Hashing and Cryptographic Hash Functions

An approach of security and integrity

Introduction to Hashing

- Hashing is a process of converting data of any size into a fixed-size value.
- The output is called a hash, digest, or checksum.
- Common in databases, password storage, and integrity checks.

Introduction to Hashing

- Input: "Hello"
- Hash: 8b1a9953c4611296a827abf8c47804d7

How Hashing Works

- A hash function takes input (message) and produces output (hash value).
- It's a one-way function: easy to compute, hard to reverse.
- Same input → same hash;
- different input → different hash.
- `Message → [Hash Function] → Hash Output`

Characteristics of a Good Hash Function

- Deterministic
- Fast to compute
- Uniform distribution
- Avalanche effect — small change in input → large change in output
- Irreversibility

Example of Avalanche Effect

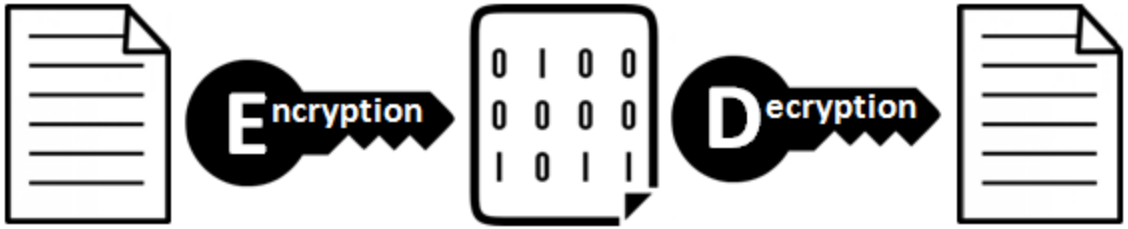
Input	Hash (SHA-1)
"abc"	a9993e364706816aba3e25717850c26c9cd0d89d
"abC"	4ca7d5d8c846c74f1b76f8f015b02ef99fcb94b3

changing a single letter drastically changes the hash.

Hashing vs. Encryption

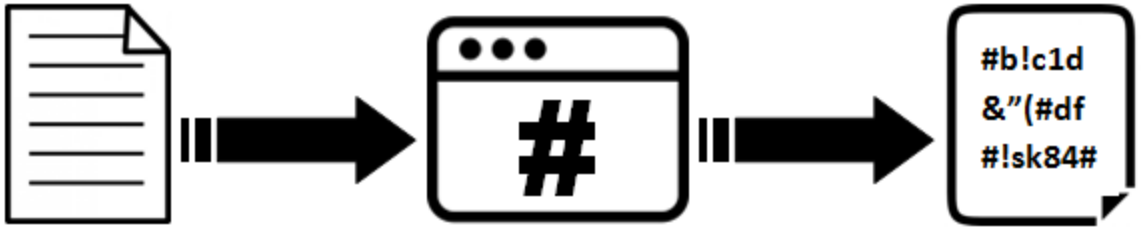
Feature	Hashing	Encryption
Purpose	Verify integrity	Protect confidentiality
Reversible	☐ No	☐ Yes (with key)
Output Size	Fixed	Variable
Example	SHA-256	AES, RSA

Encryption & Decryption



Plain Text | Encrypted Text | Plain Text

Hashing Algorithm



Plain Text | Hash Function | Hashed Text

Applications of Hashing

- Data integrity verification
- Password storage
- Digital signatures
- Blockchain and cryptocurrencies
- File and data deduplication

Cryptographic Hash Functions

- A cryptographic hash function is a special type of hash function designed for security.
- It must satisfy:
  - Pre-image resistance: Hard to find input from hash.
  - Second pre-image resistance: Hard to find a different input with same hash.
  - Collision resistance: Hard to find two inputs producing same hash.

Common Cryptographic Hash Algorithms

Algorithm	Output Size	Status
MD5	128-bit	Weak (deprecated)
SHA-1	160-bit	Weak (deprecated)
SHA-256	256-bit	Strong
SHA-512	512-bit	Strong

MD5 Example

- `echo -n "SAEED" | md5sum`

- `e9d71f5ee7c92d6dc9e92ffdad17b8bd`

Visualizing Collision Resistance

- Two different files produce different hashes:
- `sha256sum file1.txt`
- `sha256sum file2.txt`

- Even one character change produces a completely new hash.

Password Hashing

Instead of storing plaintext passwords:

- user: SAEED
- password: `MySecret123`

System stores:

- user: SAEED
- hash: `2bb80d537b1da3e38bd30361aa855686bde0ba`

Hashing in Digital Signatures

- Hash the message
- Encrypt the hash with sender's private key
- Receiver decrypts and compares hash for integrity and authenticity.

