BCA THIRD SEMESTER

BCA312 – UNIT IV

ERROR DETECTION AND CORRECTION CODES

We know that the bits 0 and 1 corresponding to two different range of analog voltages. So, during transmission of binary data from one system to the other, the noise may also be added. Due to this, there may be errors in the received data at other system.

That means a bit 0 may change to 1 or a bit 1 may change to 0. We can't avoid the interference of noise. But we can get back the original data first by detecting whether any error is present and then correcting those errors.

For this purpose, we can use the following codes.

- Error detection codes
- Error correction codes

ERROR DETECTION CODES

Error detection codes − are used to detect the errors present in the received data bitstream. These codes contain some bits, which are appended to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data bitstream. **Example** − Parity code, Hamming code.

ERROR CORRECTION CODES

Error correction codes − are used to correct the errors present in the received data bitstream so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes. **Example** − Hamming code.

Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission.

EVEN PARITY CODES

It is easy to include one parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

Even Parity Code

The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one. So that, even number of ones present in **even parity code**. Even parity code contains the data bits and even parity bit.

The following table shows the **even parity codes** corresponding to each 3-bit binary code. Here, the even parity bit is included to the right of LSB of binary code.

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.

- If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.
- If the other system receives other than even parity codes, then there will be errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

ODD PARITY CODES

The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in **odd parity code**. Odd parity code contains the data bits and odd parity bit.

The following table shows the **odd parity codes** corresponding to each 3-bit binary code. Here, the odd parity bit is included to the right of LSB of binary code.

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes are 1 & 3.

- If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.
- If the other system receives other than odd parity codes, then there is an error in the received data.
- In this case, we can't predict the original binary code because we don't know the bit position of error.

Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

ERROR CORRECTION CODES – HAMMING CODES

Hamming code is useful for both detection and correction of error present in the received data. This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2.

The **minimum value of 'k'** for which the following relation is correct valid is nothing but the required number of parity bits.

$2^k \geq n+k+1$

Where,

'n' is the number of bits in the binary code information

'k' is the number of parity bits

Therefore, the number of bits in the Hamming code is equal to n + k.

Let the **Hamming code** is $b_{n+k}\ b_{n+k-1}.....b_3b_2b_1$ & parity bits $p_k\ p_{k-1}.....p_3p_2p_1$. We can place the 'k' parity bits in powers of 2 positions only. In remaining bit positions, we can place the 'n' bits of binary code.

Based on requirement, we can use either even parity or odd parity while forming a Hamming code. But, the same parity technique should be used in order to find whether any error present in the received data.

Follow this procedure for finding **parity bits**-

- Find the value of $\mathbf{p_1}$, based on the number of ones present in bit positions b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.
- Find the value of $\mathbf{p_2}$, based on the number of ones present in bit positions b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.
- Find the value of $\mathbf{p_3}$, based on the number of ones present in bit positions b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.
- Similarly, find other values of parity bits.

Follow this procedure for finding **check bits**.

- Find the value of c_1, based on the number of ones present in bit positions b_1, b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.
- Find the value of c_2, based on the number of ones present in bit positions b_2, b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.
- Find the value of c_3, based on the number of ones present in bit positions b_4, b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.
- Similarly, find other values of check bits.
- The decimal equivalent of the check bits in the received data gives the value of bit position, where the error is present. Just complement the value present in that bit position. Therefore, we will get the original binary code after removing parity bits.

Example 1

Let us find the Hamming code for binary code, $d_4d_3d_2d_1$ = 1000. Consider even parity bits.

The number of bits in the given binary code is n=4.

We can find the required number of parity bits by using the following mathematical relation.

$2^k \geq n+k+1$

Substitute, n=4 in the above mathematical relation.

$\Rightarrow 2^k \geq n+k+1 \Rightarrow 2^k \geq 4+k+1$

$\Rightarrow 2^k \geq 5+k$

The minimum value of k that satisfied the above relation is 3. Hence, we require 3 parity bits p_1, p_2, and p_3. Therefore, the number of bits in Hamming code will be 7, since there are 4 bits in binary code and 3 parity bits. We have to place the parity bits and bits of binary code in the Hamming code as shown below.

