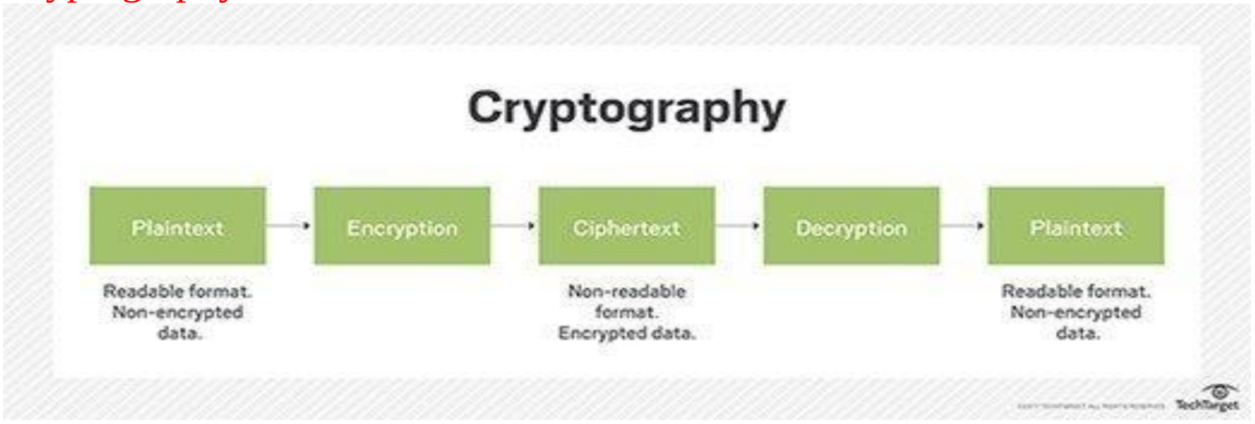
Unit-III
Cryptography and Hash Functions

Cryptography:


Cryptography
Plaintext
Encryption
Ciphertext
Decryption
Plaintext
Readable format. Non-encrypted data.
Non-readable format. Encrypted data.
Readable format. Non-encrypted data.
TechTarget

Cryptography Is technique of securing information and communications through use of codes so that only those person for whom the information is intended can understand it and process it? Thus preventing unauthorized access to information. The prefix "crypt" means "hidden" and suffix "graph" means "writing". In Cryptography the techniques which are used to protect information are obtained from mathematical concepts and a set of rule based calculations known as algorithms to convert messages in ways that make it hard to decode it. These algorithms are used for cryptographic key generation, digital signing, and verification to protect data privacy, web browsing on internet and to protect confidential transactions such as credit card and debit card transactions.

Techniques used For Cryptography: In today's age of computers cryptography is often associated with the process where an ordinary plain text is converted to cipher text which is the text made such that intended receiver of the text can only decode it and hence this process is known as encryption. The process of conversion of cipher text to plain text this is known as decryption.

Features of Cryptography are as follows:

1. **Confidentiality:** Information can only be accessed by the person for whom it is intended and no other person except him can access it.
2. **Integrity:** Information cannot be modified in storage or transition between sender and intended receiver without any addition to information being detected.
3. **Non-repudiation:** The creator/sender of information cannot deny his intention to send information at later stage.
4. **Authentication:** The identities of sender and receiver are confirmed. As well as destination/origin of information is confirmed.

Types of Cryptography: In general there are three types Of cryptography:

1. **Symmetric Key Cryptography:** It is an encryption system where the sender and receiver of message use a single common key to encrypt and decrypt messages. Symmetric Key Systems are faster and simpler but the problem is that sender and receiver have to somehow exchange key in a secure manner. The most popular symmetric key cryptography system are Data Encryption System (DES) and Advanced Encryption System (AES).
2. **Hash Functions:** There is no usage of any key in this algorithm. A hash value with fixed length is calculated as per the plain text which makes it impossible for contents of plain text to be recovered. Many operating systems use hash functions to encrypt passwords.
3. **Asymmetric Key Cryptography:** Under this system a pair of keys is used to encrypt and decrypt information. A receiver's public key is used for encryption and a receiver's private key is used for decryption. Public key and Private Key are different. Even if the public key is known by everyone the intended receiver can only decode it because he alone know his private key. The most popular asymmetric key cryptography algorithm is RSA algorithm.

Applications Of Cryptography:

1. **Computer passwords:** Cryptography is widely utilized in computer security, particularly when creating and maintaining passwords. When a user logs in, their password is hashed and compared to the hash that was previously stored. Passwords are hashed and encrypted before being stored. In this technique, the passwords are encrypted so that even if a hacker gains access to the password database, they cannot read the passwords.
2. **Digital Currencies:** To safeguard transactions and prevent fraud, digital currencies like Bitcoin also use cryptography. Complex algorithms and cryptographic keys are used to safeguard transactions, making it nearly hard to tamper with or forge the transactions.
3. **Secure web browsing:** Online browsing security is provided by the use of cryptography, which shields users from eavesdropping and man-in-the-middle assaults. Public key cryptography is used by the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols to encrypt data sent between the web server and the client, establishing a secure channel for communication.
4. **Electronic signatures:** Electronic signatures serve as the digital equivalent of a handwritten signature and are used to sign documents. Digital signatures are created using cryptography and can be validated using public key cryptography. In many nations, electronic signatures are enforceable by law, and their use is expanding quickly.
5. **Authentication:** Cryptography is used for authentication in many different situations, such as when accessing a bank account, logging into a computer, or using a secure network. Cryptographic methods are employed by authentication