Hashing in Blockchain

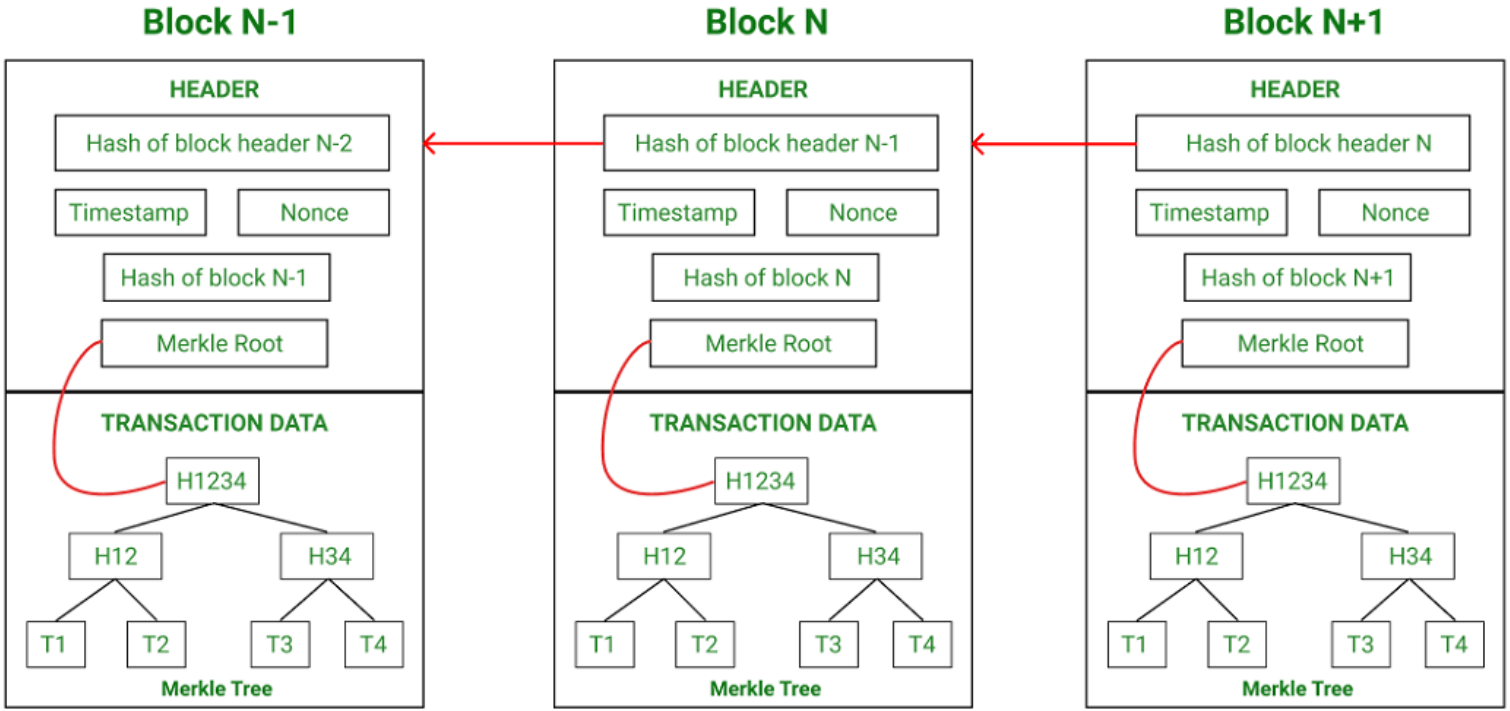
- Every block contains and depends the hash of the previous block.
- Creates a tamper-proof chain.
- If there is a changing in one block, that changes all subsequent hashes → tamper-evident chain.
- Used in Bitcoin, Ethereum, Hyperledger, etc.

Block Structure Example

Block Header
- Previous Hash
- Timestamp
- Nonce
- Merkle Root

- Previous Hash ensures each block is chained securely to its predecessor.
- The Timestamp helps verify when transactions occurred.
- The Nonce is critical in Proof-of-Work mining; it's changed until the block hash meets the difficulty target.
- The Merkle Root summarizes all transactions using the Merkle Tree, providing integrity verification without storing all data.

Merkle Tree


Block N-1
HEADER
Hash of block header N-2
Timestamp
Nonce
Hash of block N-1
Merkle Root
TRANSACTION DATA
H1234
H12
H34
T1
T2
T3
T4
Merkle Tree
Block N
HEADER
Hash of block header N-1
Timestamp
Nonce
Hash of block N
Merkle Root
TRANSACTION DATA
H1234
H12
H34
T1
T2
T3
T4
Merkle Tree
Block N+1
HEADER
Hash of block header N
Timestamp
Nonce
Hash of block N+1
Merkle Root
TRANSACTION DATA
H1234
H12
H34
T1
T2
T3
T4
Merkle Tree

Thank You