The **7-bit Hamming code** is $b_7b_6b_5b_4b_3b_2b_1 = d_4d_3d_2p_3d_1p_2p_1$

By substituting the bits of binary code, the Hamming code will be $b_7b_6b_5b_4b_3b_2b_1 = 100p_30p_2p_1$ Now, let us find the parity bits.

$p_1=b_7 \oplus b_5 \oplus b_3=1 \oplus 0 \oplus 0=1$

$p_2=b_7 \oplus b_6 \oplus b_3=1 \oplus 0 \oplus 0=1$

$p_3=b_7 \oplus b_6 \oplus b_5=1 \oplus 0 \oplus 0=1$

By substituting these parity bits, the **Hamming code** will be $b_7b_6b_5b_4b_3b_2b_1 = 100p_30p_2p_1$ =1001011.

Example 2

In the above example, we got the Hamming code as $b_7b_6b_5b_4b_3b_2b_1 = 100p_30p_2p_1$ =1001011. Now, let us find the error position when the code received is $b_7b_6b_5b_4b_3b_2b_1$ = 1001111.

Now, let us find the check bits.

$c_1=b_7 \oplus b_5 \oplus b_3 \oplus b_1 =1 \oplus 0 \oplus 1 \oplus 1=1$

$c_2=b_7 \oplus b_6 \oplus b_3 \oplus b_2 =1 \oplus 0 \oplus 1 \oplus 1=1$

$c_3=b_7 \oplus b_6 \oplus b_5 \oplus b_4 =1 \oplus 0 \oplus 0 \oplus 1=0$

The decimal value of check bits gives the position of error in received Hamming code.

$c_3c_2c_1 = (011)_2 = 3_{10}$

Therefore, the error present in third bit (b_3) of Hamming code. Just complement the value present in that bit and remove parity bits in order to get the original binary code.

Example 3 and 4

Let us find the Hamming code for binary code, $d_4d_3d_2d_1$ = 1010 which when transmitted was received as 1110. Consider even parity bits.

Detect the error and correct it using Hamming codes.

Let us find the Hamming code for binary code, $d_4d_3d_2d_1$ = 1010. Consider even parity bits.

The number of bits in the given binary code is n=4.

We can find the required number of parity bits by using the following mathematical relation.

$2^k \geq n+k+1$

Substitute, n=4 in the above mathematical relation.

$\Rightarrow 2^k \geq n+k+1 \Rightarrow 2^k \geq 4+k+1$

$\Rightarrow 2^k \geq 5+k$

The minimum value of k that satisfied the above relation is 3. Hence, we require 3 parity bits p_1, p_2, and p_3. Therefore, the number of bits in Hamming code will be 7, since there are 4 bits in binary code and 3 parity bits. We have to place the parity bits and bits of binary code in the Hamming code as shown below.

The **7-bit Hamming code** is $b_7b_6b_5b_4b_3b_2b_1$ = $d_4d_3d_2p_3d_1p_2p_1$

By substituting the bits of binary code, the Hamming code will be $b_7b_6b_5b_4b_3b_2b_1$ = $101p_30p_2p_1$ Now, let us find the parity bits.

p_1=$b_7 \oplus b_5 \oplus b_3$=$1 \oplus 1 \oplus 0$=0

p_2=$b_7 \oplus b_6 \oplus b_3$=$1 \oplus 0 \oplus 0$=1

p_3=$b_7 \oplus b_6 \oplus b_5$=$1 \oplus 0 \oplus 1$=0

By substituting these parity bits, the **Hamming code** will be $b_7b_6b_5b_4b_3b_2b_1$ = $111p_30p_2p_1$ =1110010.

Example 4

In the above example, we got the Hamming code as $b_7b_6b_5b_4b_3b_2b_1$ = $111p_30p_2p_1$ =1110010. Now, let us find the error position when the code received is $b_7b_6b_5b_4b_3b_2b_1$ = 1110010.

Now, let us find the check bits.

c_1=$b_7 \oplus b_5 \oplus b_3 \oplus b_1$ =$1 \oplus 1 \oplus 0 \oplus 0$=0

c_2=$b_7 \oplus b_6 \oplus b_3 \oplus b_2$ =$1 \oplus 1 \oplus 0 \oplus 1$=1

c_3=$b_7 \oplus b_6 \oplus b_5 \oplus b_4$ =$1 \oplus 1 \oplus 1 \oplus 0$=1

The decimal value of check bits gives the position of error in received Hamming code.

