Secure Hash Algorithm(SHA)

19Z701-CRYPTOGRAPHY

22z220 Gobika R
22z258 Sandhiya R
22z267 Thamina anzum A
22z272 Vijeyasri T
22z273 Vinithaa P

Introduction to Secure Hash Algorithm (SHA)

What is cryptographic hash function ?

A mathematical algorithm that converts input data into a fixed-size hash value.

Acts as a digital fingerprint of the data.

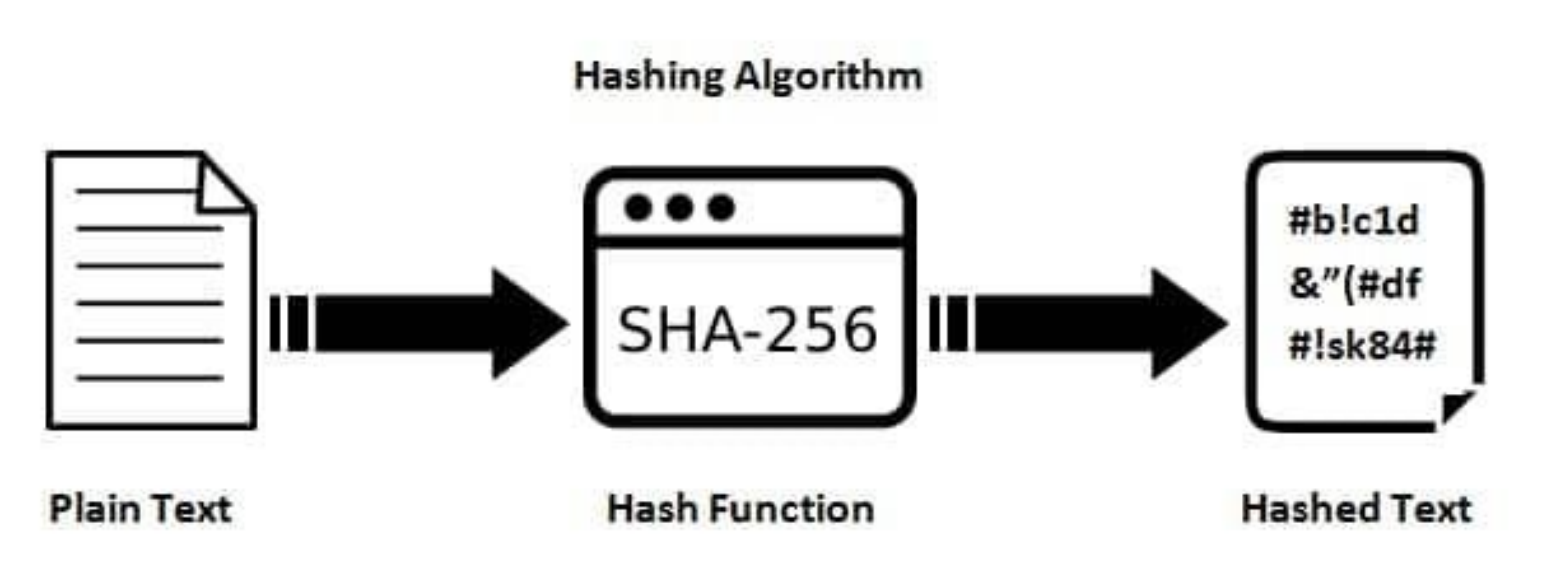
Properties:

- Deterministic – same input gives same output.
- Irreversible – cannot retrieve input from hash.
- Collision-resistant – hard to find two inputs with same hash.

What is SHA?

- SHA (Secure Hash Algorithm) is a family of cryptographic hash functions designed by the National Security Agency (NSA) and published by NIST.
- It converts any input (message) into a fixed-length hash value or digest that uniquely represents the data.
- Common versions include SHA-1, SHA-2, and SHA-3.
- SHA ensures data integrity by detecting any changes made to the original data.

What is SHA?


Hashing Algorithm
SHA-256
#b!c1d
&"(#df
#!sk84#
Plain Text
Hash Function
Hashed Text

Why SHA is important ?

