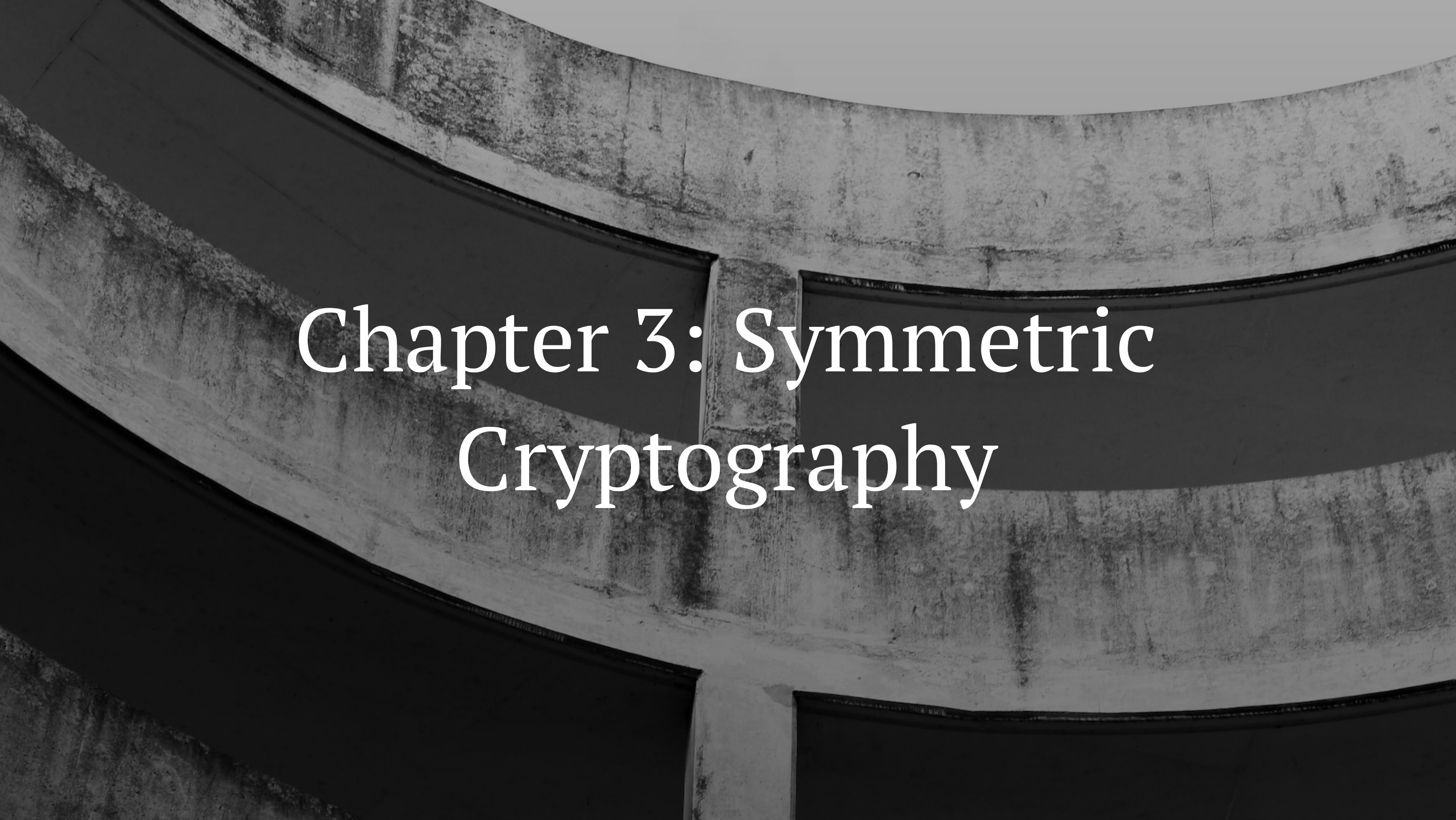


Chapter 3: Symmetric Cryptography

Introduction: What is Cryptography?

> **Cryptography is the science of making information secure in the presence of adversaries.**

- It does so under the assumption that <u>limitless</u> resources are available to adversaries.

> *Cryptography is the science of making information secure in the presence of adversaries.*

What Does Cryptography Look Like?

In this diagram, P, E, C, and D represent plaintext, encryption, ciphertext, and decryption, respectively. This model utilizes concepts such as entities, senders, receivers, adversaries, keys, and channels.

Entity: Either a person or system that sends, receives, or performs operations on data

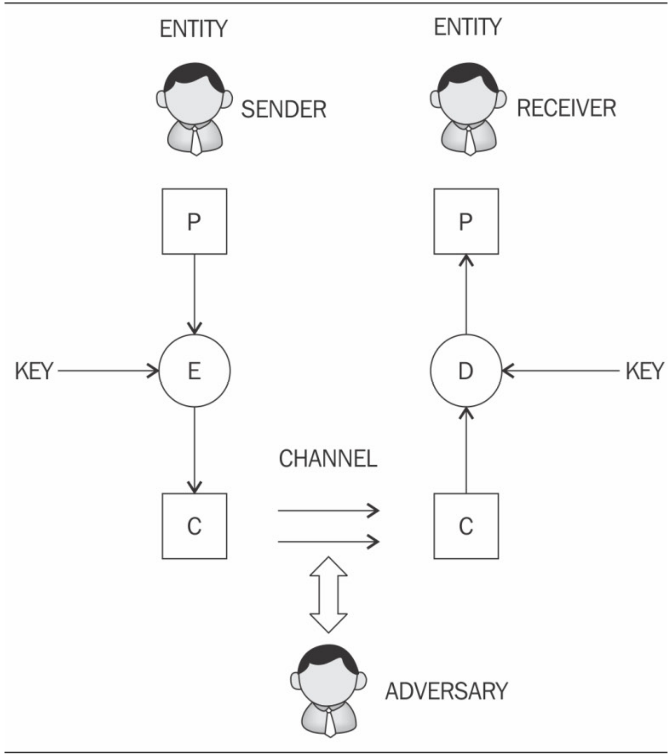
Sender: This is an entity that transmits the data

Receiver: This is an entity that takes delivery of the data

Adversary: This is an entity that tries to circumvent the security service

Key: A key is data that is used to encrypt the plaintext and also to decrypt the ciphertext

Channel: A channel provides a medium of communication between entities



What is Cryptography Used For?

