Compiled Notes of

INTRODUCTION TO CRYPTOGRAPHY

For BSc. CSIT

SANDIP PAUDEL

2019

Compiled Notes of

INTRODUCTION TO CRYPTOGRAPHY

For BSc. CSIT

Sandip Paudel

Revised and Compiled by Sandip Paudel

Contents

Acknowledgement

The compilation of this study material would not have been possible without the contents and guidance covered in the textbooks; "*Cryptography and Network Security: Principles and practice, 5th edition by* ***William Stallings***", "C*ryptography: Theory and Practice, 3rd edition by* ***Douglas R. Stinson***". I am indebted for these authors for their endless effort in the field of cryptography, which helped me find the way past all the obstacles. My sincere gratitude goes to **Mr. Tej Shahi,** the original author of this compiled version for his support and belief on me for the revision of his original compilation work. It was a great honor and real pleasure researching his works.

Having said all above, I am solely responsible for the originality of data and this work.

Unit 1

Introduction to Cryptography

Revised and Compiled by Sandip Paudel

Cryptology

The combined study of ***cryptography*** and ***cryptanalysis*** is the cryptology. Most of the times, we use cryptography and cryptology in the same way (Stallings, 2011).

FIGURE 1: COMPONENTS OF CRYPTOLOGY

Cryptography

The word '***Cryptography***' is from the combination of two Greek words '***kryptós***' meaning cryptic, secret or hidden and '***graphein***' meaning representation or writing. Thus, cryptography in this sense means '***secret writing***', which is considered as the art and science of the information hiding.

In the ancient days, cryptography meant 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**) of the message.

This field is at the intersection of the mathematics and the computer science with application in many fields like computer security, electronic commerce, telecommunication, etc.

Cryptanalysis

Literally, cryptanalysis means the breaking down of the encrypted codes. Cryptanalysis encompasses all the techniques to recover the plaintext and/or key from the ciphertext. It is the reverse process of the cryptography.

William Friedman coined the term cryptanalysis in 1920, by combining two Greek words '***kryptós***' meaning hidden and '***analỳein***' meaning to loosen or to unite. In this sense, cryptanalysis is the study of the methods for obtaining the meaning of encrypted information, without access to the secret information that is normally required to do so.

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

The Objectives of Cryptography

Confidentiality: only the authorized recipient should be able to extract the content of the cypher. In addition, obtaining information about the content of the message (such as a statistical distribution of certain characters) should not be possible.

Message Integrity: the recipient must be able to determine if the message was altered during transmission.

Authentication: the recipient should be able to identify the sender and verify if it was him who sent the message.

Irrecoverability: it should not be possible to deny the authorship of the message.

Encryption

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.

The use of encryption techniques is historic, dating back to the time when Julius Caesar started using a technique called Caesar's cipher for passing information to his soldiers. Since then, encryption techniques have been extensively used in military purposes to conceal the information from the enemy. Nowadays, to gain the confidentiality or to provide the secure environment for the various businesses and end users of the system, encryption is used in the areas like communication, internet banking, digital right management, etc.

The consideration of cryptography was on message confidentiality (encryption) in the past; however, cryptography evolved encompassing the study and practices of authentication, digital signatures, integrity checking, and key management, etc. Encryption mostly provides the secrecy on the message transmission over the communication network, also known as the 'confidentiality of message'. The sender and the receiver are aware of the keys and can decipher the message.

Decryption

Decryption is the reverse process, transforming an encrypted message back into its normal or original form. The use of the key is equally important in decryption process.

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.

FIGURE 2: ENCRYPTION-DECRYPTION

Security Attacks

If the cryptanalyst learns or make use of the information of the system but does not influence the system, the kind of attack is considered passive. Contrary to it, the attack is active if it involves the modification of the message.

The following are the types of possible **passive attacks**:

1. **Release of message contents**: Eve (the eavesdropper) reads the message from Bob (the sender) to Alice (the recipient), the confidentiality of the message from Bob to Alice only reveals to Eve.
2. **Traffic analysis**: Eve just observes the pattern of message from Bob to Alice.

The following are the types of possible **active attacks**:

1. **Masquerade**: Eve captures the authentication sequence of Bob and sends message to Alice.
2. **Replay**: Eve passively captures the message form Bob before reaching Alice and then retransmits the message to Alice, creating unauthorized effect.
3. **Modification of message**: Eve captures and modifies the message from Bob to Alice.
4. **Denial of services**: Eve disrupts the services of the communication channel to Bob and establishes communication with Alice.

Cryptanalysis

Cryptanalysis generally depends on the assumptions about how much the system under observation could be available. It is normally assumed that the general algorithm is known, based on the Kerckhoff's principle of "*the enemy knows the system.*"

1. **Ciphertext – only**: the attacker has access only to the collection of ciphertexts.
2. **Known – plaintext**: the attacker has a set of ciphertexts to which he knows the corresponding plaintext.
3. **Chosen – plaintext/ chosen – ciphertext**: the attacker can get ciphertext/plaintext related to any arbitrary set of the corresponding plaintext/ciphertext of his own choice.

4. **Adaptively-chosen-plaintext**:"... the attacker has the ability to make his choice of the inputs to the encryption function based on the previous chosen plaintext queries and their corresponding ciphertexts. The scenario is clearly more powerful than the basic chosen plaintext attack, but is probably less practical in real life since it requires interaction of the attacker with the encryption device." (Biryukov, Adaptive Chosen Plaintext Attack, 2005)
5. **Adaptively-chosen-ciphertext**:"... the attacker has the ability to make his or her choice of the inputs to the decryption function based on the previous chosen ciphertext queries. The scenario is clearly more powerful than the basic chosen ciphertext attack and thus less realistic. However, the attack may be quite practical in the public-key setting ..." (Biryukov, Adaptive Chosen Ciphertext Attack, 2011)

Every attack depends on the mathematics or statistics. Mathematical attacks analyzes the underlying mathematics of the cryptographic algorithm whereas the statistical attacks make assumptions about the distribution of the letters, pairs of letters (diagrams), triplets of letters (trigrams), etc. and examine ciphertext by correlating properties with the assumptions.

Brute force Attack

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.

Key

A key is a parameter or a piece of information used to determine the output of cryptographic algorithm. During the encryption process, key determines the transformation of plaintext to the cipher text and the vice versa on decryption.
Keys are equally popular 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. Nevertheless, the length of the key also stands as the important issue in cryptography. 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. The random choice of the key is the must, which might take time long enough for the attackers to guess and use easily unlike the systematic or planned key.

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.

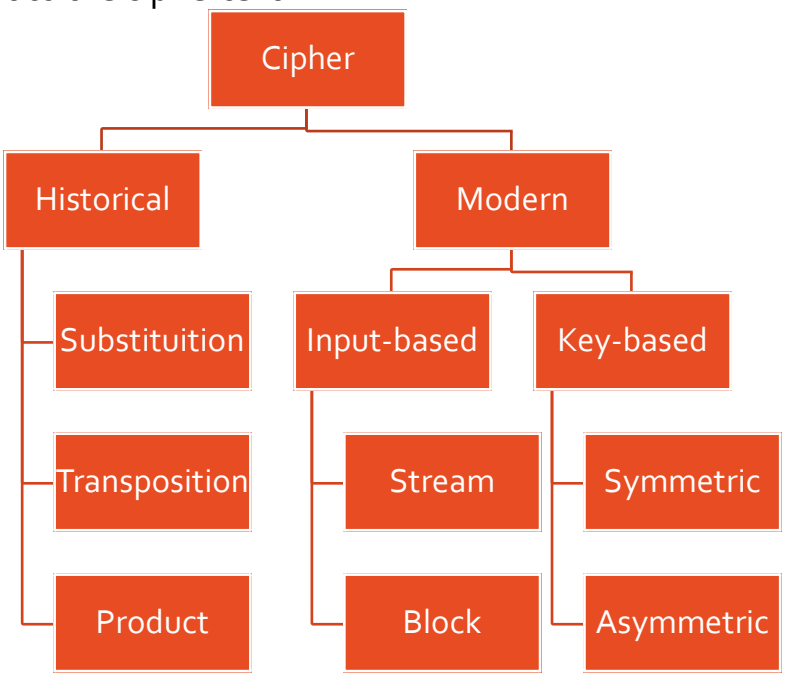


FIGURE 3: CLASSIFICATION OF THE CIPHERS

Cryptosystem

Definition 1.1: A *cryptosystem* is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where the following conditions are satisfied:

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

FIGURE 4: DEFINITION: CRYPTOSYSTEM (STINSON, 2005)

Alternatively, Cryptosystem is a 5-tuple/quintuple (E, D, M, K, C), where,

E set of encryption functions e: M x K → C,
D set of decryption functions d: C x K → M,
M set of plaintexts,
K set of keys,
C set of ciphertexts,

Example:

$M = \{\text{sequences of letters}\}$

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

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

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

$C = M$

FIGURE 5: AN EXAMPLE OF CAESAR CIPHER

Shannon's Conventional Cryptosystem

Shannon in 1948 introduced two concepts on information processing those act as basic building blocks for designing cryptographic systems. They are:

Diffusion

It means to obscure or conceal the statistical structure of the plaintext from the ciphertext, by having each ciphertext digit be affected by more than one plaintext digit. In other words, to have the statistical structure of the plaintext be dissipated into long-range statistics of the ciphertext.

Confusion

It means to obscure the statistical dependence between the encryption key and the ciphertext to thwart attempts to discover the key. Even a simple linear substitution generates little confusion. The higher confusion can be achieved by using complex substitution algorithms.

On this basis, a simple symmetric cryptography relies on three algorithms:

1. a key generator which generates a secret key in a cryptographically random or pseudorandom way;
2. an encryption algorithm which transforms a plaintext into a ciphertext using a secret key;
3. a decryption algorithm which transforms a ciphertext back into the plaintext using the secret key

Malicious Logic

Malicious logic is a set of instructions that cause a site's security policy to be violated. 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 codes: 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 effects to the system. In the other words, the malicious codes are the programmed threats. The following figure provides an overall taxonomy of Malicious Code.

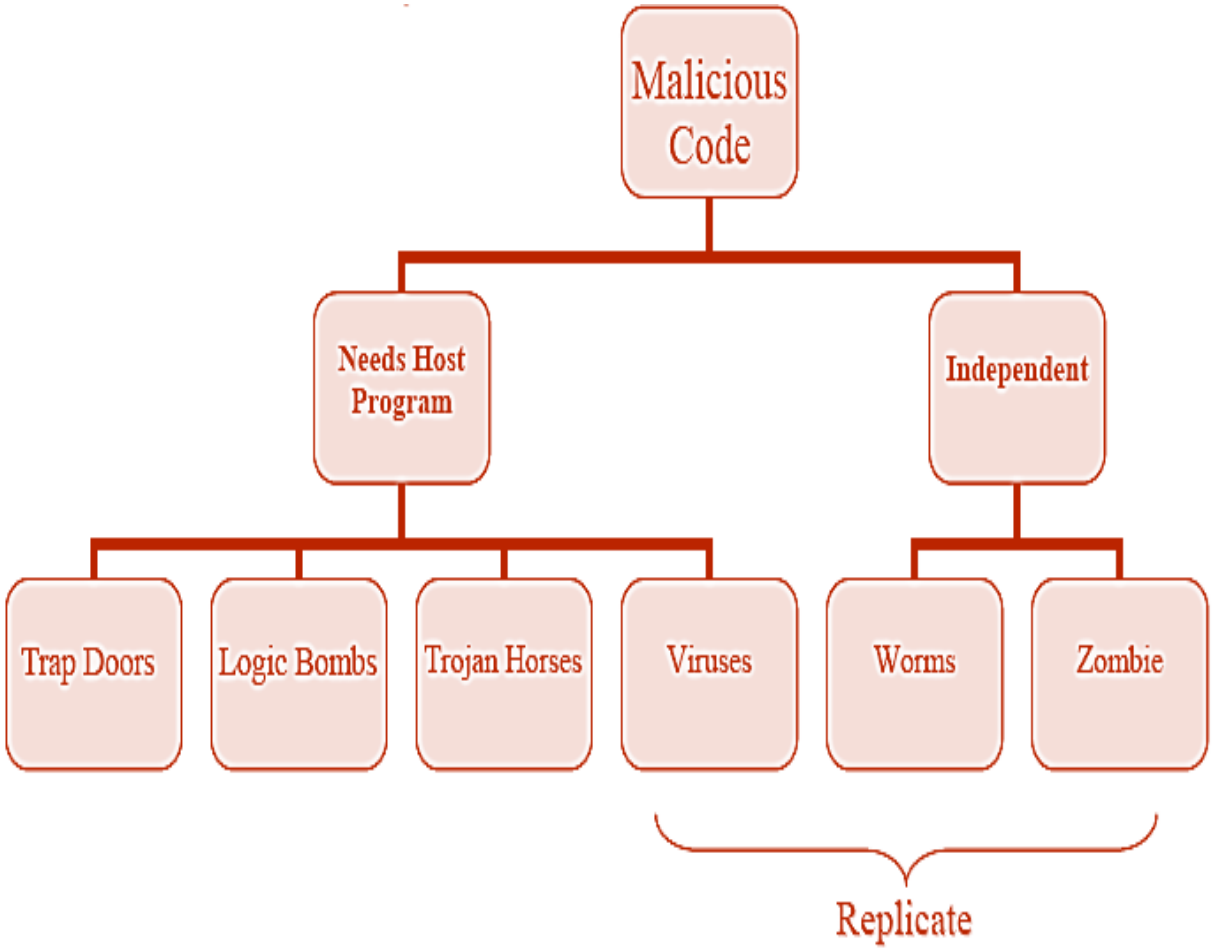


FIGURE 6: CLASSIFICATION OF MALICIOUS CODES

Trojan Horse

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.

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 1: 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 2: 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 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.

Virus

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.

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.

Example 1: 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 2: 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.

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.

Classical Cryptography

A classical cipher, sometimes also called the historical cipher, uses the processes like substitution or transposition or the combination of both called as the product. These ciphers use the single key for both encryption and decryption. To reduce the cipher attacks in substitution, instead of monoalphabetic – a letter for letter, polyalphabetic – one or more letters for a single letter substitution can be used.

Caesar Cipher

It is the simple shift monoalphabetic classical cipher adopting substitution mechanism. 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.

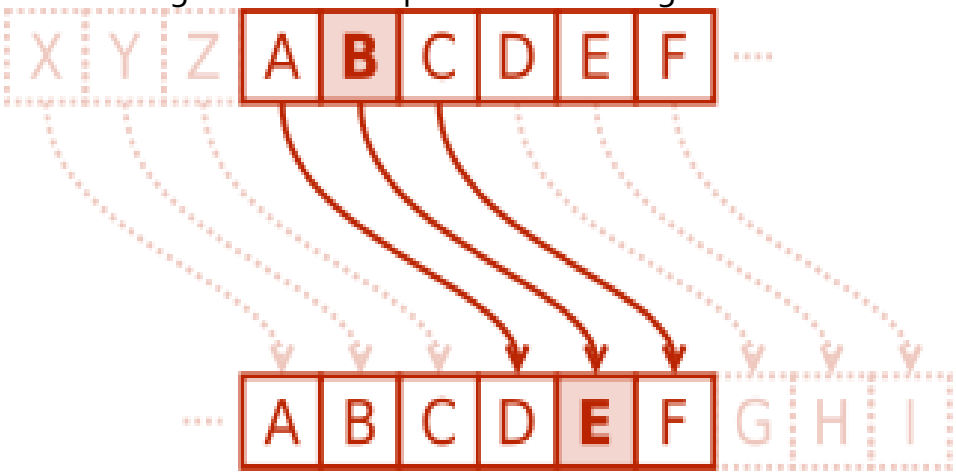


FIGURE 7: CAESAR CIPHER

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.

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. Mathematically, encryption of a letter c by a shift k is 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: 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 One-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.

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 ciphered message 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

Rail Fence's Problem

Similarly, if we have 3 "rails" and a message of "THIS IS THE PLAINTEXT", the cipher message writes out "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):

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

We 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.

Vigenère Cipher

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

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

FIGURE 8: A VIGNERE TABLEU

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.

One-time Pad (Simple X-OR Cipher)

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.

Play-Fair Cipher

The best-known digraph substitution cipher is the Playfair, invented in 1854 by Sir Charles Wheatstone, which treats digrams in the plaintext as single units and translates these units into ciphertext digrams. Here, the mnemonic aid used to carry out the encryption is a 5 × 5-square matrix containing the letters of the alphabet (I and J are treated as the same letter). A keyword, MONARCHY in this example, is filled in first, and the remaining unused letters of the alphabet are entered in their lexicographic order:

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

Plaintext digraphs are encrypted with the matrix by first locating the two plaintext letters in the matrix. They are (1) in different rows and columns; (2) in the same row; (3) in the same column; or (4) alike. The corresponding encryption (replacement) rules are the following:

1. When the two letters are in different rows and columns, each is replaced by the letter that is in the same row but in the other column; i.e., to encrypt WE, W is replaced by U and E by G.
2. When A and R are in the same row, A is encrypted as R and R (reading the row cyclically) as M.
3. When I and S are in the same column, I is encrypted as S and S as X.
4. When a double letter occurs, a spurious symbol, say Q, is introduced so that the MM in SUMMER is encrypted as NL for MQ and CL for ME.
5. An X is appended to the end of the plaintext if necessary to give the plaintext an even number of letters.

