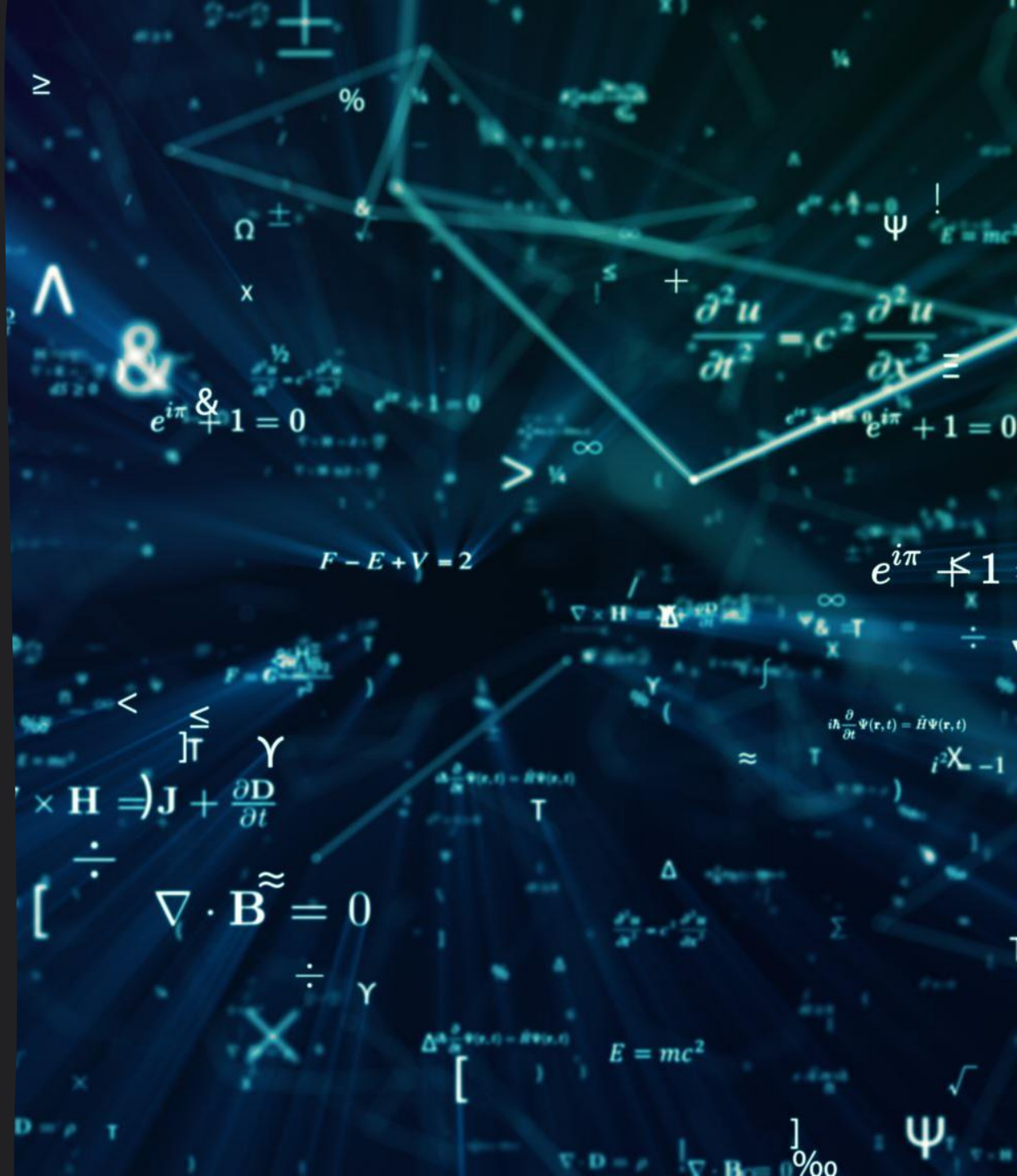
Hashing & Hashing Algorithms

Task no.1



What is hashing?

Hashing is the process of transforming any given key or a string of characters into another value. This is usually represented by a shorter, fixed-length value or key that represents and makes it easier to find or employ the original string.

