OCR A Level Computer Science

Your notes

3.1 Compression, Encryption & Hashing

Contents

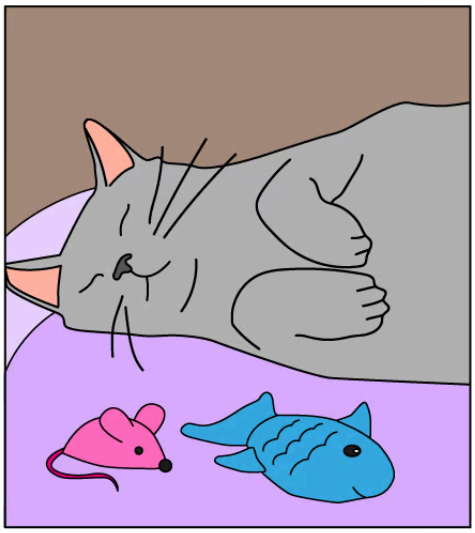
Your notes

Lossy & Lossless Compression

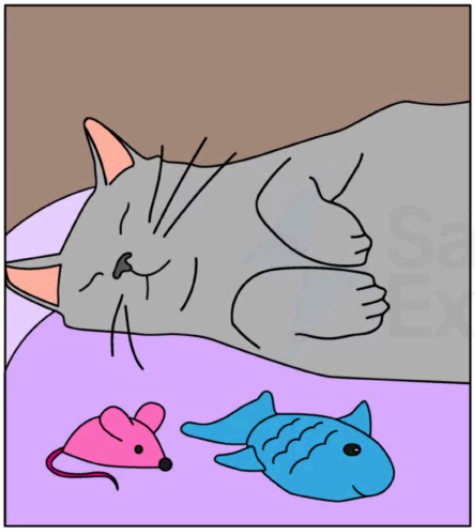
Why is Compression needed in Data Transfer?

- **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**

Your notes

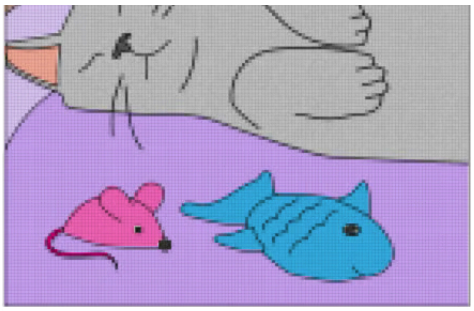


ORIGINAL
8mb FILE SIZE
2Mbit UPLOAD SPEED
32 SECONDS UPLOAD TIME



LOSSLESS
2.4mb FILE SIZE
2Mbit UPLOAD SPEED
9 SECONDS UPLOAD TIME

LOSSY
706kb FILE SIZE
2Mbit UPLOAD SPEED
2 SECONDS UPLOAD TIME



Your notes

Comparing Types of Compression

- There are two main methods of shrinking digital files: **Lossy** and **Lossless** compression
- In simplified terms,
  - **Lossy**: Some data is discarded
  - **Lossless**: Keeps everything, no data lost
- Thanks to **Moore's Law**, technology such as cameras continue to improve each year and are capable of **capturing more data**
- A Standard Definition film doesn't contain as much detail as a 4K film, because it fundamentally contains less data in the file
- New advancements in camera technology make it possible to capture more colour, contrast and audio data

Below is a table highlighting the benefits and drawbacks of Lossy and Lossless compression:

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**

Your notes

EXAMINER TIP

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

WORKED EXAMPLE

A graphic designer has two tasks: selecting an appropriate compression method for a high-resolution logo intended for a client's brand identity and for an image used in a social media campaign.

Recommend a compression type for each file and justify your choices.

[4]

How to answer this question:

To secure all 4 marks, you need to specify the type of compression for each file type and offer a detailed rationale for your choices. Make sure to highlight the benefits and potential drawbacks of each situation.

- **High-Resolution Logo**: For this, **Lossless Compression** is the best choice. This method ensures no loss of image quality, which is crucial for professional branding. Lossless Compression is particularly advantageous when the logo is to be printed or included in high-definition media where pixel-perfect clarity is required.
- **Social Media Campaign Image**: **Lossy Compression** is preferable in this scenario. This compression type drastically reduces file sizes, making the image easier to upload and quicker to load on social media platforms. Although there may be a slight loss in quality, the trade-off is generally acceptable for social media's fast-paced and transient nature.

Answer:

Example answer that gets full marks:

I recommend **Lossless Compression** for the high-resolution logo that is part of the client's brand identity. Lossless Compression preserves the image's original quality, which is key for professional presentations and branding. This choice is essential so that the logo remains high-quality for use in prints or high-definition media where every detail matters.