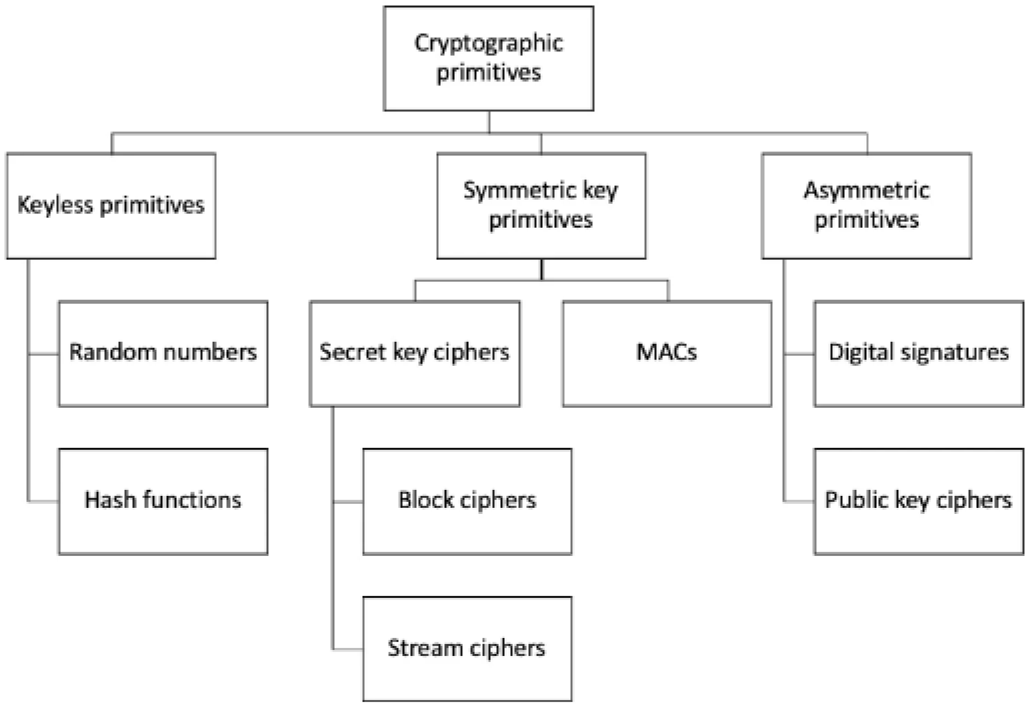
protocols to confirm the user’s identity and confirm that they have the required access rights to the resource.

6. **Cryptocurrencies:** Cryptography is heavily used by cryptocurrencies like Bitcoin and Ethereum to safeguard transactions, thwart fraud, and maintain the network’s integrity. Complex algorithms and cryptographic keys are used to safeguard transactions, making it nearly hard to tamper with or forge the transactions.
7. **End-to-End Encryption:** End-to-end encryption is used to protect two-way communications like video conversations, instant messages, and email. Even if the message is encrypted, it assures that only the intended receivers can read the message. End-to-end encryption is widely used in communication apps like WhatsApp and Signal, and it provides a high level of security and privacy for users.

Advantages

1. **Access Control:** Cryptography can be used for access control to ensure that only parties with the proper permissions have access to a resource. Only those with the correct decryption key can access the resource thanks to encryption.
2. **Secure Communication:** For secure online communication, cryptography is crucial. It offers secure mechanisms for transmitting private information like passwords, bank account numbers, and other sensitive data over the internet.
3. **Protection against attacks:** Cryptography aids in the defence against various types of assaults, including replay and man-in-the-middle attacks. It offers strategies for spotting and stopping these assaults.
4. **Compliance with legal requirements:** Cryptography can assist firms in meeting a variety of legal requirements, including data protection and privacy legislation.

Cryptography Primitives:



Cryptography primitives are fundamental building blocks or techniques used in cryptographic systems to achieve specific security goals. These primitives serve as the foundation for various cryptographic applications and protocols. Here are some common cryptography primitives:

Encryption/Decryption: Encryption is a fundamental primitive that involves transforming plaintext data into cipher text to protect its confidentiality. Decryption is the reverse process of converting cipher text back into plaintext using a cryptographic key. Common encryption algorithms include Advanced Encryption Standard (AES), Data Encryption Standard (DES), and RSA.

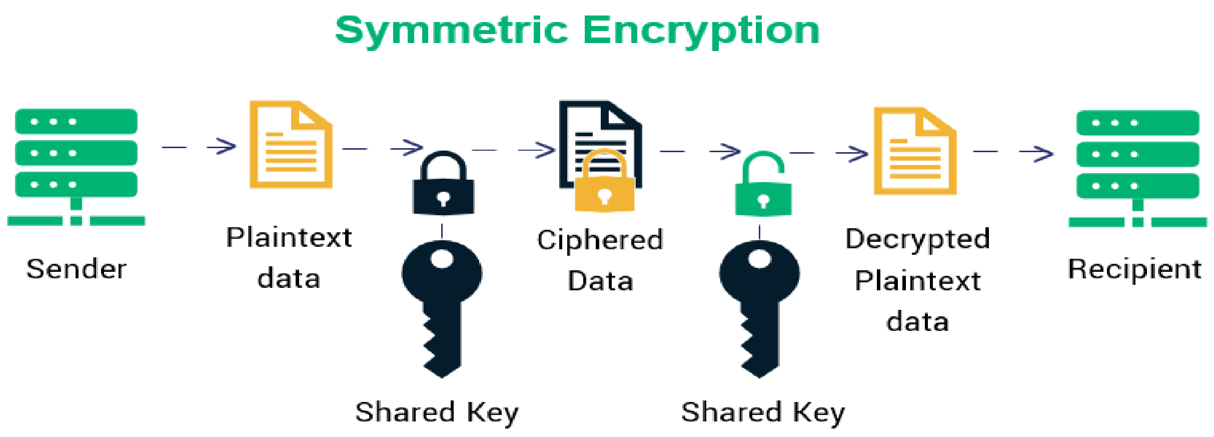
Hash Functions: Hash functions are one-way functions that take an input (message or data) and produce a fixed-size hash value or digest. Hash functions are used to verify data integrity, create digital signatures, and generate unique identifiers for data. Examples include SHA-256 and MD5 (though MD5 is now considered weak and unsuitable for security-sensitive applications).

Digital Signatures: Digital signatures use asymmetric cryptography to provide authentication and integrity verification. A digital signature is created using a private key and can be verified using the corresponding public key. It proves that a message was signed by the private key holder and that the message hasn't been tampered with.