Cryptography is primarily used to provide a **confidentiality** service. On its own, it cannot be considered a complete solution, rather it serves as a crucial building block within a more extensive security system to address a security problem.

> **Confidentiality** is the assurance that information is only available to authorized entities.

In addition to a confidentiality service, cryptography also provides other security services such as **integrity**, **authentication** (*entity authentication* and *data origin authentication*), and **non-repudiation**. Additionally, accountability is also provided, which is a requirement in many security systems.

> **Integrity** is the assurance that information is modifiable only by authorized entities.

Entity Authentication and Data Origin Authentication

Entity Authentication

> **Entity authentication** is the assurance that an entity is currently involved and active in a communication session.

There are various authentication practices, such as **single-factor authentication** (such as authentication via a username/password combo) and it’s more secure version **multi-factor authentication**, which consists of additional techniques for user identification (such as confirming a session on your phone as well).

Authentication Methods:

1. Using *something you have* (such as a hardware token or a smart card)

- A user who has access to the hardware token and/or knows login credentials will be able to access the system.

2. Using *something you are* (such as the utilization of biometric features)

- In this way it can be ensured that the user was indeed present during the authentication process, as biometric features are unique to every individual. However, careful implementation is required to guarentee a high level of security, as some research has suggested that biometric systems can be circumvented under specific conditions.

Authentication: Data Origin Authentication and Accountability

Data Origin Authentication

> Also known as **message authentication**, **data origin authentication** is the assurance that the source of the information is indeed verified.

- Data origin authentication guarantees data integrity because, if a source is corroborated, then the data must not have been altered.
- Various methods, such as **Message Authentication Codes (MACs)** and **digital signatures** are common methods for data origin authentication, which will be explored later on in the chapter.

Accountability

> **Accountability** is the assurance that actions affecting security can be traced back to the responsible party.

- This is usually done via logging and audit mechanisms in systems where detailed audits are required due to the nature of the business (such as in electronic trading systems)

Non-Repudiation

> **Non-repudiation** is the assurance that an entity cannot deny a previous commitment or action by providing undeniable or undisputable proof.

- This property is essential in debatable situations whereby an entity has denied the actions performed, for example, the placement of an order on an e-commerce system.
- Such as in electronic transactions, so in case of disputes, it can be used as a "receipt" of sorts.

The primary requirements of a non-repudiation protocol are **fairness**, **effectiveness**, and **timeliness**.

A lot of the time, multiple participants are involved in a transaction, and two-party non-repudiation protocols are no longer useful.

- To address this problem, **multi-party non-repudiation (MNPR)** protocols have been developed.

> **MNPR** refers to non-repudiation protocols that run between multiple parties instead of the traditional two parties.

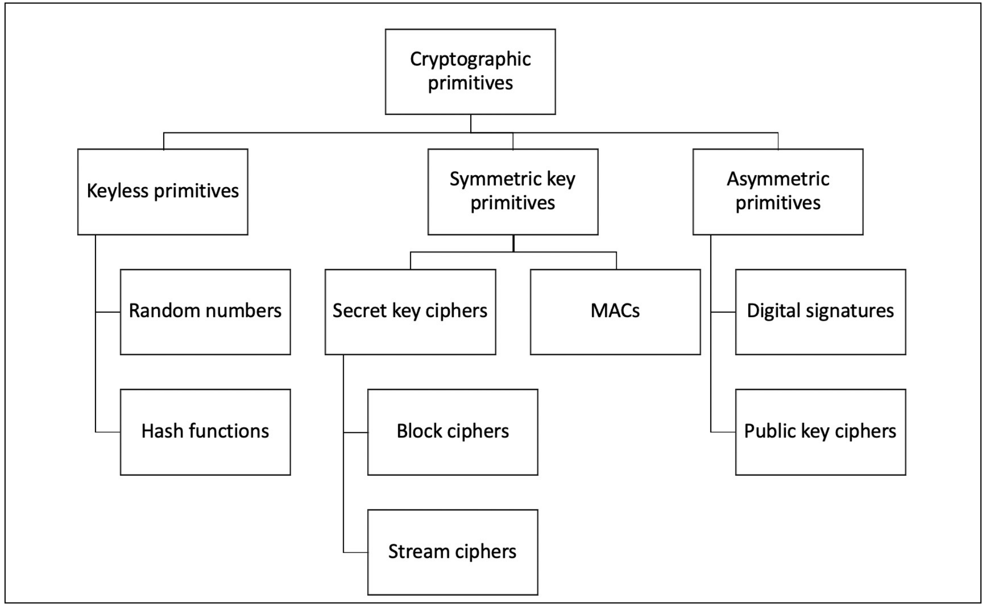
Cryptographic Primitives

A thing of note here is that to provide all of the aforementioned services, different *cryptographic primitives* are used, which are essential for building secure protocols and systems.

> **Cryptographic primitives** are the basic building blocks of a security protocol or system.

> A **security protocol** is a set of steps taken to achieve the required security goals by utilizing appropriate security mechanisms.

Cryptographic Primitices (Cont.)



As shown in this diagram, cryptography is mainly divided into three categories: **keyless primitives**, **symmetric key primitives, or cryptography** , and **asymmetric key primitives, or cryptography**.

- *In this chapter we'll be discussing keyless and symmetric key primitives, and in the next chapter we'll talk about asymmetric key primitives.*

Keyless Primitive: Randomness

Randomness provides an indispensable element for the security of the cryptographic protocols. It is used for the generation of keys and in encryption algorithms.

Randomness ensures that operations of a cryptographic algorithm do not become predictable enough to allow cryptanalysts to predict the outputs and operations of the algorithm, which will make the algorithm insecure.

- It is quite a feat to generate suitable randomness with a high degree of uncertainty, but there are methods that ensure an adequate level of randomness is generated for use in cryptographic algorithms.

There are two categories of source of randomness, namely, **Random Number Generators (RNGs)** and **Pseudorandom Number Generators (PRNGs)**.

