

Syllabus Requirements:

2 Data transmission

2.1 Types and methods of data transmission


Candidates should be able to:

- (a) Understand that data is broken down into packets to be transmitted
- (b) Describe the structure of a packet

Notes and guidance

A packet of data in a unit of data contains a: **packet header** , **payload** and **trailer**

The packet header includes the:

- o destination address
 - o packet number
 - o originator's address
- (c) Describe the process of packet switching
 -  Data is broken down into packets
 - o Each packet could take a different route
 - o A router controls the route a packet takes
 - o Packets may arrive out of order
 - o Once the last packet has arrived, packets are reordered
 - (a) Describe how data is transmitted from one device to another using different methods of data transmission
 - (b) Explain the suitability of each method of data transmission, for a given scenario
- Understand the universal serial bus (USB) interface and explain how it is used to transmit data
 - o Including: – serial – parallel – simplex – half-duplex – full-duplex
 - o advantages and disadvantages of each method
 - o Including the benefits and drawbacks of the interface

2.2 Methods of error detection

Candidates should be able to: 1 Understand the need to check for errors after data transmission and how these errors can occur

Notes and guidance

- o Errors can occur during data transmission due to interference, e.g. data loss, data gain and data change
- Describe the processes involved in each of the following error detection methods for detecting errors in data after transmission: parity check (odd and even), checksum and echo check

Notes and guidance

- o Including parity byte and parity block check
- Describe how a check digit is used to detect errors in data entry and identify examples of when a check digit is used, including international standard book numbers (ISBN) and bar codes
 - Describe how an automatic repeat query (ARQ) can be used to establish that data is received without error
- Notes and guidance
- o Including the use of: – positive/negative acknowledgements – timeout







2.3 Encryption Candidates should be able to:

- Understand the need for and purpose of encryption when transmitting data
- Understand how data is encrypted using symmetric and asymmetric encryption

Notes and guidance • Asymmetric encryption includes the use of public and private keys




When data is sent from one device to another, it is important to consider how that data is transmitted.

Data Packet:

-  Information on the internet is broken down into **packets** sometimes called **datagram**, and are created by **TCP** and transmitted over the internet
-  Packets are small chunks of information/data
-  TCP stands for **Transmission Control Protocol** and is used for organising data transmission over networks
-  Small chunks of data are easier and quicker to **route** over the internet than big chunks of data
-  **Routing** involves finding the most optimal path over a network
-  Data can include anything from text, images, audio, video, animations, etc, or any combination of these

What do packets contain?

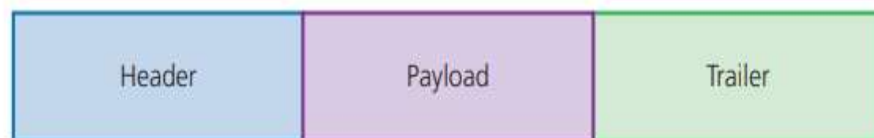
Packets are “chunks” of information. This information is called the “**payload**”
Packets act like postage letters, each one has: a delivery address (**destination IP address**), a return address (**source IP address**), and a message (**data payload**)
Packets are split into three parts:

-  Packet header
-  Payload (the actual data)
-  Trailer

The header contains:

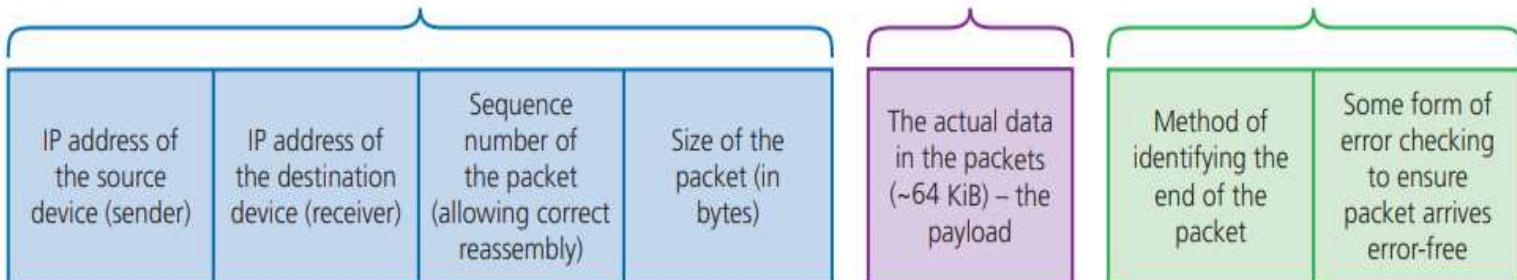
- **Source IP** (IP address of sending device)
- **Destination IP** (IP address of receiving device)
- **Packet sequence number** (eg packet 31 out of 100)
- **Packet size**

Payload
(actual data being sent)



The trailer contains:

- **Error checks** (e.g. Cyclic Redundancy Check, CRC)
- **End of packet notification**