Encrypting the familiar plaintext example using Playfair array yields:

```
Plaintext: WE ARE DISCOVERED SAVE YOURSELFX
Cipher:    UG RMK CSXHMUFMKB TOXG CMVATLUIV
```

Vernam's Cipher

Gilbert Vernam, 1917, proposed a bit-wise exclusive-OR of the message stream with a truly random zero-one stream which was shared by sender and recipient. For example:

```
SENDING
-------
message: 0 0 1 0 1 1 0 1 0 1 1 1 ...
pad:     1 0 0 1 1 1 0 0 1 0 1 1 ...
XOR      --------------------------
cipher:  1 0 1 1 0 0 0 1 1 1 0 0 ...
RECEIVING
---------
cipher:  1 0 1 1 0 0 0 1 1 1 0 0 ...
pad:     1 0 0 1 1 1 0 0 1 0 1 1 ...
XOR      --------------------------
message: 0 0 1 0 1 1 0 1 0 1 1 1 ...
```

This cipher is unbreakable in a very strong sense. The intuition is that any message can be transformed into any cipher (of the same length) by a pad, and all transformations are equally likely. Given a two letter message, there is a pad which adds to the message to give OK, and another pad which adds to the message to give NO. Since either of these pads are equally likely, the message is equally likely to be OK or NO.

Rotor Machines

In the 1920s, various mechanical encryption devices were invented to automate the process of encryption. Most were based on the concept of a rotor, a mechanical wheel wired to perform a general substitution.

A rotor machine has a keyboard and a series of rotors, and implements a version of the Vigenère cipher. Each rotor is an arbitrary permutation of the alphabet, has 26 positions, and performs a simple substitution. For example, a rotor might be wired to substitute "F" for "A," "U" for "B," "L" for "C," and so on. In addition, the output pins of one rotor are connected to the input pins of the next.

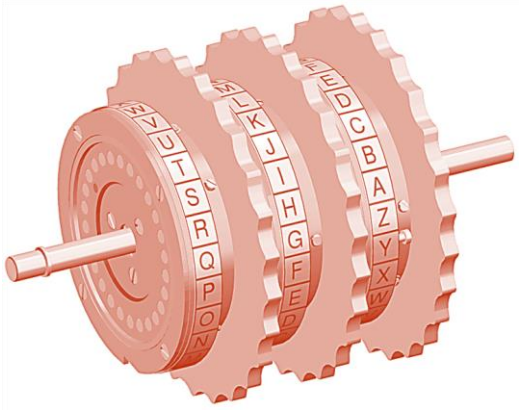


FIGURE 9: A 3-STEP ROTOR MACHINE

For example, in a 4-rotor machine the first rotor might substitute "F" for "A," the second might substitute "Y" for "F," the third might substitute "E" for "Y," and the fourth might substitute "C" for "E"; "C" would be the output ciphertext. Then some of the rotors shift, so next time the substitutions will be different.

It is the combination of several rotors and the gears moving them that makes the machine secure. Because the rotors all move at different rates, the period for an n-rotor machine is 26n. Some rotor machines can also have a different number of positions on each rotor, further frustrating cryptanalysis.

Enigma

The best-known rotor device is the Enigma. The Germans used the Enigma during World War II. Arthur Scherbius and Arvid Gerhard Damm in Europe invented the

idea. Arthur Scherbius patented it in the United States. The Germans beefed up the basic design considerably for wartime use. The German Enigma had three rotors, chosen from a set of five, a plug-board that slightly permuted the plaintext, and a reflecting rotor that caused each rotor to operate on each plaintext letter twice. As complicated as the Enigma was, it was broken during World War II. First, a team of Polish cryptographers broke the German Enigma and explained their attack to the British. The Germans modified their Enigma as the war progressed, and the British continued to cryptanalyze the new versions.

Hagelin

Arvid Gerhard Damm of Sweden, invested by K.W. Hagelin and Emanuel Nobel, was the last of the four cipher machine inventors at the end of WW1 to file for a patent on his rotor based cipher machine. His was the only rotor machine to have irregular stepping, but his machines performed erratically. K.W. Hagelin had his son, Boris, join the firm in 1922 in order to protect his investment.

Boris Hagelin (1892-1983) was a Swedish engineer, born in Russia, who took over management of Damm's company in 1925. The Swedish army made the first large purchase in 1926 and Damm died early the following year. Hagelin bought the firm and went on to have the most successful and controversial cipher machine company in history.

The first cipher machine manufactured by Damm was the B-21, which had two rotors and two pairs of pinwheels to produce an irregular stepping action for the two rotors. The B-21 had a light panel like the Enigma machine. In 1932, the French Army wanted a smaller version of the B-21 with printing capability, so Hagelin designed and produced the B-211, which was successful and helped to fund further development. In 1934, Hagelin got the idea to adapt the calculating function from a mechanical moneychanger into a new type of cipher machine. He adapted the pinwheel in the moneychanger into his famous pin and lug mechanism. This machine was called the C-35, the model number designating the year it was first produced.

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 vector represents the first three letters of the plaintext $\begin{pmatrix}15\\ 0\\ 24\end{pmatrix}$

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

The ciphertext thus obtained 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.

Steganography

Steganography serves to hide secret messages in other messages, such that the secret's very existence is concealed. Generally, the sender writes an innocuous message and then conceals a secret message on the same piece of paper. Historical tricks include invisible inks, tiny pin punctures on selected characters, minute differences between handwritten characters, pencil marks on typewritten characters, grilles that cover most of the message except for a few characters, and so on.

More recently, people are hiding secret messages in graphic images. Replace the least significant bit of each byte of the image with the bits of the message. The graphical image will not change appreciably—most graphics standards specify more gradations of color than the human eye can notice—and the message can be stripped out at the receiving end. One can store a 64-kilobyte message in a 1024 × 1024 grey-scale picture this way.

Unit 2

Basics of Modern Cryptography

Revised and Compiled by Sandip Paudel

Modern Ciphers

Modern encryption methods are based on the criteria as input data and key used during encryption.

Based on Input data

Stream Ciphers: the plaintext is converted into ciphertext stream by stream to encrypt the continuous stream of data
Block Ciphers: the plaintext is converted into ciphertext block by block to encrypt the data of fixed size

Based on Key used

Symmetric key: uses single public key for encryption as well as decryption
Asymmetric key: uses two keys, namely, public key for encryption and private key for decryption

Public Key Cryptography Protocol (Model)

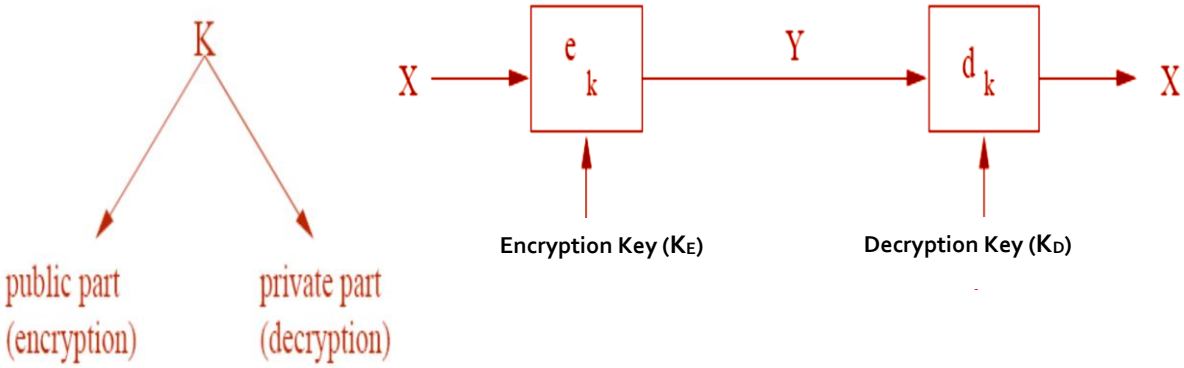


FIGURE 10: A PUBLIC KEY CRYPTOGRAPHY MODEL

Assumptions

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

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

Families of Public-Key (PK) algorithms

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

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:

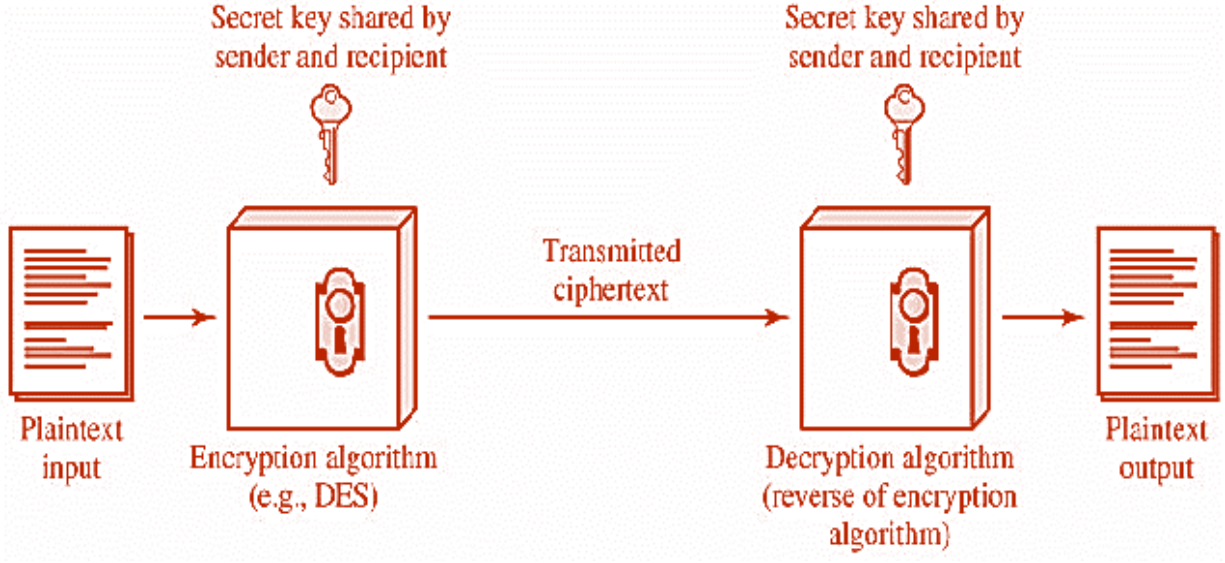


FIGURE 11: A SYMMETRIC CRYPTOGRAPHY

A symmetric cipher model can be defined with the help of the following components:

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

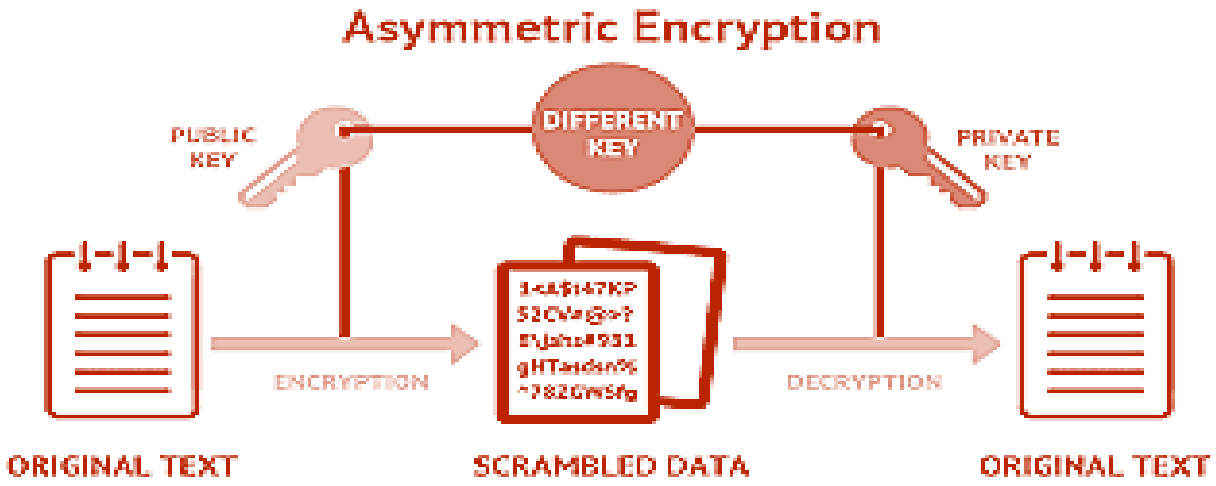


FIGURE 12: AN ASYMMETRIC CRYPTOGRAPHY

An asymmetric cryptography was developed to solve the following issues in the symmetric cryptography:

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**.

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 distribution	Must be out-of-band	Public key can be used to distribute other keys
Speed	Fast	Slow; typically, 10,000 times slower than secret key

Digital signature

In the following 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.

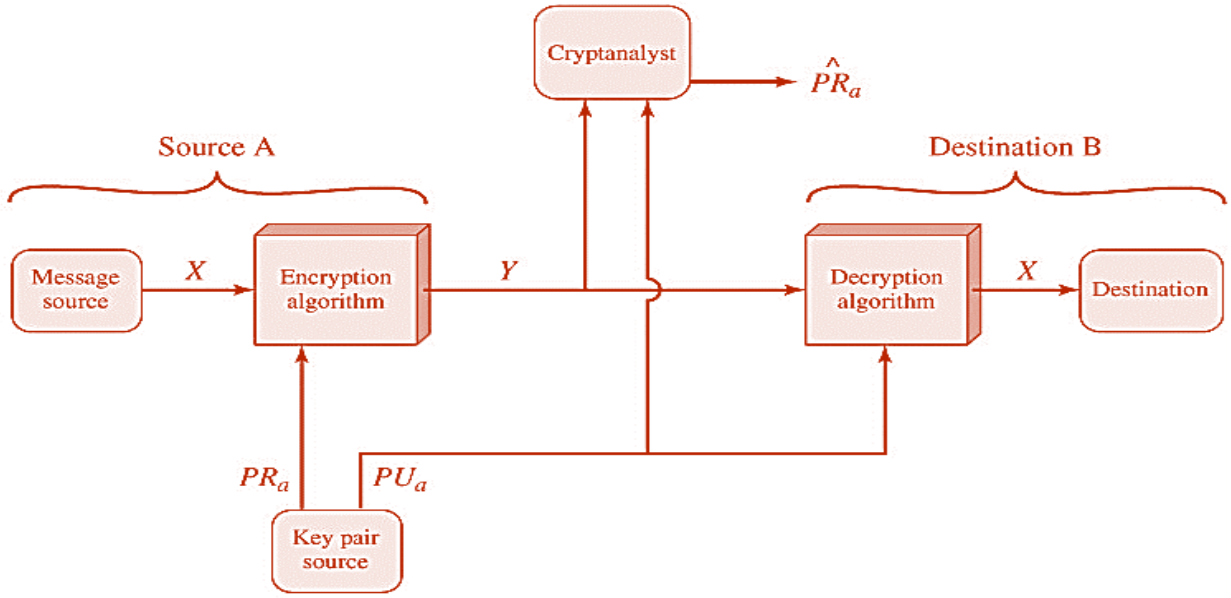


FIGURE 13: A DIGITAL SIGNATURE MODEL

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**.

Unit - 3

Conventional Encryption / Secret Key Cryptography

Shannon's Theory

Shannon's theoretical components act as the basic building block for the modern cryptography. The security of the modern information system depends upon the computational security, provable security and unconditional security (Stinson, 2005).

Computational Security

The information system is computationally secure if it requires the cryptanalyst at least 'N' iterations of operations to breakdown the encryption algorithm, where 'N' is a very large but finite number. Thus, the computational security deals with the computational effort required to breakdown the encryption algorithm. Other way, from this view it could be understood that no any cryptosystem is fully secure or it could be decrypted.

Provable Security

In addition to the computational aspect of the security, if we could demonstrate that a problem reaches its solution by some efficient algorithm, it is possible that using some relatively difficult problem as algorithm for cryptosystem could provide high-end security too the information system. This view provides the way for using NP-complete problems as algorithm for a given cryptosystem. This whole provides the provable security to any modern information system.

Unconditional Security

Any modern information system reaches the unconditional security state one its encryption algorithm proves itself to be unbreakable. It means that the cryptosystem is unconditionally secure because it is impossible to breakdown the encryption algorithm even with the infinite computational resources.

Encryption Algorithms

Block cipher and stream cipher

A stream cipher is one that encrypts a digital data stream 1 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 the 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.

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.
- **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.

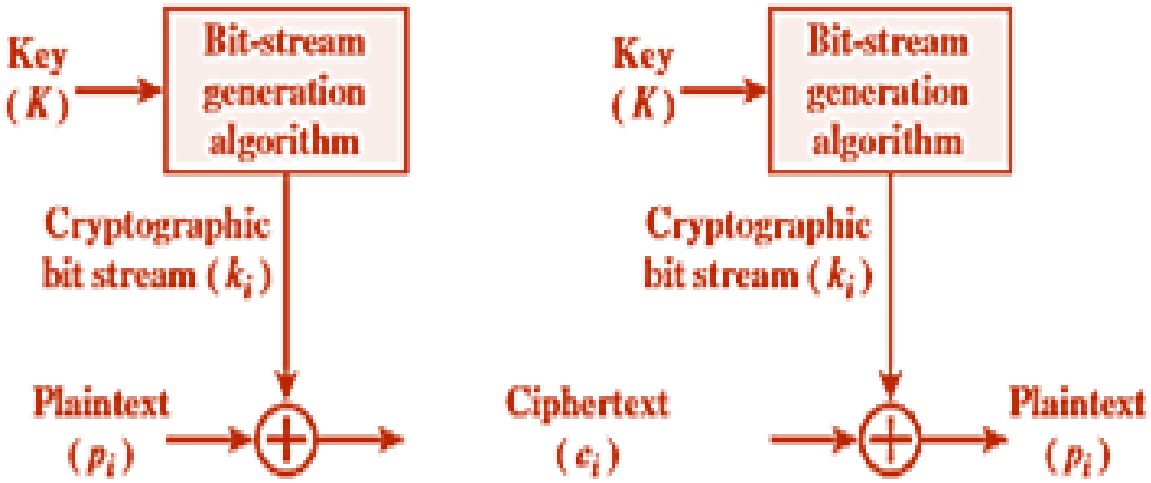


FIGURE 14: STREAM CIPHERS

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.
- **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.