$c_3c_2c_1$ = $(110)_2$ = 6_{10}

Therefore, the error present in sixth bit (b_6) of Hamming code. Just complement the value present in that bit and remove parity bits in order to get the original binary code.

ERROR DETECTING CODES – CHECK SUM

This is a block code method where a checksum is created based on the data values in the data blocks to be transmitted using some algorithm and appended to the data. When the receiver gets this data, a new checksum is calculated and compared with the existing checksum. A non-match indicates an error.

Error Detection by Checksums

For error detection by checksums, data is divided into fixed sized frames or segments.

- **Sender's End** – The sender adds the segments using 1's complement arithmetic to get the sum. It then complements the sum to get the checksum and sends it along with the data frames.
- **Receiver's End** – The receiver adds the incoming segments along with the checksum using 1's complement arithmetic to get the sum and then complements it.

If the result is zero, the received frames are accepted; otherwise they are discarded.

Example

Suppose that the sender wants to send 4 frames each of 8 bits, where the frames are 11001100, 10101010, 11110000 and 11000011.

The sender adds the bits using 1s complement arithmetic. While adding two numbers using 1s complement arithmetic, if there is a carry over, it is added to the sum.

After adding all the 4 frames, the sender complements the sum to get the checksum, 11010011, and sends it along with the data frames.

Example

The receiver performs 1s complement arithmetic sum of all the frames including the checksum. The result is complemented and found to be 0. Hence, the receiver assumes that no error has occurred.

This is a block code method where a checksum is created based on the data values in the data blocks to be transmitted using some algorithm and appended to the data. When the receiver gets this data, a new checksum is calculated and compared with the existing checksum. A non-match indicates an error.

<u>Sender's End</u>

```
Frame 1 :          11001100
Frame 2 :        + 10101010
             -----------------------------
Partial sum:     1 01110110
                         +1
             -----------------------------
                   01110111
Frame 3:         + 11110000
             -----------------------------
Partial Sum:     1 00101011
                         +1
                 ------------------------------
                   01101000
Frame 4:           11000011
                  -----------------------------
Partial Sum:     1  01100111
```

```
                           +1
                 ------------------------------
Sum :              00101100
                  ------------------------------
Check Sum:          11010011
                 ------------------------------
```

Receiver's End

```
Frame 1 :            11001100
Frame 2 :          + 10101010
               -----------------------------
Partial sum:       1 01110110
                             +1
               -----------------------------
                      01110111
Frame 3:           + 11110000
               -----------------------------
Partial Sum:       1 00101011
                             +1
                 ------------------------------
                      01101000
Frame 4:              11000011
                  -----------------------------
Partial Sum:       1  01100111
                              +1
                 ------------------------------
Sum :                 00101100
Check Sum:            11010011
                 ------------------------------
Sum                   11111111

Complement            00000000

                 ------------------------------
```

Hence accepted frames.