Random Number Generators (RNGs)

RNGs are software or hardware systems that make use of the randomness available in the real world, that is, the analog world, where uncertainty in the environment produces randomness.

- Some sources include temperature variations, thermal noises from various electronic components, acoustic noise, or even randomness generated on the computer via running processes such as keystrokes or disk movements.
- This is what is called *real* randomness.
- Not very practical due to the difficulty of acquiring the data or not having enough entropy. Additionally, sources might not be always available.

Entropy is the measure of randomness.

Pseudo Random Number Generators (PRNGs)

> **PRNGs** are deterministic functions that work on the principle of using a random initial value called a seed to produce a random looking set of elements.

- Commonly used to generate keys for encryption algorithms
- Better alternative to RNGs due to their reliability and deterministic nature.

Hash Functions

Another incredibly important keyless primitive are *hash functions*. Hash functions aren't used for encrypting data, but instead produce a fixed-length digest of the data it is provided as input. They are **very** important in the blockchain.

More formally:

> **Hash functions** are used to create fixed-length digests of arbitrarily long input strings.

- Hash functions are keyless, and they provide a data integrity service.
- They are usually built using iterated and dedicated hash function construction techniques.
- Various families of hash functions are available, such as MD, SHA1, SHA-2, SHA-3, RIPEMD, and Whirlpool
- In addition to blockchains & p2p networks, hash functions are commonly used for digital signatures and message authentication codes, such as HMACs.

Hash Function Properties

There are two practical and three security properties of hash functions that must be met depending on the level of integrity required.

Practical Properties

- **Compression of arbitrary messages into fixed-length digests**: This property relates to the fact that a hash function must be able to take an input text of any length and output a fixed-length compressed message. Hash functions produce a compressed output in various bit sizes, usually between 128-bits and 512-bits.

- **Easy to compute**: Hash functions are efficient and fast one-way functions. It is required that hash functions be very quick to compute regardless of the message size. The efficiency may decrease if the message is too big, but the function should still be fast enough for practical use.

Hash Function Security Properties

- **Pre-image resistance**:

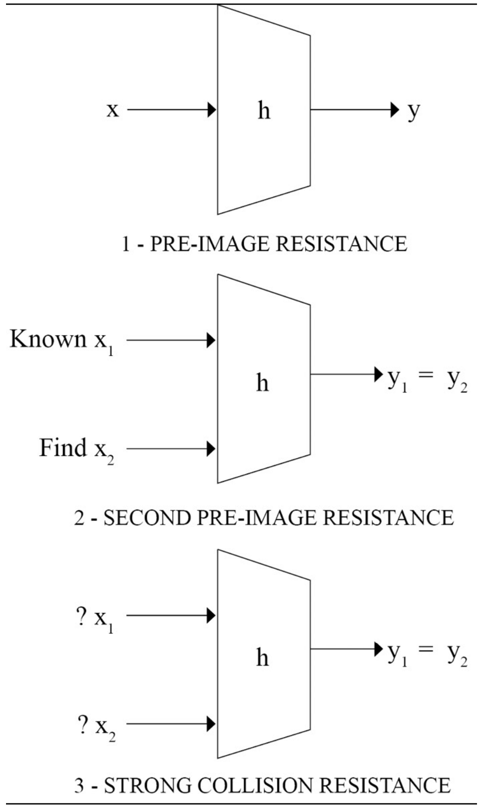
Also called the *one-way property*, it states that given the output of the hash function, it should be impossible to compute what has been hashed. This property can also be explained by this equation:

$$h(x) = y$$

Here, h is the hash function, x is the input, and y is the hash. x is considered a pre-image of y, hence the name of this property.

- **Second pre-image resistance**:

Also known as *weak collision resistance*, it states that it should be almost impossible two find two distinct messages with the same hash. More formally pre-image resistance property requires that given x and $h(x)$, it is almost impossible to find any other message m, where $m \neq x$ and $h(m) = h(x)$.

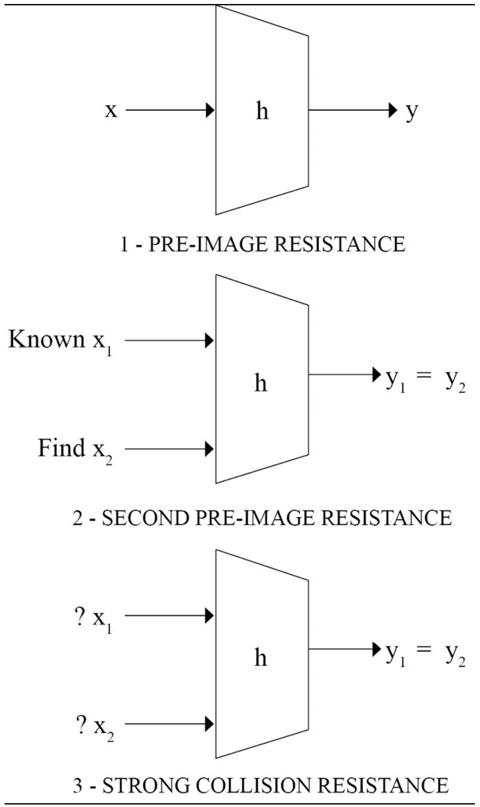


Hash Function Security Properties

- **Collision resistance**:

The collision resistance property requires that two different input messages should not hash to the same output. In other words, $h(x) \neq h(z)$.

- This property is also known as *strong collision resistance*.



Hash Functions Cont.

- *Is collision resistance guarenteed?*

Due to their very nature, hash functions will always have some collisions. This is a situation where two different messages hash to the same output. However, they should be computationally impractical to find.

There’s a concept known as the **avalanche effect** that’s very desirable in all cryptographic hash functions.

> The avalanche effect specifies that a small change, even a single character change in the input text, will result in an entirely different hash output.

Hash Functions Cont.

Hash functions are usually designed by following an *iterated hash functions* approach.

With this method, the input message is compressed in multiple rounds on a block-by-block basis in order to produce the compressed output.

A popular type of iterated hash function is the Merkle-Damgard construction.

- This construction is based on the idea of dividing the input data into equal block sizes and then feeding them through the compression functions in an iterative manner.