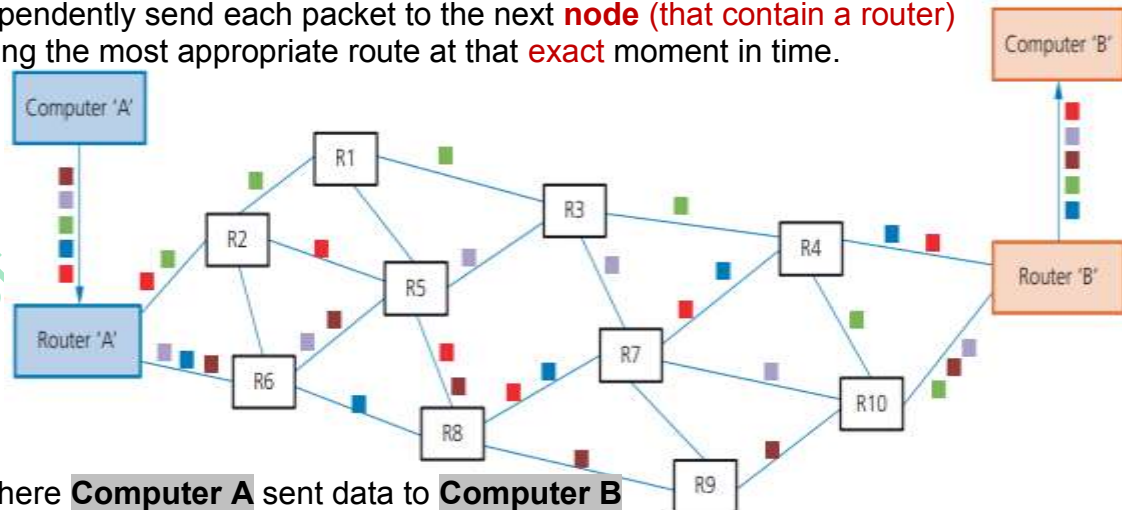
Cyclic redundancy checks (CRCs) are used to check data packets: –

- Sender computer adds up all the **1-bits** in the **payload** and **stores this as hex value in the trailer** before it is sent.
- once packet arrives, **receiving computer recalculates** number of **1-bits** in the payload.
- the computer then compares both values (calculated value with the value in trailer)
- if values match, then no error; otherwise the packet needs to be **re-sent**.

Packet Switching:







- Packet switching is the process of transmitting data by splitting it into a series of small data packets (as described above)
- A data packet will encounter several nodes as it makes its journey from A to B.
- At each node is a **router**.
- The **router** will process the data packets, using the information found in the head section of the packet, and decides where to send the data packet.
- A router will independently send each packet to the next **node (that contain a router)** of its journey, using the most appropriate route at that **exact** moment in time.






In above scenario where **Computer A** sent data to **Computer B**

- each packet will follow its own path (route)
- routers will determine the route of each packet
- routing selection depends on the number of packets waiting to be processed at each node
- the shortest possible path available is always selected – this may not always be the shortest path that could be taken, since certain parts of the route may be too busy or not suitable
- unfortunately, packets can reach the destination in a different order to that in which they were sent
- Also note that packets can arrive in a different order compared to the way they were sent.

Pros of packet switching:

-  High transmission rates can be achieved through packet switching
-  No single connection is fully occupying a communication line
-  Busy or faulty lines will not stop data from reaching its destination
-  Expansion is easy to achieve

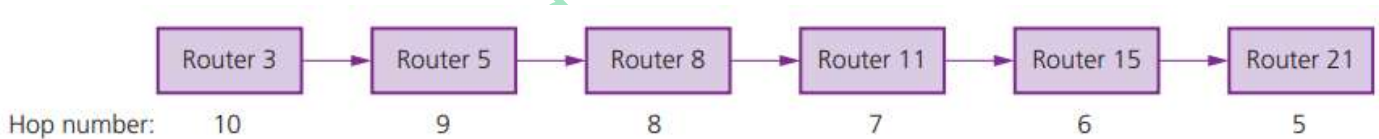
Cons of packet switching:




-  Lost packets will need to be resent
-  Putting packets back into the correct sequence causes a small delay
-  The method is more prone to errors in **real-time streaming**.

Sometimes it is possible for packets to get lost because they keep 'bouncing' around from router to router and never actually reach their destination.

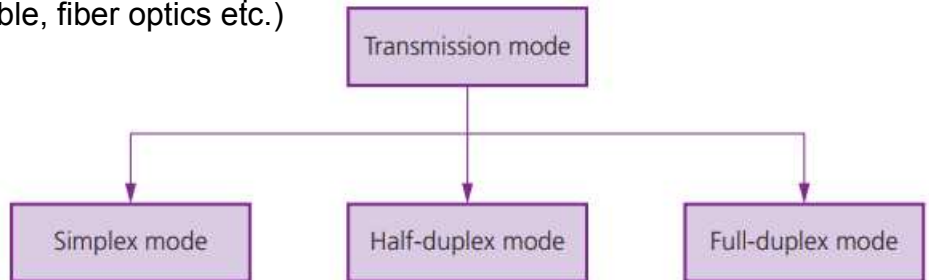
Eventually the network would just grind to a halt as the number of lost packets mount up, clogging up the system.

