the length of the message is 64 bits less than multiple of 512. The remaining 64 bits is used for providing length of the original message i.e. unpadded message.

Step 2: MD4 Message Digest Computation

MD4 processes message in 512 bits (16 32 bits words) each time. At first message digest is initialized to a fixed value and then each stage of the algorithm takes current value of message digest and modifies it using the next block of message. The function producing 128 bits output from the 512 bits block is called **compression function**. Each stage makes three passes with different methods of mangling for each pass. Each word of mangled message digest is added to

its pre-stage value to produce post-stage value that becomes the pre-stage value for the next stage.

Each stage starts with 16-words message block and 4-words message digest values. Say message words as m_0, m_1, …m_{15} and message digest words as d_0, d_1, d_2,d_3 with initial values $d_0 = 67452301_{16}$, $d_1 = efcdab89_{16}$, $d_2 = 98badcfe_{16}$, $d_3 = 10325476_{16}$, these numbers seems to be random but are initialized to the following values in hexadecimal, low-order bytes first 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10.

Each pass modifies d_0, d_1, d_2, d_3 using m_0, m_1, … m_{15} with the following operations.

- **floor(x):** the greatest integer not greater than x.
- **~x:** bitwise complement of 32 bits quantity x.
- **$x \wedge y$/ $x \vee y$/ $x \oplus y$:** bitwise AND/OR/XOR of 32 bits quantities x and y.
- **x + y:** bitwise binary sum of 32 bits quantities x and y with carry discarded.
- **x ↵ y:** called left rotate generates 32 bits quantity by shifting bit position of x to the left by y times. The shifted bits occupy the right position.

Pass 1: This pass uses the **selection function** [$F(x, y, z) = (x \wedge y) \vee (\sim x \wedge z)$] that takes three 32 bits words as input and produced a 32 bits output. This function's output depending upon n^{th} bits of x since it selects n^{th} bit of y if n^{th} bit of x is 1, otherwise it selects n^{th} bit of z. Each of the 16 words of the messages is separately processed using the following relation, where i goes from 0 to 15.

$d_{(-i) \wedge 3} = (d_{(-i) \wedge 3} + F(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_i) \leftarrow S_1(i \wedge 3)$, where $S_1(i) = 3 + 4i$ so ↵s cycles over the values 3, 7, 11, 15.

The above relation's "$\wedge$ 3" signifies that only the bottom two bits are used since bitwise AND with 11_2 changes last two bits. In our relation $(-i) \wedge 3$ gives us cycle 0, 3, 2, 1, 0, similarly $(1-i) \wedge 3$ gives 1, 0, 3, 2, 1; $(2-i) \wedge 3$ gives 2, 1, 0, 3, 2; and so on. So expanding the above relation we get first few steps as:

$d_0 = (d_0 + F(d_1, d_2, d_3) + m_0) \leftarrow 3;$

$d_3 = (d_3 + F(d_0, d_1, d_2) + m_1) \leftarrow 7;$

$d_2 = (d_2 + F(d_3, d_0, d_1) + m_2) \leftarrow 11;$

$d_1 = (d_1 + F(d_2, d_3, d_0) + m_3) \leftarrow 15;$

$d_0 = (d_0 + F(d_1, d_2, d_3) + m_4) \lfloor\!\lfloor 3$; and so on.

Pass 2: This pass uses **majority function** [$G(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$] that takes three 32 bits words gives a 32 bits output. The function's output of n^{th} bit is 1 if and only if any two of the three input's n^{th} bits are 1. In this pass we introduce one strange constant **floor**$(2^{30}\sqrt{2}) = 5a827999_{16}$. Each of the 16 words of the messages is separately processed, not in order, using the following relation, where i goes from 0 to 15.

$d_{(-i) \wedge 3} = (d_{(-i) \wedge 3} + G(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_{X(i)} + 5a827999_{16}) \lfloor\!\lfloor S_2(i \wedge 3)$, where X(i) is 4-bits number formed by exchanging the low order and high order pairs of bits in 4-bits number **i** [so, $X(i) = 4i - 15$**floor**$(i/4)$], and $S_2(0) = 3$, $S_2(1) = 5$, $S_2(2) = 9$, $S_2(3) = 13$, so $\lfloor\!\lfloor$s cycles over the values 3, 5, 9, 13.

$d_0 = (d_0 + G(d_1, d_2, d_3) + m_0 + 5a827999_{16}) \lfloor\!\lfloor 3$;

$d_3 = (d_3 + G(d_0, d_1, d_2) + m_4 + 5a827999_{16}) \lfloor\!\lfloor 5$;

$d_2 = (d_1 + G(d_3, d_0, d_1) + m_8 + 5a827999_{16}) \lfloor\!\lfloor 9$;

$d_1 = (d_1 + G(d_2, d_3, d_1) + m_{12} + 5a827999_{16}) \lfloor\!\lfloor 13$;

$d_0 = (d_0 + G(d_1, d_2, d_3) + m_1 + 5a827999_{16}) \lfloor\!\lfloor 3$; and so on.

Pass 3: This pass uses the function [$H(x, y, z) = x \oplus y \oplus z$] that takes three 32 bits words gives a 32 bits output. In this pass a different strange constant **floor**$(2^{30}\sqrt{3}) = 6ed9eba1_{16}$. Each of the 16 words of the messages is separately processed, not in order, using the following relation, where i goes from 0 to 15.

$d_{(-i) \wedge 3} = (d_{(-i) \wedge 3} + H(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_{R(i)} + 6ed9eba1_{16}) \lfloor\!\lfloor S_3(i \wedge 3)$, where X(i) is 4-bits number formed by reversing the order of bits in 4-bits number **i** [so, $R(i) = 8i - 12$**floor**$(i/2) - 6$**floor**$(i/4) - 3$**floor**$(i/8)$], and $S_3(0) = 3$, $S_3(1) = 9$, $S_3(2) = 11$, $S_3(3) = 15$, so $\lfloor\!\lfloor$s cycles over the values 3, 9, 11, 15.

$d_0 = (d_0 + H(d_1, d_2, d_3) + m_0 + 6ed9eba1_{16}) \lfloor\!\lfloor 3$;

$d_3 = (d_3 + H(d_0, d_1, d_2) + m_8 + 6ed9eba1_{16}) \lfloor\!\lfloor 9$;

$d_2 = (d_1 + H(d_3, d_0, d_1) + m_4 + 6ed9eba1_{16}) \lfloor\!\lfloor 11$;

$d_1 = (d_1 + H(d_2, d_3, d_1) + m_{12} + 6ed9eba1_{16}) \lfloor\!\lfloor 15$;

$d_0 = (d_0 + H(d_1, d_2, d_3) + m_2 + 6ed9eba1_{16}) \lfloor\!\lfloor 3$; and so on.