- **Ensures Data Integrity:** Detects any alteration in transmitted or stored data by comparing hash values before and after transfer.

- **Supports Digital Signatures:** Generates message digests used to verify the authenticity and non-repudiation of the sender.

- **Secures Passwords:** Converts plain-text passwords into irreversible hashes, protecting user credentials from exposure.

- **Strengthens Blockchain Security:** Ensures the immutability of transaction data by linking blocks using cryptographic hashes.

- **Used in TLS/SSL and Certificates:** Validates website authenticity and establishes secure HTTPS connections.

Evolution of SHA algorithms

SHA-0 (1993):

- First version of SHA released by NIST.
- Withdrawn quickly due to security vulnerabilities.

SHA-1 (1995):

- Improved over SHA-0 with a 160-bit hash output.
- Widely used initially but now considered insecure due to collision attacks.

SHA-2 (2001):

- Introduced stronger hash variants: SHA-224, SHA-256, SHA-384, SHA-512.
- Provides robust security and is still widely used today.

SHA-3 (2015):

- Based on the Keccak algorithm, unlike SHA-1/2.
- Offers a new internal structure and additional security features.

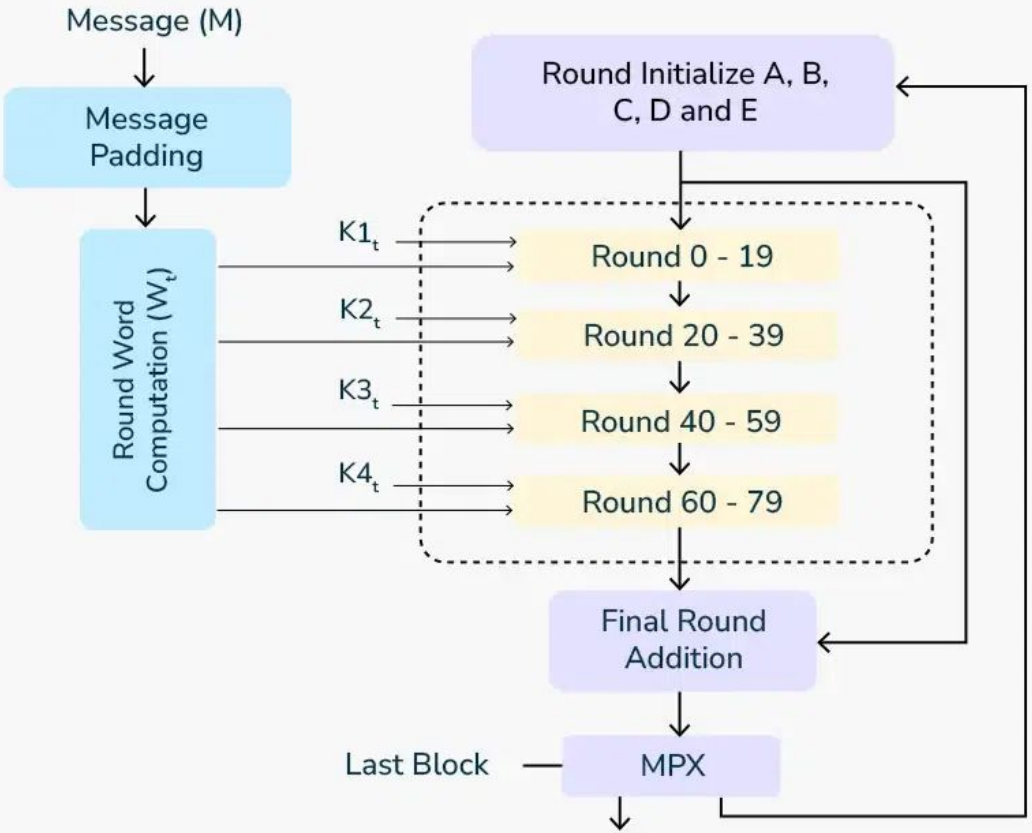
SHA – 1

Introduction to SHA-1:

- SHA-1 is a cryptographic hash function that produces a 160-bit hash value from an input message of any size, up to $2^{64} - 1$ bits.
- SHA-1 was designed by the NSA and published by the NIST in 1995 as a part of the SHS.
- SHA-1 is a one-way function.