-  To overcome this, a method called **hopping** is used. A **hop** number is added to the header of each packet, and this number is reduced by 1 every time it leaves a router



-  Each packet has a maximum hop number to start with.
-  Once a hop number reaches zero, and the packet hasn't reached its destination, then the packet is deleted when it reaches the next router.
-  The missing packets will then be flagged by the receiving computer and a request to re-send these packets will be made.

Data transmission refers to the movement of data in the form of bits between two or more digital devices. This transfer of data takes place via some form of transmission media (for example, coaxial cable, fiber optics etc.)



Data transmission

Data transmission can be either over a short distance (for example, from computer to printer) or over longer distances (for example, over a telephone network).

Essentially, three factors need to be considered when transmitting data (each factor has to be agreed by both sender and receiver for this to work without error):



the direction of the data transmission (i.e. in one direction only or in both directions)



the method of transmission (how many bits are sent at the same time)



the method of synchronization between the two devices.

Simplex, half-duplex and full-duplex

SIMPLEX DATA TRANSMISSION: is in *one direction* only (i.e. from sender to receiver).

Example: data being sent from a computer to a printer.

010010010100010101000011

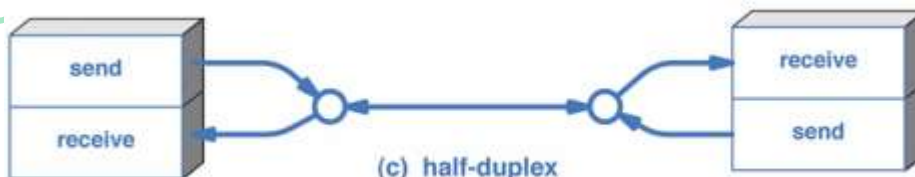
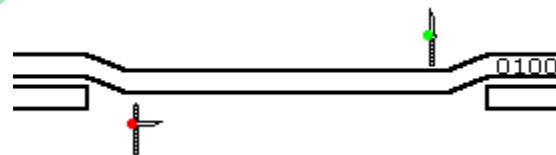


(a) simplex

HALF-DUPLEX DATA TRANSMISSION: is in *both directions* **but not at the same time**

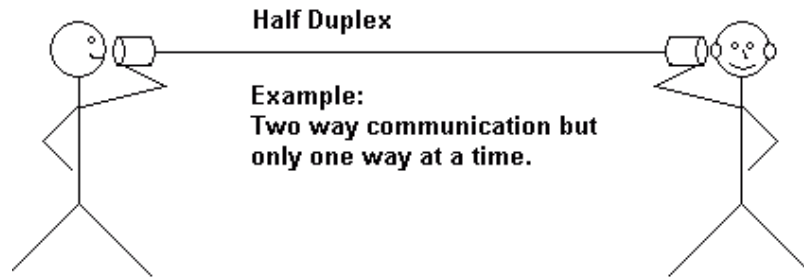
(i.e. data can be sent from 'A' to 'B' or from 'B' to 'A' along the same line, but not at the same time).

Example: a phone conversation between two people where only one person speaks at a time a **Walkie Talkie**.



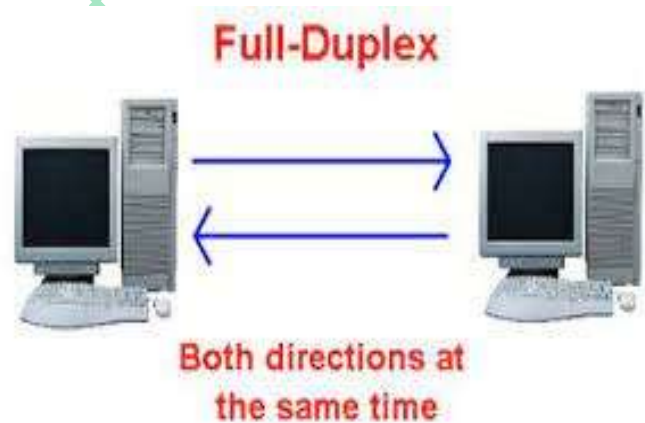
(c) half-duplex

Half-Duplex is like the dreaded "one lane" road you may have run into at construction sites. Only one direction will be allowed through at a time. Railroads have to deal with this scenario more often since it's cheaper to lay a single track.



FULL-DUPLEX DATA TRANSMISSION is in *both directions simultaneously* (i.e. data can be sent from 'A' to 'B' and from 'B' to 'A' along the same line, *both at the same time*). Example: broadband connection on a phone line.

Full-Duplex



In communications, this is most common with networking.