MD 5 (Message Digest 5)

MD5 was designed by Ron Rivest in 1991 to replace an earlier hash function, MD4. MD5 is a widely used hash function with a 128-bit hash value. An MD5 hash is typically expressed as a sequence of 32 hexadecimal digits.

Differences Between MD4 and MD5

- A fourth pass has been added.
- Each step now has a unique additive constant (T_i), so there are 64 constants.
- The function G in pass 2 was changed from $((x \wedge y) \vee (x \wedge z) \vee (y \wedge z))$ to $((x \wedge z) \vee (y \wedge \sim z))$ to make g less symmetric.
- Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".
- The order in which input words are accessed in passes 2 and 3 is changed, to make these patterns less like each other.
- The shift amounts in each pass have been approximately optimized, to yield a faster "avalanche effect." The shifts in different passes are distinct.

MD2, MD4, and MD5 Hashes

The 128-bit (16-byte) MD2, MD4, and MD5 hashes (also termed message digests) are typically represented as 32-digit hexadecimal numbers. The following demonstrates a 43-byte ASCII input and the corresponding MD2 hash:

MD2("The quick brown fox jumps over the lazy dog")
= 03d85a0d629d2c442e987525319fc471
MD4("The quick brown fox jumps over the lazy dog")
= 1bee69a46ba811185c194762abaeae90
MD5("The quick brown fox jumps over the lazy dog")
= 9e107d9d372bb6826bd81d3542a419d6

Even a small change in the message will (with overwhelming probability) result in a completely different hash, due to the avalanche effect. For example, changing d to c:

MD2("The quick brown fox jumps over the lazy cog")
= 6b890c9292668cdbbfda00a4ebf31f05
MD4("The quick brown fox jumps over the lazy cog")
= b86e130ce7028da59e672d56ad0113df
MD5("The quick brown fox jumps over the lazy cog")
= 1055d3e698d289f2af8663725127bd4b

The hash of the zero-length string is:

MD2("") = 8350e5a3e24c153df2275c9f80692773
MD4("") = 31d6cfe0d16ae931b73c59d7e0c089c0
MD5("") = d41d8cd98f00b204e9800998ecf8427e

Secure Hash Standard (SHS)

The Secure Hash Standard (SHS) is a set of cryptographically secure hash algorithms specified by the National Institute of Standards and Technology. The SHS standard specifies a number of Secure Hash Algorithms (SHA), for example SHA-1, SHA-256 and SHA-512.

SHA-1 (Secure Hash Algorithm 1)

When a message of length $< 2^{64}$ bits is input, the SHA produces a 160-bit representation of the message called the message digest. The SHA is designed to have the following properties: it is computationally infeasible to recover a message corresponding to a given message digest, or to find two different messages which produce the same message digest.

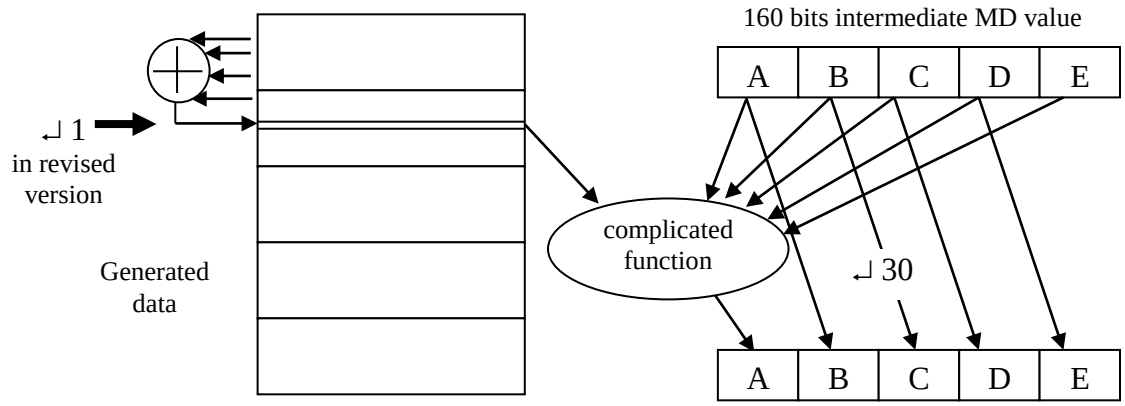
Operations

Step 1: Message Padding

The padding is same as that of MD4 or MD5 except that the length of the message is not longer than 2^{64} bits.

Step 2: SHA-1 Message Digest Computation

SHA-1 processes message in 512 bits each time where it mangles the current message block using pre-stage message digest and sequence of operations. Each word of mangled message digest is added to its pre-stage value to produce post-stage value that becomes the pre-stage value for the next stage. Initial values for five 32 bits message digest words are: A = 67452301_{16}, B = $efcdab89_{16}$, C = $98badcfe_{16}$, D = 10325476_{16}, E = $c3d2e1f0_{16}$.



SHA-1 operations on a 512-bits block (see figure above): At each stage, the 512-bits message block is used to create 5×512-bits chunk. The first 512 bits is the actual message block and the rest chunks are filled 32 bits at a time using a rule: n^{th} word (starts from 16, since 0-15 are for actual message block) is the left rotation (this rotation differs SHA-1 from SHA) of ⊕ of words (n-3), (n-8), (n-14), and (n-16). Now, we have the buffer of 80 32 bits words, lets denote them by W_0, W_1, …, W_{79}. here as from the above discussion we can find n^{th} 32 bits word W_n as: $W_n = (W_{n-3} \oplus W_{n-8} \oplus W_{n-14} \oplus W_{n-16})$

The change of A, B, C, D, and E are done as: for t = 0 to 79, B = old A; C = old B ↵ 30; D = old C; E = old D; A = E + (A ↵ 5) + W_t + K_t + f(t,B,C,D); Here at first A is calculated using old values of A, B, C, D, E (complicated function) and other calculations for B, C, D, and E are done. In the calculation of A, W_t is the t^{th} 32 bits word block and K_t is the constant depending upon the value of t given by the following relations:

K_t = **floor**($2^{30}\sqrt{2}$) = $5a827999_{16}$ if $0 \le t \le 19$.

K_t = **floor**($2^{30}\sqrt{3}$) = $6ed9eba1_{16}$ if $20 \le t \le 39$.

K_t = **floor**($2^{30}\sqrt{5}$) = $8f1bbcdc_{16}$ if $40 \le t \le 59$.