For the image designated for the social media campaign, **Lossy Compression** is more appropriate. This method significantly reduces file size, facilitating faster uploads and quicker loading times on

social media. While there's a slight degradation in image quality, this is a reasonable trade-off given social media platforms' temporary and fast-paced environment.

Acceptable answers you could have given instead:

Lossless Compression should be used for the high-resolution logo as it maintains data integrity, which is vital for a consistent and professional brand image.

Lossy Compression is the better choice for social media images due to its efficiency in reducing file sizes, which is beneficial for quick data transmission and loading times.

Your notes

Your notes

Run Length Encoding & Dictionary Coding

What is Run-Length Encoding?

- **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**
- The phrase "for example" could be coded as 'FE' if 'FE' doesn't appear in the original text
- A sequence of binary numbers '1010' could be replaced by a shorter unique code

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
}
```

Your notes

- Replace the sequences
  - 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 TIP

- 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

WORKED EXAMPLE

A survey focuses on the kinds of vehicles travelling on a motorway. For each vehicle that passes, a letter is noted:

- **For a car, 'C' is entered.**
- **For a motorbike, 'M' is entered.**
- **For a lorry, 'L' is entered.**
- **For any other vehicle, 'O' is entered.**

It's decided to compress the data generated.

Run Length Encoding has compressed the following sequence:

3C3M4C

Show the result of decompressing the sequence.

[2]

How to answer this question:
Run Length Encoding (RLE) shows the number of consecutive occurrences of a character in a sequence. To decompress, repeat each character the number of times indicated by the preceding number.

Answer:

Example answer that gets full marks:
CCCMMMCCCC

Your notes

WORKED EXAMPLE

The following sequence represents the raw data collected during the survey:
CCCCOLLLCCCCCMOCCCCC

Show the result of compressing the sequence.

[2]

How to answer this question:
To compress using RLE, count consecutive occurrences of each character and append the count before the character itself.

Answer:

Example answer that gets full marks:
4C1O3L5C1M1O5C

WORKED EXAMPLE

Write pseudocode for the function "longest", which takes in a string of characters as an argument and returns an integer representing the longest continuous sequence of 'C's.

[6]

How to answer this question
Write pseudocode for a function that iterates through the string, counting continuous sequences of 'C's and keeping track of the longest such sequence.

For example, in ABCCCBAACC the longest consecutive string of 'C's is 3.

Answer:

Example answer that gets full marks

How this answer might look in pseudocode:

```
Function longest(s: String) -> Integer:
  max_count = 0
  current_count = 0

  For each character in s:
    If char equals 'C':
      Increment current_count by 1
```

```
        If current_count > max_count:
            Set max_count to current_count
        Else:
            Set current_count to 0

    Return max_count
```

How this answer might look in Python:

```
def longest(s):
    max_count = 0
    current_count = 0

    for char in s:
        if char == 'C':
            current_count += 1
        else:
            current_count = 0
            if current_count > max_count:
                max_count = current_count

    return max_count
```

How this answer might look in Java:

```
public class Main {
    public static void main(String[] args) {
        System.out.println(longest("ABCACCCCCABAC"));  // Output should be 5
    }

    public static int longest(String s) {
        int maxCount = 0;
        int currentCount = 0;

        for (int i = 0; i < s.length(); i++) {
            char currentChar = s.charAt(i);

            if (currentChar == 'C') {
                currentCount++;
                if (currentCount > maxCount) {
                    maxCount = currentCount;
                }
            } else {
                currentCount = 0;
            }
        }