Full-duplex also called **Duplex** mode occurs when data can be sent in BOTH DIRECTIONS AT THE SAME TIME



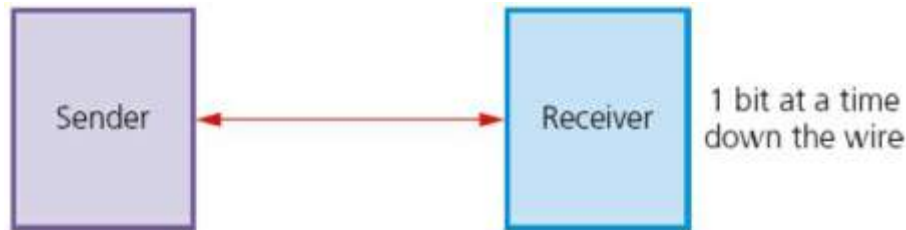
(for example, data can be sent from 'A' to 'B' and from 'B' to 'A' along the same transmission line simultaneously).



An example of this would be a broadband internet connection.

Serial and parallel data transmission

SERIAL DATA TRANSMISSION is when data is sent, *one bit at a time*, over a *single wire or channel* (bits are sent one after the other in a single stream).

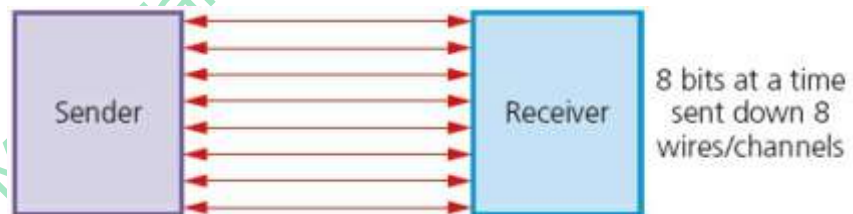


(Note: bits can be transmitted as simplex, half-duplex or full-duplex.)

- This method of data transmission works well over long distances.
- However, data is transmitted at a slower rate than parallel data transmission.
- Since only one wire or channel is used, there is no problem of data arriving at its destination out of synchronisation.

An example of its use is sending data from a computer to a modem for transmission over a telephone line.

PARALLEL DATA TRANSMISSION: is when *several bits of data (usually 1 byte)* are sent down *several wires or channels at the same time*; one wire or channel is used to transmit each bit.



(Note: bits can be transmitted as simplex, half-duplex or full-duplex.)

PARALLEL DATA TRANSMISSION:

- This method of data transmission works very well over short distances (over longer distances, the bits can become 'skewed' – this means they will no longer be synchronised).
- It is, however, a faster method of data transmission than serial.
- An example of its use is when sending data to a printer from a computer using a ribbon connector.



A common use for serial data transmission is **(Universal Serial Bus (USB))**.

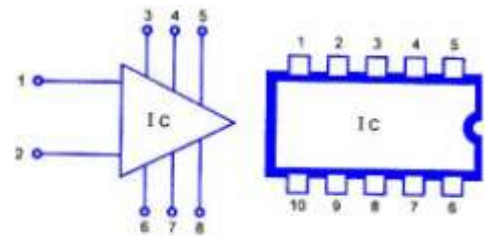
Parallel data transmission is used in the **internal electronics of the computer** system. The pathways between the CPU and the memory all use this method of data transmission.

Pros and Cons of Serial and Parallel data transmission:

Serial	Parallel
less risk of external interference than with parallel (due to fewer wires)	faster rate of data transmission than serial
more reliable transmission over longer distances	works well over shorter distances (for example, used in internal pathways on computer circuit boards)
transmitted bits won't have the risk of being skewed (that is, out of synchronisation)	since several channels/wires used to transmit data, the bits can arrive out of synchronisation (skewed)
used if the amount of data being sent is relatively small since transmission rate is slower than parallel (for example, USB uses this method of data transmission)	preferred method when speed is important
used to send data over long distances (for example, telephone lines)	if data is time-sensitive, parallel is the most appropriate transmission method
less expensive than parallel due to fewer hardware requirements	parallel ports require more hardware, making them more expensive to implement than serial ports
	easier to program input/output operations when parallel used

Integrated circuits:

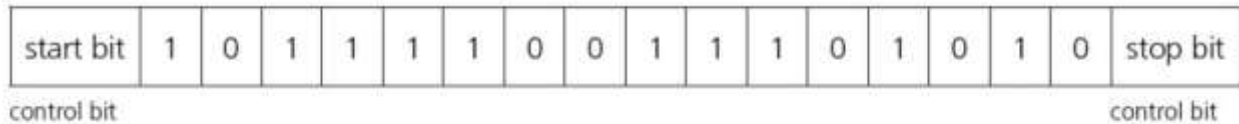
Busess and other internal components all **use parallel data transmission** because of the need for high speed data transfer.



- The use of 8-bit, 16-bit, 32-bit and 64-bit buses, for example, allow much faster data transmission rates than could be achieved with single channel serial data transfer.
- An internal clock is used to ensure the correct timing of data transfer; it is essentially synchronous in nature and the short distances between components mean that none of the issues described earlier have any real impact on the accuracy of the data.