Hashing Characteristics

1. **Deterministic:**
   - The same input will always produce the same hash value. This is important in verification and comparison purposes.
2. **Fixed Output Length:**
   - Regardless of the input size, the hash function will always generate a fixed-length output. This is handy for creating consistency in data representation.
3. **Efficient:**
   - Hashing is designed to be fast and efficient. This is important in applications like encryption, where speed is a critical factor.

Common Use Cases:

Data Integrity:

Hashing is used to verify the integrity of data. By comparing hash values before and after transmission, one can ensure that the data has not been altered.

Password Storage:

Hash functions play a crucial role in securing passwords. Instead of storing actual passwords, systems store the hash values, adding an extra layer of security.

File Deduplication:

In systems where duplicate files are to be avoided, hashing is used to identify identical files. If two files have the same hash value, they are considered duplicates.

Hashing in Cryptography & Cyber Security

Hashing serves various critical purposes in encryption and cybersecurity, contributing to the overall security of digital systems. It’s used in many fields related to Cyber Security like:

Malware Detection

- Hashing is used in cybersecurity for malware detection.
- Antivirus programs often use hash databases to compare the hash values of files on a system against known malicious files. If a match is found, it can indicate the presence of malware.

Cryptographic Hash Functions in Protocols

- Many cryptographic protocols like SSH and HTTPS use hash functions to ensure security.

- For example, secure key exchange protocols like Diffie-Hellman key exchange (*For two parties to communicate confidentially, they must first exchange the secret key so that each party is able to encrypt messages before sending, and decrypt received ones. This process is known as the key exchange.*) may use hashes to verify the integrity of exchanged messages and prevent man-in-the-middle attacks.

Blockchain Technology

- Blockchain relies heavily on hashing for creating secure blocks.
- Each block in a blockchain contains a hash of the previous block, forming a chain.

Most Popular Hashing Algorithms

MD5

Argon2

Bcrypt

(Secure Hash Algorithm) SHA Family:

- SHA-256
- SHA-1
- SHA-3

Tiger

MD5

MD5 (Message Digest 5 Algorithm) is a cryptographic hash algorithm used to generate a 128-bit hash from a string of any length. It represents the hash as 32 digit **hexadecimal** (4bit each) numbers.

Ronald Rivest (Also contributed in RSA algorithm) designed this algorithm in 1991 to provide the means for digital signature verification. Eventually, it was integrated into multiple programming languages and frameworks.

Use Of MD5 Algorithm:

- It is used for file authentication.
- In a web application, it is used for security purposes. e.g. Secure password of users etc.
- Using this algorithm, We can store our password in 128 bits format.

Steps in MD5 Algorithm

There are four major sections of the algorithm:

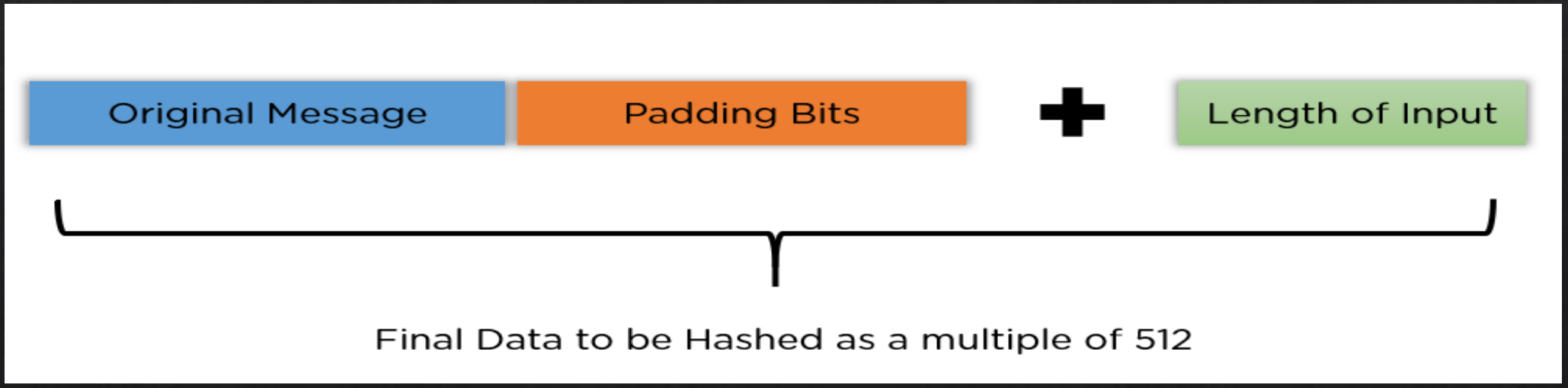
1. Padding Bits
2. Padding Length
3. Initialize MD Buffer
4. Process Each Block

1. Padding Bits

When you receive the input string, you have to make sure the size is 64 bits short of a multiple of 512. When it comes to padding the bits, you must add one(1) first, followed by zeroes to round out the extra characters.