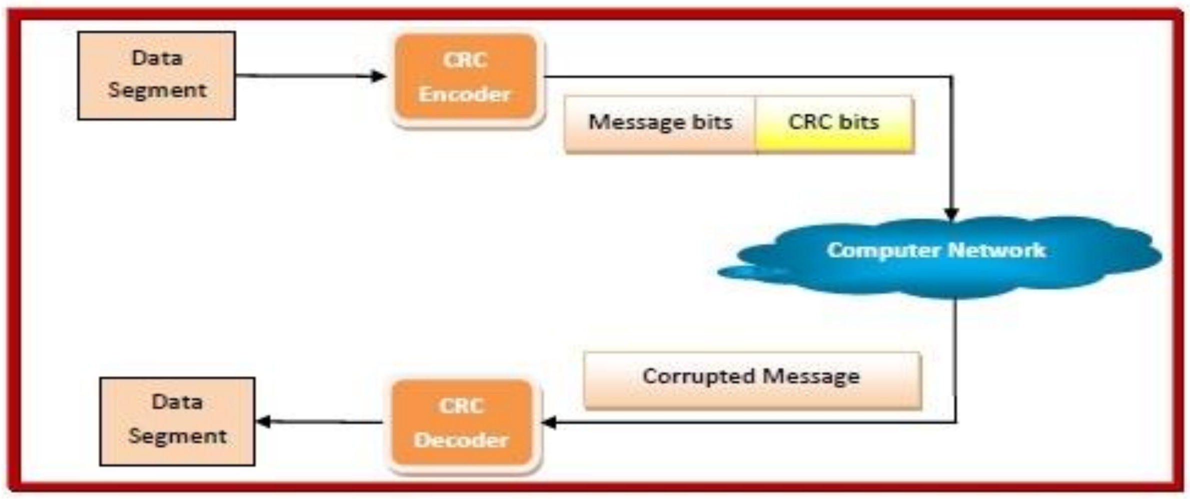
ERROR DETECTING CODES – CRC

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) is a block code and is commonly used to detect accidental changes to data transmitted via telecommunications networks and storage devices.

CRC involves binary division of the data bits being sent by a predetermined divisor agreed upon by the communicating system. The divisor is generated using polynomials. So, CRC is also called polynomial code checksum.

The process is illustrated as follows –



Encoding using CRC

- The communicating parties agrees upon the size of message block and the CRC divisor. For example, the block chosen may be CRC (7, 4), where 7 is the total length of the block and 4 is the number of bits in the data segment. The divisor chosen may be 1011.
- The sender performs binary division of the data segment by the divisor.
- It then appends the remainder called CRC bits to the end of data segment. This makes the resulting data unit exactly divisible by the divisor.

Decoding

- The receiver divides the incoming data unit by the divisor.
- If there is no remainder, the data unit is assumed to be correct and is accepted.
- Otherwise, it is understood that the data is corrupted and is therefore rejected. The receiver may then send an erroneous acknowledgement back to the sender for retransmission.

CRYPTOGRAPHY

Cryptography is the practice and study of techniques for secure communication in the presence of third parties called adversaries. More generally, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages; various aspects in information security such as data confidentiality, data integrity and authentication are central to modern cryptography. Applications of cryptography include e-commerce, chip based payment cards, digital currencies, computer passwords and military communications.

Cryptography prior to the modern age was effectively synonymous with encryption, the conversion of information from a readable state to apparent nonsense. The originator of an encrypted message shares the decoding technique only with intended recipients to preclude access from adversaries.

Modern cryptography is heavily based on mathematical theory and computer science practice and cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in practice by any adversary. It is theoretically possible to break such a system, but it is infeasible to do so by any known practical means. These schemes are therefore termed computationally secure.

CEASER CIPHER

In cryptography, a **Caesar cipher** is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.

The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. For instance, here is a Caesar cipher using a left rotation of three places, equivalent to a right shift of 23 (the shift parameter is used as the key):

Plain:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:	XYZABCDEFGHIJKLMNOPQRSTUVW

When encrypting, a person looks up each letter of the message in the "plain" line and writes down the corresponding letter in the "cipher" line.

Plaintext:	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Ciphertext:	QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD

Deciphering is done in reverse, with a right shift of 3.

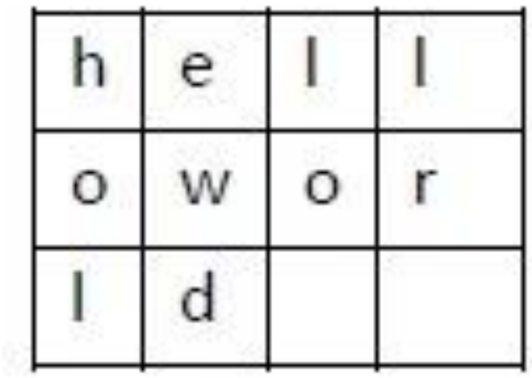
TRANSPOSITION CIPHER

Transposition Cipher is a cryptographic algorithm where the order of alphabets in the plaintext is rearranged to form a cipher text. In this process, the actual plain text alphabets are not included.

Example

A simple example for a transposition cipher is **columnar transposition cipher** where each character in the plain text is written horizontally with specified alphabet width. The cipher is written vertically, which creates an entirely different cipher text.