ASYNCHRONOUS DATA TRANSMISSION:

Asynchronous data transmission refers to data being transmitted in an agreed bit pattern. Data bits (1s and 0s) are grouped together and sent with **CONTROL BITS**:



This means that the receiver of the data knows when the data starts and when it ends.

This prevents data becoming mixed up; without these control bits, it would be impossible to separate groups of data as they arrived.




SYNCHRONOUS DATA TRANSMISSION

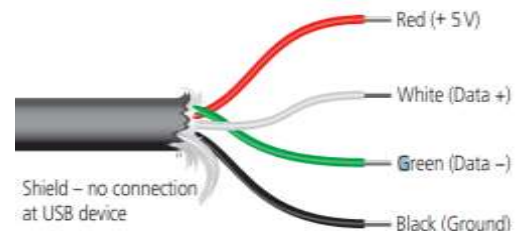
Synchronous data transmission is a continuous stream of data (unlike asynchronous data which is sent in discrete groups). The data is accompanied by timing signals generated by an internal clock. This ensures that the sender and receiver are synchronised with each other. The receiver counts how many bits (1s and 0s) were sent and then reassembles them into bytes of data.

The timing must be very accurate here since there are no control bits sent in this type of data transmission. However, it is a faster data transfer method than asynchronous and is therefore used where this is an important issue (for example, in network communications).




Universal Serial Bus (USB)

The **UNIVERSAL SERIAL BUS (USB)** is an asynchronous serial data transmission method. It has quickly become the standard method for transferring data between a computer and a number of devices. Essentially, the USB cable consists of:

-  a four-wire shielded cable
-  two of the wires are used for power and the earth
-  two of the wires are used in the data transmission.



When a device is plugged into a computer using one of the USB ports:





-  the computer automatically detects that a device is present (this is due to a small change in the voltage level on the data signal wires in the cable)
-  the device is automatically recognised, and the appropriate **DEVICE DRIVER** is loaded up so that computer and device can communicate effectively
-  if a new device is detected, the computer will look for the device driver which matches the device; if this is not available, the user is prompted to download the appropriate software.

Pros and Cons of USB:





✓	✗
Devices plugged into the computer are automatically detected; device drivers are automatically uploaded	—
The connectors can only fit one way; this prevents incorrect connections being made	The maximum cable length is presently about 5 metres
This has become the industry standard; this means that considerable support is available to users	—
Several different data transmission rates are supported	The present transmission rate is limited to less than 500 megabits per second
Newer USB standards are backward compatible with older USB standards	The older USB standard (e.g. 1.1) may not be supported in the near future

Error-checking methods

Following data transmission, there is always the risk that the data has been corrupted or changed in some way.

-  This can occur whether data is being transmitted over short distances or over long distances.
-  Checking for errors is important since computers aren't able to check that text is correct; they can only recognise whether a word is in their built-in dictionary or not.
-  This is why error checking is such an important part of computer technology.
-  This section considers a number of ways that can be used to check for errors.

A number of methods exist which can detect errors and, in some cases, actually correct the error. The methods covered in this section are:

-  parity checking
-  checksum
-  echo checking
-  automatic repeat request (ARQ)

Parity checking

PARITY CHECKING is one method used to check whether data has been changed or corrupted following transmission from one device or medium to another device or medium.

A byte of data, for example, is allocated a **PARITY BIT**. This is allocated before transmission takes place. Systems that use **EVEN PARITY** have an even number of 1-bits; systems that use **ODD PARITY** have an odd number of 1-bits.

Consider the following byte:

	1	1	0	1	1	0	0
--	---	---	---	---	---	---	---

If this byte is using even parity, then the parity bit needs to be 0 since there is already an even number of 1-bits (in this case, 4).

If odd parity is being used, then the parity bit needs to be 1 to make the number of 1-bits odd.

Therefore, the byte just before transmission would be: either (even parity)

0	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---

parity bit

or (odd parity)

1	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---

parity bit

If two of the bits change value following data transmission, it may be impossible to locate the error using parity checking.

0	1	0	1	1	1	0	0
---	---	---	---	---	---	---	---

Let us imagine we are transmitting the following byte, using even parity:

Suppose more than one bit has been modified during data transmission.

This means the byte could have reached the destination as any of the following, parity remains **even** and no error is detected although errors have occurred:

0	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---

six 1-bits

In all cases where parity remains even, no errors are detected.

0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

two 1-bits

0	1	0	1	0	1	1	0
---	---	---	---	---	---	---	---

four 1-bits

Parity blocks:

When error has been flagged, it is impossible to know exactly which bit is in error.



One of the ways round this problem is to **use parity blocks**.



In this method, a block of data is sent and the number of 1-bits are totalled horizontally and vertically (in other words, a parity check is done in both horizontal and vertical directions).

As the following example shows, this method not only identifies that an error has occurred but also indicates where the error is.