2. Padding Length

You need to add a few more characters to make your final string a multiple of 512. To do so, take the length of the initial input and express it in the form of 64 bits. On combining the two, the final string is ready to be hashed.



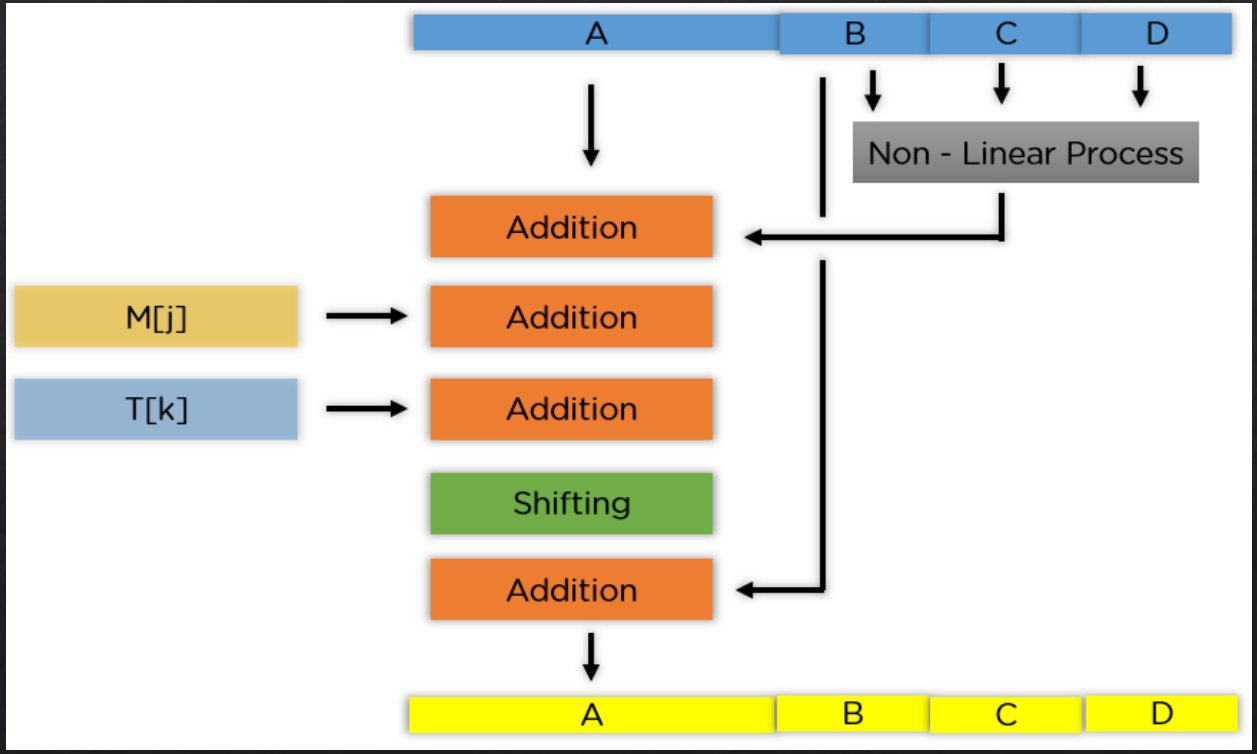
3. Initialize MD Buffer

The entire string is converted into multiple blocks of 512 bits each. You also need to initialize four different buffers, namely A, B, C, and D. These buffers are 32 bits each and are initialized as follows:

$$A = 01\ 23\ 45\ 67$$
$$B = 89\ ab\ cd\ ef$$
$$C = fe\ dc\ ba\ 98$$
$$D = 76\ 54\ 32\ 10$$