K_t = **floor**($2^{30}\sqrt{10}$) = $ca62c1d6_{16}$ if $60 \le t \le 79$.

Again f(t,B,C,D) is a function that varies according to the following relations:

$f(t,B,C,D) = (B \wedge C) \vee (\sim B \wedge D)$ if $0 \le t \le 19$.

$f(t,B,C,D) = B \oplus C \oplus D$ if $20 \le t \le 39$.

$f(t,B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ if $40 \le t \le 59$.

$f(t,B,C,D) = B \oplus C \oplus D$ if $60 \le t \le 79$.

Chapter 7

Authentication and Public Key Infrastructure (PKI)

Key management

Key management refers to the distribution of cryptographic keys; the mechanisms used to bind an identity to a key; and the generation, maintenance, and revoking of such keys. Here we assume that the key gives us the true identity. So in our consideration we assume that we already have authentication and the key has been assigned to the user. In this chapter we discuss the following concepts: **Key exchange:** Session vs. interchange keys; Classical, public key methods; Key generation. **Cryptographic key infrastructure:** Certificates. **Key storage:** Key escrow; Key revocation.

Notation

- **$X \rightarrow Y : \{ Z \| W \} k_{X,Y}$**: X sends Y the message produced by concatenating Z and W enciphered by key $k_{X,Y}$, which is shared by users X and Y.
- **$A \rightarrow T : \{ Z \} k_A \| \{ W \} k_{A,T}$:** A sends T a message with the concatenation of Z enciphered using k_A, A's key, and W enciphered using $k_{A,T}$, key shared by A and T.
- **$rand_1$, $rand_2$:** nonces (nonrepeating random numbers)

Session, Interchange Keys

An interchange key is a cryptographic key associated with a principal to a communication. A session key is a cryptographic key associated with the communication itself. For e.g. A wants to send a message m to B, assume public key encryption, here A generates a random cryptographic key $k_{session}$ and uses it to encipher m that is to be used for this message only called a session key. Now A enciphers $k_{session}$ with B's public key k_B (k_B enciphers all session keys A uses to communicate with B called an interchange key). Finally A sends $\{m\} k_{session} \| \{k_{session}\} k_B$.

Benefits

It limits amount of traffic enciphered with single key, here standard practice is to decrease the amount of traffic an attacker can obtain.

Prevents some attacks: E.g.- Suppose Alice is a client of Bob's stockbrokering firm. She needs to send Bob one of two messages: BUY or SELL. The attacker, Cathy, enciphers both messages with Bob's public key. When Alice sends her message, Cathy compares it with her messages and sees which one it matches.

Key Exchange

Key exchange is any method in cryptography by which cryptographic keys are exchanged between users, allowing use of a cryptographic algorithm. The **key exchange problem** is how to exchange whatever keys or other information needed so that no one else can obtain a copy. Traditionally, this required trusted couriers (with or without briefcases handcuffed to their wrists), or diplomatic bags, or some other secure channel. With the advent of public key / private key cipher algorithms, the encrypting key could be made public, since (at least for high quality algorithms) no one without the decrypting key could decrypt the message.

In principle, then, the only remaining problem was to be sure (or at least confident) that a public key actually belonged to its supposed owner. Because it is possible to 'spoof' another's identity in any of several ways, this is not a trivial or easily solved problem, particularly when the two users involved have never met and know nothing about each other.

The goal of key exchange is to enable A to B, and vice versa communication secret, using a shared cryptographic key. Solutions to this problem must meet the following criteria.

1. The key that A and B are to share cannot be transmitted in the clear. Either it must be enciphered when sent, or A and B must derive it without an exchange of data from which the key can be derived. (A and B can exchange data, but a third party cannot derive the key from the data exchanged.)
2. A and B may decide to trust a third party (called "C" here).
3. The cryptosystems and protocols are publicly known. The only secret data is to be the cryptographic keys involved.

Classical Key Exchange

Suppose Alice(A) and Bob(B) wish to communicate. If they share a common key, they can use a classical cryptosystem. But how do they agree on a common key? If A sends one to B, Eve the eavesdropper will see it and be able to read the traffic between them. In this context A can't send the key to be shared to B in the clear. To avoid this bootstrapping problem, classical protocols rely on a trusted third party, Cathy(C). A and C share a secret key, k_A, and B and C share a (different) secret key, k_B. The goal is to provide a secret key, k_S that A and B share. The following simple protocol provides a starting point

Simple Protocol

To avoid bootstrap problem we can use the following simple protocol.

1. $A \rightarrow C$: {request for session key to B} k_A.
2. $C \rightarrow A$: $\{ k_{session} \}k_A \| \{ k_{session} \}k_B$.
3. $A \rightarrow B$: $\{ k_{session} \}k_B$.

B now deciphers the message and uses $k_{session}$ to communicate with A.

Problems: This protocol is the basis for many more sophisticated protocols. However, B does not know to whom he is talking. This problem leads us to **replay attack**: Eve records message from A to B, later replays it; B may think he's talking to A, but he isn't and **session key reuse**: Eve replays message from A to B, so B re-uses session key. So the protocols must provide authentication and defense against replay attack. The following algorithm provides solution to the above problem.

Needham-Schroeder Protocol

1. $A \rightarrow C : \{ A \| B \| rand_1 \}$

   Alice sends a message to the server identifying herself and Bob, telling the server she wants to communicate with Bob.
2. $C \rightarrow A : \{ A \| B \| rand_1 \| k_{session} \| \{A \| k_{session}\} k_B \} k_A$

   The server (Cathy) generates $k_{session}$ and sends back to Alice a copy encrypted under k_B for Alice to forward to Bob and also a copy for Alice. Since Alice may be requesting keys for several different people, the nonce assures Alice that the message is fresh and that the server (Cathy) is replying to that particular message and the inclusion of Bob's name tells Alice who she is to share this key with.
3. $A \rightarrow B : \{ A \| k_{session} \} k_B$

Alice forwards the key to Bob who can decrypt it with the key he shares with the server (Cathy), thus authenticating the data.

4. $B \rightarrow A : \{ rand_2 \} k_{session}$

   Bob sends Alice a nonce encrypted under $k_{session}$ to show that he has the key.

5. $A \rightarrow B : \{ rand_2 - 1 \} k_{session}$

   Alice performs a simple operation on the nonce, re-encrypts it and sends it back verifying that she is still alive and that she holds the key.

Argument: A talking to B

Second message: This is the response to the first message (since $rand_1$ in second message is same as of in $rand_1$ first message) enciphered using key only A and C knows.

Third message: A knows only B can read it since only B can derive session key from message and any messages enciphered with that key are from B.