	Parity bit	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Byte 1	1	1	1	1	0	1	1	0
Byte 2	1	0	0	1	0	1	0	1
Byte 3	0	1	1	1	1	1	1	0
Byte 4	1	0	0	0	0	0	1	0
Byte 5	0	1	1	0	1	0	0	1
Byte 6	1	0	0	0	1	0	0	0
Byte 7	1	0	1	0	1	1	1	1
Byte 8	0	0	0	1	1	0	1	0
Byte 9	0	0	0	1	0	0	1	0
Parity byte	1	1	0	1	0	0	0	1

Before data is transferred, an agreement is made between sender and receiver regarding which of the two types of parity are used. This is called **HANDSHAKING**.

Automatic Repeat Request (ARQ)

AUTOMATIC REPEAT REQUEST (ARQ) is another method used to check whether data has been correctly transmitted.

It uses an **ACKNOWLEDGEMENT** (a message sent by the receiver indicating that data has been received correctly) and **TIMEOUT** (this is the time allowed to elapse before an acknowledgement is received).

If an acknowledgement isn't sent back to the sender before timeout occurs, then the message is automatically resent.

Checksum

CHECKSUM is another way to check if data has been changed or corrupted following data transmission. Data is sent in blocks and an additional value, the checksum, is also sent at the end of the block of data.

To explain how this works, we will assume the checksum of a block of data is 1 byte in length. This gives a maximum value of $2^8 - 1$ (i.e. 255). The value 0000 0000 is ignored in this calculation. Example 3 explains how a checksum is generated.

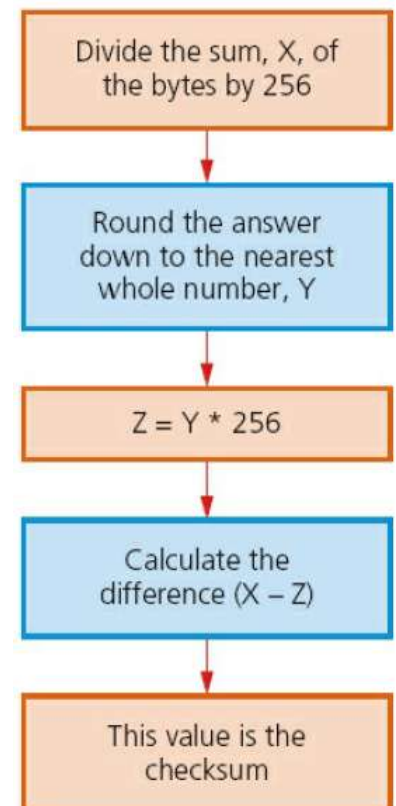
Example





If the sum of all the bytes in the transmitted block of data is ≤ 255 , then the checksum is this value.

However, if the sum of all the bytes in the data block > 255 , then the checksum is found using the simple algorithm in

Suppose the value of X is 1185, then tracing through the algorithm, we get:
X = 1185

- 1185/256 = 4.629
- Rounding down to nearest whole number gives $Y = 4$
- Multiplying by 256 gives $Z = Y * 256 = 1024$
- The difference $(X - Z)$ gives the checksum: $(1185 - 1024) = 161$
- This gives the checksum = 161



-  When a block of data is about to be transmitted, the checksum for the bytes is first of all calculated.
-  This value is then transmitted with the block of data. At the receiving end, the checksum is recalculated from the block of data received.
-  This calculated value is then compared to the checksum transmitted.
-  If they are the same value, then the data was transmitted without any errors; if the values are different, then a request is sent for the data to be retransmitted.

CHECK DIGIT: (ISBN13)

Number added to a code (such as a bar code or account number) to derive a further number as a means of verifying the accuracy or validity of the code as it is printed or transmitted. A code consisting of three digits, for example, such as 135 may include 9 (sum of 1, 3, and 5) as the last digit and be communicated as 1359.

Check digits can identify 3 types of error:

- (1) If 2 digits have been inverted e.g. "23459" instead of "23549"
- (2) An incorrect digit entered e.g. 23559 instead of 23549
- (3) A digit missed out altogether e.g. 2359 instead of 23549

Example 1: Generation of the check digit from the other 12 digits in a number

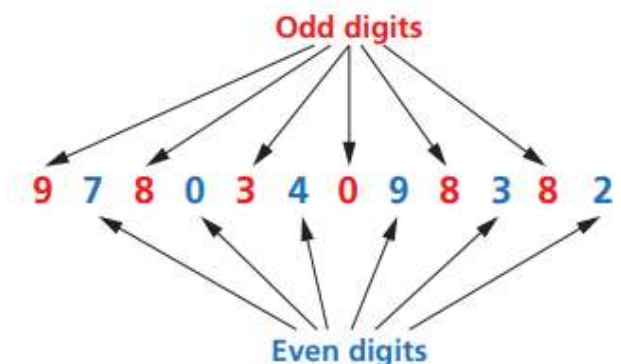
The following algorithm generates the check digit from the 12 other digits:

- 1- add all the **odd numbered (positioned) digits** together
- 2- add all the **even numbered (positioned) digits** together and **multiply the result by 3**
- 3- add the results from **step1** and **step2** together and divide by **10**
- 4- take the remainder, if it is zero then use this value
- 5- otherwise subtract the remainder from 10 to find the check digit.