Working of SHA-1

SHA-1 Algorithm Block Diagram



Implementation of SHA-1:

```python
import hashlib

def sha1(message):
    # generate a SHA-1 hash object
    sha1_hash = hashlib.sha1()

    # update the hash object
    sha1_hash.update(message.encode('utf-8'))

    # hexadecimal digest of the hash
    return sha1_hash.hexdigest()

# execute the function
message = "Hello, everyone!"
sha1_hash = sha1(message)
print("SHA-1 hash:", sha1_hash)
```

Output:

```
SHA-1 hash:
```

```
9a8bfaaace7b9e422f79ef862956e3512aa9d8ea
```

Properties of SHA-1:

- Collision Resistance
- One-way Function
- Fixed Output Length

Applications of SHA-1:

- Digital Signatures
- Password Storage
- Secure Communications

Vulnerabilities of SHA-1:

- **Birthday Attack:** SHA-1 collision vulnerability allows a birthday-style collision with roughly 2^{80} work.
- **Man-in-the-Middle:** An attacker can alter intercepted messages and supply a different message with the same SHA-1 hash to evade detection.
- **Certificate Forgery:** Collision weakness lets attackers craft fraudulent certificates that match the SHA-1 hash of legitimate ones.

SHA – 2

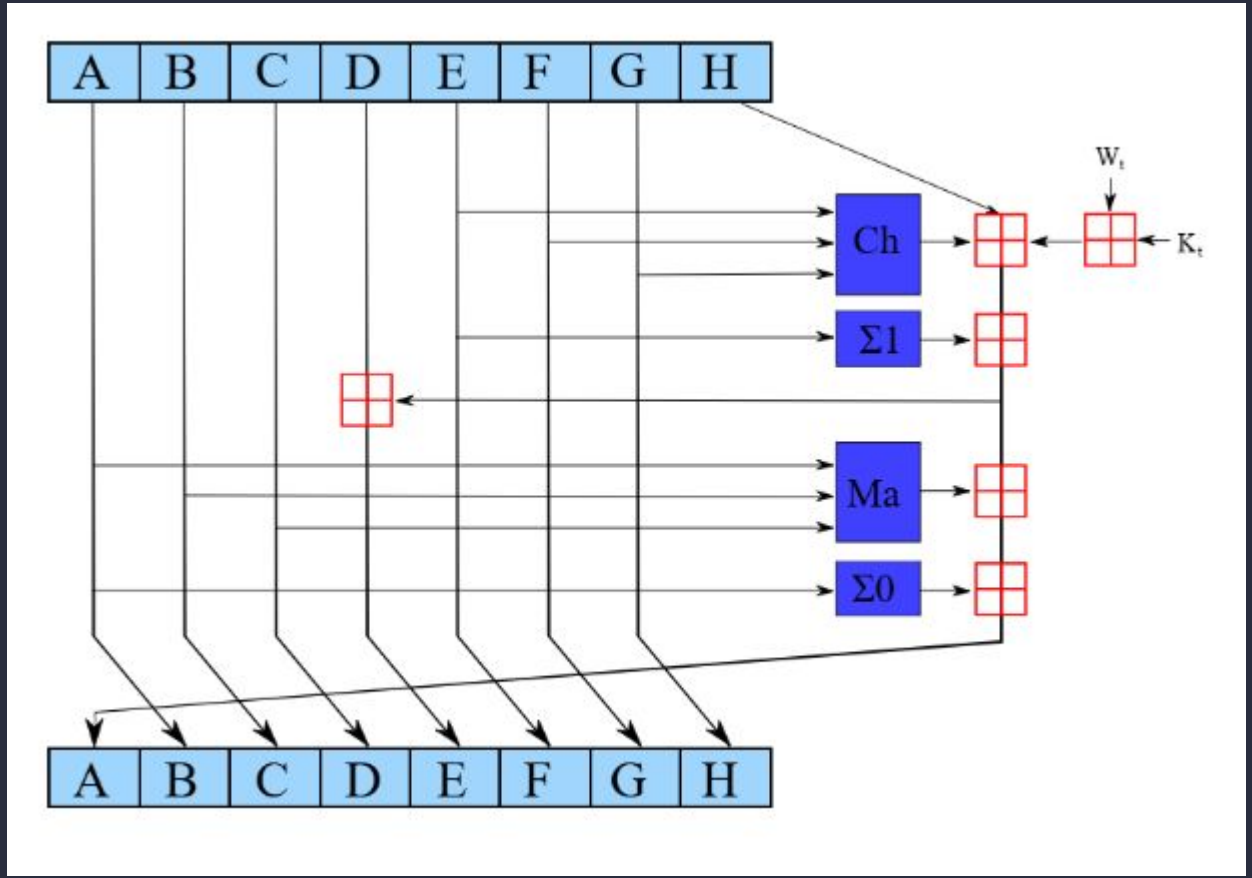
Introduction to SHA-2:

- SHA-2 (Secure Hash Algorithm 2) is a family of cryptographic hash functions developed by the NSA in 2001.
- It converts any input message into a fixed-length hash value.
- Main variants include **SHA-224, SHA-256, SHA-384, and SHA-512**.
- Each differs in output size and internal word size but follows the same design principles.
- SHA-2 is widely used for data integrity, authentication, and digital signatures.

Difference between SHA-1 and SHA-2

SHA-1	SHA-2
It is a cryptographic hash function designed by U.S. National Security Agency to replace SHA-0.	It is a cryptographic hash function designed by U.S. National Security Agency to replace SHA-1.
It produces 160 bits hash value.	It produces 224, 256, 384, or 512 bits hash value.
It is less secure.	While it is more secure.
SHA-1 certificates are not reliable.	SHA-2 has more improved certificates.
It generates smaller hash.	While it generates larger hash.
Hash generated by SHA-1 is weak.	While hash generated by SHA-2 is strong.
It is not widely used now-a-days.	While it is used widely.

WORKING OF SHA-2



Working Principle of SHA-2

Main Steps:

1. **Preprocessing:**
    - Message padding → ensures length ≡ 448 mod block size.
    - Append 64-bit (or 128-bit) length field.
    - Divide message into **blocks (512 or 1024 bits)**.
2. **Message Schedule:**
    - Each block expands into **64 (SHA-256)** or **80 (SHA-512)** 32/64-bit words.

3.Compression Function:

- 8 working registers (a–h).

For each round:

$T1 = h + \Sigma1(e) + Ch(e,f,g) + Kt + Wt$
$T2 = \Sigma0(a) + Maj(a,b,c)$

-
- Update (a–h) each round.

4.Final Output:

- Add compressed block to current hash value.
- Final digest = concatenation of H0–H7.

Applications and Importance of SHA-2

- SHA-2 ensures data integrity and authenticity in digital systems.
- It's used in **digital signatures, SSL/TLS security, blockchain, and password hashing**.
- The algorithm prevents unauthorized data modification and forgery.
- SHA-2 plays a vital role in securing communication and stored information.
- It's a core component of many cybersecurity and encryption protocols.