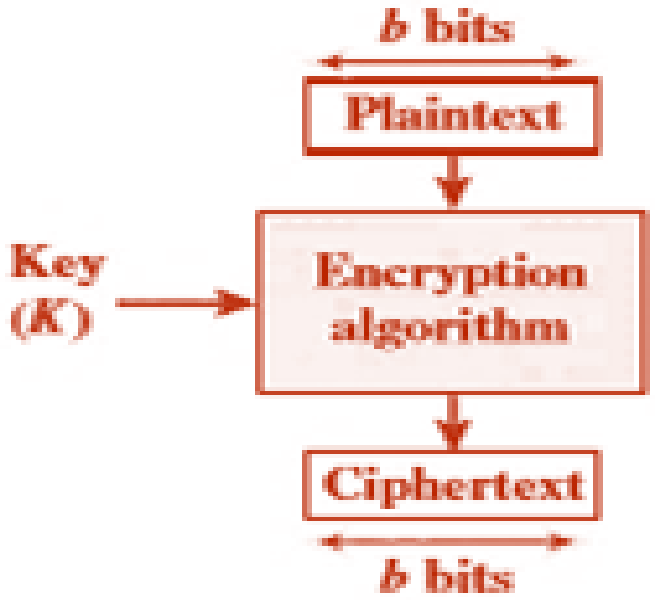


FIGURE 15: BLOCK CIPHER

Feistel Cipher

Feistel proposed that we could 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 result or product is cryptographically stronger than 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 2k possible transformations, rather than the 2n! transformations available with the ideal block cipher.

Structure

In the structure proposed by Feistel, the inputs to the encryption algorithm are a plaintext block of length '2w' bits and a key 'K'. The input block is divided into two sub-blocks and each of them passes through the 'n+1' rounds of the encryption mechanism. The first 'n' rounds are similar to each other while the last round ('n+1st' round) is just the permutation of the received outputs for each sub-blocks after 'n+1' rounds. 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.

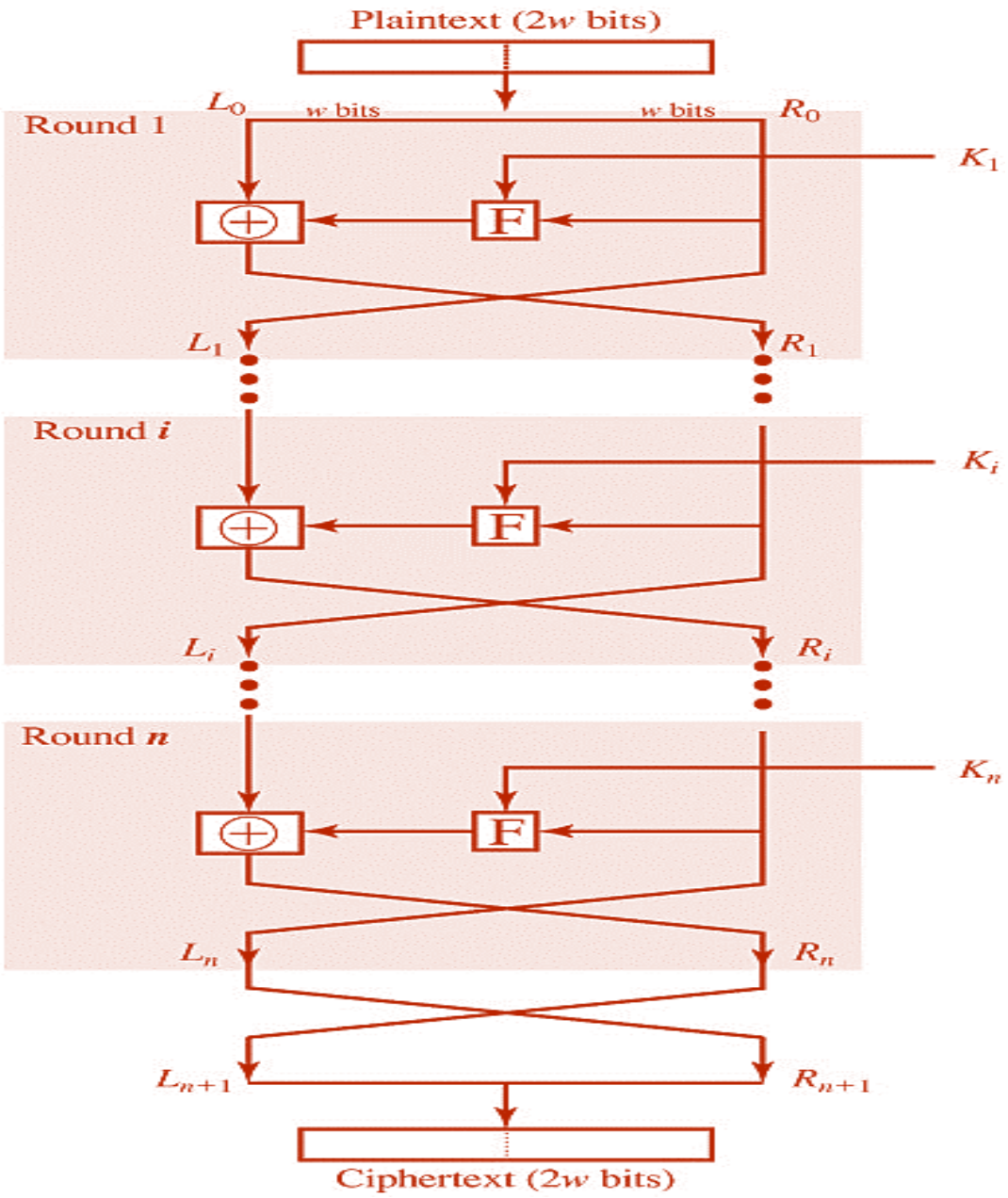


FIGURE 16: FEISTEL CIPHER STRUCTURE

Mechanism

The plaintext block, which of 2w bits is divided into two halves, Lo and Ro, w bits each. The two halves of the data pass through 'n' rounds of processing followed by the one round of permutation and then combine to produce the ciphertext block of 2w bits. 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. one of the most common way to generate the sub-keys from the main key is the randomization of each bits in the main key or, sometimes the permutation of the bits in the main key. The

sub-keys are generated in such a way that each sub-keys Ki are uniquely distinguishable and are different from the main key K.

- On the right half of the cipher, Ri-1 is passed through the **round function (F)**, which encrypts the Ri-1 with the sub-key Ki for the current round. The round function for all the 'n' rounds have the same general structure but constrained by the sub-key for each round.
  F: (Ri-1, Ki) → Ri-1´
- The output of the round function is the used to calculate the exclusive-OR (XOR) with the left half sub-block input for the current round.
  (Ri-1´ ⊕ Li-1) → Li-1´
- Finally, the produced left half sub-block output is permuted with the right half sub-block input for the same round to generate the outputs for the current round (i^{th} round). Here, **permutation** means the simple interchange of the outputs (left half sub-block is substituted with the right half sub-block, and vice-versa).
  Permute (Li-1´, Ri-1) → Li
  Permute (Ri-1, Li-1´) → Ri

Interestingly, the produced outputs on every round are different and they serve as the sub-block inputs for the next round ('$i+1^{st}$' round).

Data Encryption Standard (DES)

The *Data Encryption Standard (DES)* is a cipher that was 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 insecure for many applications. This is chiefly due to the smaller size 56-bits key. In January 1999, distributed.net and the Electronic Frontier Foundation (EFF) collaborated to break a DES key publicly in 22 hours and 15 minutes.

Structure

DES is the standard version of the Feistel cipher repeated for the 16 rounds of encryption process. It is a block cipher with the block size of 64 bits. 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. DES uses a key to customize the transformation, so that those who

know the particular key used to encrypt can supposedly only perform decryption. The key consists of 64 bits; however, the encryption algorithm uses only 56 of these and he remaining 8 bits are solely responsible for checking parity, and are thereafter discarded. Hence, the effective key length is 56-bits.

Mechanism

There are 16 identical stages of processing, termed rounds and two permutation rounds, an initial permutation (IP) at the beginning of the encryption and a final permutation (FP) after the 16 rounds. IP and FP are inverses, that is to say, IP "undoes" the action of FP, and vice versa.

The plaintext block of 64-bits is divided into two halves, Lo and Ro, 32-bits each. The two halves of the data pass through IP round followed by 16 encryption rounds and the FP round. The 32-bits sized outputs of the FP round are combined to produce the ciphertext block of 64-bits. Sixteen 48-bits sub-key are derived from the 56-bits main key, one to be used in each round. The generation of the sub-keys is similar to the process in the Feistel cipher.

- Before the main rounds, the input block is divided into two 32-bit halves and interchanged; this crisscrossing is known as IP of the ***Feistel scheme***.
- 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 ***Mangler Function (F)***, an equivalent to the Feistel function (F) in Feistel cipher, accommodates the whole encryption process for one round of the DES. The F-function 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 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.
    - **Substitution**: after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by the substitution boxes.
    - **Permutation**: finally, the 32 outputs from the substitution are rearranged according to a fixed permutation.
- Finally the FP round permutes the two 32-bits sub-blocks obtained as the output of the 16^{th} round of encryption. These sub-blocks are arranged again to produce the 64-bits ciphertext block.

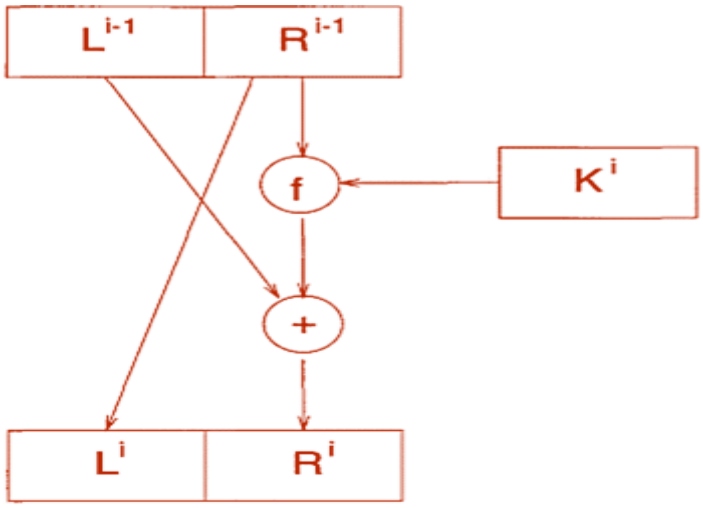


FIGURE 17: ONE ROUND OF DES

Mathematically, the DES is the combination of the following functions altogether:

M: IP, F, FP

F: e_k(Ki, Ri-1) → Ri

- ***divide*** *(64-bits Pi) → (32-bits L, 32-bits R)*
- ***IP*** *(L, R) → Lo (32-bits)*
  IP *(R, L) → Ro (32-bits)*
- ***Mangler Function: F(expand, keyMix, XOR, substitute, permute)***
  - *expand (Lo) → Lo´ (48-bits)*
    expand (Ro) → Ro´ (48-bits)
  - *keyMix (Ro´, k) → Ro1´ (48-bits)*
  - *XOR: Lo´ ⊕ Ro1´ → Lo1´ (48-bits)*
  - *substitute (Lo1´) → L1´ (32-bits)*
    substitute (Ro´) → R1´ (32-bits)
  - *permute (L1´, R1´) → L1 (32-bits)*
    permute (R1´, L1´) → R1 (32-bits)
- ***FP*** *(L16, R16) → L17 (32-bits)*
  FP *(R16, L16) → R17 (32-bits)*
- ***combine*** *(L17, R17) → Ci (64-bits)*

Multiple Encryption DES

Given the potential vulnerability of DES to a brute-force attack, there has been considerable interest in finding an alternative. One approach is to design a completely new algorithm, of which AES is a prime example. Another alternative, which would preserve the existing investment in software and equipment, is to use multiple encryption with DES and multiple keys. We here consider the simplest

example of the multiple encryption with DES and the widely accepted triple DES (3DES).

The simplest form of multiple encryption has two encryption stages and two keys. Given a plaintext and two encryption keys and, ciphertext is generated as:

$C = E(K_2, E(K_1, P))$

Decryption requires that the keys be applied in reverse order:

$P = D(K_1, D(K_2, C))$

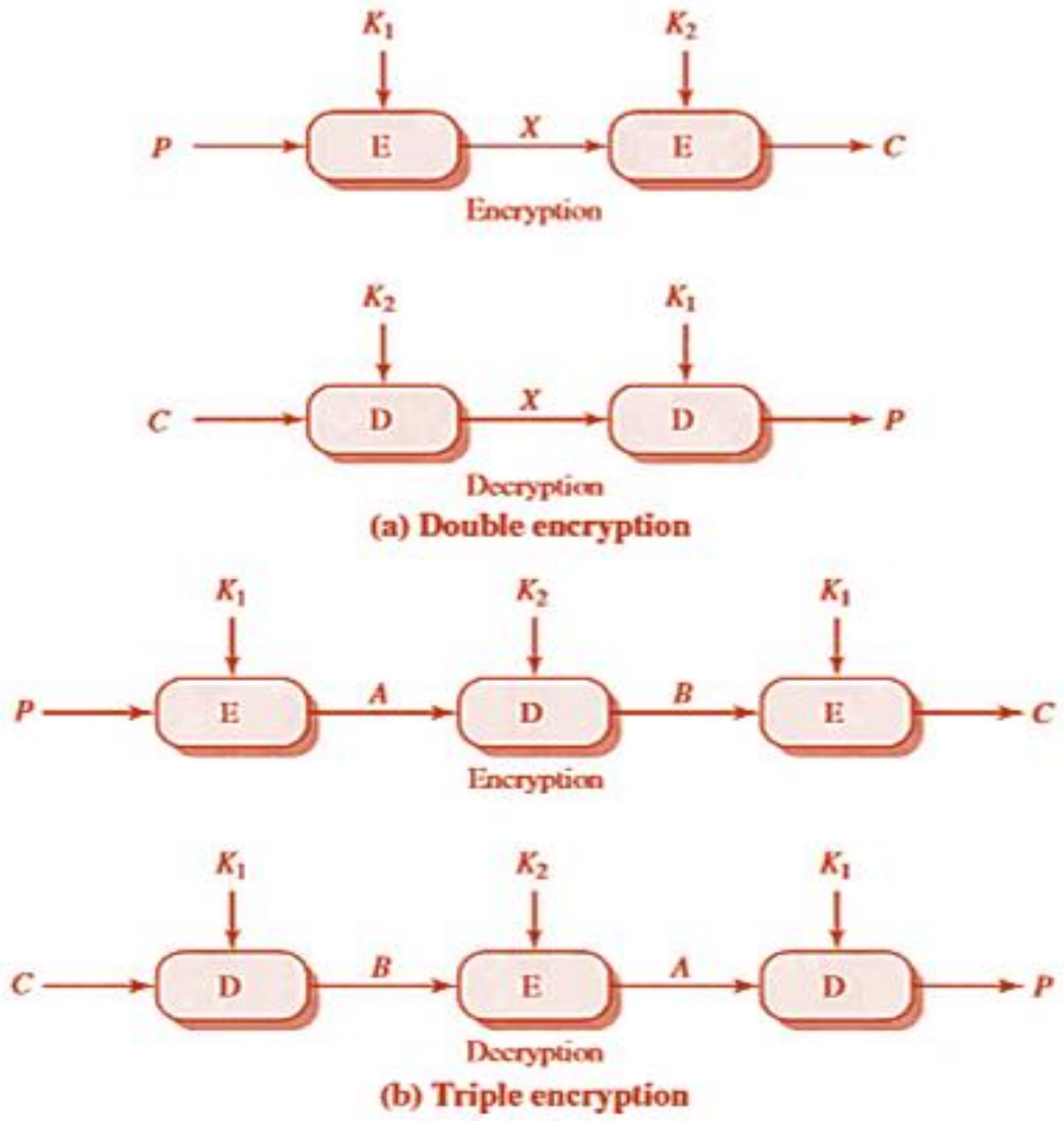


FIGURE 18: MULTIPLE ENCRYPTION DES MODEL

It is reasonable to assume that if DES is used twice with different keys, it will produce one of the many mappings that are not defined by a single application of DES.

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.

Structure

IDEA operates on 64-bit blocks using a 128-bit key, and consists of a series of eight identical rounds and an output round (the half-round). 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 ʘ), and bitwise eXclusive OR (XOR) (denoted by⊕).

