Exchanging Data KO

Why is Compression needed in Data Transfer?

- **Compression** is crucial for **reducing file size** to facilitate **efficient data transfer** over the Internet
- Smaller files have faster **transmission times** and require less **bandwidth consumption**

Comparing Types of Compression

In simplified terms,

- **Lossy**: Some data is discarded
- **Lossless**: Ensures that nothing is permanently lost and everything can be restored to its original state

Compression	Benefits	Drawbacks
Lossy	• Greatly **reduced** file sizes • Suitable for **media streaming** where some data loss is acceptable	• **Irreversible loss** of data quality • Not suitable for text or **archival storage**
Lossless	• Maintains **original data** • Best for text and data that require **integrity**	• **Larger** file sizes than lossy • Requires high bandwidth when streaming

Recommending a type of Compression

To recommend a type of compression, **evaluate the application's requirements**:

- If **data integrity** is imperative, recommend **lossless compression**
- If achieving a **small file size** or **quick data transfer** is prioritised, and some data loss is acceptable, choose **lossy compression**

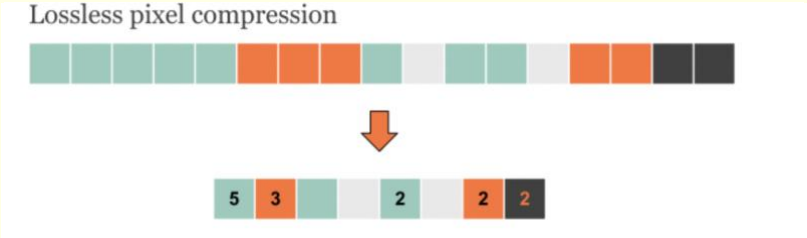
Examiner Tips and Tricks

When asked to recommend a compression type, the **choice will always** depend on the specific **scenario requirements**

Run Length Encoding & Dictionary Coding

What is run length encoding?

- In A Level Computer Science, run length encoding (RLE) is **a form of data compression** that **condenses identical elements** into a single value with a count.
- For a text file, "AAAABBBCCDAA" is compressed to "4A3B2C1D2A"
- The string has four 'A's, followed by three 'B's, two 'C's, one 'D', and two 'A's. RLE shows this as "4A3B2C1D2A"
- It is used in bitmap images to compress sequences of the same colour
- For example, a line in an image with 5 red pixels followed by 3 blue pixels could be represented as "5R3B"



What is dictionary coding?

- Dictionary coding replaces recurring sequences with shorter, unique codes
- A 'dictionary' is compiled to map original sequences to special codes
- This method is effective for both **text** and **binary data**

Example of dictionary coding

- Consider this sentence where some algorithm names are **repeatedly** mentioned:
  - QuickSort is faster than BubbleSort but MergeSort is more stable than QuickSort and BubbleSort.

- Here, the names QuickSort, BubbleSort, and MergeSort are repeated.
- Create a dictionary:
  - Start with an empty dictionary.
  - Scan the sentence for recurring sequences.
  - The dictionary might look like this:

{
QuickSort: Q
BubbleSort: B
MergeSort: M
}
	The **repetitive** words in the sentence can be replaced with the dictionary values:
	
	Original: QuickSort is faster than BubbleSort but MergeSort is more stable than QuickSort and BubbleSort
	
	Compressed: Q is faster than B but M is more stable than Q and B.
	
	The original string was **95 characters long**, and the dictionary-coded example is **53 characters long.** The shorter string will **require less space** in memory or storage

Examiner Tips and Tricks

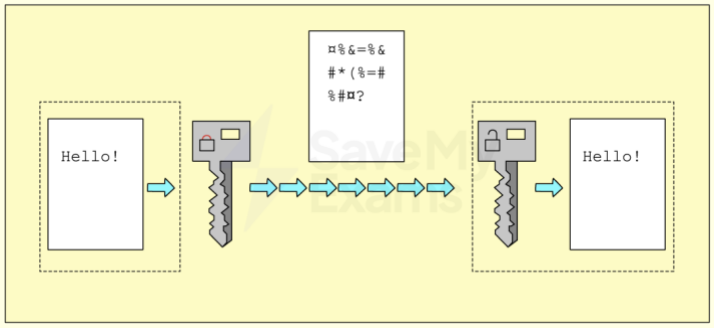
- RLE and Dictionary Coding serve different needs and have their advantages and disadvantages
- **RLE**: More effective when data has lots of repetition
- **Dictionary Coding**: More versatile but may require more computational resources

Symmetric & Asymmetric Encryption
What is encryption?