Argument: B talking to A

Third message: This message contains A (name) and session key provided by C that is enciphered using key only B and C know.

Fourth message: Uses session key to determine if it is replay from Eve the eavesdropper. If Eve recorded the message, she could have replayed it to Bob. In that case, Eve would not have known the session key, so Bob sets out to verify that his unknown recipient does know it. He sends a random message enciphered by $k_{session}$ to Alice. If Eve intercepts the message, she will not know what to return; should she send anything, the odds of her randomly selecting a message that is correct is very low and Bob will detect the attempted replay. But if Alice is indeed initiating the communication, when she gets the message she can decipher it (because she knows $k_{session}$), apply some fixed function to the random data (here, decrement it by 1), and encipher the result and return it to Bob. Then Bob will be sure he is talking to Alice.

Denning-Sacco Modification

Needham-Schroeder protocol assumes all keys are secret but suppose that if Eve can obtain session key. How does that affect protocol? In this context Eve knows $k_{session}$. So we have situation where B thinks that A has sent the message as seen from below:

1. Eve $\rightarrow$ B : { A || $k_{session}$ } k_B
2. B $\rightarrow$ A : { $rand_2$ } $k_{session}$ [intercepted by Eve]
3. Eve $\rightarrow$ B : { $rand_2 - 1$ }$k_{session}$.

Solution: In protocol (Needham-Schroeder), Eve impersonates A. So we have replay in third step (first in above). For this solution can be use of **time stamp T** to detect replay.

Needham-Schroeder with Denning-Sacco Modification

Denning and Sacco suggest using timestamps to enable B to detect replay.

1. A $\rightarrow$C : { A || B || $rand_1$ }
2. C $\rightarrow$A : { A || B || $rand_1$ || $k_{session}$ || {A || T || $k_{session}$} k_B } k_A
3. A $\rightarrow$B : {A || T || $k_{session}$} k_B
4. B $\rightarrow$A : { $rand_2$ } $k_{session}$
5. A $\rightarrow$B : { $rand_2 - 1$ }$k_{session}$

Where, T is a timestamp. When B gets the message in step 3, he rejects it if the timestamp is too old (too old being determined from the system in use). This modification requires synchronized clocks. The weakness with this solution is a party with a slow clock is vulnerable to a replay attack adds that a party with a fast clock is also vulnerable, and simply resetting the clock does not eliminate the vulnerability.

Otway-Rees Protocol

This protocol corrects problem of Eve replaying the third message in the protocol and does not use timestamps so it is not vulnerable to the problems that Denning-Sacco modification has. It uses integer **num** to associate all messages with particular exchange. The following are the steps in the protocol.

1. A $\rightarrow$B : num || A || B || { $rand_1$ || num || A || B }k_A
2. B $\rightarrow$C : num || A || B || { $rand_1$ || num || A || B }k_A || {$rand_2$ || num || A || B }k_B
3. C $\rightarrow$B : num || { $rand_1$ || $k_{session}$ }k_A || { $rand_2$ || $k_{session}$ }k_B

4. $B \rightarrow A : num \| \{ rand_1 \| k_{session} \} k_A$

The purpose of the integer **num** is to associate all messages with a particular exchange.

Argument: A talking to B

Fourth message: When Alice receives the fourth message from Bob, she checks that the num agrees with the num in the first message that she sent to Bob. If so, she knows that this is part of the exchange. She also trusts that Cathy generated the session key because only Cathy and Alice know k_{Alice}, and the random number $rand_1$ agrees with what Alice put in the enciphered portion of the message. Combining these factors, Alice is now convinced that she is talking to Bob.

Argument: B talking to A

Third message: When Bob receives the message from Cathy, he determines that the num corresponds to the one he received from Alice and sent to Cathy. He deciphers that portion of the message enciphered with his key, and checks that $rand_2$ is what he sent. He then knows that Cathy sent the reply, and that it applies to the exchange with Alice.

Suppose Eve gets old $k_{session}$, message in third step $num \| \{rand_1 \| k_{session}\}k_A \| \{rand_2 \| k_{session}\}k_B$. Eve forwards appropriate part to A

- A has no ongoing key exchange with B: num matches nothing, so is rejected
- A has ongoing key exchange with B: num does not match, so is again rejected
  - If replay is for the current key exchange, and Eve sent the relevant part before B did, Eve could simply listen to traffic; no replay involved.

Kerberos

It is a secret key based service for providing authentication in a network. It is based on Needham-Schroeder with Denning-Sacco modification. It makes use of a trusted third party, termed a key distribution center (KDC), which consists of two logically separate parts: an Authentication Server (AS) and a Ticket Granting Server (TGS). Kerberos works on the basis of "tickets" which serve to prove the identity of users.

The KDC maintains a database of secret keys; each entity on the network — whether a client or a server — shares a secret key known only to itself and to the KDC. Knowledge of this key serves

to prove an entity's identity. For communication between two entities, the KDC generates a session key which they can use to secure their interactions

Here once authenticator authenticates the user, the ticket must be used by the client to request the service from the server.

Idea

- User u authenticates to Kerberos server and obtains ticket $T_{u,TGS}$ for ticket granting service (TGS).
- For using service s by u: User sends authenticator A_u, ticket $T_{u,TGS}$ to TGS asking for ticket for service. TGS sends ticket $T_{u,s}$ to user and user sends Au, $T_{u,s}$ to server as request to use s.

Ticket

It is the credential saying issuer has identified ticket requester. Example ticket issued to user u for service s $\mathbf{T_{u,s} = s \,||\, \{ u \,||\, u\text{'s address} \,||\, valid\ time \,||\, k_{u,s} \}\, k_s}$, where: $k_{u,s}$ is session key for user and service; Valid time is interval for which ticket valid; u's address may be IP address or something else.

Authenticator

It is the system containing identity of sender of ticket that is used to confirm sender is entity to which ticket was issued. Example: authenticator user u generates for service s

$\mathbf{A_{u,s} = \{ u \,||\, generation\ time \,||\, k_t \}\, k_{u,s}}$, where: k_t is alternate session key; Generation time is when authenticator generated.