The collision resistance of the property of compression functions ensures that the hash output is also collision-resistant.

- Compression functions can be built using block ciphers.

In addition to Merkle-Damgard, there are various other constructions of compression functions proposed by researchers, for example, Miyaguchi-Preneel and Davies-Meyer.

Hash Function Categories

Message Digest:

- Message Digest (MD) functions were prevalent in the early 1990s.
- MD4 and MD5 fall into this category.
- Both MD functions were found to be insecure and are not recommended for use anymore.
- MD5 is a 128-bit hash function that was commonly used for file integrity checks.

Secure Hash Algorithms:

The following list describes the most common Secure Hash Algorithms (SHAs):

- **SHA-0**: This is a 160-bit function introduced by the U.S. National Institute of Standards and Technology (NIST) in 1993.
- **SHA-1**: SHA-1 was introduced in 1995 by NIST as a replacement for SHA-0. This is also a 160-bit hash function. SHA-1 is used commonly in SSL and TLS implementations. It should be noted that SHA-1 is now considered insecure, and it is being deprecated by certificate authorities. Its usage is discouraged in any new implementations.

Hash Function Categories

SHA Continued:

- **SHA-2**: This category includes four functions defined by the number of bits of the hash: SHA-224, SHA-256, SHA-384, and SHA-512.
- **SHA-3**: This is the latest family of SHA functions. SHA3-224, SHA3-256, SHA3-384, and SHA3-512 are members of this family. SHA3 is a NIST-standardized version of *Keccak*. Keccak uses a new approach called sponge construction instead of the commonly used Merkle-Damgard transformation.

RIPEMD

- RIPEMD is the acronym for RACE Integrity Primitives Evaluation Message Digest.
- It is based on the design ideas used to build MD4.
- There are multiple versions of RIPEMD, including 128-bit, 160-bit, 256-bit, and 320-bit.

Whirlpool

- This is based on a modified version of the Rijndael cipher known as Whirlpool.
- It uses the Miyaguchi-Preneel compression function, which is a type of one-way function used for the compression of two fixed-length inputs into a single fixed-length output.
- It is a single block length compression function.

But What About Hash Functions and Blockchains?

Hash functions play a vital role in blockchain.

The Proof-of-Work (PoW) function used in the consensus mechanism in particular uses SHA-256 twice in order to verify the computational effort spent by miners. RIPEMD 160 is used to produce Bitcoin addresses.

Ethereum uses Keccak, which is the original algorithm presented to NIST, rather than NIST standard SHA-3. NIST, after some modifications, such as an increase in the number of rounds and simpler message padding, standardized Keccak as SHA-3.

Implementation details will be explored in later chapters.

SHA-256

SHA-256 has an input message size limit of $2^{64}-1$ bits. The block size is 512 bits, and it has a word size of 32 bits. The output is a 256- bit digest.

The compression function processes a 512-bit message block and a 256-bit intermediate hash value. There are two main components of this function: the compression function and a message schedule.

This algorithm works in nine steps, described in the following slides.

SHA-256 Steps

Pre-processing

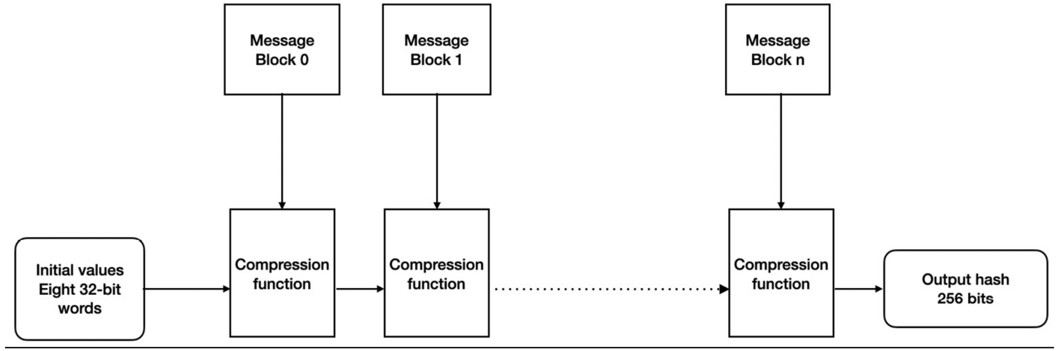
1. Padding of the message is used to adjust the length of a block to 512 bits if it is smaller than the required block size of 512 bits.
2. Parsing the message into message blocks, which ensures that the message and its padding is divided into equal blocks of 512 bits.
3. Setting up the initial hash value, which consists of the eight 32-bit words obtained by taking the first 32 bits of the fractional parts of the square roots of the first eight prime numbers. These initial values are fixed and chosen to initialize the process. They provide a level of confidence that no backdoor exists in the algorithm.

SHA-256 Steps Cont.

Hash computation

4. Each message block is then processed in a sequence, and it requires 64 rounds to compute the full hash output. Each round uses slightly different constants to ensure that no two rounds are the same.
5. The message schedule is prepared.
6. Eight working variables are initialized.
7. The compression function runs 64 times.
8. The intermediate hash value is calculated.
9. Finally, after repeating steps 5 through 8 until all blocks (chunks of data) in the input message are processed, the output hash is produced by concatenating intermediate hash values.

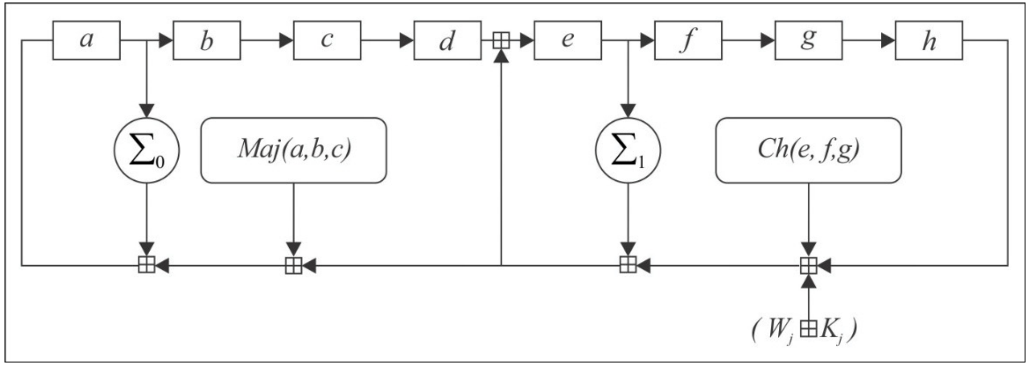
SHA-256



- As shown in the preceding diagram, SHA-256 is a Merkle Damgard construction that takes the input message and divides it into equal blocks (chunks of data) of 512 bits.
- Initial values (or initial hash values) or the initialization vector are composed of eight 32 bit words (256 bits) that are fed into the compression function with the first message.
- Subsequent blocks are fed into the compression function until all blocks are processed and finally, the output hash is produced.