Using the ISBN **9 7 8 0 3 4 0 9 8 3 8 2** (note this is the same ISBN as in Figure

1. $9 + 8 + 3 + 0 + 8 + 8 = 36$
2. $3 \times (7 + 0 + 4 + 9 + 3 + 2) = 75$
3. $(36 + 75)/10 = 111$
4. $111/10 = 11$ remainder 1
5. $10 - 1 = 9$

9 is the check digit the check digit



CHECK DIGIT: (Modulo 11)

Calculation 1 –

Generation of the check digit from the other digits in a number (In this example, we will assume the original number contained only 7 digits.)

The following algorithm generates the check digit from the other 7 digits:

1. each digit in the number is given a weighting of 8, 7, 6, 5, 4, 3 or 2 starting from the left (weightings start from 8 since the number will become eight-digit when the check digit is added)
2. the digit is multiplied by its weighting and then each value is added to make a total
3. the total is divided by 11
4. the remainder is then subtracted from 11 to find the check digit (note if the remainder is 10 then the check digit 'X' is used).

The example to be used has the following seven-digit number:

1. 7-digit number: **4 1 5 6 7 1 0**
 8 7 6 5 4 3 2 (weighting values)
2. sum: $(8 \times 4) + (7 \times 1) + (6 \times 5) + (5 \times 6) + (4 \times 7) + (3 \times 1) + (2 \times 0)$
 $= 32 + 7 + 30 + 30 + 28 + 3 + 0$ total = 130
3. divide total by **11**:
 $130/11 = 11$ remainder 9
4. subtract remainder from 11:
 $11 - 9 = 2$ (check digit)
5. So we end up with the following eight-digit: **4 1 5 6 7 1 0 2**

Echo check

With **ECHO CHECK**, when data is sent to another device, this data is sent back again to the sender. The sender compares the two sets of data to check if any errors occurred during the transmission process.






As you will have no doubt worked out, this isn't very reliable. If the two sets of data are different, it isn't known whether the error occurred when sending the data in the first place, or if the error occurred when sending the data back for checking! However, if no errors occurred then it is another way to check that the data was transmitted correctly.

Automatic Repeat Request (ARQ):

We have already considered parity checks and echo checks as methods to verify that data has arrived at its destination unchanged.

An Automatic Repeat Request (ARQ) is a third way used to check data following data transmission.

This method can best be summarised as follows:

-  ARQ uses positive and negative acknowledgements (messages sent to the receiver indicating that data has/has not been received correctly) and timeout (this is the time interval allowed to elapse before an acknowledgement is received)
-  the receiving device receives an error detection code as part of the data transmission (this is typically a Cyclic Redundancy Check); this is used to detect whether the received data contains any transmission errors
-  if no error is detected, a **positive acknowledgement** is sent back to the sending device
-  if an error is detected, receiving device sends a **negative acknowledgement** to the sending device and requests re-transmission of the data
-  a time-out is used by the sending device by waiting a pre-determined amount of time ... » ... and if no acknowledgement of any type has been received by the sending device within this time limit, it automatically re-sends the data until a positive acknowledgement is received or until a pre-determined number of re-transmissions has taken place » ARQ is often used by mobile phone networks to guarantee data integrity.

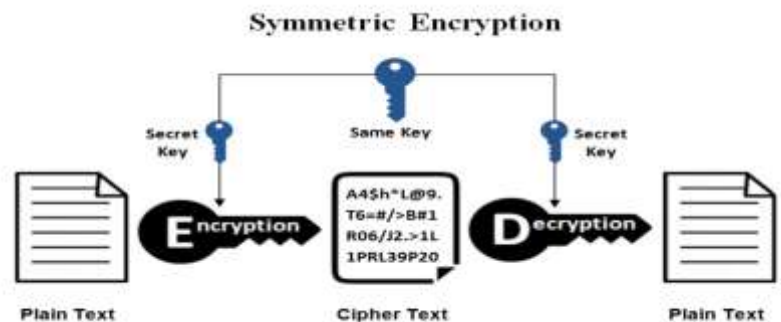
ENCRYPTION

When data is transmitted over any public network (wired or wireless), there is always a risk of it being intercepted by, for example, a hacker.