Consider the plain text **hello world**, and let us apply the simple columnar transposition technique as shown below



h	e	l	l
o	w	o	r
l	d		

The plain text characters are placed horizontally and the cipher text is created with vertical format as **: holewdlo lr.**

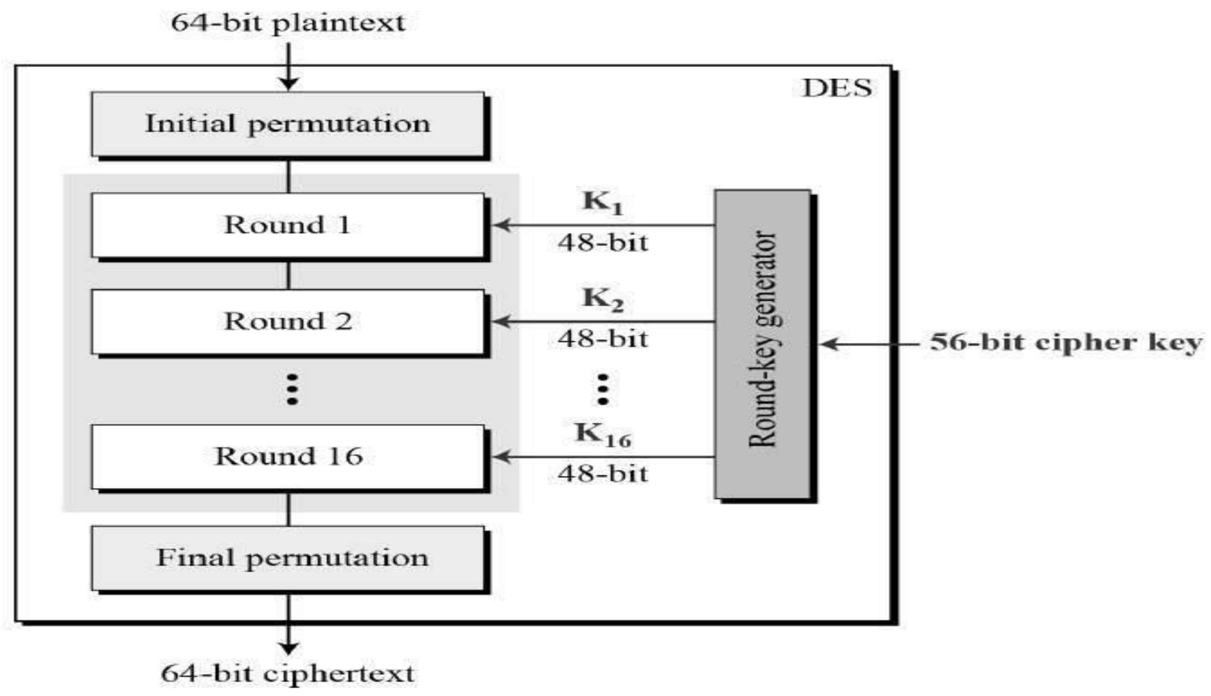
Now, the receiver has to use the same table to decrypt the cipher text to plain text.

DES – DATA ENCRYPTION STANDARD

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only).

General Structure of DES is depicted in the following illustration −



Since DES is based on the Feistel Cipher, all that is required to specify DES is −

- Round function
- Key schedule
- Any additional processing − Initial and final permutation

DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- **Avalanche effect** − A small change in plaintext results in the very great change in the ciphertext.
- **Completeness** − Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

CONCEPT OF KEYS

A network security key is a kind of network password/digital signature that one enters as authorization to gain access to a wireless network.

Network security keys also establish secure connections between the user requesting access and the network or wireless device. This protects a network and its associated devices from unwanted access.

When we have a weak network security key, we leave our network open to access by cyber criminals. If a cyber criminal is able to access our personal information, it can be sold or can lead to identity theft and other serious consequences.

The most common types of network security keys are Wired Equivalent Privacy (WEP) and WiFi Protected Access (WPA/WPA2) .

WEP

A WEP key is **a security passcode for WiFi-enabled devices**. WEP keys let devices on a network exchange encrypted messages with each other while blocking those messages from being easily decoded and read by outsiders.

WEP keys are a sequence of characters taken from the numbers 0 through 9 and the letters A through F. For instance, a WEP key could be F45HI00WR3.

The required length of a WEP key could be 10, 26, or 58 characters long, depending on which WEP version is running. WEP keys can be automatically generated for ease of use. When they're not, there are websites that generate random hard-to-guess WEP keys for us.

WEP keys **lost public favor when people began to realize that they are easy to crack**, which leaves our network potentially open to hackers.