SHA-256

The compression function of SHA-256 is shown in the following diagram:



- In the preceding diagram, a, b, c, d, e, f, g, and h are the registers for 8 working variables.
- Maj and Ch functions are applied bitwise.
- Σ_0 and Σ_1 perform bitwise rotation.
- The round constants are W_j and K_j, which are added in the main loop (compressor function) of the hash function, which runs 64 times.

SHA-3 (Keccak)

The structure of SHA-3 is very different from that of SHA-1 and SHA-2.

The key idea behind SHA-3 is based on unkeyed permutations, as opposed to other typical hash function constructions that used keyed permutations.

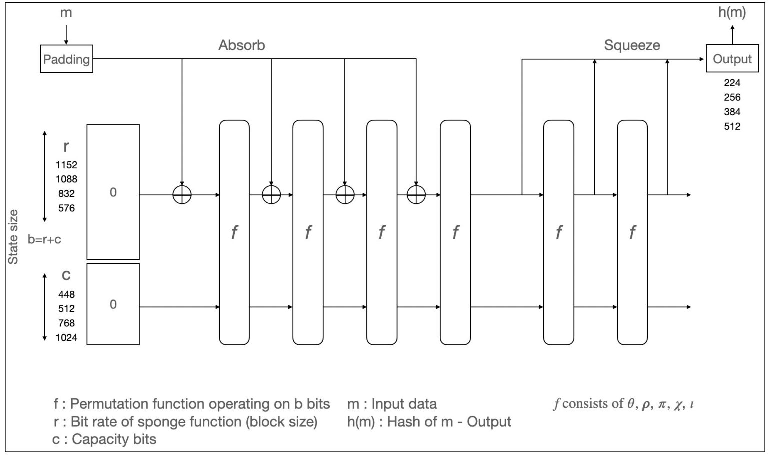
Keccak also does not make use of the Merkle-Damgard transformation that is commonly used to handle arbitrary-length input messages in hash functions.

A newer approach, called *sponge and squeeze construction*, is used in Keccak. It is a random permutation model.

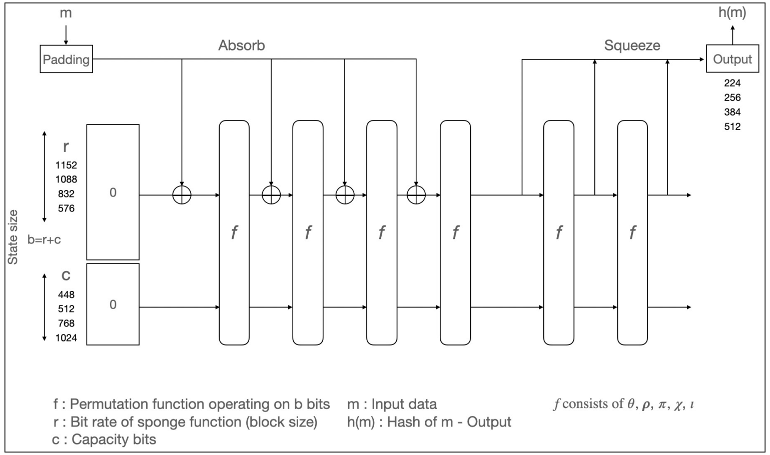
SHA-3 (Keccak) Cont.

The following diagram shows the sponge and squeeze model, which is the basis of SHA-3 or Keccak.

- Analogous to a sponge, the data (m input data) is first absorbed into the sponge after applying padding.
- It is then changed into a subset of permutation state using XOR (exclusive OR), and then the output is squeezed out of the sponge function that represents the transformed state.
- The rate r is the input block size of the sponge function, while capacity c determines the security level:


m
h(m)
Padding
Absorb
Squeeze
Output
224
256
384
512
r
1152
1088
832
576
0
State size
b=r+c
c
448
512
768
1024
0
f
f
f
f
f
f
f : Permutation function operating on b bits
r : Bit rate of sponge function (block size)
c : Capacity bits
m : Input data
h(m) : Hash of m - Output
f consists of θ, ρ, π, χ, ι

SHA-3 (Keccak) Cont.


m
h(m)
Padding
Absorb
Squeeze
Output
224
256
384
512
r
1152
1088
832
576
State size
b=r+c
c
448
512
768
1024
0
0
f
f : Permutation function operating on b bits
r : Bit rate of sponge function (block size)
c : Capacity bits
m : Input data
h(m) : Hash of m - Output
f consists of θ, ρ, π, χ, ι

- In the preceding diagram, state size b is calculated by adding bit rate r and capacity bits c.
- r and c can be any values as long as sizes of $r + c$ are 25, 50, 100, 200, 400, 800, or 1600.
- The state is a 3-dimensional bit matrix.
- The initial state is set to 0.
- The data m is entered into the absorb phase block by block via XOR after applying padding.

SHA-3 (Keccak)

The following table shows the value of bit rate r (block size) and capacity c required to achieve the desired output hash size under the most efficient setting of $r + c = 1600$:

r (block size)	**c (capacity)**	**Output Hash Size**
1152	448	224
1088	512	256
832	768	384
576	1024	512

The function f is a permutation function consisting of five transformation operations, which are beyond the scope of this class.

- The key idea behind applying these transformations is to achieve the avalanche effect
- Said five transformation operations put together form a *round*. In the SHA-3 standard, the number of rounds is 24 to achieve the desired level of security.