        return maxCount;
    }
}
```

Your notes

Acceptable answers you could have given instead:
You could use different variable names or loop structures, but the logic should be the same to get full marks.


Your notes

Symmetric vs Asymmetric Encryption

Your notes

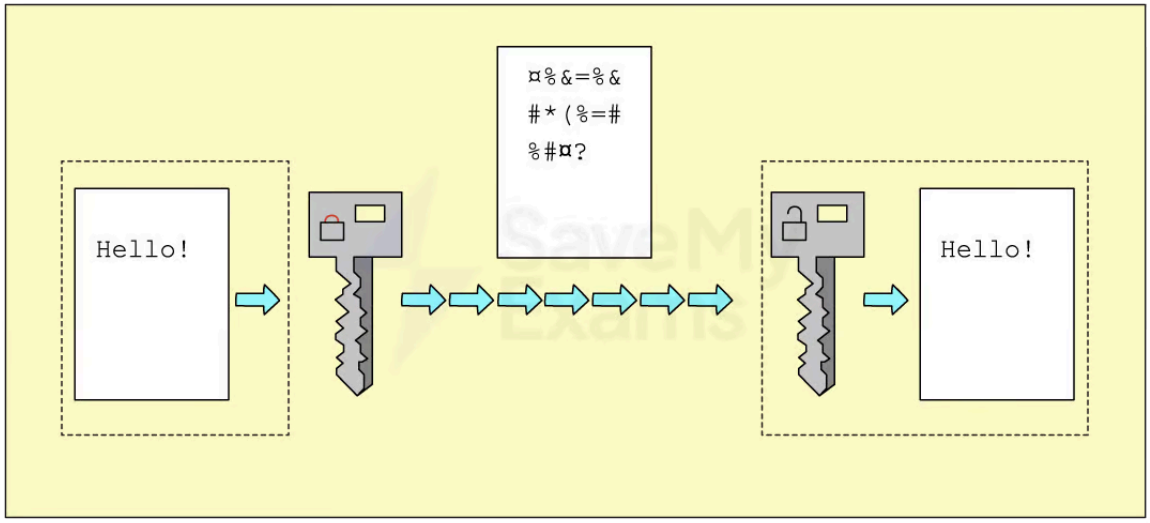
Symmetric & Asymmetric Encryption

What is Encryption?

- 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



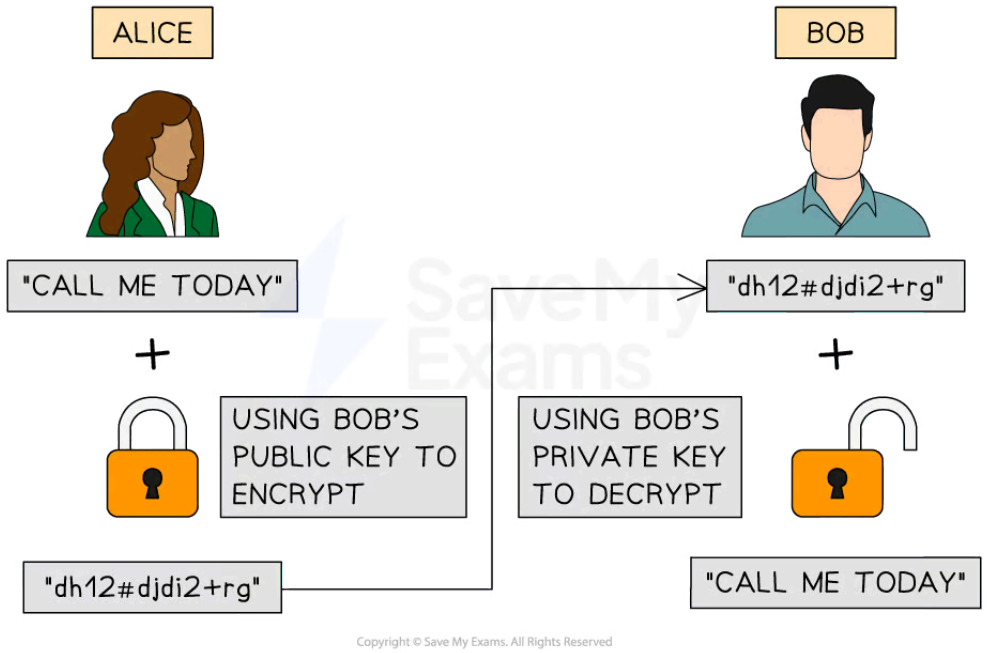
Structure of Symmetric Encryption

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**

Your notes

Your notes



Structure of Asymmetric Encryption

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

WORKED EXAMPLE

Discuss the impact of modern encryption on society. You should refer to:
• The importance of asymmetric encryption and how it differs from symmetric encryption.
• Different circumstances in which symmetric and asymmetric encryption may be used.

[9]

Your notes

How to Answer This Question:

Structuring your answer carefully is the best approach for gaining the most marks for the question. Use the list below as a guide:

1. Start with an introduction outlining the importance of modern encryption in society. Consider some scenarios where private data is being transferred, such as at school or the workplace.
2. Discuss the **key features of asymmetric encryption**, emphasising its significance and how it **resolves key distribution issues**.
3. **Compare asymmetric and symmetric encryption**, pointing out the differences and mentioning the single key used in the latter.
4. Provide **examples of scenarios** where symmetric and asymmetric encryption are most suitable, discussing the pros and cons of each.
5. Conclude by **summarising the overall impact of both types** of encryption on society, including data security, online transactions, and individual privacy.

Answer:

Example Answer That Gets Full Marks:

Encryption is critical to protect people from data misuse in today's digital society. It's a fundamental aspect of secret communications, e-commerce, and the protection of personal information.

Asymmetric encryption uses a pair of keys: a public key for encryption and a private key for decryption. This solves a crucial issue in key distribution, as the public key can be openly shared without compromising the secure private key. Asymmetric encryption forms the backbone of secure online transactions and communications, enabling features like digital signatures and secure key exchange.

Unlike asymmetric encryption, symmetric encryption uses a single key for both encryption and decryption. While it's usually faster and more efficient for handling large data, it poses a challenge in securely sharing the key between parties. Asymmetric encryption, though generally slower, alleviates this issue by using a public-private key pair, adding an extra layer of security.