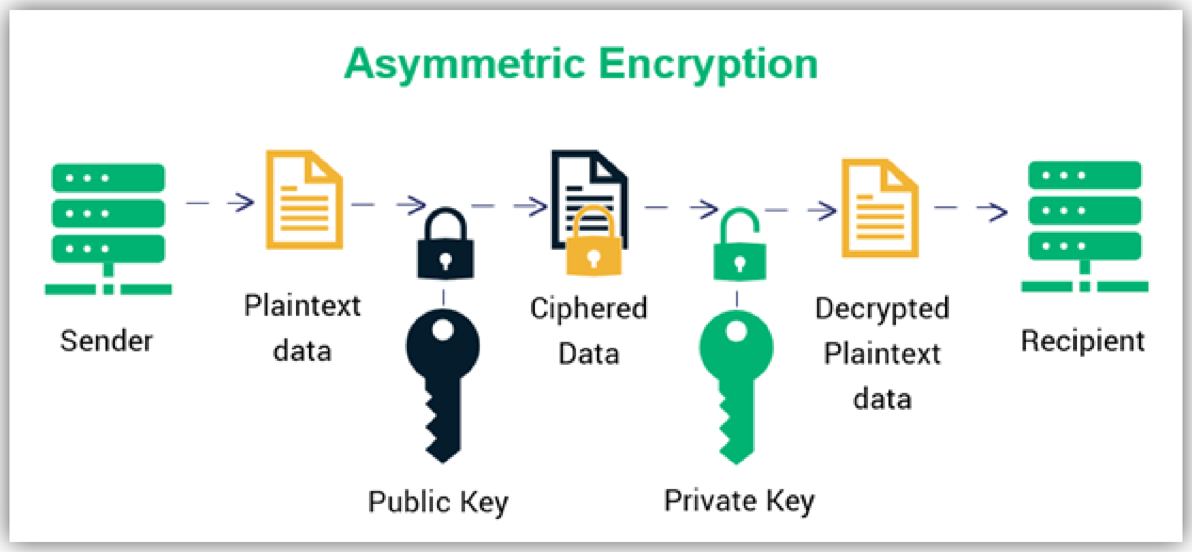
Key Exchange: Key exchange primitives enable two or more parties to securely exchange cryptographic keys over an insecure communication channel. The Diffie-Hellman key exchange is a widely used method for securely sharing keys.

Message Authentication Codes (MACs): MACs are cryptographic constructs used to verify the authenticity and integrity of a message or data. They are often used in combination with symmetric-key encryption to ensure that the cipher text has not been modified.
Public Key Infrastructure (PKI): PKI is a framework that combines various cryptographic primitives to establish trust in a networked environment. It involves the use of digital certificates, Certificate Authorities (CAs), and public key cryptography to verify the authenticity of entities in a network.
Random Number Generation: Cryptographically secure random number generation is essential for many cryptographic applications, including key generation. Weak or predictable random numbers can lead to security vulnerabilities.
Diffusion and Confusion: These are cryptographic principles introduced by Claude Shannon. Diffusion aims to spread the influence of individual plaintext bits across the entire cipher text, making it harder to discern patterns. Confusion involves making the relationship between the key and cipher text as complex as possible, increasing the difficulty of cryptanalysis.
Zero-Knowledge Proofs: These cryptographic techniques allow one party (the prover) to prove to another party (the verifier) that they know a particular piece of information without revealing the information itself. Zero-knowledge proofs are used in authentication and authentication protocols.
Holomorphic Encryption: Holomorphic encryption allows operations to be performed on encrypted data without decrypting it. This is useful in secure computation and privacy-preserving applications.
These cryptography primitives are combined and used in various ways to create secure communication protocols, data protection schemes, authentication mechanisms, and more. Understanding these building blocks is essential for designing and implementing secure cryptographic systems.

Symmetric Cryptography:



Symmetric Key Encryption: Encryption is a process to change the form of any message in order to protect it from reading by anyone. In Symmetric-key encryption the message is encrypted by using a key and the same key is used to decrypt the message which makes it easy to use but less secure. It also requires a safe method to transfer the key from one party to another.



Asymmetric Key Encryption: Asymmetric Key Encryption is based on public and private key encryption techniques. It uses two different key to encrypt and decrypt the message. It is more secure than the symmetric key encryption technique but is much slower.

Sr No	**Symmetric Key Encryption**	**Asymmetric Key Encryption**
1	It only requires a single key for both encryption and decryption.	It requires two keys, a public key and a private key, one to encrypt and the other one to decrypt.
2	The size of cipher text is the same or smaller than the original plain text.	The size of cipher text is the same or larger than the original plain text.
3	The encryption process is very fast.	The encryption process is slow.
4	It is used when a large amount of data is required to transfer.	It is used to transfer small amounts of data.
5	It only provides confidentiality.	It provides confidentiality, authenticity, and non-repudiation.
6	The length of key used is 128 or 256 bits	The length of key used is 2048 or higher

7	In symmetric key encryption, resource utilization is low as compared to asymmetric key encryption.	In asymmetric key encryption, resource utilization is high.
8	It is efficient as it is used for handling large amount of data.	It is comparatively less efficient as it can handle a small amount of data.
9	Security is less as only one key is used for both encryption and decryption purpose.	It is more secure as two keys are used here- one for encryption and the other for decryption.
10	The Mathematical Representation is as follows- P = D (K, E(K, P)) where K –> encryption and decryption key P –> plain text D –> Decryption E(K, P) –> Encryption of plain text using K	The Mathematical Representation is as follows- P = D(Kd, E (Ke,P)) where Ke –> encryption key Kd –> decryption key D –> Decryption E(Ke, P) –> Encryption of plain text using encryption key Ke. P –> plain text
11	Examples: 3DES, AES, DES and RC4	Examples: Diffie-Hellman, ECC, El Gamal, DSA and RSA

Introduction of Hash:

A hash, in the context of computer science and cryptography, is a fixed-size alphanumeric string or value generated by a mathematical function known as a hash function. Hash functions take an input (often referred to as a "message") and produce a unique, fixed-size output, which is typically a string of numbers and letters. These outputs are commonly referred to as "hash values" or simply "hashes."