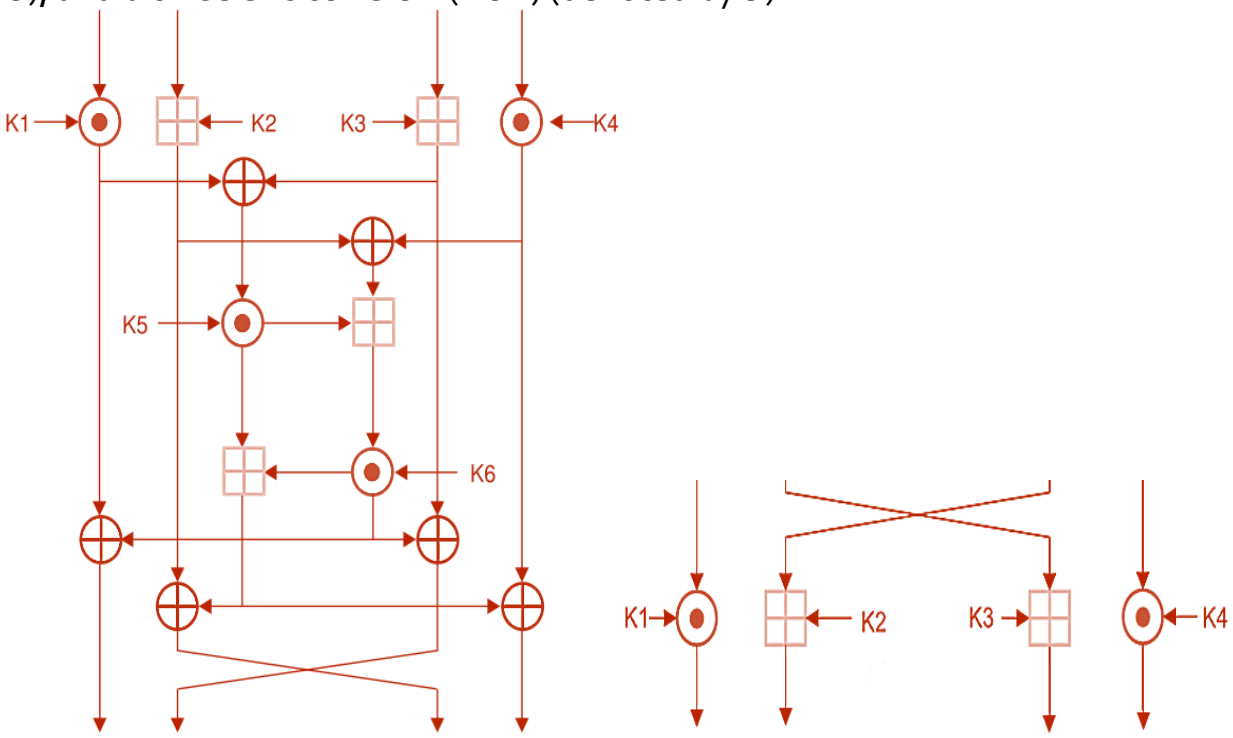


FIGURE 19: AN IDEA APPROACH

Mechanism

The ideal block size for the IDEA is of the 64-bits, and there are 4 input sub-blocks of size 16-bits each. The 128-bits key is used to generate 52 sub-keys of length 16-bits each; six of those sub-keys are used in each round and the remaining 4 keys are used in final half round. Alternatively, 6 keys are used in each round making the usage of 48 sub-key in 8 full rounds of encryption process and remaining 4 sub-keys are used in the final half round of the encryption process. The whole encryption process is as follow:

Key Schedule

The key scheduling refers to the sub-keys generation process, where sub-keys start mark of their beginning. The simple key generation approach starts producing the sub-keys starting from the first position of the 128 bits key. In each round of sub key generation, the 128-bits main keys undergoes a 25-bits right shift to generate next set of sub-keys i.e. start with bit position 0, 25, 50 and so on until all 52 sub-keys are derived. Thus, in each round, we get 8 sub-keys.

In the first round, starting from the first bit of the main key, the sub-keys K1 through K8 are generated:

0	16	32	48	64	80	96	112	0
K1	K2	K3	K4	K5	K6	K7	K8	--

In the second round, the starting point of the sub-keys generation is 25-bits right shift from the previous starting position.

24	25	41	57	73	89	105	121	09	25
--	K9	K10	K11	K12	K13	K14	K15	K16	--

In the third round, the starting point of the sub-keys generation is 25-bits right shift from the starting position in the second round.

49	50	66	82	98	114	02	18	34	50
--	K17	K18	K19	K20	K21	K22	K23	K24	--

Fourth round: start from 75 and generate sub-keys **K25** to **K32**

Fifth round: start from 100 and generate sub-keys **K33** to **K40**

Sixth round: start from 125 and generate sub-keys **K41** to **K48**

Seventh round: start from 22 (*SHIFT RIGHT 25 from 125*th position) and generate sub-keys **K49** to **K52.**

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:

$X_a = X_a \odot K_a$;

$X_b = X_b \boxplus K_b$;

$X_b = X_b \boxplus K_b$;

$X_d = X_d \odot K_d$;

- **Even Round**

Even rounds are more complicated than the odd ones. 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 third input sub-blocks are subjected to XOR operation to get single 16 bits output (say, Y_{in}) together with second and fourth input sub-blocks subjected to XOR operation to get single 16 bits output (say, Z_{in}). Y_{in} and Z_{in} are fed to the series of modular multiplication and modular addition along with K_e, and K_f to get outputs Y_{out} and Z_{out}, respectively. Y_{out} is XOR'ed with X_a and X_c, to get first two input sub-blocks for next round (say, X_{a1} and X_{b1}) and Z_{out} is XOR'ed with X_b and X_d, to get last two input sub-blocks for next round (say, X_{c1} and X_{d1}). Algebraic expressions for even round can be written as:

$Y_{in} = X_a \oplus X_c$;

$Z_{in} = X_b \oplus X_d$;

$Y_{out} = (Y_{in} \odot K_e) \boxplus Z_{out}$;

$Z_{out} = ((Y_{in} \odot K_e) \boxplus Z_{in}) \odot K_f$;

$X_{a1} = X_a \oplus Y_{out}$;

$X_{b1} = X_b \oplus Y_{out}$;

$X_{c1} = X_b \oplus Z_{out}$;

$X_{d1} = X_d \oplus Z_{out}$;

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).

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

FIGURE 20: AN AES DESCRIPTION TABLE

Structure

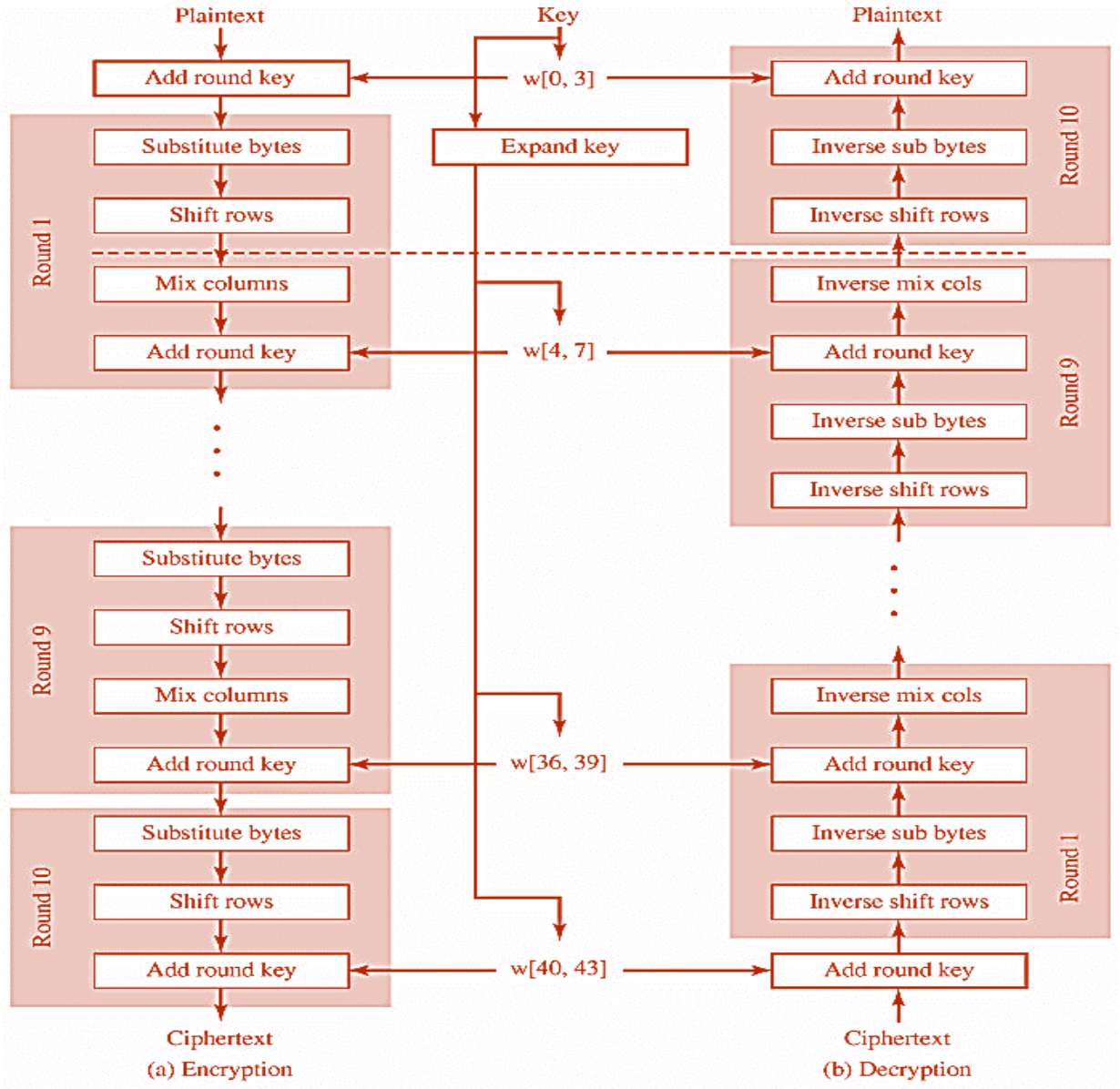


FIGURE 21: THE ENCRYPTION/DECRYPTION ROUNDS ON AES

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.

Algorithm

1. Given a plaintext x, initialize **State** to be x and perform an operation ADD-ROUNDKEY, which x-ors the **RoundKey** with **State**.
2. For each of the first Nr — 1 rounds, perform a substitution operation called SUBBYTES on **State** using an S-box; perform a permutation SHIFTROWS on **State**; perform an operation MIXCOLUMNS on **State**; and perform ADD-ROUNDKEY.
3. Perform SUBBYTES; perform SHIFTROWS; and perform ADDROUND-KEY.
4. Define the ciphertext y to be **State**.

FIGURE 22: AES ALGORITHM

AES Data Structures

The following matrixes are used in AES encryption and Decryptions:

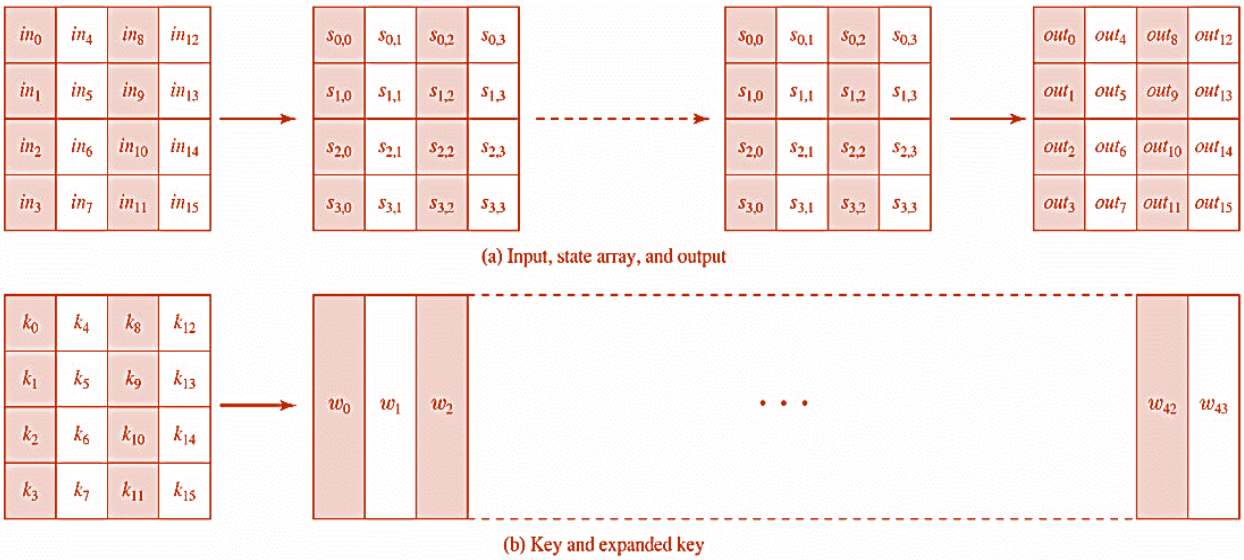


FIGURE 23: AN AES DATA STRUCTURES

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.

Substitute Bytes Transformation

The forward substitute byte transformation, called SubBytes, is a simple table lookup. 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}.

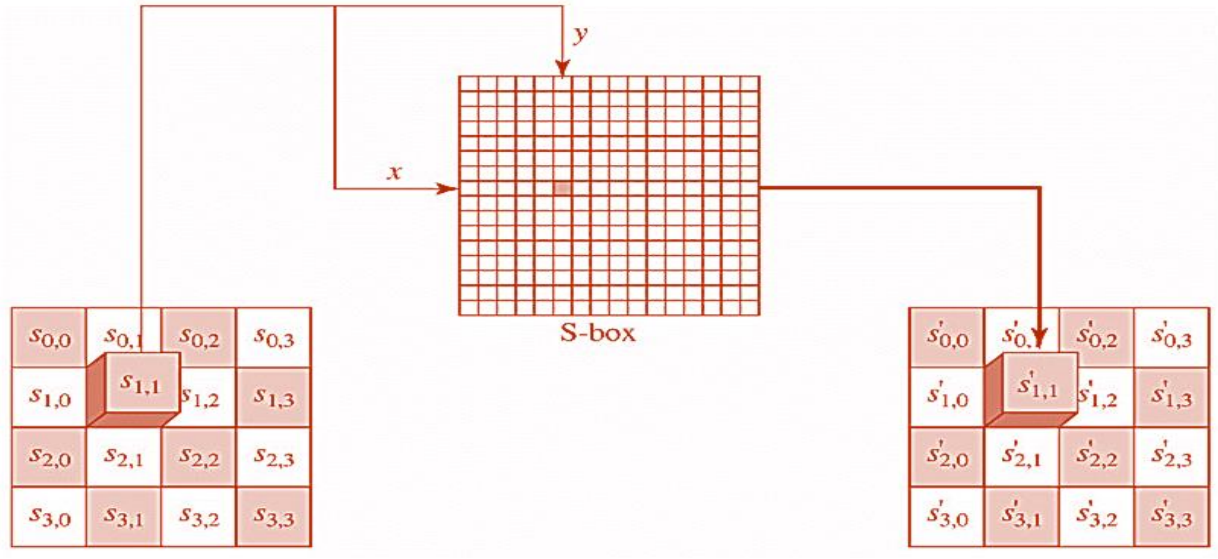


FIGURE 24: SUBSTITUTE BYTE TRANSFORMATION IN AES

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

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.

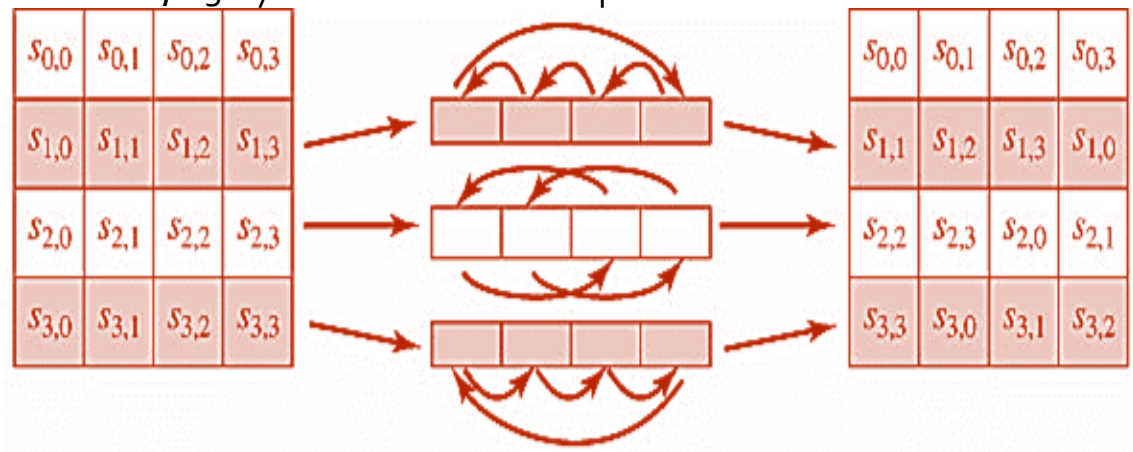


FIGURE 25: SHIFTROWS TRANSFORMATION IN AES

Mix Columns 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.

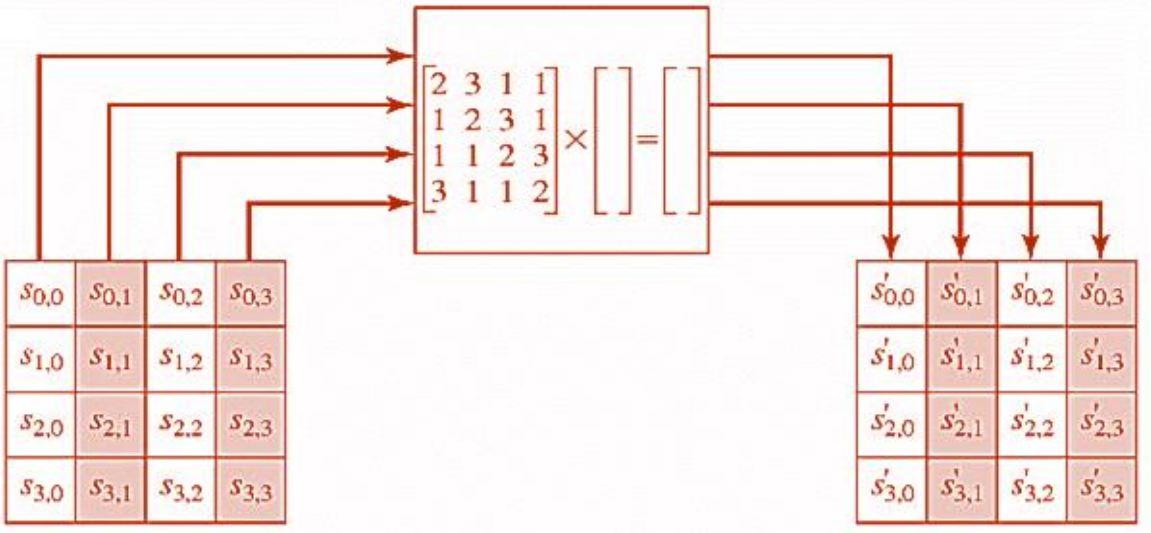


FIGURE 26: MIXCOLUMNS TRANSFORMATION IN AES

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.