WPA/WPA2

WPA and WPA2 are more secure network security keys than WEP. A WPA comes with a password/passphrase that we can obtain from the network's owner. **If you have a WiFi router at home, the password that you sometimes see printed on the side of it is the WPA key.**

Usually with these kind of networks, the owner can reset the WPA key to be whatever password or passphrase they want.

WPA2 is an even more secure version of WPA because it uses the AES algorithm, a newer, faster, and more advanced algorithm than what was previously used.

WPA2 is usually the recommended version for businesses, who tend to need heightened security measures. Keep in mind, however, that to use WPA2, we might need hardware with a higher processing power.

NETWORK SECURITY MODEL

When we send our data from source side to destination side we have to use some transfer method like the internet or any other communication channel by which we are able to send our message. The two parties, who are the principals in this transaction, must cooperate for the exchange to take place. When the transfer of data happened from one source to another source some logical information channel is established between them by defining a route through the internet from source to destination and by the cooperative use of communication protocols (e.g., TCP/IP) by the two principals.

When we use the protocol for this logical information channel the main aspect security has come. who may present a threat to confidentiality, authenticity, and so on. All the technique for providing security have to components:

1. A security-related transformation on the information to be sent.
2. Some secret information shared by the two principals and, it is hoped, unknown to the opponent.

A trusted third party may be needed to achieve secure transmission. For example, a third party may be responsible for distributing the secret information to the two principals while keeping it from any opponent. Or a third party may be needed to arbitrate disputes between the two principals concerning the authenticity of a message transmission.

This model shows that there are four basic tasks in designing a particular security service:

1. Design an algorithm for performing the security-related transformation.
2. Generate the secret information to be used with the algorithm.
3. Develop methods for the distribution and sharing of secret information.
4. Specify a protocol to be used by the two principals that make use of the security algorithm and the secret information to achieve a particular security service.

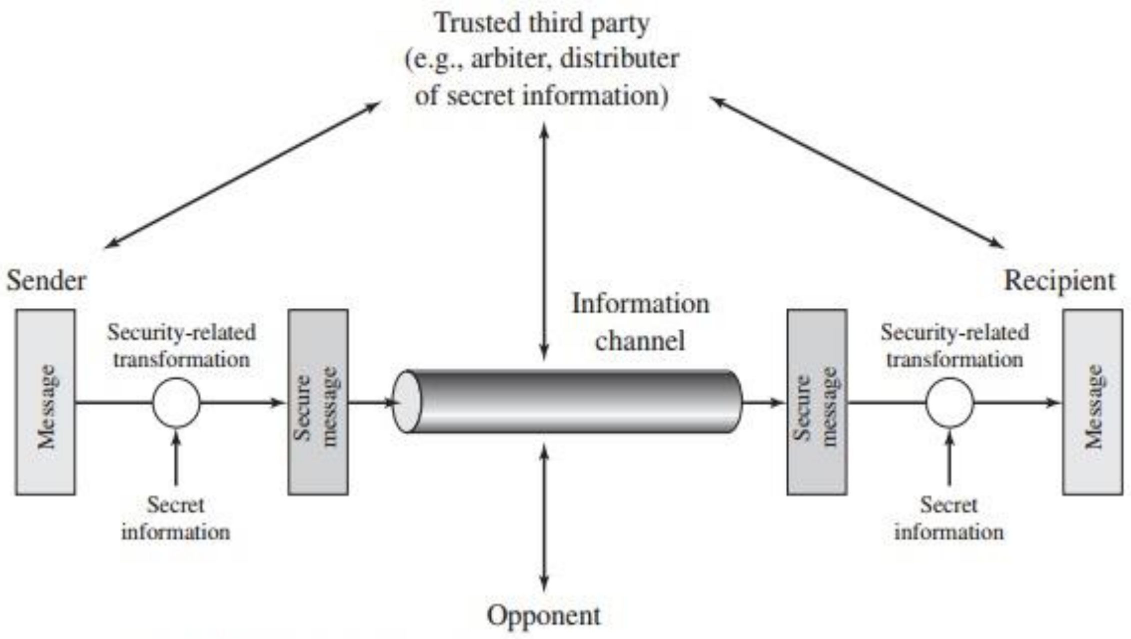


Figure 1.4 Model for Network Security