Kriptográfia

December 3, 2025

1 Information security

Az információ és az információs rendszerek védelme a jogosulatlan hozzáféréstől, felhasználástól, felfedéstől, megzavarástól, módosítástól vagy megsemmisítéstől, a bizalmasság, a sértetlenség és a rendelkezésre állás biztosítása érdekében

Three basic property:

- **Confidentiality**: Secret or private information is only available for the permitted entities. This must be met when the data is being processed, transmitted and stored.
- **Integrity**: It consists of two concepts:
  - *Data Integrity*: The data is free of unauthorized modifications, when it is being transmitted, stored or processed
  - *System Integrity*: The system is free of unauthorized modifications
- **Availability**: The system is available for the entitled entities in the corresponding time and for the needed duration

Furthermore:

- **Accountablity (Nyomon kovethetoseg)**: The acts of a given entity can be traced back to the entity.
- **Assurance (biztositek)**: The assurance is the base of the trust, that the security measures (technical, operative) function as intended in order to protect the system and the informations proccessed by it.

2 Simmetric Encryption Schemes

A private-key encrtyption scheme consists of three randomized, polinomial type algorithm (Gen, Enc, Dec) so:

- *Gen* key generation algorithm with 1^n security parameter, and outputs the secret key $k \in \mathcal{K}$ ($k \leftarrow Gen(1^k)$).
- *Enc* encryption randomized algorithm with input k and open message $m \in \mathcal{M}$, and outputs cypertext $c \in \mathcal{C}$ ($c \leftarrow Enc_k(m)$).
- *Dec* decryption deterministic algorythm has the inputs key k and cypertext c and outputs the original m message or error symbol $\perp$. ($m \leftarrow Dec_k(c)$).

Neccessary condition, that $Dec_k(Enc_k(m)) = m$.

3 Blockciphers and Stream Ciphers

3.1 Data Encryption Standard (DES)

Properties:

- Implementation of the *Fiestel Cipher*
- Uses 16 round Fiestel structure.
- Block size: 64 bit
- Key length: 64 bit
- Effective key length: 54 bit.

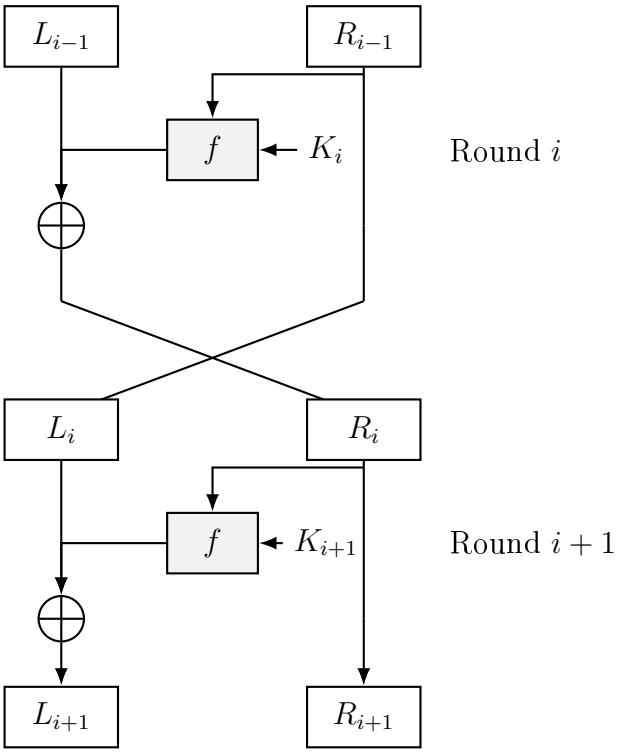

Figure 1: Two rounds of a Feistel Network structure. Note how R_{i-1} is preserved to become L_i, while L_{i-1} is modified by the round function.

3.1.1 Round Key Generation

1. **Parity Drop (PC-1)**: The process starts with a 64-bit key. However, every 8th bit is a parity bit and is ignored. The remaining 56 bits are shuffled according to "Permuted Choice 1" (PC-1).