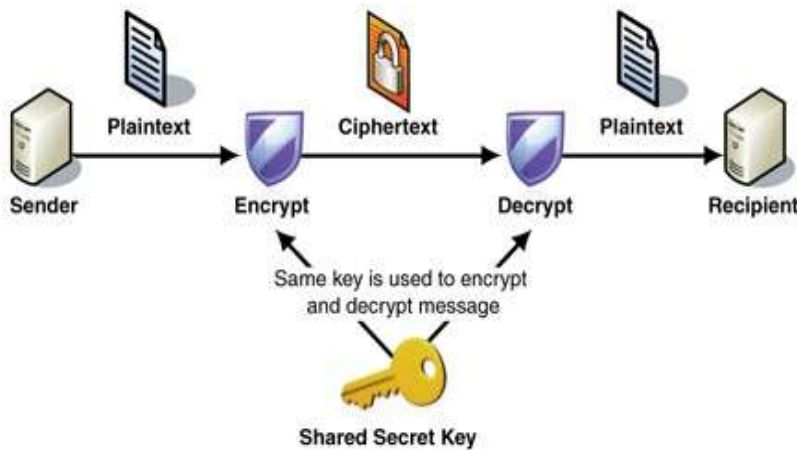
- Under these circumstances, a hacker is often referred to as an eavesdropper.
- Using encryption helps to minimise this risk.
- Encryption alters data into a form that is unreadable by anybody for whom the data is not intended.
- It cannot prevent the data being intercepted, but it stops it from making any sense to the hacker/eavesdropper.

Plaintext and ciphertext

The original data being sent is known as **plaintext**. Once it has gone through an encryption algorithm, it produces **ciphertext**



SYMMETRIC ENCRYPTION



SYMMETRIC ENCRYPTION is a secret key which can be a combination of characters.

- If this key is applied to a message, its content is changed which makes it unreadable unless the recipient also has the decryption key.
- Key is needed to encrypt a message and **same key** is needed to decrypt a message.
- It is obviously important that the sender and receiver have the same encryption and decryption key.
- There is clearly a security risk here, since the sender has to supply the key to the recipient.
- This key could be intercepted by, for example, a hacker which puts the security of the encrypted message at risk.
- This situation is referred to as the **KEY DISTRIBUTION PROBLEM**

Suppose our encryption key is: 4 2 9 1 3 6 2 8 5 6

which means every letter in a word is shifted across the alphabet +4, +2, +9, +1, and so on, places.

For example, here is the message COMPUTER SCIENCE IS EXCITING (plaintext on the top line)

C	O	M	P	U	T	E	R	S	C	I	E	N	C	E	I	S	E	X	C	I	T	I	N	G
4	2	9	1	3	6	2	8	5	6	4	2	9	1	3	6	2	8	5	6	4	2	9	1	3
G	Q	V	Q	X	Z	G	Z	X	I	M	G	W	D	H	O	U	M	C	I	M	V	R	O	J

After applying encryption key, the **Ciphertext** in the bottom line becomes meaningless. If symmetric key encryption is used, there needs to be a secure method for the sender and receiver to be provided with the secret key

ASYMMETRIC ENCRYPTION

Using **asymmetric** key encryption, the process actually starts with the **receiver**.
The **receiver** must be in **possession of Pair of keys**.

One is a **public key** which is not secret. The other is a **private key** which is secret and **known only to the receiver**.

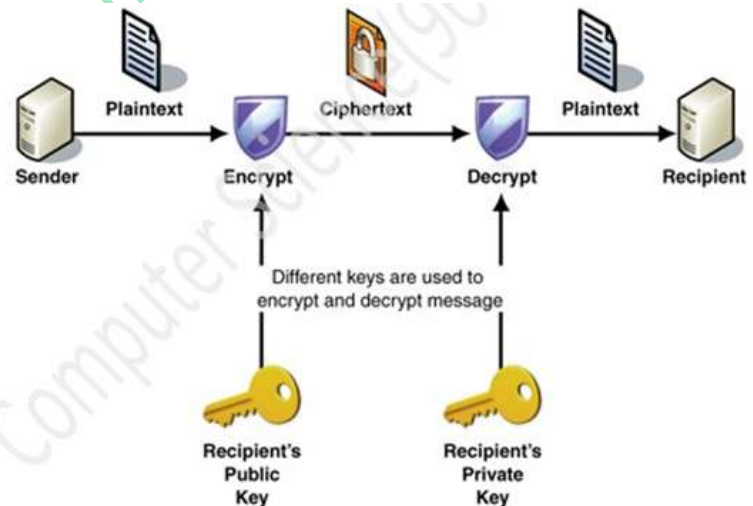
The **receiver** can send the **public key** to a **sender**, who uses the public key for encryption and sends the **cipher text** to the receiver

The **receiver** is the only person who can decrypt the message because the private and public keys are a matched pair.

The public key can be provided to any number of different people allowing the receiver to receive a private message from any of them.

Note, however, that if two individuals require **two-way communication**, both communicators need a private key and must send the matching public key to the other person.

There are two requirements to ensure confidentiality should the transmission be intercepted and the message extracted: the encryption algorithm must be complex and number of bits used to define key must be large.



Reference:

Hodder education Book by David Watson and Hellen Williams

<https://www.computerscience.gcse.guru/theory/data-packets-and-packet-switching>

<https://www.savemyexams.co.uk/igcse/computer-science/cie/23/revision-notes/2-data-transmission/2-1-types-and-methods-of-data-transmission/data-packets/>

https://en.wikipedia.org/wiki/Public-key_cryptography