Hash functions are designed to be one-way functions, meaning it is computationally infeasible to reverse the process and determine the original input from the hash value. Additionally, a good hash function ensures that even a tiny change in the input results in a significantly different hash value. This property is called the "avalanche effect."

Here are some key characteristics and uses of hash functions:

Deterministic: For the same input, a hash function will always produce the same hash value. This determinism is crucial for data verification and consistency checks.

Fixed Output Size: Hash functions produce hash values of a fixed length, regardless of the length of the input data. Common hash lengths include 128 bits, 160 bits, 256 bits, and 512 bits.

Efficiency: Hash functions are designed to be computationally efficient and should produce hash values quickly, even for large inputs.

Pre-image Resistance: It should be computationally infeasible to find an input that corresponds to a given hash value. In other words, given a hash value, it should be difficult to reverse-engineer the original input.

Collision Resistance: A good hash function makes it extremely unlikely that two different inputs will produce the same hash value. When two different inputs produce the same hash, it's called a "collision," and hash functions aim to minimize the likelihood of collisions.

Hash functions have a wide range of practical applications, including:

Data Integrity: Hashes are used to verify the integrity of data during transmission or storage. By comparing the hash of received data with a precomputed hash, one can detect if the data has been tampered with.

Password Storage: Hash functions are used to securely store passwords in databases. Instead of storing plaintext passwords, systems store the hashes of passwords. During login, the system hashes the entered password and compares it to the stored hash.

Digital Signatures: Hash functions play a crucial role in digital signatures. To create a digital signature, a hash of the message is signed with a private key. The recipient can verify the signature by hashing the received message and comparing it to the decrypted signature.

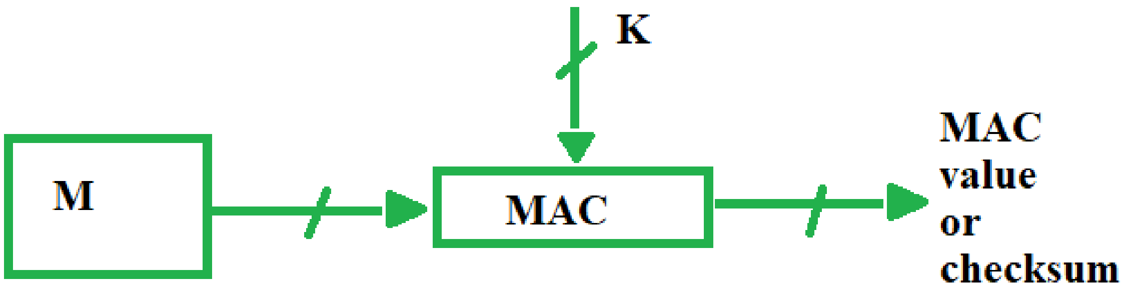
Cryptographic Applications: Hash functions are used in various cryptographic protocols, including blockchain technology (where they are used to create blocks and secure transactions), digital certificates (to verify their integrity), and more.

Data Structures: Hash functions are used in data structures like hash tables and hash maps to quickly retrieve data based on a key.

In summary, hash functions are essential tools in computer science and cryptography, providing a means to verify data integrity, securely store passwords, enable digital signatures, and support various other security and data management tasks. Their deterministic and collision-resistant properties make them a cornerstone of modern computing.

Message Authentication Code:

Apart from intruders, the transfer of message between two people also faces other external problems like noise, which may alter the original message constructed by the sender. To ensure that the message is not altered there’s this cool method MAC.

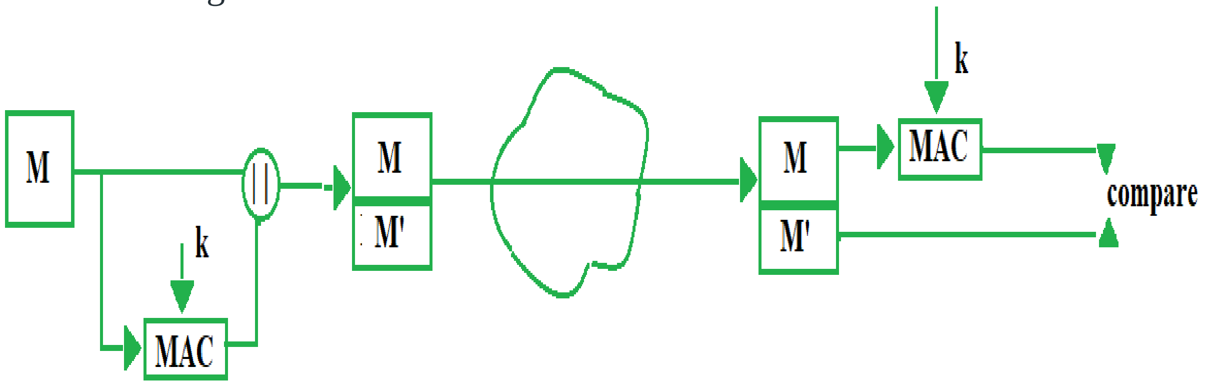


MAC stands for Message Authentication Code. Here in MAC, sender and receiver share same key where sender generates a fixed size output called Cryptographic checksum or Message Authentication code and appends it to the original message. On receiver's side, receiver also generates the code and compares it with what he/she received thus ensuring the originality of the message. These are components:

- Message
- Key
- MAC algorithm
- MAC value

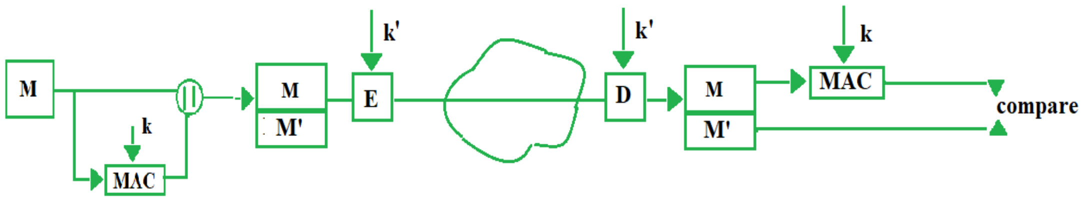
There are different types of models Of Message Authentication Code (MAC) as following below:

1. **MAC without encryption –**
   This model can provide authentication but not confidentiality as anyone can see the message.