Symmetric encryption is ideal for quick scenarios, such as encrypting large files or databases within a secure network where key distribution is not an issue. Asymmetric encryption is preferred in situations demanding high security, like secure email communications or online banking, where a compromise in key distribution can lead to significant risks.

Using symmetric and asymmetric encryption profoundly impacts society, enabling secure data transmission, enhancing online commerce, and protecting individual privacy. However, the consequences of providing individuals with highly secure methods of communication mean that regular users can communicate for nefarious reasons, such as organised crime. For situations where

an individual must choose between the two methods, they should consider if the data transfer needs to happen quickly or securely.


Your notes

Hashing

Your notes

Hashing

What is Hashing?

- **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**


Your notes

Comparison of Encryption and Hashing

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

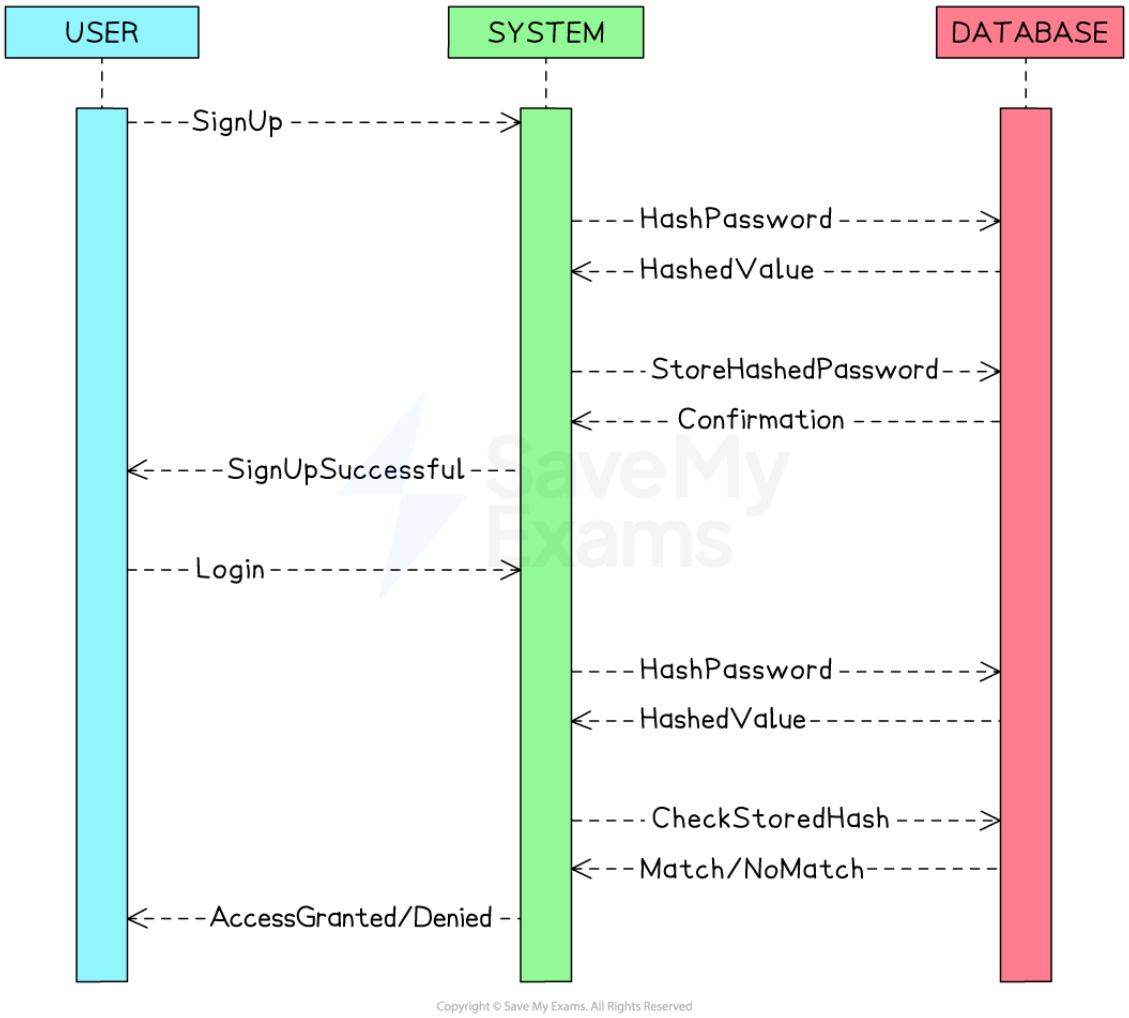
	Encryption	Hashing
Purpose	Securing data for transmission or storage; reversible	Data verification, quick data retrieval, irreversible
Reversibility	Can be decrypted to the original data	It cannot be reversed to the original data
Keys	Uses keys for encryption and decryption	No keys involved
Processing 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


Your notes

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



Your notes

- **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


Your notes

- 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 interference,** 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

WORKED EXAMPLE

A developer is designing a network security system. She is developing a component that logs websites users can access. This software records the websites' URLs and details about the allowed users and their access times.

Your notes

For each website, the following details are captured:

- **Required user rank (A-D)**
- **If it's accessible 24/7 (true) or only during breaks and outside office hours (false)**

For instance, a website that can be accessed by users of rank B and higher throughout the day would have the data [B, true] associated with it.

A site that ranks C and above users but only outside office hours would be recorded as [C, false].

Identify an appropriate data structure to keep the details of a single website.

[1]

Answer:

Answer that gets full marks:

Hash table or tuple.

WORKED EXAMPLE

Every website's URL, along with its corresponding data, is saved in a hash map.

The hash function of this map processes the website's URL (serving as the key). The hashing procedure is as follows:

1. **Remove characters up to and inclusive of the first dot.**
2. **Eliminate characters from and after the next dot present.**
3. **Convert the remaining string to uppercase.**