Protocol

1. user → C: {user || TGS}
2. C → user: $\{k_{u,TGS}\}k_u \,||\, T_{u,TGS}$
3. user → TGS: service || $A_{u,TGS}$ || $T_{u,TGS}$
4. TGS → user: user || $\{k_{u,s}\}k_{u,TGS}$ || $T_{u,s}$
5. user → service: $A_{u,s}$ || $T_{u,s}$
6. service → user: $(t + 1)\ k_{u,s}$

Analysis

- First two steps get user ticket to use TGS. Here user u can obtain session key only if u knows key shared with C.
- Next four steps show how u gets and uses ticket for service s
    - Service s validates request by checking sender (using $A_{u,s}$) is same as entity ticket issued to.
    - Step 6 optional; used when u requests confirmation

Problems

Kerberos relies on clocks being synchronized to prevent replay attacks. If the clocks are not synchronized, and if old tickets and authenticators are not cached, replay is possible.
The tickets have some fixed fields so a dictionary attack can be used to determine keys shared by services or users and the ticket-granting service or the authentication service.

Public Key Cryptographic Key Exchange

In this approach interchange keys are known as of e_A, e_B A and B's public keys known to all and d_A, d_B A and B's private keys known only to owner. The simple protocol with $k_{session}$ as desired session key is A $\rightarrow$B: $\{k_{session}\}e_B$.

Problem and Solution

It is vulnerable to forgery or replay because e_B known to anyone, B has no assurance that A sent message. A simple fix uses A's private key, where $k_{session}$ is desired session key and A $\rightarrow$B: A || $\{\{k_{session}\}d_A\}e_B$.

Notes:

A can include message enciphered with $k_{session}$. The above solution assumes B has A's public key, and vice versa. If not, each must get it from public server. If keys not bound to identity of owner, attacker Eve can launch a man-in-the-middle attack. Solution to this (binding identity to keys) discussed later as public key infrastructure (PKI).

Man-in-the-Middle Attack

1. A $\rightarrow$P : { send me B's public key } [intercepted by Eve]
2. Eve $\rightarrow$P : { send me B's public key }
3. P $\rightarrow$Eve : e_B
4. Eve $\rightarrow$A : e_{Eve}
5. A $\rightarrow$B : { $k_{session}$ } e_{Eve} [intercepted by Eve]
6. Eve $\rightarrow$B : { $k_{session}$ }e_B

Key Generation

The secrecy that cryptosystems provide resides in the selection of the cryptographic key. If an attacker can determine someone else's key, the attacker can read all traffic enciphered using that key or can use that key to impersonate its owner. Hence, generating keys that are difficult to guess or to determine from available information is critical.

Goal: generate keys that are difficult to guess

Problem statement: given a set of K potential keys, choose one randomly. This is equivalent to selecting a random number between 0 and K–1 inclusive.

Why is this hard: generating random numbers is hard since numbers are usually pseudo-random, that is, generated by an algorithm.

What is "Random"?

A sequence of random numbers is a sequence of numbers n_1, n_2, ... such that for any positive integer k, an observer cannot predict n_k even if n_1, ..., n_{k-1} are known.

Best physical source of randomness: Random pulses, Electromagnetic phenomena, Characteristics of computing environment like disk latency, Ambient background noise.

What is "Pseudorandom"?

A sequence of pseudorandom numbers is a sequence of numbers intended to simulate a sequence of cryptographically random numbers but generated by an algorithm.

Very difficult to do this well

Linear congruential generators [$n_k = (an_{k-1} + b) \bmod n$] this is broken; Polynomial congruential generators [$n_k = (a_j n_{k-1}^j + \ldots + a_1 n_{k-1} + a_0) \bmod n$] broken too. Here, "broken" means next number in sequence can be determined.

Best Pseudorandom Numbers

A strong mixing function is a function of two or more inputs that produces an output each bit of which depends on some nonlinear function of all the bits of the input. Examples: DES, MD5, SHA-1.

E.g.: On a UNIX system, the status of the processes is highly variable. An attacker is unlikely to reproduce the state at a future time. So the command **(date ; ps gaux) | md5** would produce acceptable pseudorandom data. In this command, ps gaux lists all information about all processes on the system.

Cryptographic Key Infrastructure

Because classical cryptosystems use shared keys, it is not possible to bind an identity to a key. Instead, two parties need to agree on a shared key. Public key cryptosystems use two keys, one of which is to be available to all. The association between the cryptographic key and the principal is critical, because it determines the public key used to encipher messages for secrecy. If the binding is erroneous, someone other than the intended recipient could read the message.

For purposes of this discussion, we assume that the principal is identified by a name of some acceptable sort and has been authenticated to the entity that generates the cryptographic keys. The question is how some (possibly different) principal can bind the public key to the representation of identity.

An obvious idea is for the originator to sign the public key with her private key, but this merely pushes the problem to another level, because the recipient would only know that whoever generated the public key also signed it. No identity is present.

Certificates

A certificate is a token that binds an identity to a cryptographic key. When B wants to communicate with A, B obtains A's certificate C_A. for e.g. Create token (message) containing: Identity of principal (here, A), Corresponding public key, Timestamp (when issued), and Other information (perhaps identity of signer) signed by trusted authority (here, C) as $C_A = \{ e_A \| A \| T \}d_C$.

Use

B gets A's certificate: If B knows C's public key, B can decipher the certificate and see when was certificate issued? Is the principal A? Now B has A's public key.

Problem: B needs C's public key to validate certificate. Two approaches deal with this problem. The first, by Merkle, eliminates Cs signature; the second structures certificates into signature chains.

Merkle's Tree Scheme

This scheme keeps certificates in a file and changing any certificate changes the file. This reduces the problem of substituting faked certificates to a data integrity problem. Cryptographic hash functions create checksums that reveal changes to files.

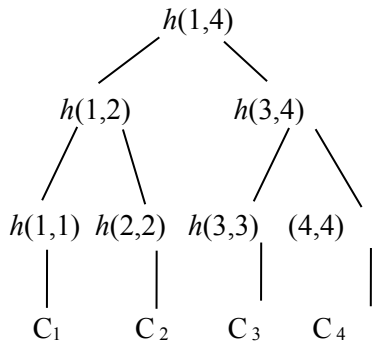
Let Y_i be an identifier and its associated public key, and let Y_1, ..., Y_n be stored in a file. Define a function f: D x D $\rightarrow$D, where D is a set of bit strings. Let h: N x N $\rightarrow$D be a cryptographic hash function, where N is integers set. Here we define