Symmetric Cryptography

> **Symmetric cryptography**, also known as *shared key cryptography* or *secret key cryptography*, refers to a type of cryptography where the key that is used to encrypt the data is the same one that is used for decrypting the data.

A key, more precisely, a *cryptographic* key, is required for the encryption and decryption process and must be kept secret, hence it is called a secret key.

There are many different types of keys depending upon the protocol.

- In symmetric key systems, only one key is required, which is shared between the sender and the receiver. It is used for both encryption and decryption, hence the name symmetric key cryptography or shared key cryptography, or just symmetric cryptography.
- Other types of keys are *public keys* and *private keys*, which are generated in pairs for public key cryptography or asymmetric cryptography. Public keys are used for encrypting the plaintext, whereas private keys are used for decryption and are expected to be kept secret by the receiver.

Asymmetric cryptography will be discussed in the next chapter.

Symmetric Keys

Keys can also be *ephemeral (temporary)* or *static*.

- Ephemeral keys are intended to be used only for a short period of time, such as in a single session between the participants, whereas static keys are intended for long-term usage.

Another type of key is called the *master key*, which is used for the protection, encryption, decryption, and generation of other keys.

Key Generation

There are different methods to generate keys:

1. **Random**, where a random number generator is used to generate a random set of bytes that can be used as a key.

2. **Key derivation-based**, where a single key or multiple keys are derived from a password. A key derivation function (KDF) is used for this purpose, taking the password as input, and converting that into a key. Commonly used key derivation functions are Password-Based Key Derivation Function 1 (PBKDF1), PBKDF2, Argon 2, and Scrypt.

3. Using a **key agreement protocol**, two or more participants run a protocol that produces a key and that key is then shared between participants. In key agreement schemes, all participants contribute equally in the effort to generate the shared secret key. The most commonly used key agreement protocol is the Diffie-Hellman key exchange protocol.

Key Generation (Cont.)

4. In **encryption schemes**, there are also some random numbers that play a vital role in the operation of the encryption process. These concepts are presented as follows:

- **The nonce**: This is a number that can be used only once in a cryptographic protocol. It must not be reused. Nonces can be generated from a large pool of random numbers or they can also be sequential. The most common use of nonces is to prevent replay attacks in cryptographic protocols.
- The **initial value** or **initialization vector (IV)** is a random number, which is basically a nonce, but it must be chosen in an unpredictable manner. This means that it cannot be sequential. IVs are used extensively in encryption algorithms to provide increased security.
- **The salt**: Salt is a cryptographically strong random value that is typically used in hash functions to provide defense against dictionary or rainbow attacks. Using dictionary attacks, hashing-based password schemes can be broken by trying hashes of millions of words from a dictionary in a brute-force manner and matching it with the hashed password. If a salt is used, then a dictionary attack becomes difficult to run because a random salt makes each password unique, and secondly, the attacker will then have to run a separate dictionary attack for random salts, which is quite unfeasible.

MACs

Usually, hash functions do not use a key. Nevertheless, if they are used with a key, then they can be used to create another cryptographic construct called **message authentication codes (MACs)**.

> **Message authentication codes (MACs)** are sometimes called keyed hash functions, and they can be used to provide message integrity and authentication.

More specifically, they are used to provide data origin authentication. These are symmetric cryptographic primitives that use a shared key between the sender and the receiver. MACs can be constructed using block ciphers or hash functions.

HMACs

Similar to the hash function, **hash-based MACs (HMACs)** produce a fixed-length output and take an arbitrarily long message as the input.

In this scheme, the sender signs a message using the MACand the receiver verifies it using the shared key.

The key is hashed with the message using either of the two methods known as secret prefix or secret suffix.

With the secret prefix method, the key is concatenated with the message; that is, the key comes first and the message comes afterward, whereas with the secret suffix method, the key comes after the message, as shown in the following equations:

Secret prefix: $M = MACk(x) = h(k||x)$

Secret suffix: $M = MACk(x) = h(x||k)$

There are various powerful applications of hash functions used in peer-to-peer networks and blockchain technologies, such as Merkle trees, Patricia trees, and distributed hash tables.

Encryption Algorithms

> In layman’s terms, to encipher or to encrypt is to turn regular text into code such that the resulting code can be turned back into regular text again.

There are two types of symmetric ciphers: **stream ciphers** and **block ciphers**.

Data Encryption Standard (DES) and *Advanced Encryption Standard (AES)* are typical examples of block ciphers, whereas RC4 and A5 are commonly used stream ciphers.

Stream Ciphers

> **Stream ciphers** are encryption algorithms that apply encryption algorithms on a bit-by-bit basis (one bit at a time) to plaintext using a keystream.

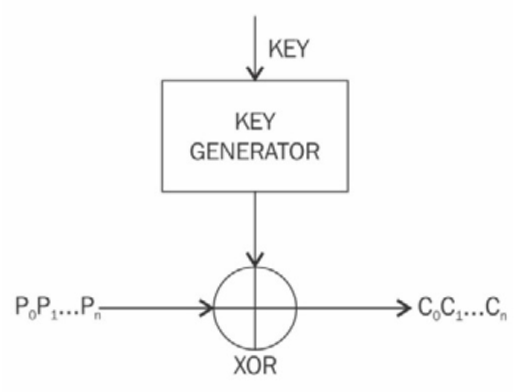
There are two types of stream ciphers: synchronous stream ciphers and asynchronous stream ciphers.

- Synchronous stream ciphers are those where the keystream is dependent only on the key.
- Asynchronous stream ciphers have a keystream that is also dependent on the encrypted data.

In stream ciphers, encryption and decryption are the same function because they are simple modulo 2 additions or XOR operations.

The fundamental requirement in stream ciphers is the security and randomness of keystreams.

Various techniques ranging from PRNGs to true hardware RNGs have been developed to generate random numbers, and it is vital that all key generators be cryptographically secure.