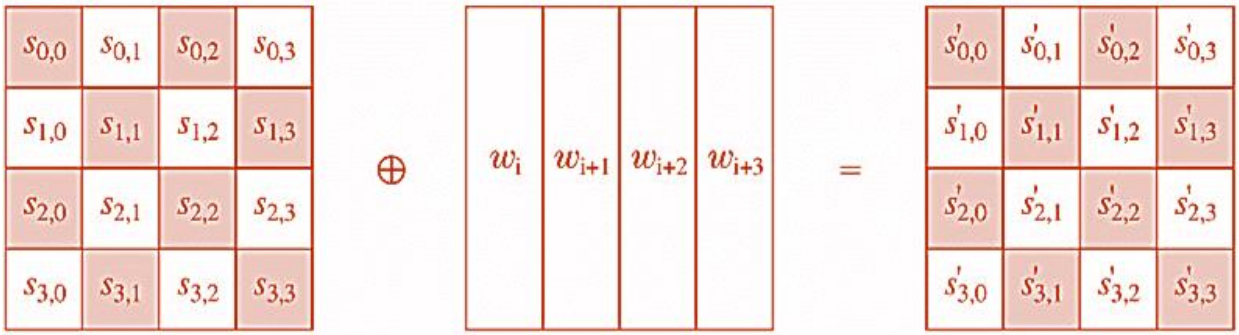


FIGURE 27: ADDROUNDKEY TRANSFORMATION IN AES

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.

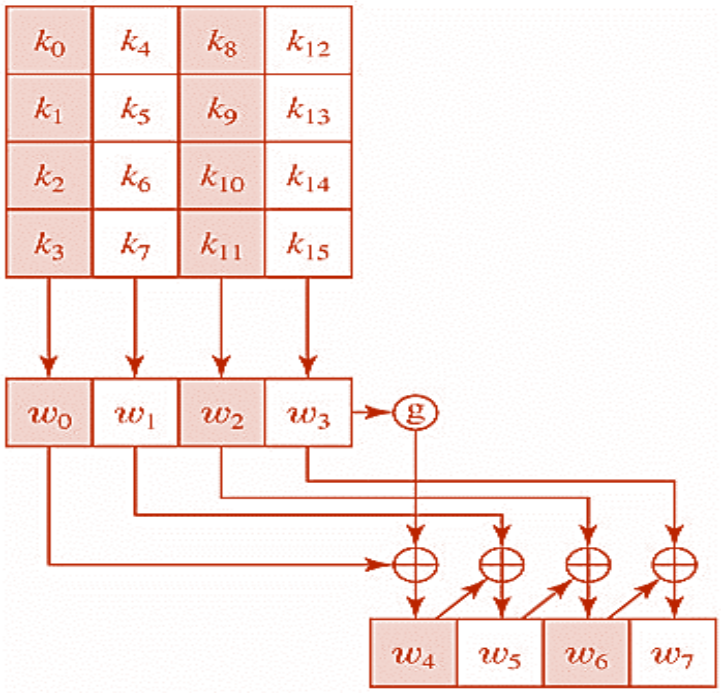


FIGURE 28: AES KEY EXPANSION ALGORITHM

Where g is a special function, consist of following sub functions:

- **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].
- **SubWord**: performs a byte substitution on each byte of its input word, using the S-box.

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.

Modes of operation

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.

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.

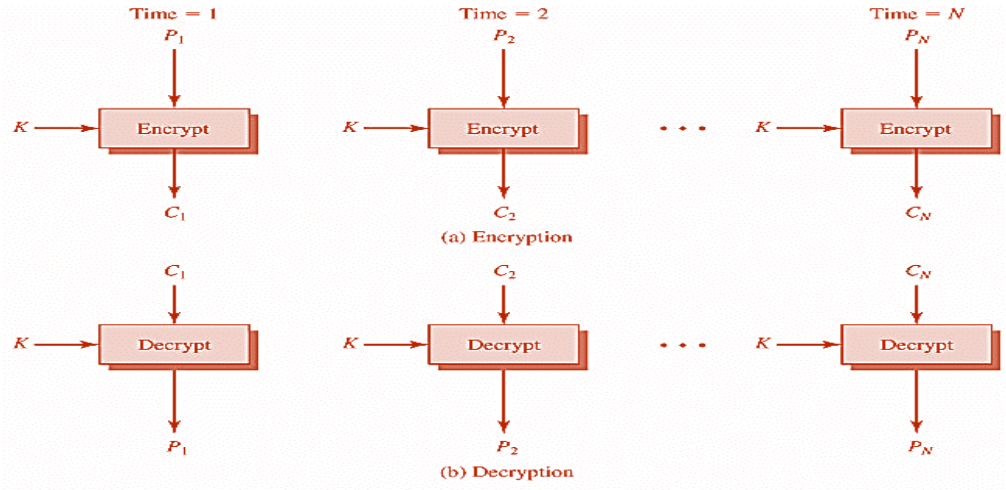


FIGURE 29: ECB MODE

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.

Cipher Block Chaining Mode

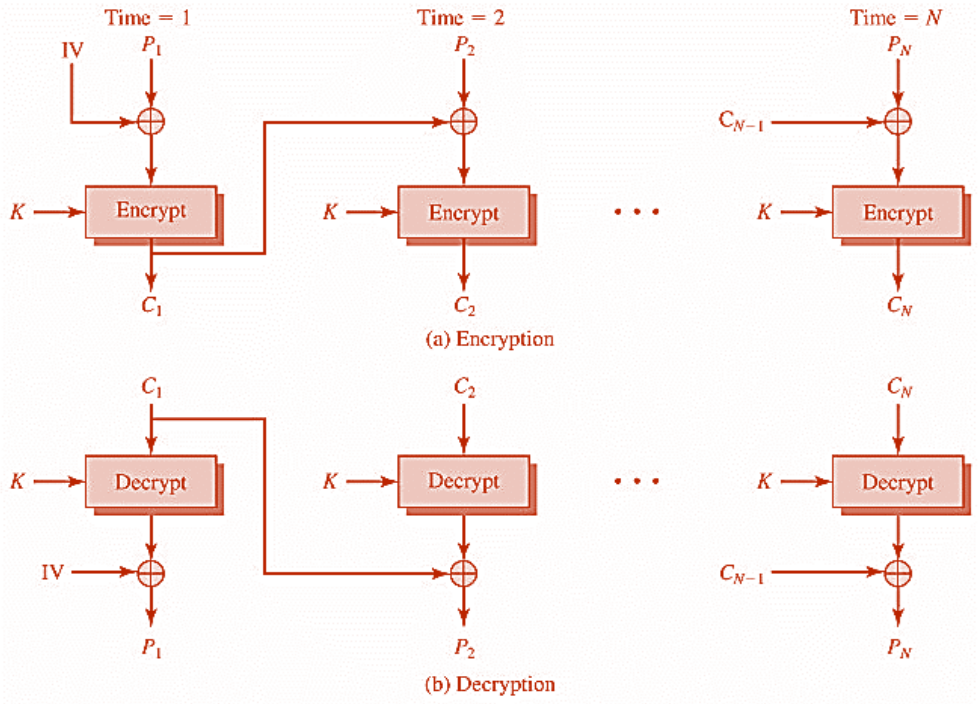


FIGURE 30: CBC MODE

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.

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.

Output Feedback Mode

The output feedback (OFB) mode is similar in structure to that of CFB. 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.

Counter 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.

Unit - 4

Public Key Cryptography

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

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. $a \equiv b \ (mod\ n)$

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'. In addition, 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 ≠o divides o.
- If b|g and b|h, then b|(mg + nh) for arbitrary integers m and n.

Congruency

Congruences have the following properties:

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

Modular Arithmetic Operations

Modular arithmetic exhibits the following properties:

- [(a mod n) + (b mod n)] mod n = (a + b) mod n
- [(a mod n) - (b mod n)] mod n = (a- b) mod n
- [(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

FIGURE 31: ARITHMETIC MODULO OF 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 i.e., 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 ϵZ_n, there exists a z such that w + z ≡0 mod n

Group

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. 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	$a \cdot b = b \cdot a$ 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 εG. 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 $aa^{-1} = (a^{-1})a = 1$.

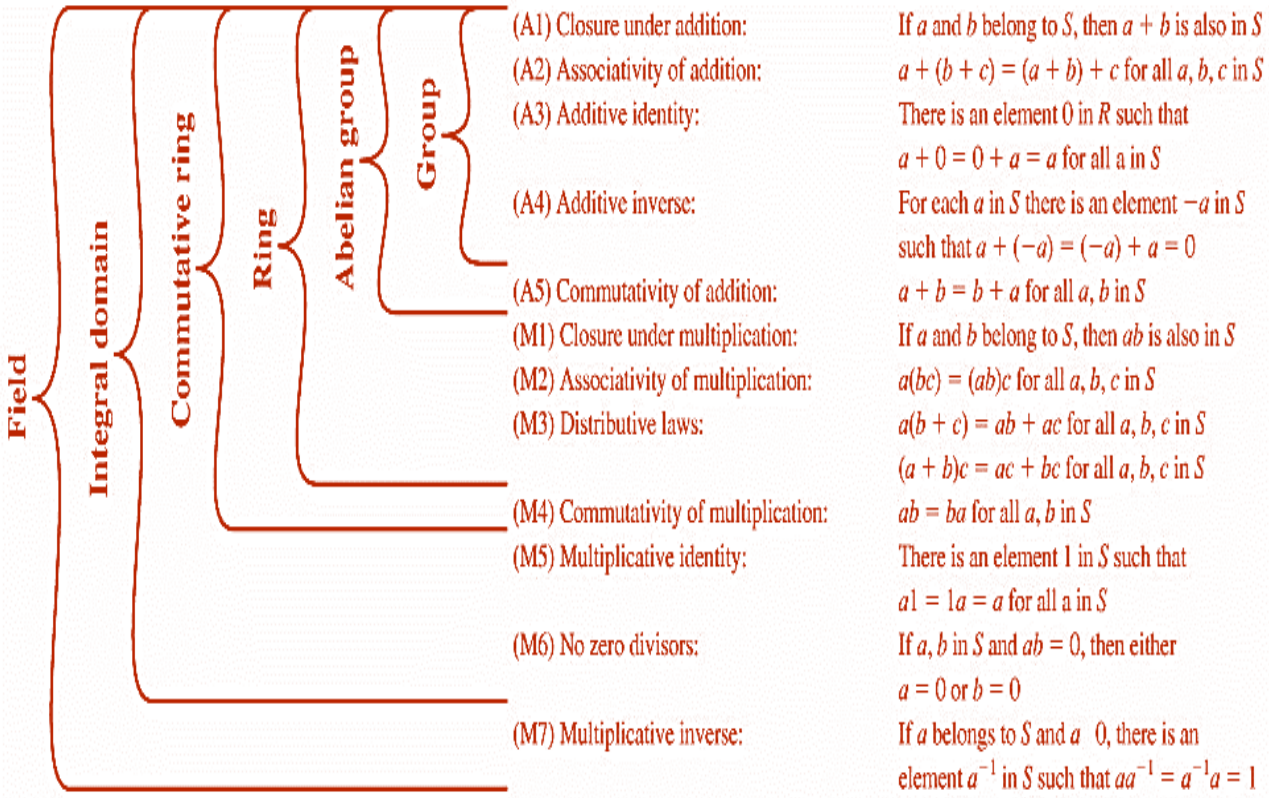


FIGURE 32: THE GROUP THEORY

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.

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

FIGURE 33: 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.

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
$7^2 = 49 \equiv 11 \pmod{19}$
$7^4 = 7^2 \times 7^2 \equiv 11 \times 11 = 121 \equiv 7 \pmod{19}$
$7^8 \equiv 49 \equiv 11 \pmod{19}$
$7^{16} = 121 = 7 \pmod{19}$
$a^{p1} = 7^{18} = 7^{16} \times 7^2 = 7 \times 11 = 1 \pmod{19}$

Alternatively, form of fermat theorem is $a^p=a \pmod p$

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 ϕ(n), defined as the number of positive integers less than n and relatively prime to n. By convention, ϕ(1) = 1.

n	ϕ(n)
1	1
3	2
13	12
14	6
15	8
19	18

If n is prime number then ϕ(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.

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

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

Two Properties of Prime Numbers

The first property

If p is prime and a is a positive integer less than p, then $a^2 \bmod p = 1$ 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 $(a \bmod p)^2 = 1$, which is true only for a mod p = 1 or a mod p = 1.

The second property:

Let p be a prime number greater than 2. We can then write $p\ 1 = 2^k q$, 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, $a^q \bmod p = 1$, 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 $a^{2^{j-1}q}$ mod p = 1 mod p = p-1, or equivalently, $a^{2^{j-1}q} \equiv 1 \pmod{p}$.

Algorithm

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.

The Discrete Logarithm

From Euler's theorem, for every 'a' and 'n' that are relatively prime,

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

Where, $\phi(n)$ Euler's totient function is the number of positive integers less than n and relatively prime to n. Now consider the more general expression:

$$a^m \equiv 1 \pmod{n}$$

If a and n are relatively prime, then there is at least one integer m that satisfies the above relation namely, $M = \phi(n)$. The least positive exponent for which the given relation holds is referred to in several ways:

- The order of ***a(mod n)***
- The exponent to which ***a*** belongs to ***(mod n)***
- The length of the period generated by ***a***

To see this last point, consider the powers of 7, modulo 19:

$$\begin{aligned}
7^1 &= && 7 \pmod{19} \\
7^2 &= 49 = 2 \times 19 + 11 &\equiv\ & 11 \pmod{19} \\
7^3 &= 343 = 18 \times 19 + 1 &\equiv\ & 1 \pmod{19} \\
7^4 &= 2401 = 126 \times 19 + 7 &\equiv\ & 7 \pmod{19} \\
7^5 &= 16807 = 884 \times 19 + 11 &\equiv\ & 11 \pmod{19}
\end{aligned}$$

There is no point in continuing because the sequence is repeating. This can be proven by noting that $7^3 \equiv 1 \pmod{19}$, and therefore, $7^{3+j} \equiv 7^3 7^j \equiv 7^j \pmod{19}$, and hence, any two powers of 7 whose exponents differ by 3 (or a multiple of 3) are congruent to each other (mod 19). In other words, the sequence is periodic, and the length of the period is the smallest positive exponent m such that $7^m \equiv 1 \pmod{19}$.

FIGURE 34: A DISCRETE LOGARITHM EXAMPLE

Primitive Root

More generally, we can say that the highest possible exponent to which a number can belong (mod n) is $\phi(n)$. If a number is of this order, it is referred to as a **primitive root** of n. The importance of this notion is that if a is a primitive root of n, then its powers

$$a, a^2, \ldots, a^{\phi(n)}$$

are distinct (mod n) and are all relatively prime to n. In particular, for a prime number p, if a is a primitive root of p, then

$$a, a^2, \ldots, a^{p-1}$$

are distinct (mod p). For the prime number 19, its primitive roots are 2, 3, 10, 13, 14, and 15.

FIGURE 35: PRIMITIVE ROOT DEFINITION

Logarithms for Modular Arithmetic

With ordinary positive real numbers, the logarithm function is the inverse of exponentiation. An analogous function exists for modular arithmetic.

Let us briefly review the properties of ordinary logarithms. The logarithm of a number is defined to be the power to which some positive base (except 1) must be raised in order to equal the number. That is, for base x and for a value y,

$$y = x^{\log_x(y)}$$

The properties of logarithms include

$$\log_x(1) = 0$$

$$\log_x(x) = 1$$

$$\log_x(yz) = log_x(y) + \log_x(z)$$

$$\log_x(y^r) = r \times \log_x(y)$$

Consider a primitive root a for some prime number p (the argument can be developed for nonprimes as well). Then we know that the powers of a from 1 through $(p - 1)$ produce each integer from 1 through $(p - 1)$ exactly once. We also know that any integer b satisfies

$$b = r \pmod{p} \quad \text{for some } r, \text{ where } 0 \leq r \leq (p - 1)$$

by the definition of modular arithmetic. It follows that for any integer b and a primitive root a of prime number p, we can find a unique exponent i such that

$$b \equiv a^i \pmod{p} \quad \text{where } 0 \leq i \leq (p - 1)$$

This exponent i is referred to as the **discrete logarithm** of the number b for the base a (mod p). We denote this value as $\text{dlog}_{a,p}(b)$.[10]

FIGURE 36: DEFINITION FOR LOGARITHMS FOR MODULAR ARITHMETIC

Public Key Cryptography

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.