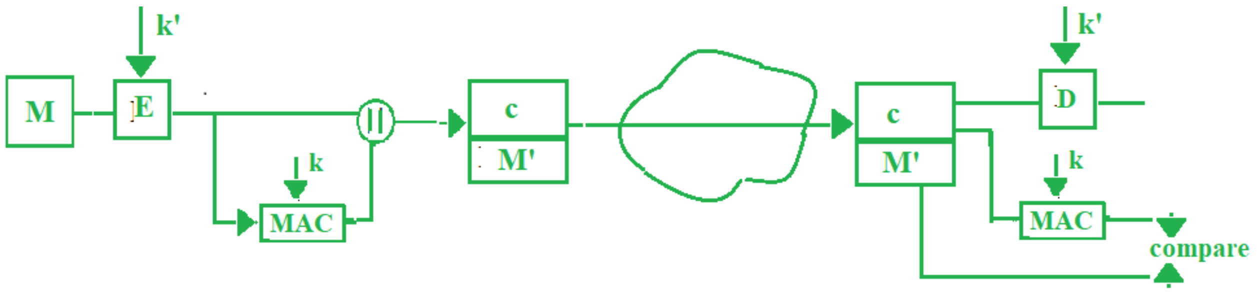


2. **Internal Error Code –**
   In this model of MAC, sender encrypts the content before sending it through network for confidentiality. Thus this model provides confidentiality as well as authentication.

   M' = MAC(M, k)

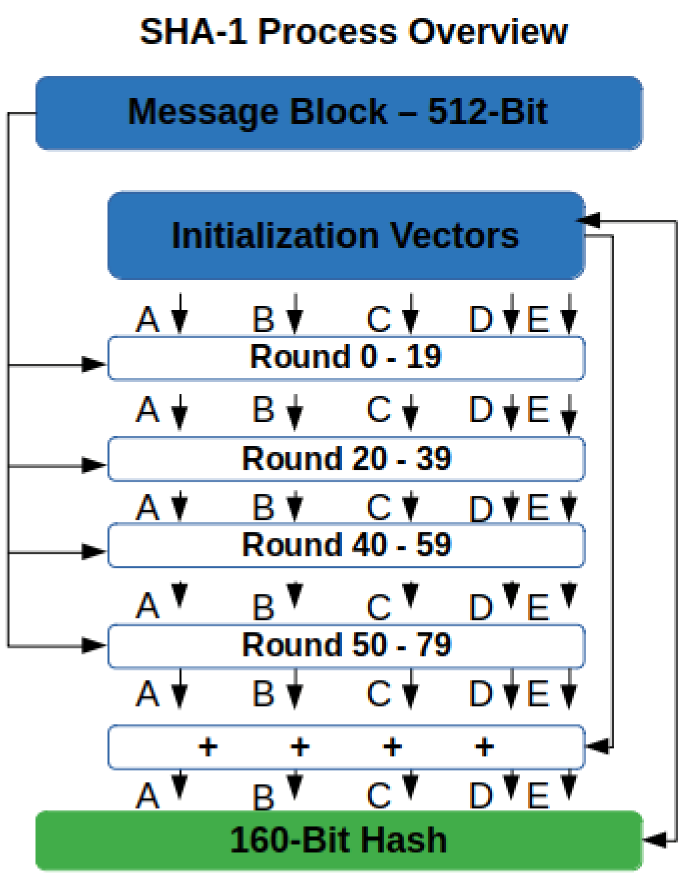


3.External Error Code –
For cases when there is an alteration in message, we decrypt it for waste, to overcome that problem, we opt for external error code. Here we first apply MAC on the encrypted message ‘c’ and compare it with received MAC value on the receiver’s side and then decrypt ‘c’ if they both are same, else we simply discard the content received. Thus it saves time.
c = E(M, k')
M' = MAC(c, k)



Problems in MAC –
If we do reverse engineering we can reach plain text or even the key. Here we have mapped input to output, to overcome this we move on to hash functions which are “One way”.

Secure Hash Algorithms (SHA-1):



Developed by the NSA (National Security Age), SHA-1 is one of the several algorithms included under the umbrella of the "secure hash algorithm" family. In a nutshell, it's a one-way cryptographic function that converts messages of any lengths and returns a 160 bits hash value as a 40 digits long hexadecimal number. This will look along the lines of this: 0aa12c48afc6ff95c43cd3f74259a184c34cde6d.

Its structure is similar to MD5, but the process to get the message-digest is more complex as summarized in the steps listed below:

1. Add padding bits to the original message. You'll extend the total length of the original input so it's 64 bits short of any multiple of 512 (i.e., 448 mod 512).

2. Add length bits to the end of the padded message. With this operation, the total number of bits in the message becomes a multiple of 512 (i.e., 64 bits).

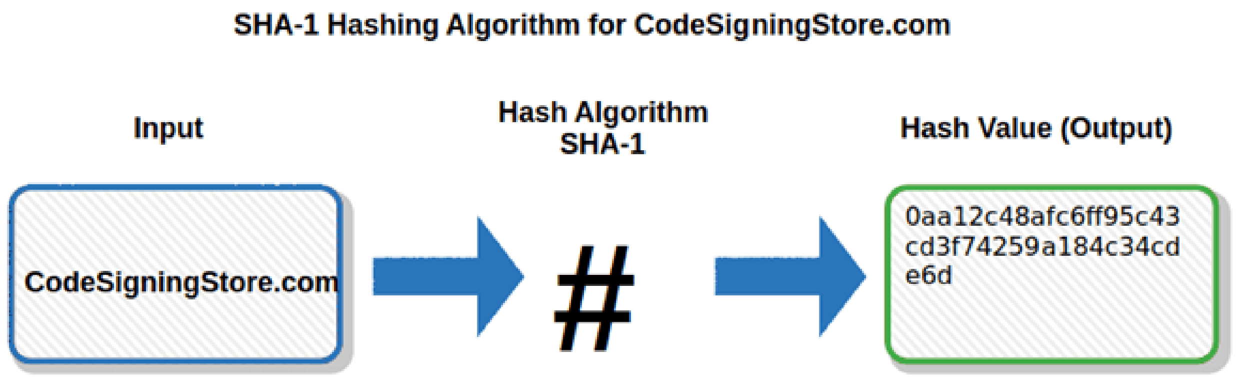
3. Initialize MD buffers to compute the message digest. This algorithm requires two buffers and a long sequence of 32-bit words:

- The two five 32-bit registers (“A, B, C, D, E” and “H0, H1, H2, H3, H4”) have specific initial values, and
- The sequence of 80 32-bit words (W[0], W[1], W[2]… W[68], W[69]).

4. Process the message in successive 512 bits blocks. Each block goes through a complex process of expansion and 80 rounds of compression of 20 steps each. The value obtained after each compression is added to the current buffer (hash state).

5. Produce a final 160 bits hash value. After the last block is processed, the current hash state is returned as the final hash value output.

SHA-1 shouldn’t be used for digital signatures or certificates anymore. Theoretically broken since 2005, it was formally deprecated by the National Institute of Standards and Technology (NIST) in 2011. In 2017, SHA-1 was officially broken (SHAttered) by Google’s academics, who managed to produce two files with the same hash.



Pros of SHA 1:
1. it’s a slow algorithm. This characteristic made it useful for storing password hashes as it slows down brute force attacks.
2. It can be used to compare files or codes to identify unintentional only corruptions.
3. Can replace SHA-2 in case of interoperability issues with legacy codes.

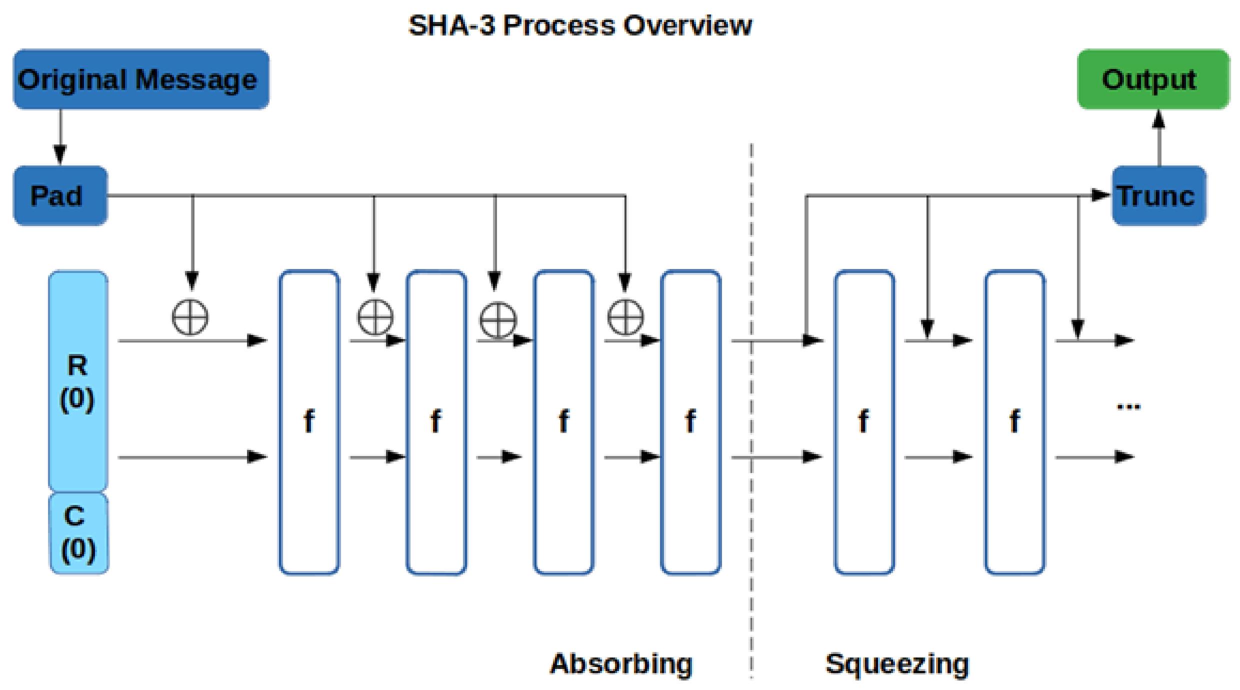
Cons of SHA1:
1. Slower than other algorithms, therefore unsuitable for many purposes other than password storage (e.g., when establishing secure connections to websites or comparing files).
2. Less secure with many vulnerabilities found during the years.
3. Collisions are easy and cheap to find.
4. Key length too short to resist to attacks.

Secure Hash Algorithm Version 3:

SHA-3 is the latest addition to the SHA family. Developed via a public competition promoted by NIST, its part of the same standard while being completely different from MD5, SHA-1 and SHA-2. SHA-3 is based on a new cryptographic approach called sponge construction, used by Keccak. Basically, the data are "absorbed" into the sponge, then the result is "squeezed" out, just like a sponge absorbs and releases water. It's important to note that NIST doesn't see SHA-3 as a full replacement of SHA-2; rather, it's a way to improve the robustness of its overall hash algorithm toolkit.SHA-3 is a family of four algorithms with different hash functions plus two extendable output functions can be used for domain hashing, randomized hashing, stream encryption, and to generate MAC addresses:

- **SHA3-224,**
- **SHA3-256,**
- **SHA3-384,**
- **SHA3-512,**
- **SHAKE-128** (output function), and
- **SHAKE-256** (output function).