- In A Level Computer Science, encryption is crucial for converting **readable data** into an **unreadable format**
- Its primary aim is to **secure** data from **unauthorised access,** making it a highly important technique for defending against cyber-attacks and data breaches
- Encryption methods use **'keys'**, which are specialised programs designed to **scramble** or **unscramble** data
- Selecting a type of encryption isn't a daily choice for most people
    - Modern devices and technologies, e.g. web browsers (HTTPS protocol), provide a basic level of encryption by default
    - Most people transfer sensitive data **without thinking about it** because of developments in technology

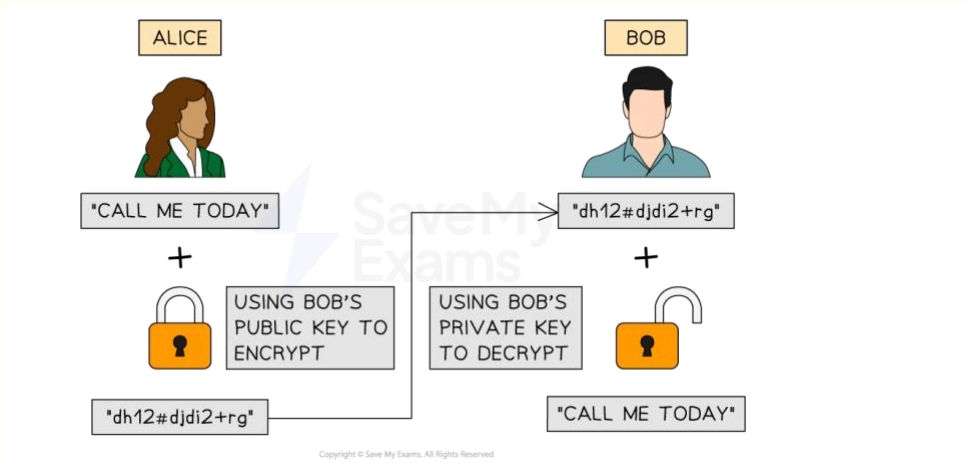
How does symmetric encryption work?

- The sender uses a key to **encrypt** the data before transmission
- The receiver **uses the same key** to **decrypt** the data
- It's usually **faster**, making it ideal for encrypting **large amounts of data**
- The significant downside is **the challenge of securely sharing this key** between the sender and receiver
- If a **bad actor** captures the key, they can **decrypt all messages intercepted** in transmission



How does asymmetric encryption work?

- Asymmetric encryption uses **two keys**:
  - a **public key** for encryption
  - and a **private key** for decryption
- **Receivers** openly share their **public key**
- **Senders** use this public key to **encrypt** the data
- The **receiver's private key** is the **only key** that can decrypt the data and is kept locally on their side
- The public and private keys are created at the same time and are **designed to work together** in this way
- It is typically **slower than symmetric encryption**
- It is generally used for **more secure and smaller** data transactions, e.g. **passwords**, **bank details**



Choosing an encryption type

- **Symmetric encryption is fast** but has key-sharing issues; **asymmetric is slower** but solves these issues.
- The choice should be made based on the **situation's needs**: whether **speed** or **security** is more critical.

Encryption Type	Suitable For	Reasons to choose
Symmetric	Large files, databases	• Fast and efficient for bulk data. • The same person encrypts and decrypts, e.g. when backing up data.
Asymmetric	Confidential/secret communications	• Sharing highly secure data, e.g. passwords, government communications

Hashing

What is hashing?

- In A Level Computer Science, hashing is a **method to convert any data** into a **fixed-size string of characters**
- This fixed-size output is often called a **digest**

Same input will always produce the same hash, providing **consistency**

Even a minor change in input produces a radically different hash, giving it **sensitivity to data changes.**

Input Text	Hashing Algorithm	Truncated Hash Digest
"hello123"	SHA-256	8d9389d5a0375bd6b028bc0368003333
"hello124"	SHA-256	9ac12bac3a0843a1917b1c4a0f77a76d
"applepie"	SHA-256	6395fc6a2522f7a27f5bdc7e31026fb6
"bpplepie"	SHA-256	af2c27fca1755bf0f7ff55a51a166ed5
"password1"	SHA-256	e99a18c428cb38d5f260853678922e03
"password2"	SHA-256	ad0234829205b9033196ba818f7a872b

Some common **hashing algorithms** are:

- **MD5 (Message Digest 5)**
  - Widely used but considered weak due to **vulnerabilities to collision attacks**
- **SHA-1 (Secure Hash Algorithm 1)**
  - Previously used in **SSL certificates** and **software repositories**, now considered weak due to vulnerabilities
- **SHA-256 (Part of the SHA-2 family)**
  - Commonly used in **cryptographic applications** and **data integrity checks**. Considered secure for most practical purposes
- **SHA-3**
  - The most recent member of the **Secure Hash Algorithm family**, designed to provide higher levels of **security**