$h(i, j) = f(C_i, C_j)$ if $i \geq j$,

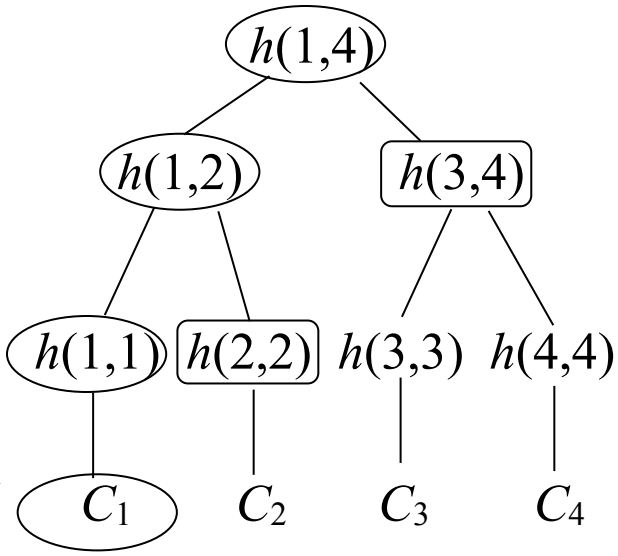
$h(i, j) = f(h(i, \lfloor (i+j)/2 \rfloor), h(\lfloor (i+j)/2 \rfloor + 1, j))$ if $i < j$,

Example:

- Construct Merkle hash tree by computing hashes recursively
  - *h* is hash function
  - *Ci* is certificate *i*
- Root hash (*h*(1,4) in example) is published and is known to all
  - Root hash is signed by the certificate authority to ensure the value's integrity



Validation



To validate C_1:

Compute h(1, 1), obtain h(2, 2)

Compute h(1, 2), obtain h(3, 4)

Compute h(1,4) and compare to known h(1, 4)

- Need to know siblings of nodes on path from C1 to the root
  - The proof from CA consists of these hashes (in rectangles on the left)

Problem

In this scheme file must be available for validation otherwise, can't recompute hash at root of tree. Not practical if there are too many certificates and users and users and certificates distributed over widely separated systems.

Issuers

Certification Authority (CA): entity that issues certificates. If all certificates have a common issuer, then the issuer's public key can be distributed out of band. However, this is infeasible. For example, it is highly unlikely that France and the United States could agree on a single issuer for their organizations' and citizens' certificates. This suggests multiple issuers, which complicates the process of validation.

Suppose Alice has a certificate from her local CA, Cathy. She wants to communicate with Bob, whose local CA is Dan. The problem is for Alice and Bob to validate each other's certificates.

Assume that X<<Y>> represents the certificate that X generated for the subject Y (X is the CA that issued the certificate). Bob's certificate is Dan<<Bob>>. If Cathy has issued a certificate to Dan, Dan has a certificate Cathy<<Dan>>; similarly, if Dan has issued a certificate to Cathy, Cathy has a certificate Dan<<Cathy>>. In this case, Dan and Cathy are said to be cross-certified.

Because Alice has Cathy's (trusted) public key, she can obtain Cathy<<Dan>> and form the signature chain

Cathy<<Dan>> Dan<<Bob>>

Because Alice can validate Dan's certificate, she can use the public key in that certificate to validate Bob's certificate. Similarly, Bob can acquire Dan<<Cathy>> and validate Alice's certificate.

Dan<<Cathy>> Cathy<<Alice>>

Storing and Recovering Keys

Key storage arises when a user needs to protect a cryptographic key in a way other than by remembering it. If the key is public, of course, any certificate-based mechanism will suffice, because the goal is to protect the key's integrity. But secret keys (for classical cryptosystems) and private keys (for public key cryptosystems) must have their confidentiality protected as well.

Storing Keys

In case of multi-user or networked systems: attackers may defeat access control mechanisms. Even the encipherment of file containing key does not help since the attacker can monitor keystrokes to decipher files as key will be resident in memory that attacker may be able to read. One of the solutions can be the use of physical devices like “smart card” where key never enters system. In this approach the card can be stolen, so use of two devices that combine bits to make single key can be used.

Key Revocation

If the certificate is invalidated before expiration i.e. the key is invalid, then it may be due to compromised key or may be due to change in circumstance (e.g., someone leaving company).

There are two problems with revoking a public key. The first is to ensure that the revocation is correct—in other words, to ensure that the entity revoking the key is authorized to do so. The second is to ensure timeliness of the revocation throughout the infrastructure. This second problem depends on reliable and highly connected servers and is a function of the infrastructure as well as of the locations of the certificates and the principals who have copies of those certificates. Ideally, notice of the revocation will be sent to all parties when received, but invariably there will be a time lag.

CRLs (Certificate Revocation Lists)

A certificate revocation list is a list of certificates that are no longer valid. A certificate revocation list contains the serial numbers of the revoked certificates and the dates on which they were revoked. It also contains the name of the issuer, the date on which the list was issued, and when the next list is expected to be issued. The issuer also signs the list. Under X.509, only the issuer of a certificate can revoke it.

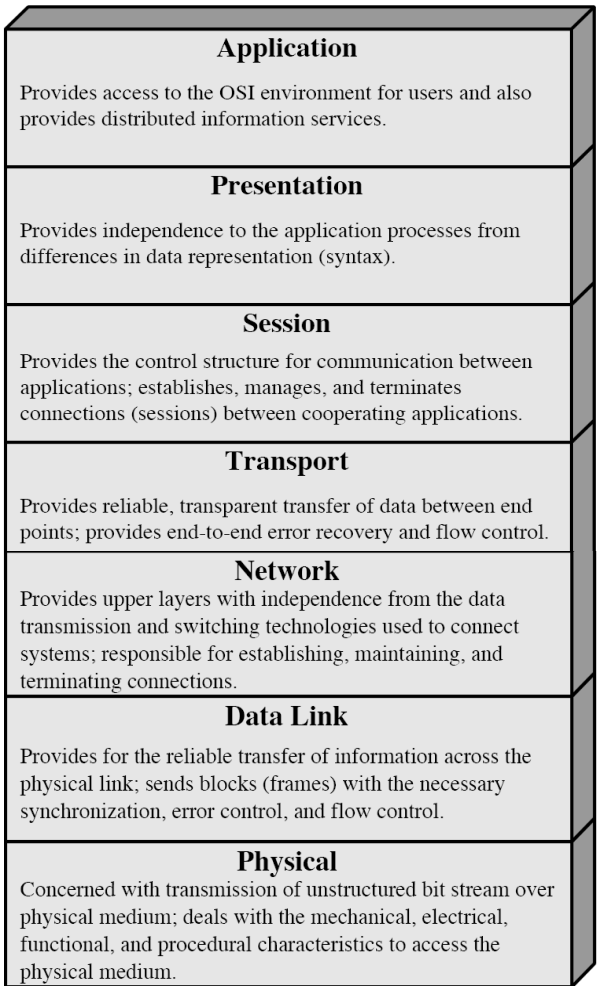
PGP allows signers of certificates to revoke their signatures as well as allowing owners of certificates, and their designees, to revoke the entire certificates. The certificate revocation is placed into a PGP packet and is signed just like a regular PGP certificate. A special flag marks it as a revocation message.