4. Process Each Block

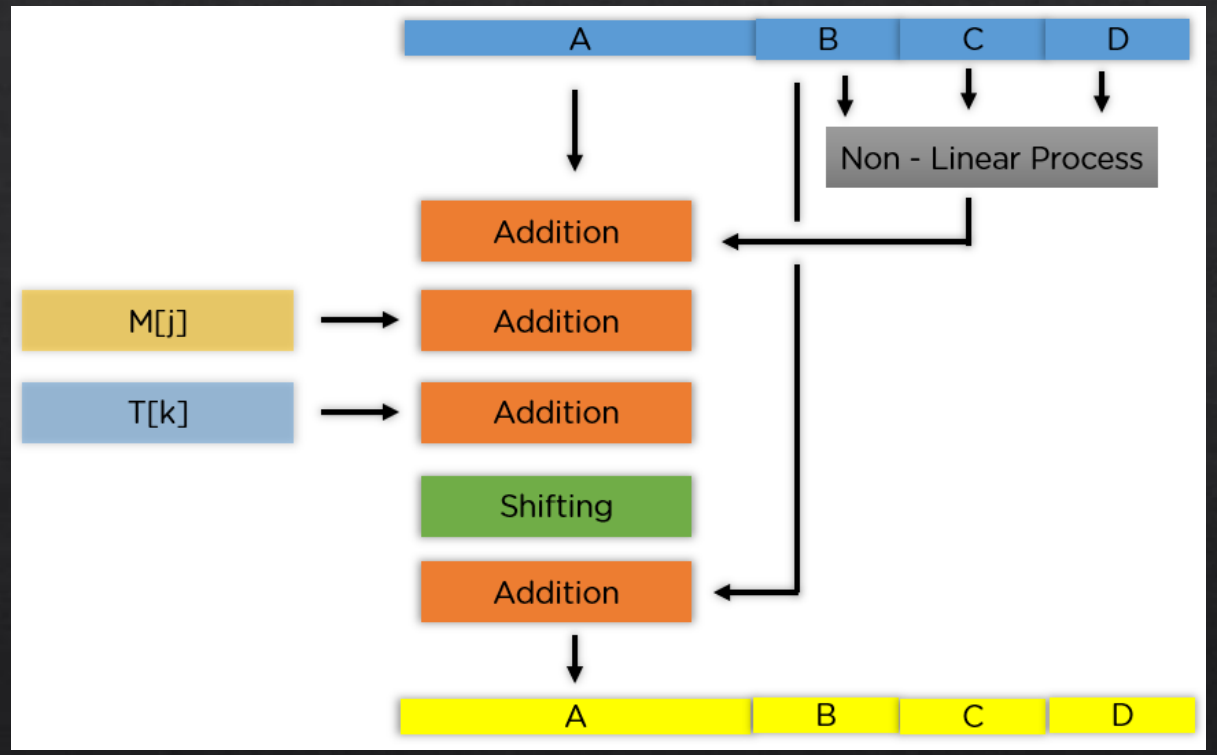
- Each 512-bit block gets broken down further into 16 sub-blocks of 32 bits each. There are four rounds of operations, with each round utilizing all the sub-blocks, the buffers, and a constant array value.
- This constant array can be denoted as $T[1] \rightarrow T[64]$.
- Each of the sub-blocks are denoted as $M[0] \rightarrow M[15]$.



According to the image, you see the values being run for a single buffer A.

The correct order is as follows:

1. It passes B, C, and D onto a non-linear process.
2. The result is added with the value present at A.
3. It adds the sub-block value to the result above.
4. Then, it adds the constant value for that particular iteration.
5. There is a circular shift applied to the string.
6. As a final step, it adds the value of B to the string and is stored in buffer A.



The steps mentioned before are run for every buffer and every sub-block. When the last block's final buffer is complete, you will receive the MD5 digest.

The non-linear process above is different for each round of the sub-block.

Round 1: $(b\ AND\ c)\ OR\ ((NOT\ b)\ AND\ (d))$

Round 2: $(b\ AND\ d)\ OR\ (c\ AND\ (NOT\ d))$

Round 3: $b\ XOR\ c\ XOR\ d$

Round 4: $c\ XOR\ (b\ OR\ (NOT\ d))$

With this, you conclude the working of the MD5 algorithm. You will now see the advantages procured when using this particular hash algorithm.