4. **Sum up the ASCII values of the characters.**

Given the ASCII values for the letters:

Letter	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
Value	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

As an example, consider the URL www.exam.net. The hashing process would be:

Step 1: exam.net

Step 2: exam

Step 3: EXAM

Step 4: 69 + 88 + 65 + 77 = 299

This results in a hashed value of 299.

State what hashed value would be given by the website www.foo.co.uk

[1]

Your notes

How to answer this question:

Follow the same steps as described in the question.

1. Remove characters up to and inclusive of the first dot.

Result: foo.co.uk

2. Eliminate characters from and after the next dot present.

Result: foo

3. Convert the remaining string to uppercase.

Result: FOO

4. Sum up the ASCII values of the characters.

- F: 70
- O: 79
- O: 79

Sum: 70 + 79 + 79 = 228

Answer:

Answer that gets full marks:

The sum of ascii values for www.foo.co.uk is **228.**

WORKED EXAMPLE

Complete the function hash**, which takes in a string and returns the hashed value.**

You can assume you have access to the following three functions:

- **asc() – this takes in a character and returns its ASCII value. For example,** asc("A") **returns 65.**
- **locate() – this takes in a string and character and returns the location of the first instance of the character (with the string starting at character 0). For example,** locate("electricity","c") **returns 3.**
- **upper() – this takes in a string and returns the UPPERCASE version. For example** upper("hello") **returns "HELLO".**

You should also assume that all website names use letters but no numbers or symbols.

```
function hash(siteName)

endfunction
```

[5]

Your notes

How to answer this question:

1. Understand the requirements:
    - You are to create a hash function.
    - Make use of the provided functions: asc(), locate(), and upper().
    - Implement the hash function as described in the earlier content.
2. Implement the steps:
    - Discard characters up to and including the first dot.
    - Discard characters including and to the right of the remaining leftmost dot.
    - Convert the characters to uppercase.
    - Add the ASCII values of the characters together.

Answer:

Answer that gets full marks:

```
function hash(siteName)
  firstDot = locate(siteName, ".")
  remainingString = siteName[firstDot+1:]

  secondDot = locate(remainingString, ".")
  requiredString = upper(remainingString[:secondDot])

  sum = 0
  for char in requiredString
    sum += asc(char)
  endfor

  return sum
endfunction
```

Acceptable answers you could have given instead:

```
function hash(siteName)
  parts = siteName.split(".")
  if len(parts) > 2:
    keyString = upper(parts[1])
  else:
    return -1  // handle error or unexpected input

  sum = 0
  for char in keyString
    sum += asc(char)
  endfor
```

```
    return sum
endfunction
```

Your notes