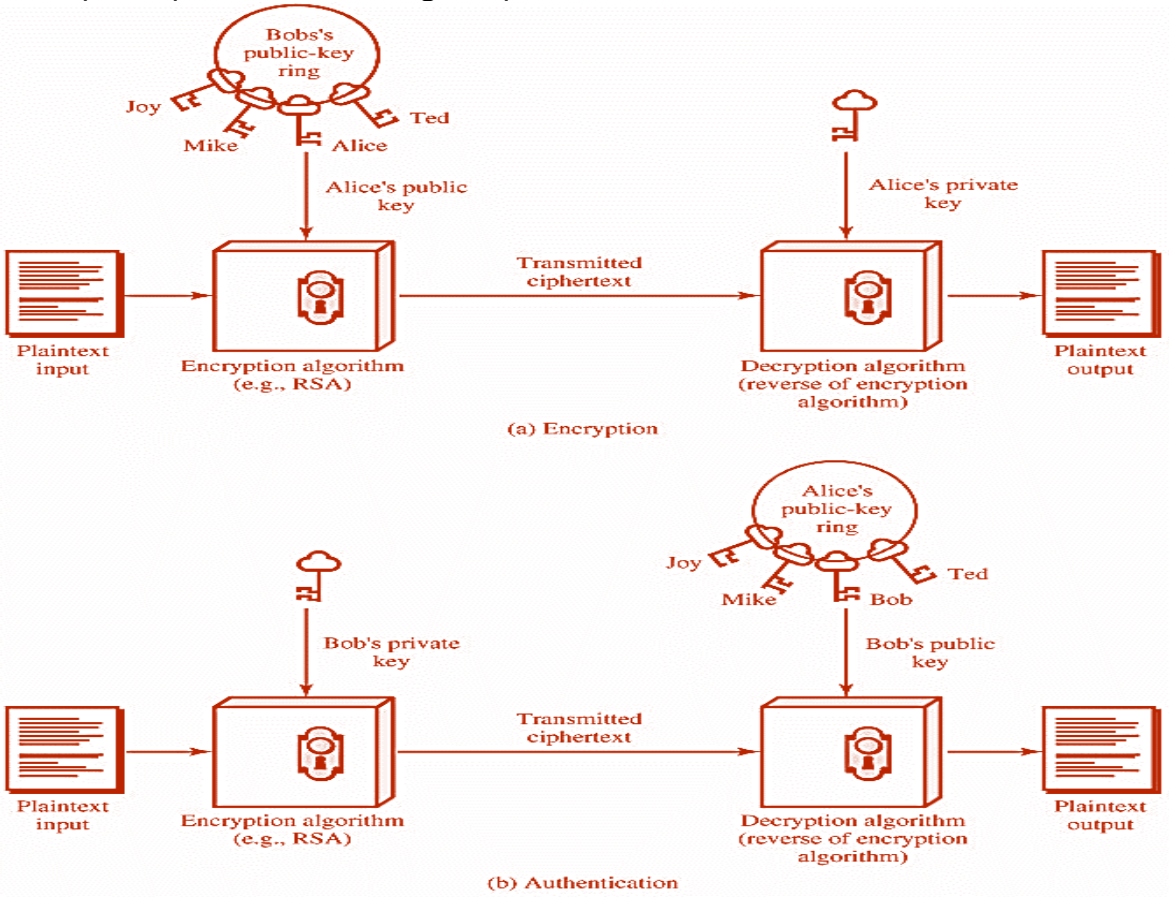


FIGURE 37: A PUBLIC KEY CRYPTOSYSTEM

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.

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: ***$X = D(PR_b, Y)$***

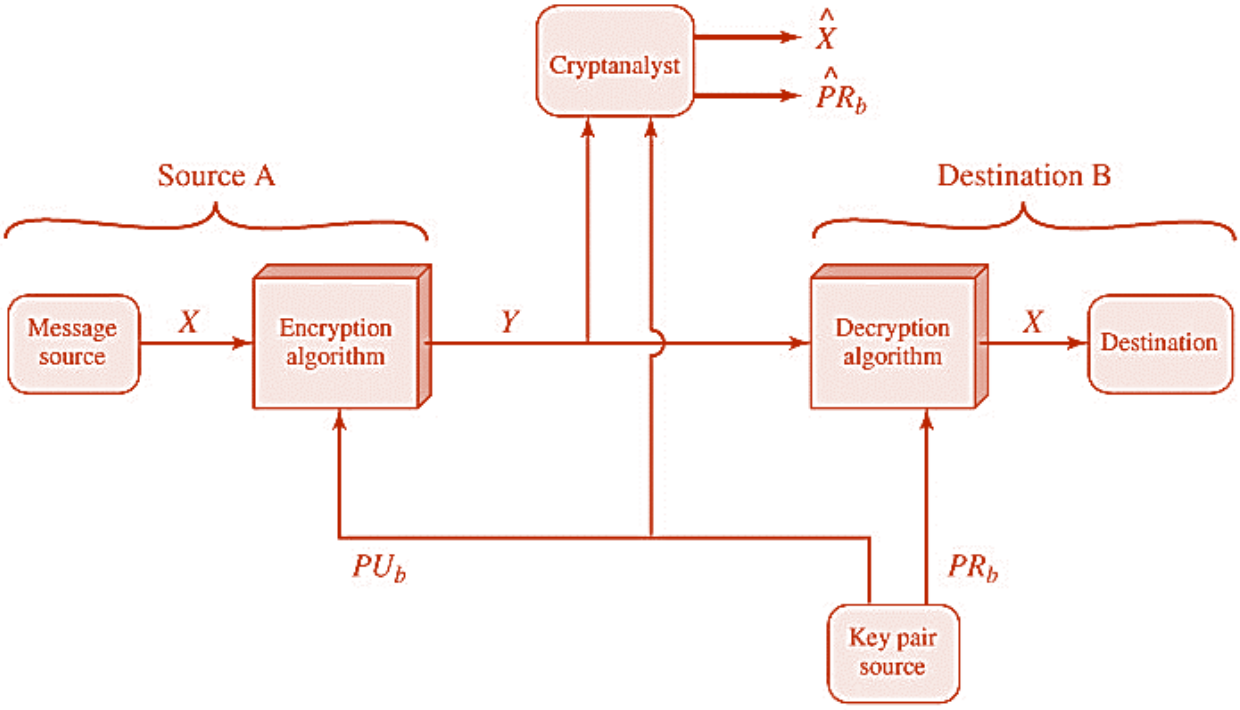


FIGURE 38: A PUBLIC KEY CRYPTOSYSTEM FOR SECRECY

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.

$Y = E(PR_a, X)$

$Y = E(PU_a, Y)$

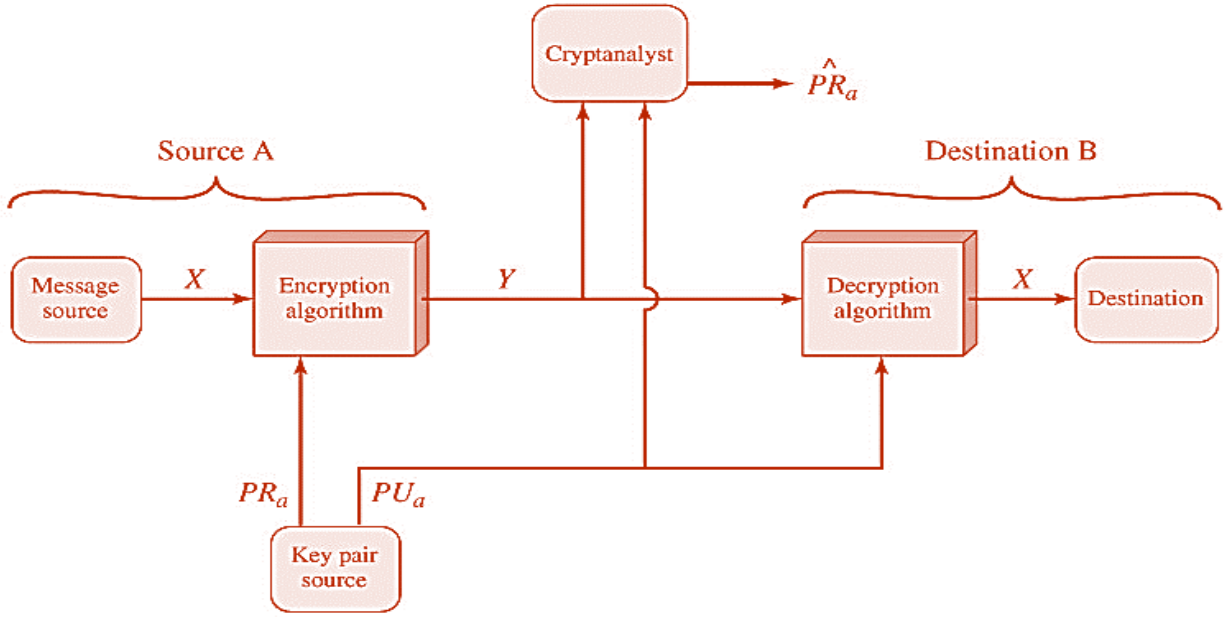


FIGURE 39: A PUBLIC KEY CRYPTOSYSTEM FOR AUTHENTICATION

Authentication and Secrecy

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

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

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

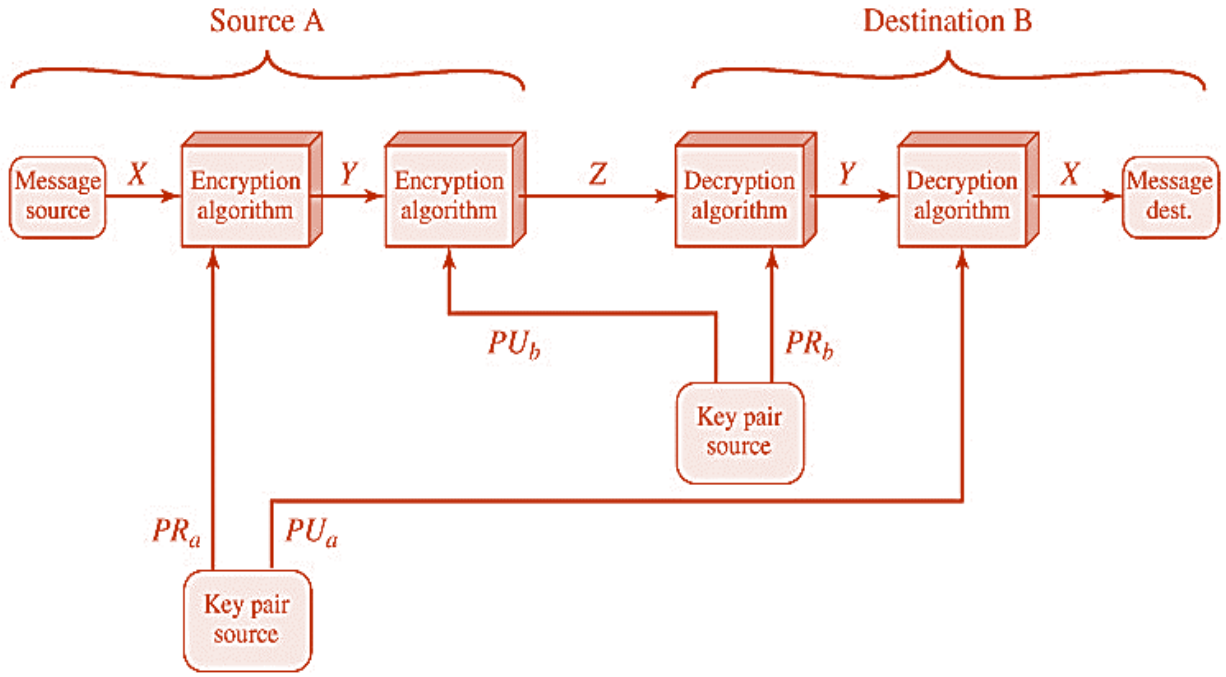


FIGURE 40: A PUBLIC KEY CRYPTOSYSTEM FOR BOTH SECRECY AND AUTHENTICATION

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. Only the intended receiver, who alone has the matching private key, can decrypt the final ciphertext. 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.
6. The two keys can be applied in either order:
   $M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$

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: $\phi(n) = (p - 1)(q - 1)$.
4. Choose an integer e such that $1 < e < \phi(n)$, and e and $\phi(n)$ share no factors other than 1 i.e. e and $\phi(n)$ are relatively prime).
5. e is released as the public key exponent.
6. Compute d to satisfy the congruence relation $ed \equiv 1 \bmod \phi(n)$; i.e. $de = 1 + k\phi(n)$ for some integer k.
7. d is kept as the private key exponent.

Considerations 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:

$$c = m^e \bmod n$$

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:

$$m = c^d \bmod n$$

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 $\phi(n) = (p - 1)(q - 1) = (61-1)(53-1) = 3120$
Choose $e > 1$ relatively prime to 3120; e = 17
Compute d such that $ed \equiv 1 \bmod \phi(n)$ e.g., by computing the modular multiplicative inverse of e modulo $\phi(n)$: 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:

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

The private key is (n = 3233, d = 2753). The decryption function is:
$m = c^d \bmod n = c^{2753} \bmod 3233$.

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.

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.

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,

$$K = (Y_A)^{X_B} \bmod p = (g^{X_A} \bmod p)^{X_B} \bmod p = (g^{X_A})^{X_B} \bmod p = g^{X_B X_A} \bmod p$$

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 $(Y_B)^{X_A} \bmod p$: $19^6 \bmod 23 = 2$.

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 humankind'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.*

Unit - 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 that 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:

1. **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.
2. **A signing algorithm S,** that, on input of a message m and a signing key SK, produces a signature σ.
3. **A signature verifying algorithm V,** which on input a message m, a verifying key PK, and a signature σ, either accepts or rejects.

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).

Hash Usage

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

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

2. **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.
3. **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

In this 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.

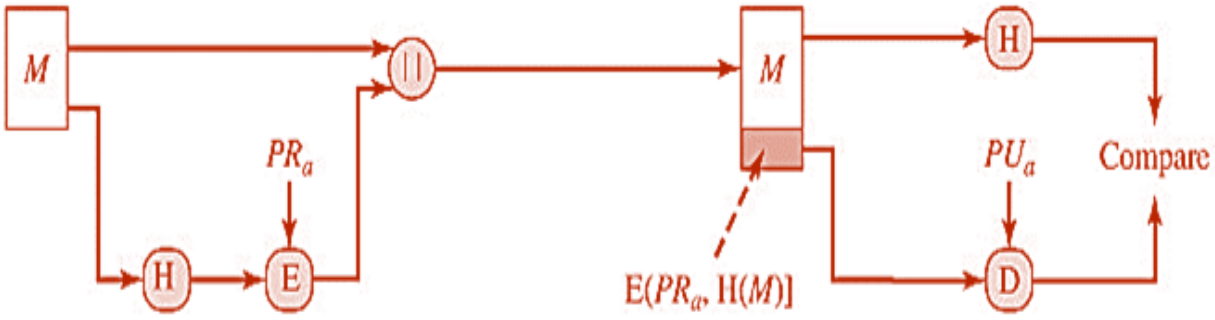


FIGURE 41: RSA APPROACH FOR DIGITAL SIGNATURE

Basic RSA signatures are computed as follows:

1. To generate RSA signature keys, one simply generates an RSA key pair. See RSA encryption fro Key generation mechanism.
2. 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

1. 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 $d_m = H(m|T_m)$.
2. 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

1. Calculate d_m using obtained m and T_m.
2. 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 (DSS)

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.

The **DSS approach** 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.