Using MD5 on Windows OS Via PowerShell:

Argon2

Argon2 is a key derivation function (KDF) and password hashing algorithm that was selected as the winner of the Password Hashing Competition (PHC) in 2015. It is designed to securely hash passwords and other sensitive information, making it difficult for attackers to reverse-engineer the original input from the hash value. Argon2 is known for its resistance against various cryptographic attacks, including brute-force and time-memory trade-off attacks. It is considered to be highly secure and is widely used for securely storing passwords and protecting sensitive data in software applications.

How does argon2 work ?

1. **Initialization**: Argon2 takes several parameters as input, salt and so on. These parameters influence how much memory and time the algorithm will consume.
2. **Memory-Hardness**: Argon2 uses a large, configurable amount of memory.
3. **Data-Dependent Operations**: Argon2 operates on the input data (password and salt) , meaning the operations performed depend on the actual data being processed.
4. **Parallel Processing**: Argon2 can be parallelized efficiently. The parallelism factor parameter controls how many threads can be used to hash the password.

5. **Iterations**: Argon2 applies a specified number of iterations to the data. This makes the hash calculation intentionally slow, which is a desirable feature for password hashing algorithms.
6. **Finalization**: After the specified number of iterations, Argon2 produces the final hash value. This hash value can be stored in a database for later authentication purposes.

All three modes allow specification by three parameters that control:

1. execution time
2. memory required
3. degree of parallelism

Using with:

1. PHP
2. Nodejs
3. Js

Main parameter to argon2

1. **Password**: The input password to be hashed.
2. **Salt**: A random value used to increase the security of the hash.
3. **Memory Cost**: The amount of memory (in kilobytes) that the algorithm uses.
4. **Time Cost**: The number of iterations the algorithm performs.
5. **Parallelism Factor**: The number of parallel threads or lanes used for computation.

Bcrypt

Bcrypt is a password hashing algorithm designed by Niels Provos and David Mazières based on the Blowfish cipher. The name “bcrypt” is made of two parts: b and crypt, where b stands for Blowfish and crypt is the name of the hashing function used by the Unix password system.

Bcrypt was created as a result of the failure of Crypt to adapt to technology and hardware advancement. Bcrypt is designed to be a slow algorithm, which is a good thing when it comes to password hashing. Therefore, bcrypt is perfect for password hashing because it reduces brute-force attacks.

How does bcrypt work?

- bcrypt takes a user-submitted plain password and converts it into a hash. The hash is what is stored in the database. This prevents attackers from accessing users' plain passwords in the event of a data breach. Unlike some other password-hashing algorithms that just hash the plain password, bcrypt uses the concept of salt.
- This unique randomly generated string provides an additional level of security for a generated hash. Before the plain password is hashed, a salt is generated. Then, it is appended to the plain password, and everything is hashed (the plain password and salt). This help protects against rainbow table attacks because attackers can randomly guess users' passwords, but they can't guess the salt.

- Bcrypt also uses a cost factor (or work factor) to determine how long it takes to generate a hash. This cost factor can be increased to make it slower as hardware power increases. The higher the cost factor, the more secure the hash and the slower the process. Therefore, you need to find the right balance between security and speed.

- The generated hash will include the salt and other things, like the hash algorithm identifier prefix, the cost factor, and the hash. The hashing process is irreversible. The hash cannot be converted back to the original plain password. Therefore, to determine whether a user provides the correct password, the provided password is hashed (using the original salt) and compared against the hash stored in the database.

Benefits of password hashing by bcrypt

- Bcrypt has significant advantages over other hashing methods like MD5, SHA1, SHA2, and SHA3. They can all perform hashing of a large number of data in less time. Suppose an attacker has a robust system capable of trying 700-900 million passwords in seconds. Your password containing alphanumeric and special character values will be cracked in a few seconds.
- So, all of these hashing methods cannot be used to encrypt the password.

Now, the main question is, how does bcrypt provide a significant advantage here?

- Bcrypt was built upon Blowfish keying schedule and used a work factor, which decides how expensive the hash function will be. After knowing it, bcrypt will get slower if an attacker makes multiple requests in a single time frame. So generally, cracking one password will take 12 damn years. Also, bcrypt uses salt, which helps prevent attacks like rainbow table attacks and is suitable for securing passwords.