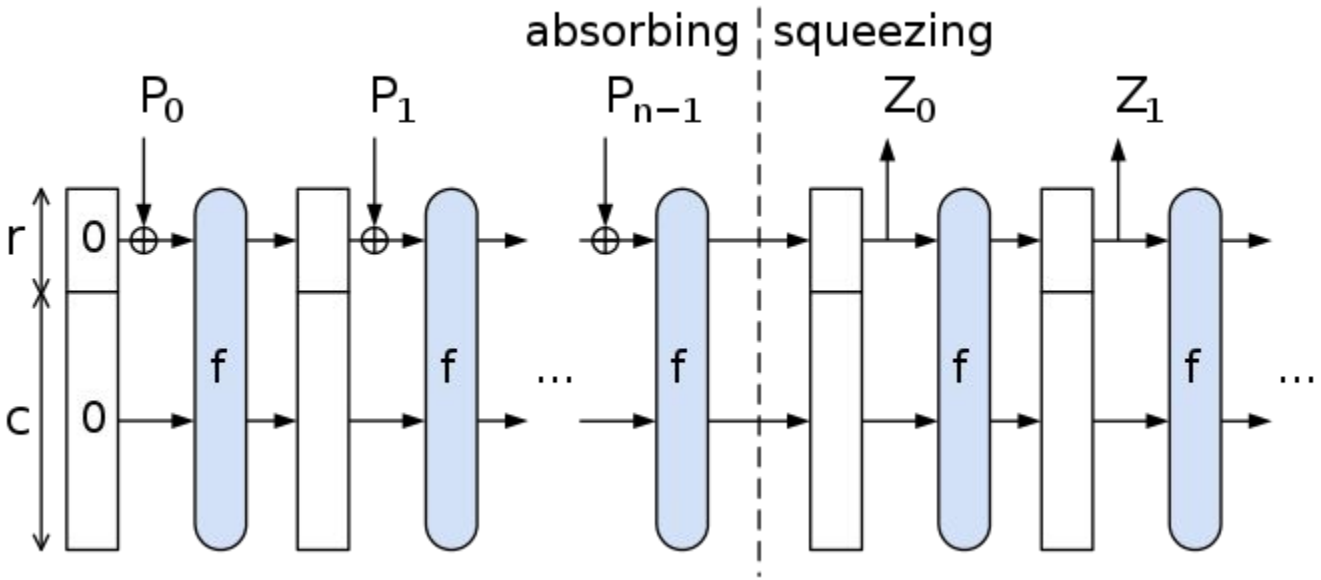
SHA – 3

Introduction to SHA-3:

- SHA-3 is a cryptographic hash function based on Keccak algorithm, designed to generate a fixed-size hash value from input of any length.
- It supports different output sizes — 224, 256, 384, and 512 bits.
- SHA-3 was developed as a new standard by the NIST in 2015.
- SHA-3 was designed to provide stronger security, resist collision and length-extension attacks.

Structure of SHA-3:

- Core of SHA-3 is the Keccak algorithm.
- Keccak uses a sponge construction model that **absorbs** input into a fixed-size state, then **squeezes** out the hash output.
- It involves a permutation function operates on a 1600-bit state.
- Highly parallel and hardware-efficient design.


absorbing
squeezing
P_0
P_1
P_{n-1}
Z_0
Z_1
r
c
0
0
f
f
f
f
f
f
...
...

Implementation:

- Python has a built-in library called *hashlib.*

```python
import hashlib

def sha3(message):
    sha3_hash = hashlib.sha3_256()
    sha3_hash.update(message.encode('utf-8'))
    return sha3_hash.hexdigest()

message = "Hello, everyone!"
sha3_hash_value = sha3(message)
print("SHA-3 hash:", sha3_hash_value)
```

Applications and Advantages:

- High resistance to collision, preimage, and length-extension attacks.
- More secure and flexible than SHA-2.
- Efficient in both hardware and software implementations.
- Used in cryptographic protocols, blockchain systems, and digital signatures.

Strength of SHA-3:

- SHA-3 is cryptographically strong, no practical breaks for full 24-round Keccak.
- Main risks are implementation (side-channels, faults) and misuse in protocols.
- Avoid using raw SHA-3 for authentication or password hashing.
- Mitigate with constant-time implementations, domain separation, and vetted libraries.

Practical Applications & Security Considerations

Practical Applications of SHA

- **Blockchain:** SHA-256 is the backbone of Bitcoin and many other blockchains. Ensures immutability of blocks and integrity of transaction data.
- **SSL/TLS (Secure Web Communication):** Used in digital certificates to verify authenticity and prevent tampering.
- **Password Hashing:** Stores only hashed passwords instead of plaintext (e.g., SHA-256 or SHA-512 with salt).
- **File Integrity Verification:** Hash values help verify that downloaded or transmitted files are unchanged.