-

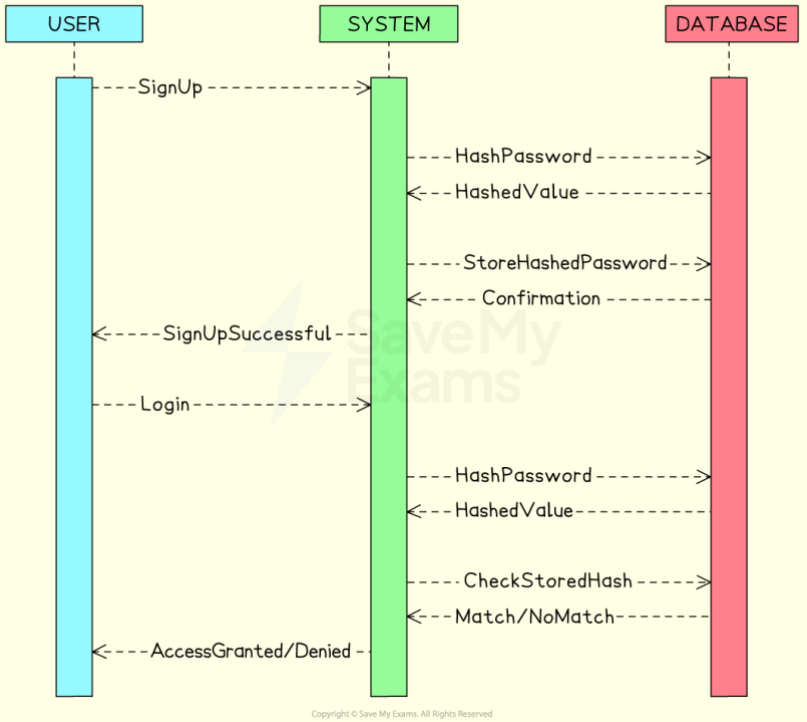
Comparison of encryption and hashing

Hashing and encryption both turn readable data into an unreadable format, but the two technologies have different purposes.

	Encryption	Hashing
Purpose	Securing data for transmission or storage; reversible	Data verification, quick data retrieval, irreversible
Reversibility	Can be decrypted to the original data	It cannot be reversed to the original data
Keys	Uses keys for encryption and decryption	No keys involved
Processing Speed	Generally slower for strong encryption methods	Generally faster
Use Cases	Secure communications, file storage	Password storage, data integrity checks
Algorithm Types	Symmetric, Asymmetric	MD5, SHA-1, SHA-256, etc.
Security	Varies; potentially strong but dependent on key management	One-way function makes it secure but susceptible to collisions
Data Length	Output length varies; could be same or longer than input	Fixed length output
Change in Output	Small change in input results in significantly different output	Small change in input results in significantly different output
Typical Operations	Encrypt, Decrypt	Hash, Verify

Hashing for password storage

- **Hashing** is commonly used for **storing passwords**
- When the user first signs up, the password they provide is **hashed**
- The hashed password is stored in the database, rather than as plaintext
- When users try to log in, they enter their username and password
- The system **hashes the password entered by the user during the login** attempt
- The hashed password is **compared against the stored hash** in the database
- If the hashes match, the user is **authenticated** and granted access
- If they don't match, access is denied



- **Hashing passwords** adds an extra layer of security
- Even if the **database** is compromised, the attacker can't use the **hashed passwords** directly
- In case of a **data breach**, not storing passwords in plaintext minimises the risk and potential legal repercussions
- Users' **raw passwords** are not exposed, reducing the impact of a **data breach**
- Since the **hash function** always produces the same output for the same input, verifying a user's password is quick

Why is hashing an efficient method for data retrieval?

Database lookup

- A good **hash function** uniformly distributes keys across the hash table, allowing for a more balanced and efficient data retrieval
- In the example below, the hashed Users table for a website is shown
    - The hashed table has **no order**
    - New users are randomly inserted into the hash table, which leads to a uniform distribution
    - If the website application needs to fetch the user's data from the table, it is computationally more **efficient** to **query using the hash digest** value than any other attribute
    - This is because hash digests have a fixed length, making it **easier** for the computer to compare hash digests **rather than variable-length strings** like email addresses
    -

Hash Table Index	0	1	2	3	4
Hash Digest	ab1c2d	ef8g9h	i1j2k3	l4m5n6	o7p8q9
Email Address	Aarav@	Mei@	Sven@	Fatima@	Tariq@
Sign-Up Date	8/8/22	11/11/22	2/2/22	6/6/22	5/5/22

The **hash digest** serves as a summarised representation of the data (email address in the above example)

Data integrity

- Another **benefit** of hashing data is being able to verify its integrity
- When data is being **transferred** over a network, it is susceptible to **loss of packets** or **malicious** i**nterference,** so if two hashes are compared and are identical, it allows a system to verify the **integrity of data**
- This is because the **same data** hashed by the **same hashing function** will produce the **same digest**
- Comparing two fixed-size hashes is **computationally less intensive** than string comparison