(Secure Hash Algorithm) SHA-1

a hash algorithm used to convert data into a segment known as hash. SHA-1 was developed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) in 1995. However, the use of SHA-1 in data security construction was warned because of its vulnerability. In 2005, a study indicated that SHA-1 could be compromised in an inexpensive way. In the following years, in-depth research provided additional evidence of SHA-1's vulnerability and exposure to collisions, a condition that occurs when there are two different series of data that are re-fragmenting. Due to these security vulnerabilities, SHA-1 has been set as an unsecured and not recommended algorithm in many sensitive uses, such as confirming emails and software signatures. Instead, it is best to use stronger and safer segment algorithms like SHA-256 or SHA-3 in apps that require high security.

Complete algorithm of SHA-1:

- Step1. Divide the original text into blocks (blocks) of data size of 512 bits.
  - Input Text: MOHAMED
  - Text to ASCII: 109, 111, 104, 97, 109, 101, 100
  - ASCII to Binary: 01101101, 01101111, 01101000, 01100001, 01101101, 01100101, 01100100
- Step2. If the block is less than 512 bits, fill the difference with zero bits until you reach 512 bits.

- Step3. Increase the original length of the original text by structural representation and add it to the end of the last block. For example, if the original data contains 64 bits, the length will be constructively represented on 64 bits and add to the end of the last block.
- Step4. Divide each block into 16 32-bit words to perform operations on it.
- Step5. Expand the word list to 80 32-bit words by applying a specific function to the original words.

- Chunk 0
  - $01101101011011110110100001100001\ W[0]$
  - $01101101011001010110010010000000\ W[1]$
  - $00000000000000000000000000000000\ W[2]$
  - $00000000000000000000000000000000\ W[3]$
  - $00000000000000000000000000000000\ W[4]$
  - $00000000000000000000000000000000\ W[5]$
  - $00000000000000000000000000000000\ W[6]$
  - $00000000000000000000000000000000\ W[7]$
  - $00000000000000000000000000000000\ W[8]$
  - $00000000000000000000000000000000\ W[9]$
  - $00000000000000000000000000000000\ W[10]$
  - $00000000000000000000000000000000\ W[11]$
  - $00000000000000000000000000000000\ W[12]$
  - $00000000000000000000000000000000\ W[13]$
  - $00000000000000000000000000000000\ W[14]$
  - $00000000000000000000000000111000\ W[15]$

- Step6. Displacement of values in words based on the laws specified in the algorithm.
- Step7. Perform 80 cycles of operations on displaced words.
- Step8. Keep a steady initial value for each course and use it in all courses.

$$for\ I\ =\ 16\ to\ 79$$

$$I\ = 16$$

$$W\ [i]\ =\ w[i-3]\ XOR\ w[i-8]\ XOR\ w[i-14]\ XOR\ w[i-16]$$

$$w[16] =\ w[16-3] XOR\ w[16-8]\ XOR\ w[16-14]\ XOR\ w[16-16]$$

$$\#w[16] = w[13] XOR\ w[8]\ XOR\ w[2]\ XOR\ w[0]$$

$$\#w[16] = 00000000000000000000000000000000$$
$$00000000000000000000000000000000$$
$$00000000000000000000000000000000$$
$$01101101011011110110100001100001$$

$$\#W[16] = 01101101011011110110100001100001$$

left rotated.

$$\#W[16] = 11011010110111101101000011000010$$

- Step9. Repeat the previous process until I reach word 80.
- Step10. We arrange the80 words and divide them into 4groups, and each group contains 20words. The first group starts from ($W0 - W19$) and takes function 1, and the second group starts from ($W20 - W39$) and takes function 2, and so on for the rest of the groups.

$$A = h0 = 01100111010001010010001100000001$$
$$B = h1 = 11101111100110110101011100010001$$
$$C = h2 = 10011000101110101101110011111110$$
$$D = h3 = 00010000001100100101010001110110$$
$$E = h4 = 11000011110100101110000111110000$$

FUNCTION1:

$$f = (B\ and\ C)\ or\ (!\ B\ and\ D)$$

FUNCTION2:

$$f = B\ xor\ C\ xor\ D$$

FUNCTION3:

$$f = (B\ and\ C)\ or (B\ and\ D)\ or (C\ and\ D)$$

FUNCTION4:

$$f = B\ xor\ C\ xor\ D$$

- step11. We calculate the value of the temp variable and change the values of the variables.