Performance Considerations

- **Speed vs Security:** SHA-1 is faster but insecure; SHA-2 and SHA-3 provide stronger protection with moderate performance cost.
- **Hardware Acceleration:** Modern CPUs and GPUs have built-in optimizations for SHA-2 and SHA-3.
- **Energy Efficiency:** Important for IoT and blockchain mining—SHA-3 (Keccak) offers better flexibility.
- **Implementation Trade-offs:** Choice depends on the platform (embedded devices vs servers).

Future Directions in SHA Research

- Development of **post-quantum-resistant** hash functions.
- Focus on **lightweight hashing** for IoT devices.
- **Hybrid schemes** combining SHA-3 with other primitives for layered security.
- Ongoing **cryptanalysis** to test resistance against new attack vectors.

Mini Blockchain using SHA-256

- Mini blockchain demonstration using **SHA-256 hashing**.
- Each block contains: **index, timestamp, transaction data, previous block hash, and its own hash**.
- **Genesis block** starts the chain with a previous hash of `"0"`.
- Subsequent blocks reference the **hash of the previous block**, forming a linked chain.
- **Immutability:** Changing a block's data changes its hash, breaking the chain.
- Tampering is **immediately detectable** due to hash mismatch.
- Demonstrates **secure data integrity** and prevention of unauthorized changes in a blockchain.

Python Implementation

```python
import hashlib
import time

class Block:
    def __init__(self, index, data, previous_hash):
        self.index = index
        self.timestamp = time.ctime()
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.calculate_hash()

    def calculate_hash(self):
        value = str(self.index) + self.timestamp + self.data + self.previous_hash
        return hashlib.sha256(value.encode()).hexdigest()

genesis_block = Block(0, "Genesis Block", "0")
second_block = Block(1, "Alice -> Bob : 5 BTC", genesis_block.hash)
third_block = Block(2, "Bob -> Charlie : 2 BTC", second_block.hash)

blockchain = [genesis_block, second_block, third_block]
```

```python
print("\n=== Original Blockchain ===\n")
for block in blockchain:
    print(f"Block {block.index}")
    print(f"Timestamp     : {block.timestamp}")
    print(f"Data          : {block.data}")
    print(f"Prev Hash     : {block.previous_hash}")
    print(f"Current Hash  : {block.hash}\n")

print("=== Tampering Example ===")
second_block.data = "Alice -> Bob : 50 BTC"
tampered_hash = second_block.calculate_hash()
print(f"Modified Data : {second_block.data}")
print(f"New Hash      : {tampered_hash}")
print(f"Old Hash      : {second_block.hash}\n")

if tampered_hash != second_block.hash:
    print("⚠️  Tampering detected! The blockchain is no longer valid.")
else:
    print("✅ Blockchain is still valid.")
```

Output:

```
=== Original Blockchain ===
Block 0
Timestamp     : Sun Oct 26 19:37:56 2025
Data          : Genesis Block
Prev Hash     : 0
Current Hash  : cebd8f7f4b70897d07cd50d2f9e379b4206afc2ead6445a06cf8b9f9ca2fe994

Block 1
Timestamp     : Sun Oct 26 19:37:56 2025
Data          : Alice -> Bob : 5 BTC
Prev Hash     : cebd8f7f4b70897d07cd50d2f9e379b4206afc2ead6445a06cf8b9f9ca2fe994
Current Hash  : c7aa06b79729b7e0bca90f452782bb2cac55d0a6d56d5d7512df88a403362776

Block 2
Timestamp     : Sun Oct 26 19:37:56 2025
Data          : Bob -> Charlie : 2 BTC
Prev Hash     : c7aa06b79729b7e0bca90f452782bb2cac55d0a6d56d5d7512df88a403362776
Current Hash  : 58e35754890ae09b6a1aa257452b76f4c473d6a1930ff548bccb010a86036c15

=== Tampering Example ===
Modified Data : Alice -> Bob : 50 BTC
New Hash      : 023fa6f8cabe5a8bf21f8ea5dbae5778c5bac5294a2a5d2eac8a3f6eb0a48427
Old Hash      : c7aa06b79729b7e0bca90f452782bb2cac55d0a6d56d5d7512df88a403362776

⚠️ Tampering detected! The blockchain is no longer valid.
```

Thank you