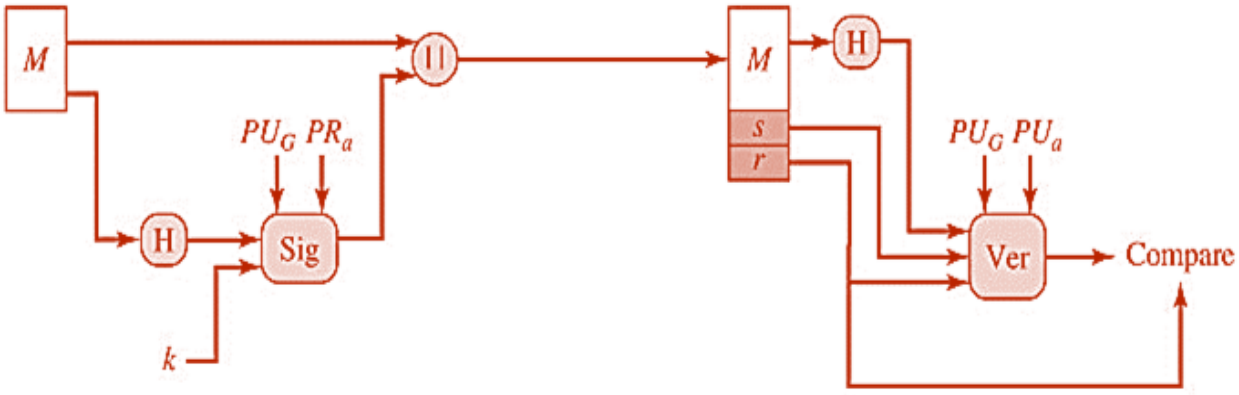


FIGURE 42: DSS APPROACH

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 mod p) mod q

s	= $[k^{-1} (H(M) + xr)] \bmod q$
	Signature = (r, s)
	Verifying
w	= $(s')^{-1} \bmod q$
u1	= $[H(M')w] \bmod q$
u2	= $(r')w \bmod q$
v	= $[(g^{u1} y^{u2}) \bmod p] \bmod q$
	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 processes in a DSS:

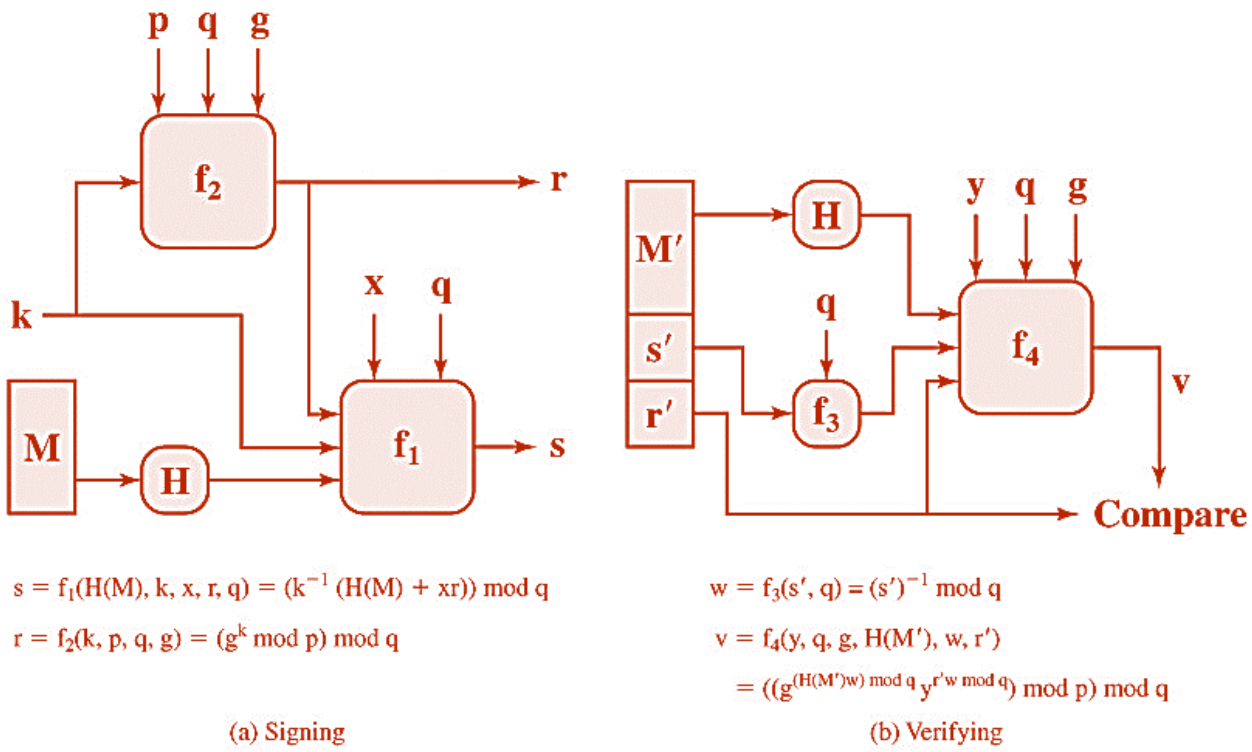


FIGURE 43: SIGNING AND VERIFYING IN DSS APPROACH

Unit – 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 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 either can be specially designed for hashing or be built from a block cipher.

There is no formal definition able to captures all of the desirable properties for a cryptographic hash function. These properties below are generally considered prerequisites:

1. **Preimage resistant:** given h it should be hard to find any m such that h = hash(m).
2. **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.

Message Digest Algorithms

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 os 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.

1. **floor(x):** the greatest integer not greater than x.
2. **~x:** bitwise complement of 32 bits quantity x.
3. **x ∧ y/ x ∨ y/ x ⊕ y:** bitwise AND/OR/XOR of 32 bits quantities x and y.
4. **x + y:** bitwise binary sum of 32 bits quantities x and y with carry discarded.
5. **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 ∧ y) ∨ (~x ∧ 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.

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

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

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

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

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

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

$d_0 = (d_0 + F(d_1, d_2, d_3) + m_4) \hookleftarrow 3;$

And so on.

Pass 2

This pass uses **majority function** [G(x, y, z) = (x ∧ y) ∨ (x ∧ z) ∨ (y ∧ 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.

$d_{(-i) \wedge 3} = (d_{(-i) \wedge 3} + G(d_{(1-i) \wedge 3}, d_{(2-i) \wedge 3}, d_{(3-i) \wedge 3}) + m_{X(i)} + 5a827999_{16}) \hookleftarrow S_2(i \wedge 3)$,

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 ↵s cycles over the values 3, 5, 9, 13.

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

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

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

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

$d_0 = (d_0 + G(d_1, d_2, d_3) + m_1 + 5a827999_{16}) \curvearrowleft 3;$

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.

$d_{(-i)\wedge 3} = (d_{(-i)\wedge 3} + H(d_{(1-i)\wedge 3}, d_{(2-i)\wedge 3}, d_{(3-i)\wedge 3}) + m_{R(i)} + 6ed9eba1_{16}) \curvearrowleft S_3(i \wedge 3),$

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 ↵s cycles over the values 3, 9, 11, 15.

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

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

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

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

$d_0 = (d_0 + H(d_1, d_2, d_3) + m_2 + 6ed9eba1_{16}) \curvearrowleft 3;$

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 result in a completely different hash, due to the overwhelming probability, also known as ***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

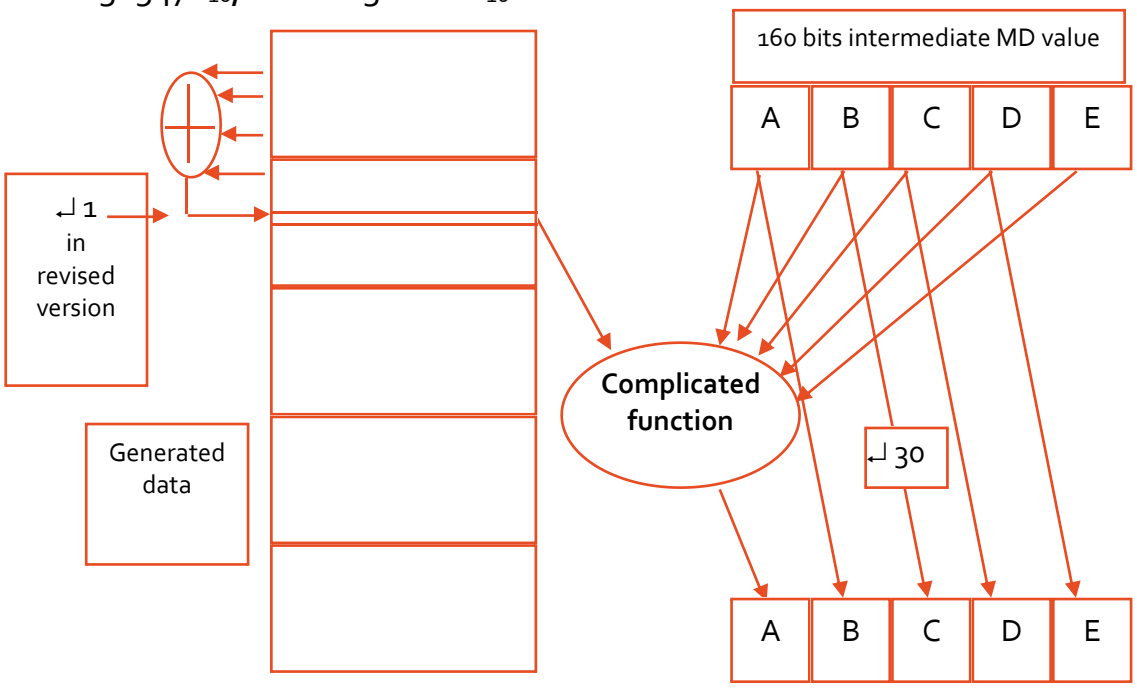
Step 1: Message Padding

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

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 ∧ C) ∨ (~B ∧ D) if $0 \le t \le 19$.
f(t, B, C, D) = B ⊕ C ⊕ D if $20 \le t \le 39$.
f(t, B, C, D) = (B ∧ C) ∨ (B ∧ D) ∨ (C ∧ D) if $40 \le t \le 59$.
f(t, B, C, D) = B ⊕ C ⊕ D if $60 \le t \le 79$.

Unit - 7

Authentication and Public Key Infrastructure

Revised and Compiled by Sandip Paudel

Overview of the Authentication System

Authentication System

Authentication system, alternatively known as the identification system allows someone to be identified in the network or the system uniquely. Generally, the authentication of the user means one or all of the following:

- What you are
- What you have and
- What you know

The physical and the behavioral attributes belong to "what you are", "what you have" asks for the documents or credentials and "what you know" encompasses passwords and personal information.

Physical Attributes

The physical attributes include the height, weight, racial origin, eye color, hair color, fingerprints, etc. More often, these physical attributes unique to each individual are more useful for the authentication of the users. These attributes allows the system or network to identify each individual uniquely like in the family and friends. The automated information systems like biometrics makes the use of the physical attributes like fingerprints, retina-structure, etc.

Credentials

In the diplomatic way, the credential may be defined as the letter of the identification. Credential may mean to the driver's license for the driver and the passport for the traveler. In other words, the credential means the trusted documents that together includes some physically attributed information of the credential bearer.

Password

The password may be defined as the knowledge that an individual requires for his identification in the system to ensure the person being identified. The difficulty of using the knowledge for the identification is that the knowledge may not be secret and in addition, it is disclosed as the part of the identification process, which allows for the possible future impersonation.

The authentication system completely depends upon the attack models and the adversarial goals. There are various attack model in the modern cryptography and the goal of the adversary (the opponent or the contender) is different in different

session, however the adversary tries to have the initiator of the scheme "accept" in some session in which the adversary is active. Mutual identification is the most often used cryptographic handshaking used in the network sessions or schemes. In mutual identification, both the parties involved in the scheme agree to identify each other and accept, using the mutual keys.

Components of the Authentication System

The major components of the authentication system are key, key management and key distribution or key exchange.

Key

The key provides the true identification of the party involved in the session scheme. The key may include the password or the PIN (Personal Identification Number).

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.

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 $\{m\}\ k_{session} \parallel \{k_{session}\}\ k_B$.

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

Key Distribution or 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.

Cryptographic Security Handshake

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.

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. Random pulses, Electromagnetic phenomena, Characteristics of computing environment like disk latency, Ambient background noise, etc. are the best sources of the random numbers.

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.

Issue in random number generation: 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.

Key Exchange

Suppose Alice (A) and Bob (B) wish to communicate. If they share a common key, they can use a classical cryptosystem. Nevertheless, 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 cannot 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.

Simple Protocol

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

$A \rightarrow C$: {request for session key to B} k_A.

$C \rightarrow A$: $\{ k_{session} \}k_A \,||\, \{ k_{session} \}k_B$.

$A \rightarrow B$: $\{ k_{session} \}k_B$.

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 is talking to A, but he is not.

Session key reuse: Eve replays message from A to B, so B re-uses session key. Therefore, the protocols must provide authentication and defense against replay attack.

The following algorithm provides solution to the above problem.

Needham-Schroeder Protocol

$A \rightarrow C : \{ A \,||\, B \,||\, rand_1 \}$

Alice sends a message to the server identifying herself and Bob, telling the server she wants to communicate with Bob.

$C \rightarrow A : \{ A \,||\, B \,||\, rand_1 \,||\, k_{session} \,||\{A \,||\, k_{session}\}\, k_B \}\, k_A$

The server (Cathy) generates $k_{session}$ and sends back to Alice a copy encrypted under k_B for Alice to forward to Bob and 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 whom she is to share this key.

$A \rightarrow B : \{ A \,||\, k_{session} \}\, k_B$

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

$B \rightarrow A : \{ rand_2 \}\, k_{session}$

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

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

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.

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:

$Eve \rightarrow B : \{ A \,||\, k_{session} \}\, k_B$

$B \rightarrow A : \{ rand_2 \}\, k_{session}$ [intercepted by Eve]

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

Solution

In protocol (Needham-Schroeder), Eve impersonates A. Therefore, 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.

$A \rightarrow C : \{ A \,||\, B \,||\, rand_1 \}$

$C \rightarrow A : \{ A \,||\, B \,||\, rand_1 \,||\, k_{session} \,||\, \{A \,||\, T \,||\, k_{session}\} k_B \} k_A$

$A \rightarrow B : \{A \,||\, T \,||\, k_{session}\} k_B$

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

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

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.

$A \rightarrow B : num \,||\, A \,||\, B \,||\, \{ rand_1 \,||\, num \,||\, A \,||\, B \} k_A$

$B \rightarrow C : num \,||\, A \,||\, B \,||\, \{ rand_1 \,||\, num \,||\, A \,||\, B \} k_A \,||\, \{rand_2 \,||\, num \,||\, A \,||\, B \} k_B$

$C \rightarrow B : num \,||\, \{ rand_1 \,||\, k_{session} \} k_A \,||\, \{ rand_2 \,||\, k_{session} \} k_B$

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

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

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 $A \rightarrow B$: $A \parallel \{\{k_{session}\}d_A\}e_B$.

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 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.

Authentication standards

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 based on "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 client to request the service from the server must use the ticket.

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 $\mathbf{T_{u,s} = s \,||\, \{ u \,||\, u\text{'s address} \,||\, \text{valid time} \,||\, k_{u,s}\} k_s}$, 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.

$A_{u,s}$ = { 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.

PKI Trust Models

Certificates

"Certificates are the building block of PKIs, and they ultimately enable secure and scalable PKIs to be built from them…" (Stinson, 2005).

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.

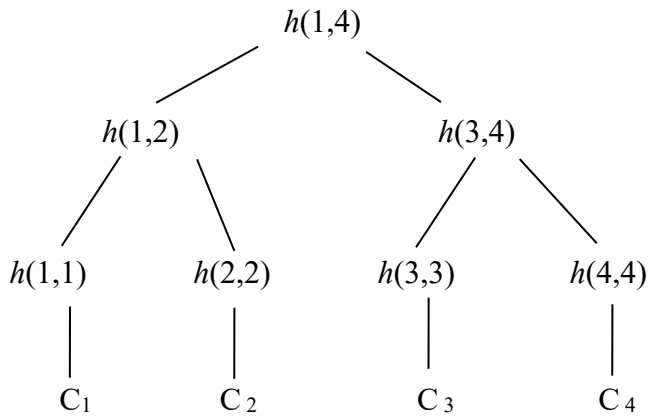
Merkle's Tree Scheme

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 $Y_1, \ldots, Y_n$ 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

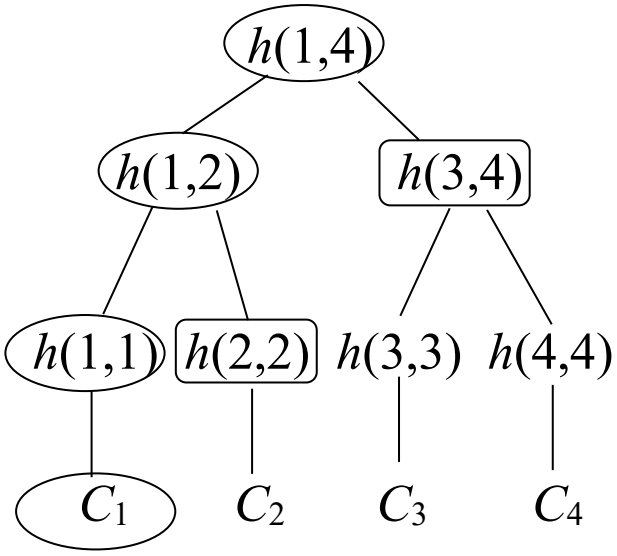
- *h* is hash function
- *Ci* is certificate *i*

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, cannot re-compute hash at the root of tree.
- Not practical if there are too many certificates and users and users and certificates distributed over widely separated systems.

Certification Authority (CA)

CA is an 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.

Example

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>>

For purposes of illustration, we now describe the format of X.509 v3 certificates, which are a popular type of certificate. X.509 certificates contain the following fields:

1. version number
2. serial number
3. signature algorithm ID
4. issuer name
5. validity period
6. subject name (i.e., the certificate owner)
7. the certificate owner's public key
8. optional fields
9. the *CA*'s signature on all the previous fields

FIGURE 44: A X.509 CERTIFICATE SAMPLE (STINSON, 2005)

Trust Models

Often, a certificate will not be signed directly by a trusted *CA*. Instead, it is necessary to follow a *certificate path* from a trusted *CA* to a given certificate. Each certificate in the path should be signed by the owner of the previous certificate in the path. By validating all the certificates in the path, the user can be confident that the last certificate in the path is valid.

A *trust model* specifies rules which determine how a certificate path should be constructed. Here are some examples of trust models that we will discuss:

1. *strict hierarchy*
2. *networked PKIs*
3. *web browser model*
4. *user-centric model* (also known as a *web of trust*).

FIGURE 45: PKI TRUST MODELS (STINSON, 2005)

Strict Hierarchy

In the strict hierarchy, the root CA has a self-signed, self-issued certificate; the root CA is called the trust anchor. The root CA may issue certificates for lower level CAs, and any CAs can issue certificates for the end users.

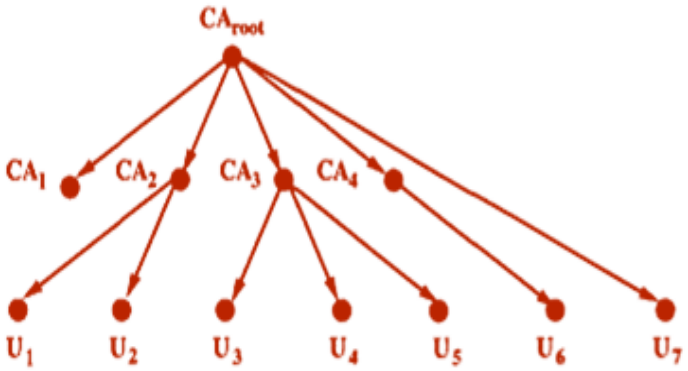


FIGURE 46: A SRICT HIERARCHY PKI

Networked PKIs

A strict hierarchy may work well within an organization, sometimes, however, it may be desirable to connect root CAs of two or more PKI domains, a process sometimes called PKI networking.