$$TEMP = (A\ left\ rotated5) + F + E + K + current\ word.$$

$$E = D$$

$$D = C$$

$$C = B\ left\ rotated30$$

$$B = A$$

$$A = TEMP$$

◈ Step12. After the courses are completed, you will get a final hash value of 160 bits. This is the full algorithm of SHA-1.

$$h0 = h0 + A$$

$$h1 = h1 + B$$

$$h2 = h2 + C$$

$$h3 = h3 + D$$

$$h4 = h4 + E$$

SHA-256

1. History:
   - sha 256 algorithm belongs to the third family of sha, which I include 6 hash function in it within sha 256 .
2. Input/output length
   - it receives an input with variable length and produced fixed size hash code represent the input data, the input variable length is in range from 1 to 2 power 64 with different sizes in this range it produce the same fixed hash code size which is 256 bit represented in hexa decimal.

3. SHA256 terminologies:

Constants values:

- consist of 64 records , each record consist of 32 bits, take initial values which are the fraction parts of the cube roots of the first 64 prime numbers, represented from $k0$ ... . $k63$.

Has values:

- consist of 8 records , each one consist of 32 bits , take initial values which are the fraction parts of the roots of the first 8 prime numbers, this 32 bits represented the has value which will be produced by the sha 256, represented from a …… h .

Collection of logical operations:

- sigma 0, sigma 1, summation 0, summation 1, ch, maj

4. SHA256 processes

- convert text to binary according to ascii and combine it together
- padding the binary text
- breaking the binary text into blocks of 512 bits
  - breaking each block to 16 blocks of 32 bits
  - add 48 blocks of 32 bits to the 16 bits to make the total number of blocks are 64 blocks represented from w0 to w63

5. Hash Computation

- convert given text to binary then combine it together
- padding the result
- initialize the 8 constant has values $h0$ ….$h7$
- initialize the 64 constant values $W0$ ….$W63$
- divide the padding result to blocks of 512 bits , for each block get there 64 blocks $(0:15),(16,63)$
- for each 64 blocks
  - logical operations carry out on the values of 8 hash records using a constant record in each iteration
- Finally combine the 8 has records together and represent it into hexa the result is the hash values for the input data .

SHA-3

- SHA-3, which stands for Secure Hash Algorithm 3, is a cryptographic hash function designed by a team of researchers led by Guido Bertoni, Joan Daemen, and Gilles Van Assche. It was standardized by the National Institute of Standards and Technology (NIST) in 2015 as part of the Secure Hash Standard (SHS) publication.
- SHA-3 is part of a family of cryptographic hash functions that includes different hash sizes (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256) to cater to different security requirements.

- The primary function of SHA-3 is to take an input message and produce a fixed-size (digest) hash value, which is typically a string of characters. This hash value is unique to the input data, meaning that even a small change in the input message will result in a significantly different hash value. SHA-3 is designed to be secure against various cryptographic attacks, including collision attacks, where two different inputs produce the same hash value.
- When considering the use of SHA-3 or any cryptographic algorithm, it's crucial to evaluate your specific use case, consult with cryptography experts if necessary, and stay informed about the latest developments in the field of cryptography to ensure that your choice of algorithm aligns with your security requirements.

SHA-3 applications in computer security and cryptography:

- **Data Integrity**: SHA-3 can be used to verify the integrity of data by generating a hash value for the original data and comparing it with the hash value of the received or stored data. If the hash values match, the data has not been tampered with.
- **Digital Signatures**: Cryptographic digital signatures often involve hashing the message and then encrypting the hash value with a private key. SHA-3 can be used as the hashing component in digital signature algorithms.
- **Password Security**: SHA-3 can be used to securely store passwords. Instead of storing the actual password, systems can store the hash of the password. During authentication, the system hashes the entered password and compares it to the stored hash value.

- **Random Number Generation**: SHA-3 can be used to generate pseudo-random numbers. By hashing an input value (such as a seed), a seemingly random output can be generated. However, it's important to note that SHA-3 is not a true random number generator; it's a deterministic algorithm producing pseudorandom output.
- **Blockchain Technology**: SHA-3 is used in various blockchain implementations for creating secure hashes of blocks, ensuring the integrity and immutability of the blockchain.