How the SHA3-224 Hashing Algorithm Works:



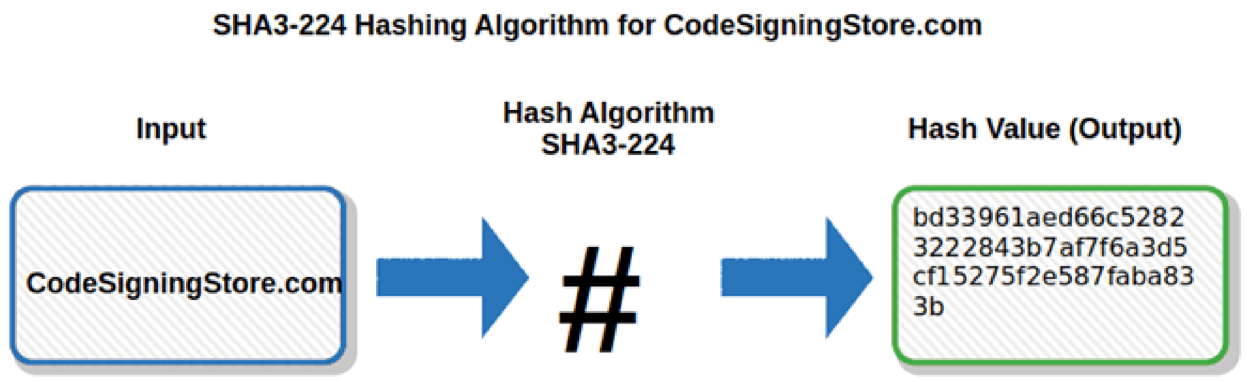
How does it work? We'll base our example on one member of the SHA-3 family: SHA3-224.

1. Add padding bits to the original message. This way the total length is an exact multiple of the rate of the corresponding hash function. In this case, as we've chosen SHA3-224, it must be a multiple of 1152 bits (144 bytes). The SHA-3 process largely falls within two main categories of actions: "absorbing" and "squeezing," each of which we'll discuss in the next sections.

2. Absorb the padded message values to start calculating the hash value. The padded message is partitioned into fixed size blocks. Then each block goes through a series of permutation rounds of five operations a total of 24 times. At the end, we get an internal state size of 1600 bits.

3. Squeeze to extract the hash value. This is where the message is extracted (squeezed out). The 1600 bits obtained with the absorption operation is segregated on the basis of the related rate and capacity (the "r" and "c" we mentioned in the image caption above).

4. Produce the final hash value. Finally, the first 224 bits are extracted from the 1152 bits (SHA3-224's rate). The extracted value of 224 bits is the hash digest of the whole message.


SHA3-224 Hashing Algorithm for CodeSigningStore.com
Input
Hash Algorithm SHA3-224
Hash Value (Output)
CodeSigningStore.com
#
bd33961aed66c5282 3222843b7af7f6a3d5 cf15275f2e587faba83 3b

Pros of SHA3:

1. it's flexible as it allows variable lengths for the input and output, making it ideal for a hash function.
2. Its instances use a single permutation for all security strengths, cutting down implementation costs.
3. The SHA3 family of algorithms enables performance-security trade-offs by choosing the suitable capacity-rate pair.
4. Not vulnerable to length extension attacks.
5. Much faster than its predecessors when cryptography is handled by hardware components.

Cons of SHA3:

1. Susceptible to collision attacks.
2. Much slower than SHA-2 (software only issue).
3. Lack of hardware and software support.

Sr.no	SHA1	SHA2
1	It is a cryptographic hash function designed by U.S National Security Agency to replace SH0.	It is a cryptographic hash function designed by U.S National Security Agency to replace SH1.
2	It was published in 1995.	While it was published in 2001.
3	It produces 160 bits hash value.	It produces 224, 256, 384 or 512 bits hash value.

4	It is successor to SH0 and predecessor to SH2.	It is successor to SH1 and predecessor to SH3.
5	It is less secure.	While it is more secure.
6	Its structure is based on Merkle–Damgard construction.	Its structure is based on Merkle–Damgard structure with Davies–Meyer compression function.
7	SHA1 certificates are not reliable.	SHA2 has more improved certificates.
8	It generates smaller hash.	While it generates larger hash.
9	Hash generated by SHA1 is weak.	While hash generated by SHA2 is strong.
10	It is not widely used now-a-days.	While it is used widely.

Distributed Hash Tables:

A distributed hash table (DHT) is a type of distributed system that provides a lookup service similar to a hash table. In a hash table, data is stored and retrieved using keys, and the keys are used to determine the location of the data in the table. A distributed hash table is similar, but the data is distributed across multiple nodes in a network rather than being stored in a single table.

In a DHT, each node is responsible for storing and managing a portion of the data. When a client wants to retrieve or store data, it sends a request to the network. The request is then forwarded to the appropriate node based on the key of the data being requested. The node then responds to the request and either retrieves or stores the data.

DHTs are used in a variety of applications, including peer-to-peer (P2P) networks, distributed databases, and distributed file systems. They are particularly useful for large-scale distributed systems, as they provide a efficient and scalable way to store and retrieve data.

Why Is a Distributed Hash Table Used?

A distributed hash table (DHT) is a type of distributed system that provides a lookup service similar to a hash table. DHTs are used for a variety of purposes, including −

- **Peer-to-peer (P2P) networks** − DHTs are often used in P2P networks to facilitate the sharing of resources, such as files or data, between peers. DHTs allow peers to locate resources on the network and download them directly from one another.
- **Distributed databases** − DHTs can be used to store and retrieve data in a distributed database. Because the data is distributed across multiple nodes in

the network, DHTs can provide a scalable and efficient way to store and retrieve large amounts of data.

- **Distributed file systems** − DHTs can be used to store and manage files in a distributed file system. By distributing the files across multiple nodes, DHTs can provide a scalable and fault-tolerant way to store and access large amounts of data.
- **Content delivery networks** − DHTs can be used to store and distribute content, such as videos or images, across a network of servers. This can help to reduce the load on a single server and improve the performance of the network.