Chapter 8

Network Security

Networks and Cryptography

ISO/OSI model



Conceptually, each host has peer at each layer and peers communicate with peers at same layer (see books on network for detail)

Link and End-to-End Protocols

Let hosts C_0, …, C_n be such that C_i and C_{i+1} are directly connected, for $0 \leq i < n$. A communications protocol that has C_0 and C_n as its endpoints is called an **end-to-end protocol**. A communications protocol that has C_j and C_{j+1} as its endpoints is called a **link protocol**.

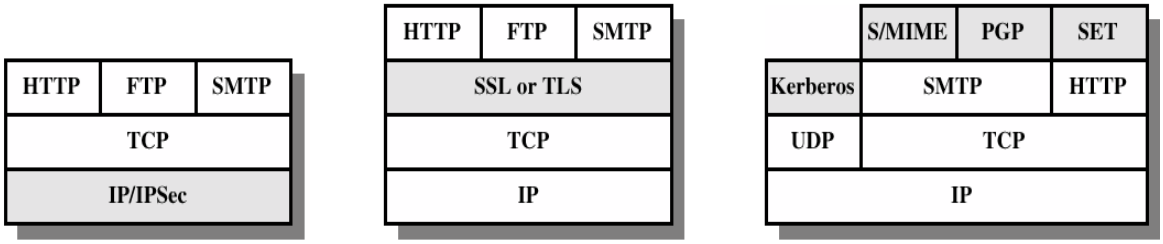
The difference between an end-to-end protocol and a link protocol is that the intermediate hosts play no part in an end-to-end protocol other than forwarding messages. Whereas, a link protocol describes how each pair of intermediate hosts processes each message.

Link encryption: Each host enciphers message and "next hop" host can read it i.e. intermediate hosts can read the message. For e.g. In PPP Encryption Control Protocol host gets message, deciphers it, figures out where to forward it, enciphers it in appropriate key and forwards it. Here each host shares key with neighbor and can be set on per-host or per-host-pair basis. Link encryption can protect headers of packets and it is possible to hide source and destination but, source can be deduced from traffic flows.

End-to-end encryption: Host enciphers message so host at other end of communication can read it i.e. message cannot be read at intermediate hosts for e.g. TELNET protocol where messages between client, server enciphered, and encipherment, decipherment occur only at these hosts. In this approach each host shares key with destination and can be set on per-host or per-host-pair basis. This approach cannot hide packet headers and attacker can read source, destination.

Security at Different Layers

The following figure shows the security at different layers



(a) Network Level (b) Transport Level (c) Application Level

Security at the Application Layer: E-Mail

a) Pretty Good Privacy (PGP):

PGP is a public key encryption package to protect e-mail and data files. It lets you communicate securely with people you've never met, with no secure channels needed for prior exchange of keys. It's well featured and fast, with sophisticated key management, digital signatures, data compression, and good ergonomic design. The actual operation of PGP is based on five services: authentication, confidentiality, compression, e-mail compatibility, and segmentation.

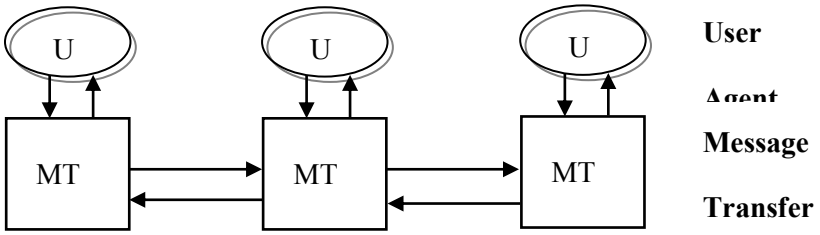
- PGP provides authentication via a digital signature scheme.
- PGP provides confidentiality by encrypting messages before transmission
- PGP compresses the message after applying the signature and before encryption. The

idea is to save space.

- PGP encrypts a message together with the signature (if not sent separately) resulting into a stream of arbitrary 8-bit octets. But since many e-mail systems permit only use of blocks consisting of ASCII text, PGP accommodates this by converting the raw 8-bit binary streams into streams of printable ASCII characters using a radix-64 conversion scheme. On receipt, the block is converted back from radix-64 format to binary.
- To accommodate e-mail size restrictions, PGP automatically segments email messages that are too long. However, the segmentation is done after all the housekeeping is done on the message, just before transmitting it. So the session key and signature appear only once at the beginning of the first segment transmitted. At receipt, the receiving PGP strips off all e-mail headers and reassembles the original mail.

b) Privacy Enchanced Mail (PEM):

The figure below shows a typical network mail service. The U (user agent) interacts directly with the sender. When the message is composed, the U hands it to the MT (message transport, or transfer, agent). The MT transfers the message to its destination host, or to another MT, which in turn transfers the message further. At the destination host, the MT invokes a user agent to deliver the message.



An attacker can read electronic mail at any of the computers on which MTs handling the message reside, as well as on the network itself. An attacker could also modify the message without the recipient detecting the change. Because authentication mechanisms are minimal and easily evaded, a sender could forge a letter from another and inject it into the message handling system at any MT, from which it would be forwarded to the destination. Finally, a sender could deny having sent a letter, and the recipient could not prove otherwise to a disinterested party. These four types of attacks (violation of confidentiality, authentication, message integrity, and nonrepudiation) make electronic mail nonsecure. So IETF with the goal of e-mail privacy develop electronic mail protocols that would provide the following services.

1. Confidentiality, by making the message unreadable except to the sender and recipient(s)
2. Origin authentication, by identifying the sender precisely
3. Data integrity, by ensuring that any changes in the message are easy to detect
4. Nonrepudiation of origin (if possible)

The protocols were named Privacy-Enhanced Electronic Mail (or PEM).

PEM vs. PGP

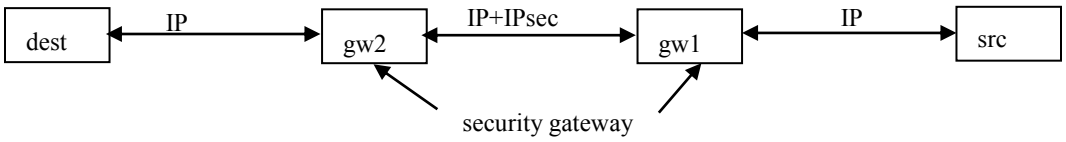
- **Use of different ciphers:** PGP uses IDEA cipher but PEM uses DES in CBC mode.
- **Use of certificate models:** PGP uses general "web of trust" but PEM uses hierarchical certification structure
- **Handling end of line:** PGP remaps end of line if message tagged "text", but leaves them alone if message tagged "binary" whereas PEM always remaps end of line.