Advantages of SHA-3:

- Security: SHA-3 is designed to be highly secure and resistant to various types of attacks, including collision and pre-image attacks. Its security has been extensively analyzed by the cryptographic community.
- Resistance to Cryptanalytic Attacks: SHA-3 has a strong security margin and is resistant to many types of cryptanalytic attacks. Its sponge construction provides a solid foundation for security.
- Versatility: SHA-3 supports hash lengths of 224, 256, 384, and 512 bits, making it suitable for a wide range of applications, including digital signatures, key derivation functions, and message authentication codes.
- Performance and Efficiency: While SHA-3 may not always be the fastest hash function, its performance is generally efficient for most practical purposes. Additionally, its modular and parallelizable design can be optimized for specific hardware. NIST Standard: SHA-3 is a NIST standard (FIPS PUB 202) and has undergone extensive scrutiny by the cryptographic community and experts, making it a reliable choice for security applications.

Disadvantages of SHA-3:

- Speed: While SHA-3 is efficient, it might not be the fastest option for applications where speed is a critical factor. Some other hash functions like SHA-256 might be faster in certain contexts.
- Implementation Complexity: Implementing SHA-3 from scratch can be complex, especially for developers without a deep understanding of cryptographic algorithms. Incorrect implementations can lead to vulnerabilities.
- Size of Output: For some applications, the 224-bit output size might be too large. For contexts where storage is limited, using a smaller hash function like SHA-256 might be more appropriate.
- Newness: While SHA-3 has been extensively analyzed, its status as a relatively newer algorithm (compared to SHA-2, for example) might make some conservative users and organizations hesitant to adopt it immediately.
- Potential Future Attacks: Although SHA-3 is currently secure, the landscape of cryptography is always evolving. Future advances in mathematics or computational power might theoretically weaken its security in the very long term.

Tiger

Tiger is a cryptographic hash function designed by Ross Anderson and Eli Biham in 1995. It produces a fixed-size hash value of 192 bits. Tiger is considered to be secure and efficient, and it has been used in various applications where data integrity and security are important.

Here's how Tiger works as a hash function:

- Initialization: Tiger initializes three 64-bit registers, often denoted as A, B, and C, with specific constant values. These registers are used to store intermediate hash values during the computation.
- Processing the Data in Blocks: Tiger processes the input message in blocks of 512 bits (64 bytes). If the message is not a multiple of 512 bits, padding is applied to make the message a multiple of the block size.
- Compression Function: The compression function operates on each block of the message. It involves several rounds of mixing and permutation operations, where the data is combined with the values in the registers in a non-linear manner.
- Finalization: After processing all blocks, the final values in the A, B, and C registers are concatenated to form the 192-bit hash value.

Advantages of Tiger Hash Function:

- Security: Tiger was designed with a focus on security and has undergone extensive analysis. While it might not be as well-known as SHA-2 or SHA-3, it is still considered a secure hash function.
- Efficiency: Tiger is relatively efficient in terms of computational resources. It performs well in software implementations and can be faster than some other hash functions, especially on platforms where bitwise operations are fast.
- Versatility: Tiger produces a 192-bit hash value, which is longer than MD5 (128-bit) and SHA-1 (160-bit). This longer hash can provide a higher level of collision resistance, making it suitable for applications where this property is important.
- Simple Design: Tiger has a relatively simple and straightforward design, making it easier to implement and understand compared to more complex hash functions.
- Checksum Verification: Tiger hash values can be used to verify data integrity and detect accidental changes or corruption in files or data transmission. It's commonly used in error-checking applications.

Disadvantages of Tiger Hash Function:

- Limited Adoption: Tiger is not as widely adopted as some other hash functions like SHA-2 or SHA-3. This means there might be fewer tools, libraries, and community support available compared to more commonly used hash functions.
- Non-Standardization: Tiger is not a NIST standard. While this doesn't necessarily mean it's insecure, standards compliance can be a critical factor in certain applications where interoperability and compliance are required.
- Limited Hash Length Options: Tiger produces a fixed- length hash of 192 bits. While this length is longer than MD5 and SHA-1, it might not provide the same level of security as longer hash functions like SHA-256 (256-bit) or SHA-3 (e.g., 256-bit or 512-bit versions) in certain use cases.
- Cryptographic Vulnerabilities: While no major vulnerabilities have been discovered in Tiger, it's worth noting that as technology advances, new attack methods might be found that could potentially weaken its security.