This model is achieved in one of the following three ways:

1. Cross-certification
2. Mesh configuration
3. Hub-and-spoke configuration

Web-browser Model

The web-browser model is the internet based, rather than the network based. Most of the web-browsers come preconfigured with the set of independent root CAs, all of which are treated by the end-users of the web-browser as trusted anchor.

User-centric Model

"Pretty-Good-Privacy" (PGP) is the most popular user-centric PKI trusted model. In PGP, every user is his or her own CA. A PGP certificates contain an email address (ID), a public key (P_k), and one or more signatures on this (ID, P_k) pair.

PKIX

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

1. ***End entity:*** *A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities typically consume and/or support PKI-related services.*
2. ***Certification authority (CA):*** *The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more Registration Authorities.*
3. ***Registration authority (RA):*** *An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the end-entity registration process but can assist in a number of other areas as well.*

4. ***CRL issuer:*** *An optional component that a CA can delegate to publish CRLs.*
5. ***Repository:*** *A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities."* (Stallings, 2011)

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.

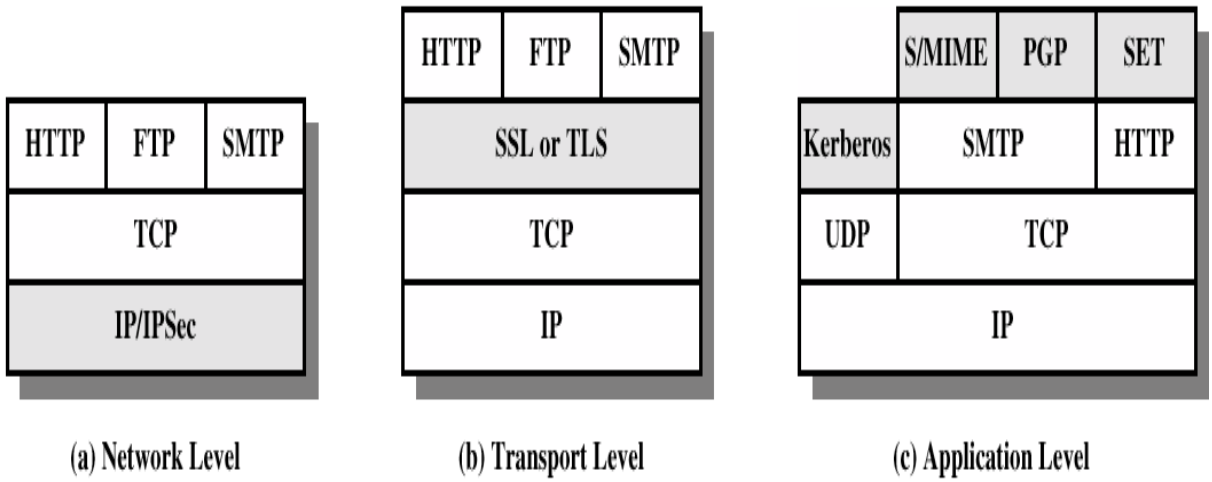
Unit - 8

Network Security

Networks and Cryptography

Security at Different Layers

The following figure shows the security at different layers:



Link and End-to-End Protocols

Let hosts C_o... C_n be such that C_i and C_{i+1} are directly connected, for o ≤i < n. A communications protocol that has C_o 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.

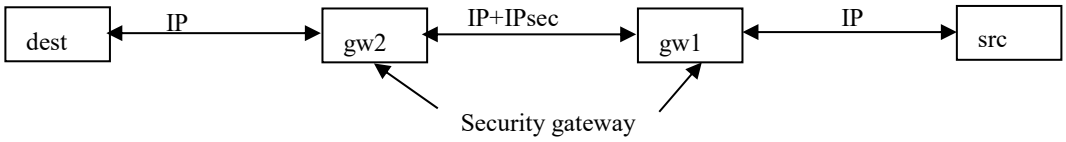
End-to-end encryption

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.

IP Security

IP Security, sometimes also called 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 to 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.



- IP security (IPsec) is a capability that can be added to either current version of the Internet Protocol (IPv4 or IPv6) by means of additional headers.
- IPsec encompasses three functional areas: authentication, confidentiality, and key management.
- Authentication makes use of the HMAC message authentication code. Authentication can be applied to the entire original IP packet (tunnel mode) or to all of the packet except for the IP header (transport mode).
- Confidentiality is provided by an encryption format known as encapsulating security payload. Both tunnel and transport modes can be accommodated.
- IKE defines a number of techniques for key management.

FIGURE 47: KEY POINTS: IPSEC

Web Security

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.

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

SL, 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.

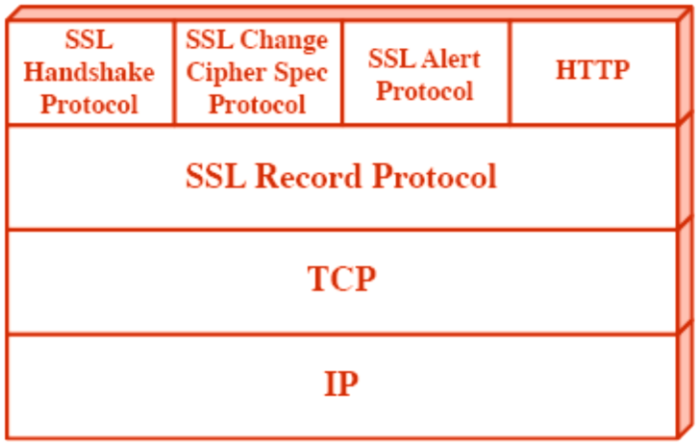


FIGURE 48: SSL STRUCTURE

SSL works in terms of **connections** and **sessions** between clients and servers. An SSL **session** is an association between two 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.

Transport Layer Security (TLS)

TLS is an IETF standardization initiative whose aim is to produce the Internet standard version of the SSL.TLS is defined as the Proposed Internet Standard in RFC 2546, which is very similar to SSLv3.

The TLS Record Format is the same as that of the SSL Record Format, and the fields in the header have the same meanings. The one difference is in version values.

There are two differences between the SSLv3 and TLS MAC schemes:

- The actual algorithm
- The scope of the MAC calculation.

TLS makes use of the HMAC algorithm defined in RFC 2104.

$$\text{HMAC}_K(M) = \text{H}[(K^+ \oplus \text{opad}) \| \text{H}[(K^+ \oplus \text{ipad}) \| M]]$$

where

H = embedded hash function (for TLS, either MD5 or SHA-1)
M = message input to HMAC
K^+ = secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1, block length = 512 bits)
ipad = 00110110 (36 in hexadecimal) repeated 64 times (512 bits)
opad = 01011100 (5C in hexadecimal) repeated 64 times (512 bits)

FIGURE 49: HMAC CALCULATION IN TLS

SNMP

"SNMP stands for Simple Network Management Protocol. It is a standard way of monitoring hardware and software from nearly any manufacturer, from Juniper, to Cisco, to Microsoft, Unix, and everything in between. SNMP requires only a couple of basic components to work, which include a management station, and an agent.

First, a management station is required. The management station is simply software that collects information from your network. Most management stations will poll your network for information regularly. Management stations range from the very simple to highly complex.

Simple software is usually very feature-limited, but can be freely available and easy to configure. For example, the free SolarWinds Network Device Manager that was recently released.

On the other hand, complex systems can manage your entire network. They will also do things like generate reports, perform inventory, and send email or SMS text alerts when systems fail. Networkmanagementsoftware.com recently

reviewed SolarWinds Network Performance Monitor, – an excellent management solution.

Second, the hardware or software that you want to monitor must have an agent running. The agent collects information, and then sends it to the monitoring station when polled. Agents can also send notification to the management station without being polled, for example if an error is detected.

Agents are usually built-in to your network hardware and software – they simply need to be enabled and configured.

SNMP is very simple, yet powerful. It has the ability to help you manage your network by:

- *Provide Read/Write abilities – for example, you could use it to reset passwords remotely, or re-configure IP addresses.*
- *Collect information on how much bandwidth is being used.*
- *Collect error reports into a log, useful for troubleshooting and identifying trends.*
- *Email an alert when your server is low on disk space.*
- *Monitor your servers' CPU and Memory use, alert when thresholds are exceeded.*
- *Page or send an SMS text-message when a device fails.*
- *Can perform active polling, i.e. Monitoring station asks devices for status every few minutes.*
- *Passive SNMP – devices can send alerts to a monitoring station on error conditions."* (https://www.networkmanagementsoftware.com/snmp-tutorial/)

Different versions of SNMPs

Several versions of SNMP are supported, v1, v2c, and v3. Nearly all monitoring stations support all three versions.

1. Version 1 is the simplest and most basic of the versions, and there may be times where it is required to support older hardware.
2. Version 2c adds several enhancements to the protocol, such as support for "Informs". Because of this v2c has become most widely used.
3. SNMP v3 adds a security features that overcome the weaknesses in v1 and v2c, and it should generally be used if possible – especially if you plan to transmit

information across unsecured links. However, the extra security makes it much more complex to configure.

Nevertheless, a major weakness of v1 and v2c is security. Community strings – the equivalent of passwords – are transmitted in clear text and there is no support for authentication. This creates risk that your community strings could become compromised. This is not good, especially considering the power SNMP has to change device configuration.

Ports

SNMP uses UDP as the transport protocol. If management traffic will traverse firewalls, make sure that the following default ports are open:

- UDP 161: Used when management stations communicate with agents, e.g. Polling
- UDP 162: Used when agents send unsolicited Traps to the management station

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.

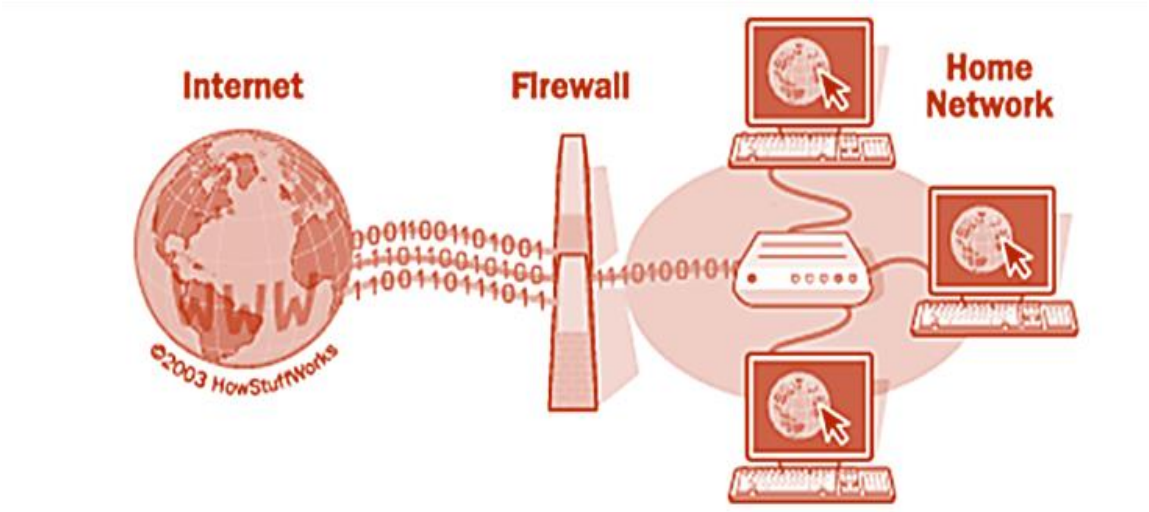


FIGURE 50: A FIREWALL

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 cannot 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.

Application Layer Security

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. However, 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.

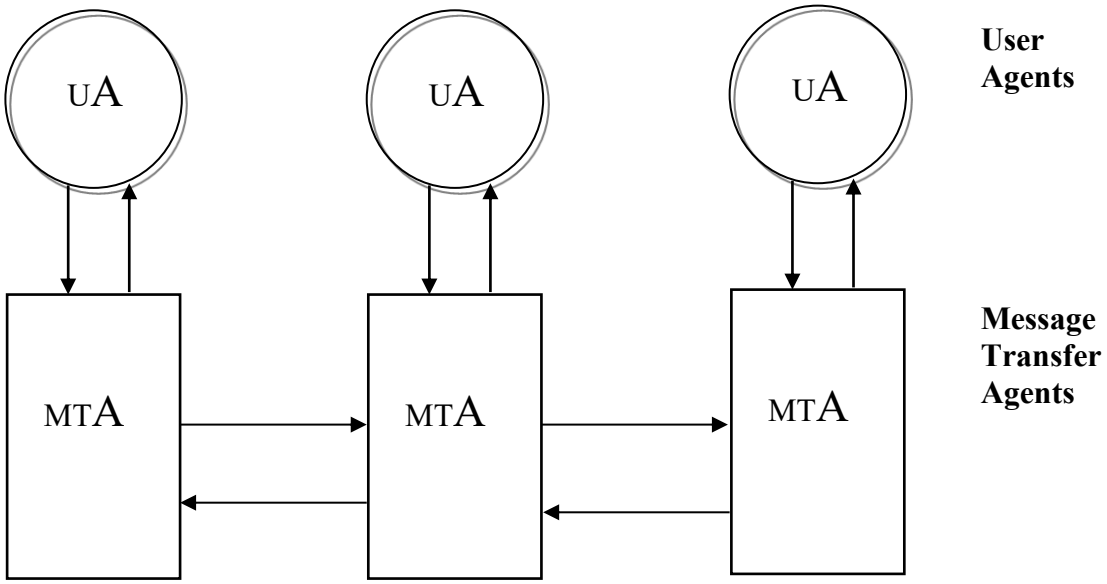
PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth.

1. It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.
2. It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.
3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
4. It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of "the establishment," this makes PGP attractive.
5. PGP is now on an Internet standards track (RFC 3156; *MIME Security with OpenPGP*). Nevertheless, PGP still has an aura of an antiestablishment endeavor.

FIGURE 51: KEY POINTS: PGP

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.

Secure Electronic Transactions (SET)

"The Secure Electronic Transactions (SET) is a protocol which has the ability to stand as an important factor in the security of e-commerce transactions. Visa and MasterCard, with IBM and other computer vendors, manufactured it. In SET, five parties are included as cardholder (customer), merchant (web server), acquirer (merchant's bank), payment gateway and issuer (cardholder's bank).The main goal of protocol is to ensure the integrity of payment information, authentication of cardholder as well as merchant and confidentiality of information. For authentication of cardholders, digital certificates are issued to cardholders, merchants and acquirers by their sponsoring organizations. It also use dual signature, which does not allow merchant to access the customer's credit card information, and hides the order information to banks, to protect privacy. In SET, message confidentiality and security is provided by cryptography and digital certificate authentication mechanism.

In SET, a 56-bit key is randomly generated and message data is encrypted by that key which is further encrypted using the message recipient's public key (RSA). Hence, —digital envelope of the message is generated. To derive the digital signature, SET uses a distinct public/private key.

Each SET participant possess two asymmetric key pairs:

- *First — key exchange pair, used for key encryption and decryption, and*
- *Second — signature pair for the creation and verification of digital signatures (160-bit message digests).*

The algorithm used for digital signature creation and verification very strictly follow the property that no two messages will have same message digest and also ensures that any one bit modification will lead to change in half of message digest bits." (Kaliya & Hussain, 2016)

"SET is an open encryption and security specification designed to protect credit card transactions on the Internet. The current version, SETv1, emerged from a call for security standards by MasterCard and Visa in February 1996. Wide ranges of companies were involved in developing the initial specification, including IBM, Microsoft, Netscape, RSA, Terisa, and Verisign. Beginning in 1996, there have been numerous tests of the concept, and by 1998, the first wave of SET-compliant products was available. SET is not itself a payment system. Rather it is a set of security formats and protocols like (Secure Sockets Layer (SSL), Microsoft's Secure Transaction Technology (STT), and Secure Hypertext Transfer Protocol (S-HTTP). SET uses S-MIME but not all aspects of a public key infrastructure (PKI) that enables users to employ the existing credit card payment infrastructure on an open network, such as the Internet, in a secure fashion. In essence, SET provides three services:

- *Provides a secure communications amongst parties.*
- *Provides trust by the use of X.509v3 digital certificates.*
- *Ensures privacy the restricted info to those who need it."* (Ghazi, 2017)

References

Biryukov, A. (2005). Adaptive Chosen Plaintext Attack. *van Tilborg H.C.A. (eds) Encyclopedia of Cryptography and Security*.

Biryukov, A. (2011). Adaptive Chosen Ciphertext Attack. *van Tilborg H.C.A., Jajodia S. (eds) Encyclopedia of Cryptography and Security*.

Ghazi, O. (2017). *Secure Electronic Transactions-SET.*

Kaliya, N., & Hussain, M. (2016). Simple Secure Electronic Transaction (SSET) Protocol.

Shoch, J. F., & Hupp, J. A. (1982). The "Worm" Programs Early Experience. *Communications of the ACM*, 172-180.

Stallings, W. (2011). *Cryptography and Network Security: Principles and Practices.* New york: Pearson Education, Inc. .

Stinson, D. R. (2005). *Cryptography: Theory and Practice.* Boca Raton: CRC Press.