Security at the Network Layer

- IPSec (Internet Protocol Security)

IPSec is a suite of authentication and encryption protocols developed by the Internet Engineering Task Force (IETF) and designed to address the inherent lack of security for IP-based networks.

It is a collection of protocols and mechanisms that provide confidentiality, authentication, message integrity, and replay detection at the IP layer. In the data transmission IPsec protect all messages sent along a path. If the IPsec mechanisms reside on an intermediate host (for example, a firewall or gateway), that host is called a security gateway.



Security at the Transport Layer

Secured Socket Layer (SSL)

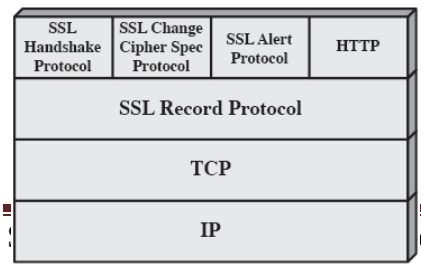
The Secure Socket Layer (SSL) is a standard developed by Netscape Corporation to provide security in WWW browsers and servers. The current version, SSLv3, is the basis for an Internet standard protocol under development. The newer protocol, the Transport Layer Security (TLS) protocol, is compatible with SSLv3 and has only minor changes. It has not yet been adopted formally.

SSL Main Goals

1. **Cryptography security:** One of the SSL protocol's primary goals is to establish a secure connection between two parties. A symmetric Encryption is used after an initial handshake to define a secret key.
2. **Reliability:** The connection is reliable. Message transport includes a message integrity check using a keyed MAC computed using secure hash functions.
3. **Interoperability:** Different applications should be able to successfully exchange cryptographic parameters without knowledge of one another's code.
4. **Extensibility:** Provide a framework that allows new public-key and bulk encryption methods to be incorporated as necessary. This will also achieve the goal of avoiding the need to implement an entire new security library.
5. **Relative efficiency:** Cryptographic operations tend to be highly CPU intensive. For this reason the SSL protocol has some options (such as caching and compression), which allow a reduction in the number of connections that need to be established from scratch and a reduction in network activity.

SSL Architecture

SSL, a set of protocols, uses TCP to provide reliable end to end service. SSLv3 consists of two layers (see figure below) supported by numerous cryptographic mechanisms. The lower layer called SSL Record Protocol provides the basic security services to various higher level protocols, particularly HTTP. There are three higher level protocols that are defined as parts of SSL namely SSL Handshake Protocol, the Change Cipher Spec Protocol, and Alert Protocol.



SSL works in terms of **connections** and **sessions** between clients and servers. An SSL **session** is an association between two peers. An SSL **connection** is the set of mechanisms used

to transport data in an SSL session. A single session may have many connections. Two peers may have many sessions active at the same time, but this is not common.

Each party keeps information related to a session with each peer. The data associated with a session includes the following information.

1. **Session identifier:** An arbitrary byte sequence chosen by the server to identify an active or resumable session state
2. **Peer certificate:** X509.v3 certificate of the peer. This may be null.
3. **Compression method:** The algorithm used to compress data prior to encryption.
4. **Cipher spec:** Specifies the bulk data encryption algorithm (null, DES, etc.) and a MAC algorithm (MD5 or SHA). It also defines attributes such as the `hash_size`.
5. **Master secret:** 48-byte secret shared between the client and server.
6. **Is resumable:** Defines whether the session can be used to initiate new connections.

A connection describes how data is sent to, and received from, the peer. Each party keeps information related to a connection. Each peer has its own parameters. The information associated with the connection includes the following.

1. **Server and client random:** Random data for server and client for each connection.
2. **Server write MAC secret:** Key for MAC operations on data written by the server.
3. **Client write MAC secret:** Key for MAC operations on data written by the client.
4. **Server write key:** Cipher key for encryption by the server and decryption by client.
5. **Client write key:** Cipher key for encryption by the client and decryption by server.
6. **Initialization vectors (IV):** When a block cipher in CBC mode is used, IV is maintained for each key. This field is first initialized by the SSL handshake protocol then final ciphertext block from each record is preserved for use with the next record.
7. **Sequence numbers:** Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a **change cipher spec** message, the appropriate sequence number is set to zero. Sequence numbers are of type uint64 and may not exceed 2^{64}-1.

The SSL Handshake Protocol

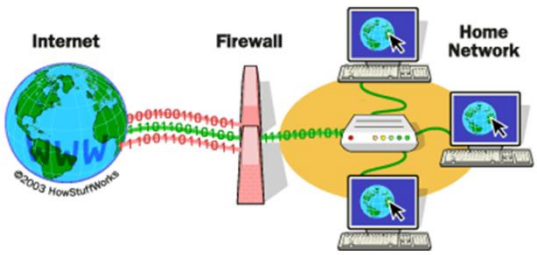
This protocol, also called the key-exchange protocol, is responsible for establishing a secure session between two parties. The SSL handshake protocol can be divided to several important stages:

1. Authenticate the server to the client.
2. Negotiation of common cryptographic algorithms, that both server and client support.
3. Authenticate the client to the server (optional).
4. Using public-key encryption to exchange cryptography parameters (shared secrets).
5. Establish an encrypted SSL connection.

Firewalls

Firewall is hardware device or software applications that act as filters between a company's private network and the internet. It protects networked computers from intentional hostile intrusion that could compromise confidentiality or result in data corruption or denial of service by enforcing an access control policy between two networks.

The main purpose of a firewall system is to control access to or from a protected network (i.e., a site). It implements a network access policy by forcing connections to pass through the firewall, where they can be examined and evaluated. A firewall system can be a router, a personal computer, a host, or a collection of hosts, set up specifically to shield a site or subnet from protocols and services that can be abused from hosts outside the subnet. A firewall system is usually located at a higher level gateway, such as a site's connection to the Internet, however firewall systems can be located at lower-level gateways to provide protection for some smaller collection of hosts or subnets. The main function of a firewall is to centralize access control. A firewall serves as the gatekeeper between the untrusted Internet and the more trusted internal networks. The earliest firewalls were simply routers.



Firewalls provide several types of protection:

- They can block unwanted traffic.
- They can direct incoming traffic to more trustworthy internal systems.

- They hide vulnerable systems, which can't easily be secured from the Internet.
- They can log traffic to and from the private network.
- They can hide information like system names, network topology, network device types, and internal user ID's from the Internet.
- They can provide more robust authentication than standard applications might be able to do.