Block Ciphers

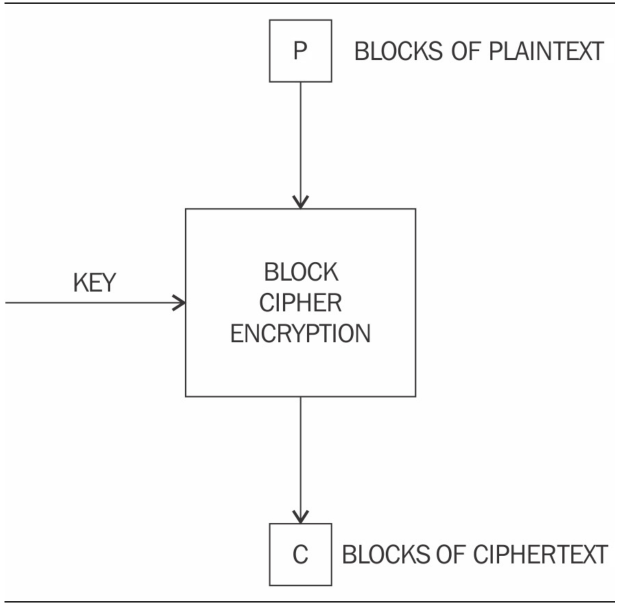
> **Block ciphers** are encryption algorithms that break up the text to be encrypted (plaintext) into blocks of a fixed length and apply the encryption block by block.

Block ciphers are generally built using a design strategy known as a Feistel cipher.

This structure is based on the idea of combining multiple rounds of repeated operations to achieve desirable cryptographic properties known as confusion and diffusion.

Feistel networks operate by dividing data into two blocks (left and right) and processing these blocks via keyed round functions in iterations to provide sufficient pseudorandom permutations.

A key advantage of using a Feistel cipher is that encryption and decryption operations are almost identical and only require a reversal of the encryption process to achieve decryption.



Confusion and Diffusion

> **Confusion** adds complexity to the relationship between the encrypted text and plaintext.

- This is achieved by substitution.
- In practice, A in plaintext is replaced by X in encrypted text. In modern cryptographic algorithms, substitution is performed using lookup tables called S-boxes.
- Confusion is required to make finding the encryption key very difficult, even if many encrypted and decrypted data pairs are created using the same key.

> The **diffusion** property spreads the plaintext statistically over the encrypted data.

- This ensures that even if a single bit is changed in the input text, it results in changing at least half (on average) of the bits in the ciphertext.
- In practice, this is achieved by transposition or permutation.

Block Cipher Operation Mode

Various modes of operation for block ciphers are **Electronic Code Book (ECB)**, **Cipher Block Chaining (CBC)**, **Output Feedback (OFB) mode**, and **Counter (CTR) mode**.

- These modes are used to specify the way in which an encryption function is applied to the plaintext.

Block Cipher Operation Modes

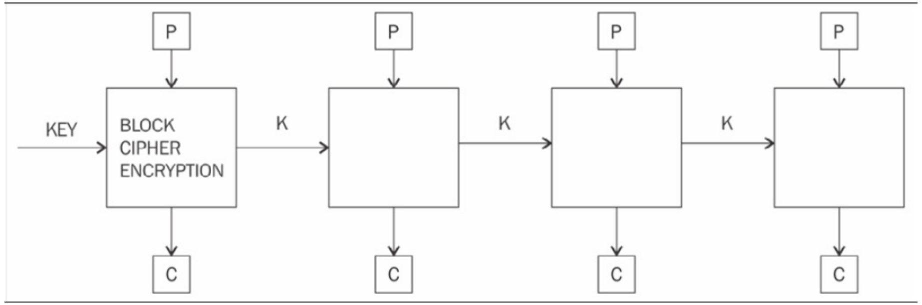
Block encryption mode

In **block encryption mode**, the plaintext is divided into blocks of fixed length depending on the type of cipher used. Then the encryption function is applied to each block. The most common block encryption modes are briefly discussed in the following sections.

Electronic codebook

Electronic codebook (ECB) is a basic mode of operation in which the encrypted data is produced as a result of applying the encryption algorithm to each block of plaintext, one by one.

This is the most straightforward mode, but it should not be used in practice as it is insecure and can reveal information:

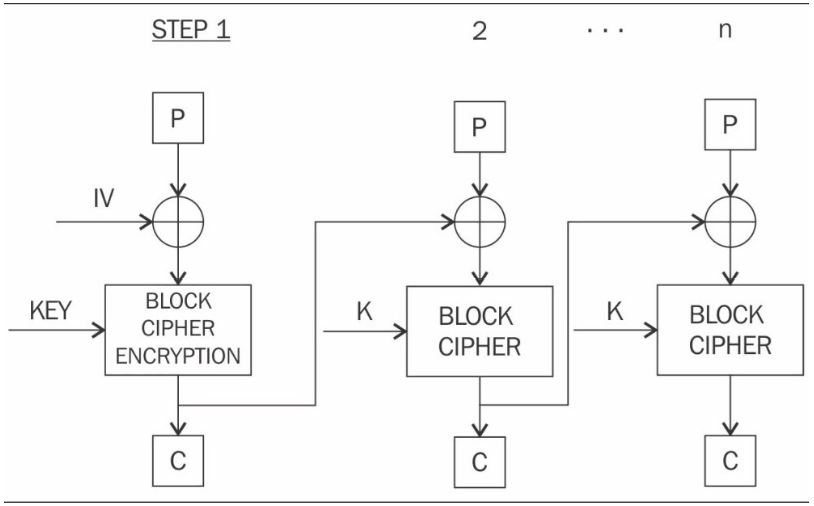


Block Cipher Operation Modes (Cont.)

Cipher block chaining

In **cipher block chaining (CBC) mode**, each block of plaintext is XORed with the previously encrypted block. CBC mode uses the IV to encrypt the first block.