Advantage of Distributed Hash Table

There are several advantages to using a distributed hash table (DHT) in a distributed system, including −

- **Scalability** − DHTs are highly scalable, as they can store and retrieve large amounts of data without requiring a central authority or server to manage the system. This makes DHTs well-suited for large-scale distributed systems.
- **Efficiency** − DHTs provide a efficient way to store and retrieve data, as they use keys to determine the location of the data in the network. This allows DHTs to quickly locate and retrieve data without having to search the entire network.
- **Fault tolerance** − DHTs are highly fault-tolerant, as they can handle node failures without requiring a central authority to manage the system. If a node fails, the data it was responsible for can be redistributed among the remaining nodes in the network.
- **Decentralization** −DHTs are decentralized, as there is no central authority or server that controls the network. This makes DHTs more resilient and less vulnerable to downtime or attack.
- **Security** − DHTs can provide a secure way to store and retrieve data, as the data is distributed across multiple nodes in the network rather than being stored in a single location. This makes it more difficult for attackers to access or modify the data.

Disadvantage of Distributed Hash Table

There are a few potential disadvantages to using a distributed hash table (DHT) in a distributed system, including −

- **Complexity** − DHTs can be complex to implement and maintain, as they require a large number of nodes to function properly. This can make DHTs more challenging to manage and maintain than other types of distributed systems.

- **Performance** − In some cases, DHTs may not perform as well as other types of distributed systems, particularly when the system is under heavy load or when the network is large and complex.
- **Security** − While DHTs can provide a secure way to store and retrieve data, they can also be vulnerable to certain types of attacks, such as distributed denial of service (DDoS) attacks or Sybil attacks.
- **Compatibility** − DHTs may not be compatible with all types of data or applications, as they may require specific data structures or formats in order to function properly.
- **Limited functionality** − DHTs are primarily designed for storing and retrieving data, and may not provide additional functionality beyond these basic capabilities.

Hashing and Data Structures:

Hashing is a fundamental technique in computer science and data structures. It involves applying a hash function to a data item (often a key) to generate a fixed-size value (hash code or hash value) that represents the original data in a way that is efficient for various operations like insertion, retrieval, and deletion. Hashing is used in various data structures and applications. Here are some key points about the relationship between hashing and data structures:

Hash Functions: A hash function takes an input (or "key") and produces a fixed-size hash value. It should be deterministic, meaning that the same input will always produce the same hash value. Good hash functions have the following properties:

Deterministic: Same input, same output. Fast to compute. Distributes values uniformly across the hash space to minimize collisions (two different inputs producing the same hash value).

Small changes in input should produce significantly different hash values (avalanche effect).

Hash Tables: Hash tables (also known as hash maps) are a common data structure that uses hashing to store and retrieve values efficiently. They consist of an array of buckets or slots, where each slot is associated with a hash code. When you want to store a value, you compute its hash code and use it to determine the index (bucket) where the value should be placed. When you want to retrieve the value, you compute the hash code again and look in the corresponding bucket.

Collisions: Collisions occur when two different inputs produce the same hash code. Hash tables must handle collisions, and there are various strategies for doing so, including chaining (each bucket contains a linked list of values), open addressing (placing the value in the next available slot), and double hashing (using a secondary hash function to resolve collisions).

Data Structures Using Hashing: Hashing is used in several data structures and algorithms, including:

Hash Sets: A data structure that stores a collection of unique values. It uses a hash table where keys are the values themselves.
Hash Maps: A data structure that stores key-value pairs, allowing efficient key-based retrieval and insertion.
Caches: Hashing is used in caching mechanisms to quickly locate and access cached data.
Bloom Filters: A probabilistic data structure used for membership testing (checking if an element is in a set).
Cryptographic Hash Functions: Hashing is used in security applications such as digital signatures and password storage.
Performance: Hashing provides fast access times (usually O(1) on average) for data retrieval, insertion, and deletion when hash functions are well-designed and collisions are handled efficiently. However, poor hash functions or high collision rates can degrade performance.
In summary, hashing is a critical technique used in various data structures and applications to optimize the storage and retrieval of data. It plays a key role in improving the efficiency of operations on large datasets and is widely used in computer science and software engineering.

Hashing in Blockchain Mining:

Transaction Verification:

-Users initiate transactions, which are then broadcast to the network.

-Nodes in the network verify the validity of each transaction by checking digital signatures, confirming sufficient funds, and ensuring it hasn't already been spent.

Block Construction:

-Valid transactions are grouped together into a block.

-Each block contains a reference to the previous block, forming a chain. This reference is the hash of the previous block (the "parent" block).

-The block also includes a timestamp and a nonce (a random number).

Mining Process:

-Miners on the network compete to find a valid hash for the current block.

-Miners change the nonce value in the block's header and calculate the hash of the entire block.

-They repeatedly adjust the nonce and hash until they find a hash that meets the Proof of Work (PoW) criteria, typically a hash with a specific number of leading zeros.

Hashing for Proof of Work:

-Miners hash the block's header, which includes the nonce, timestamp, previous block's hash, and transaction data.

-They calculate the hash using a cryptographic hash function (e.g., SHA-256 in the case of Bitcoin).

-If the resulting hash doesn't meet the PoW criteria, they increment the nonce and hash again, repeating this process until they find a valid hash.

Difficulty Adjustment:

-The network adjusts the PoW difficulty level periodically to maintain a consistent block creation rate.

-If blocks are being mined too quickly, the difficulty increases, making it harder to find a valid hash.

-If blocks are being mined too slowly, the difficulty decreases, making it easier to find a valid hash.

Consensus:

-Once a miner finds a valid hash that satisfies the PoW criteria, they broadcast the new block to the network.

-Other nodes in the network receive the block, verify its validity by checking transactions and the hash, and confirm that it meets the PoW criteria.

-If the block is valid, it is added to the blockchain, and the miner is rewarded with cryptocurrencies and transaction fees.

Immutability:

- Once added to the blockchain, a block is extremely difficult to alter because changing any part of the block would require recalculating its hash.

- Since each block references the hash of the previous block, changing one block would necessitate changing all subsequent blocks, which is computationally infeasible.

This process repeats for each new block in the blockchain, ensuring the security, integrity, and immutability of the blockchains transaction history through the use of cryptographic hashing and the Proof of Work mechanism.