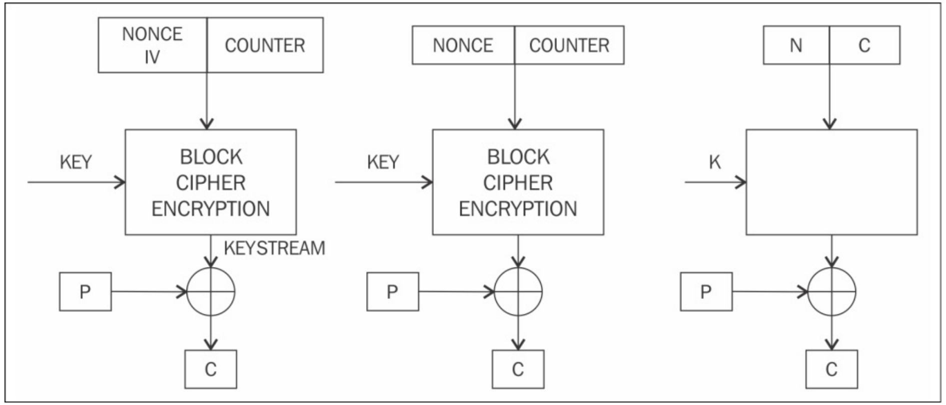
It is recommended that the IV be randomly chosen:



Block Cipher Operation Modes (Cont.)

Counter mode

The **counter (CTR) mode** effectively uses a block cipher as a stream cipher. In this case, a unique nonce is supplied that is concatenated with the counter value to produce a keystream:



CTR mode, as shown in the preceding diagram, works by utilizing a nonce (N) and a counter (C) that feed into the block cipherencryption function.

The block cipher encryption function takes the secret key (KEY) as input and produces a **keystream** (a stream of pseudorandom or random characters), which, when XORed with the plaintext (P), produces the ciphertext (C).

Message Authentication, and

In message authentication mode, a **message authentication code (MAC)** is produced from an encryption function.

> A MAC is a cryptographic checksum that provides an integrity service.

A MAC can be used to check if a message has been modified by an unauthorized entity.

This can be achieved by encrypting the message with a key using the MAC function. The resulting message and the MAC of the message, once received by the receiver, are checked by encrypting the message received, again with the key, and comparing it with the MAC received from the sender.

- If they both match, then it means that the message has not been modified by some unauthorized entity, thus an integrity service is provided.
- If they don't match, then it means that the message has been altered by some unauthorized entity during transmission.

It should also be noted that MACs work like digital signatures. However, they cannot provide a non-repudiation service due to their symmetric nature.

Advanced Encryption Standard (AES)

As the name implies, **AES** is a standard algorithm for encryption.

- Introduced in 2001 by Joan Daemen and Vincent Rijmen after an open competition, no attack has been found against AES that is more effective than the brute-force method.
- In this algorithm key sizes of 128 bits, 192 bits, and 256 bits are permissible.

Various cryptocurrency wallets use AES encryption to encrypt locally-stored data.

- Bitcoin wallets use AES-256 in CBC mode to encrypt the private keys.
- In Ethereum wallets, AES-128-CTR is used; that is, AES 128-bit in counter mode is used to encrypt the privatekey.
- Peers in Ethereum also use AES in counter mode (AES CTR) to encrypt their Peer to Peer (P2P) communication.

How AES Works

During AES algorithm processing, a 4 × 4 array of bytes known as the state is modified using multiple rounds. Full encryption requires 10 to 14 rounds, depending on the size of the key.

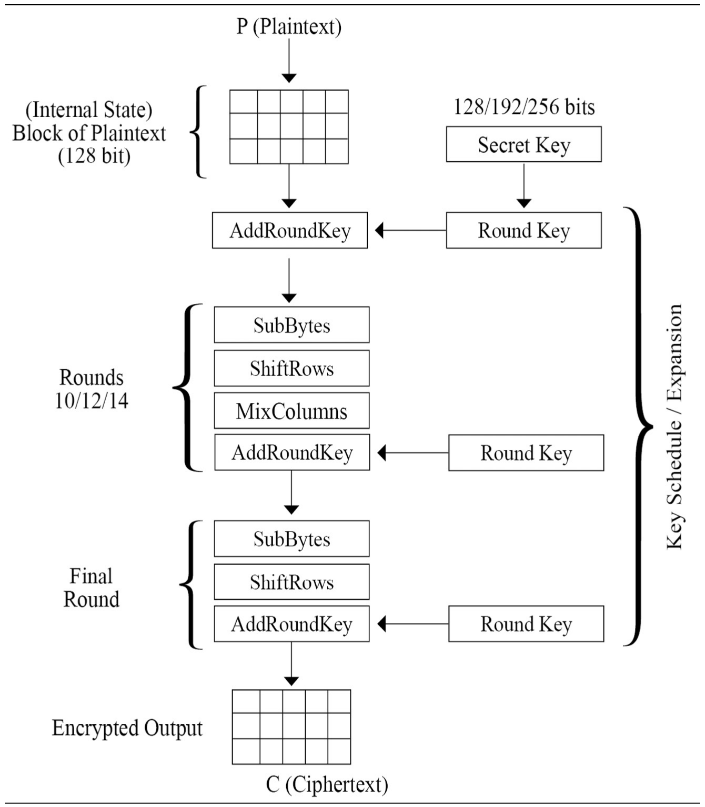
The following table shows the key sizes and the required number of rounds:

Key Size	**Number of rounds required**
128-bit	10 rounds
192-bit	12 rounds
256-bit	14 rounds

How AES Works Cont.

Once the state is initialized with the input to the cipher, the following four operations are performed sequentially step by step to encrypt the input:

1. **AddRoundKey:** In this step, the state array is XORed with a subkey, which is derived from the master key.
2. **SubBytes:** This is the substitution step where a lookup table (S- box) is used to replace all bytes of the state array.
3. **ShiftRows:** This step is used to shift each row to the left, except for the first one, in the state array in a cyclic and incremental manner.
4. **MixColumns:** Finally, all bytes are mixed in a linear fashion (linear transformation), column-wise. This is one round of AES. In the final round (either the 10th, 12th, or 14th round, depending on the key size), stage 4 is replaced with AddRoundKey to ensure that the first three steps cannot be simply reversed, as shown in this diagram.


P (Plaintext)
(Internal State) Block of Plaintext (128 bit)
128/192/256 bits
Secret Key
AddRoundKey
Round Key
Rounds 10/12/14
SubBytes
ShiftRows
MixColumns
AddRoundKey
Round Key
Final Round
SubBytes
ShiftRows
AddRoundKey
Round Key
Key Schedule / Expansion
Encrypted Output
C (Ciphertext)