2. **Splitting**: The 56-bit result is split into two 28-bit halves: C_0 and D_0.

3. **Circular Shifts (Left Rotations)**: In every round, both halves are rotated left by either 1 or 2 bits.

   - 1 bit rotation in rounds: 1, 2, 9, 16.
   - 2 bit rotation in all other rounds

4. **Compression (PC-2)**: After shifting, the two halves (56 bits total) are glued back together and passed through "Permuted Choice 2" (PC-2). This selects only 48 specific bits to create the Round Key (K_i).

5. **Loop**: The shifted halves (C_i, D_i) are used as the starting point for the next round's shift.

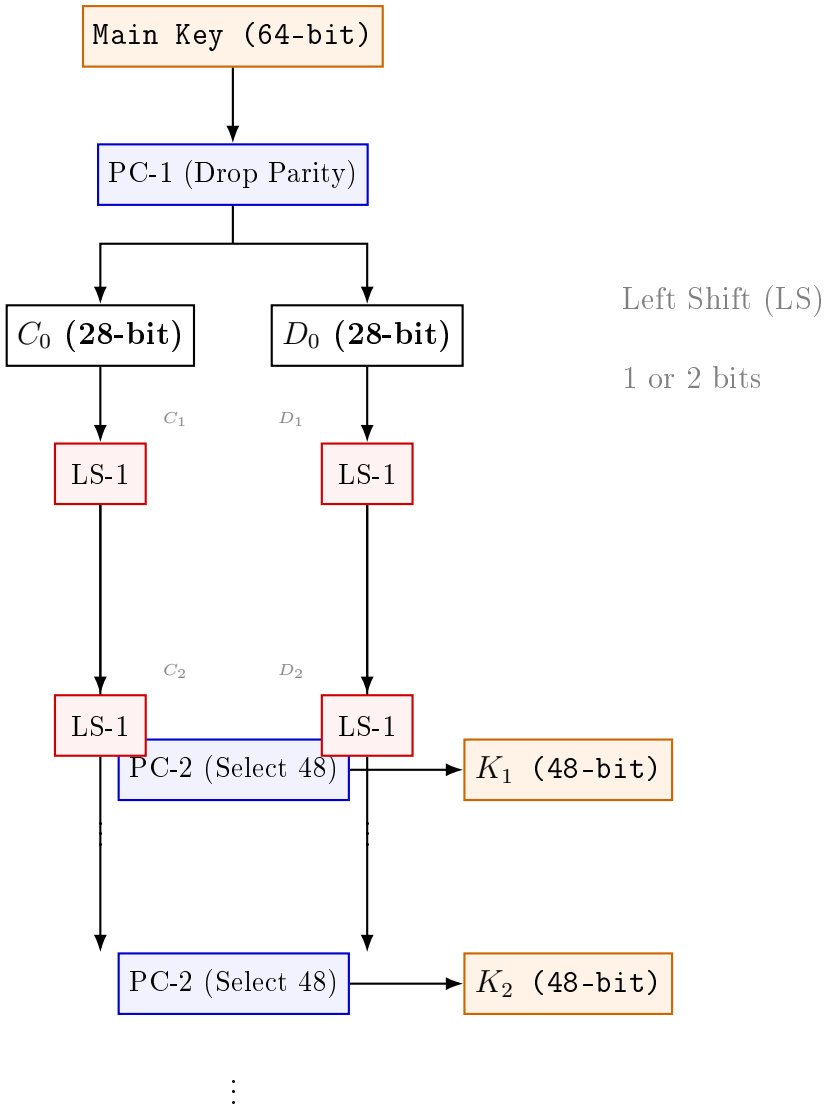

Figure 2: The DES Key Schedule. The 56-bit key state (C, D) is rotated left every round. The PC-2 permutation compresses the state to produce the 48-bit round key.

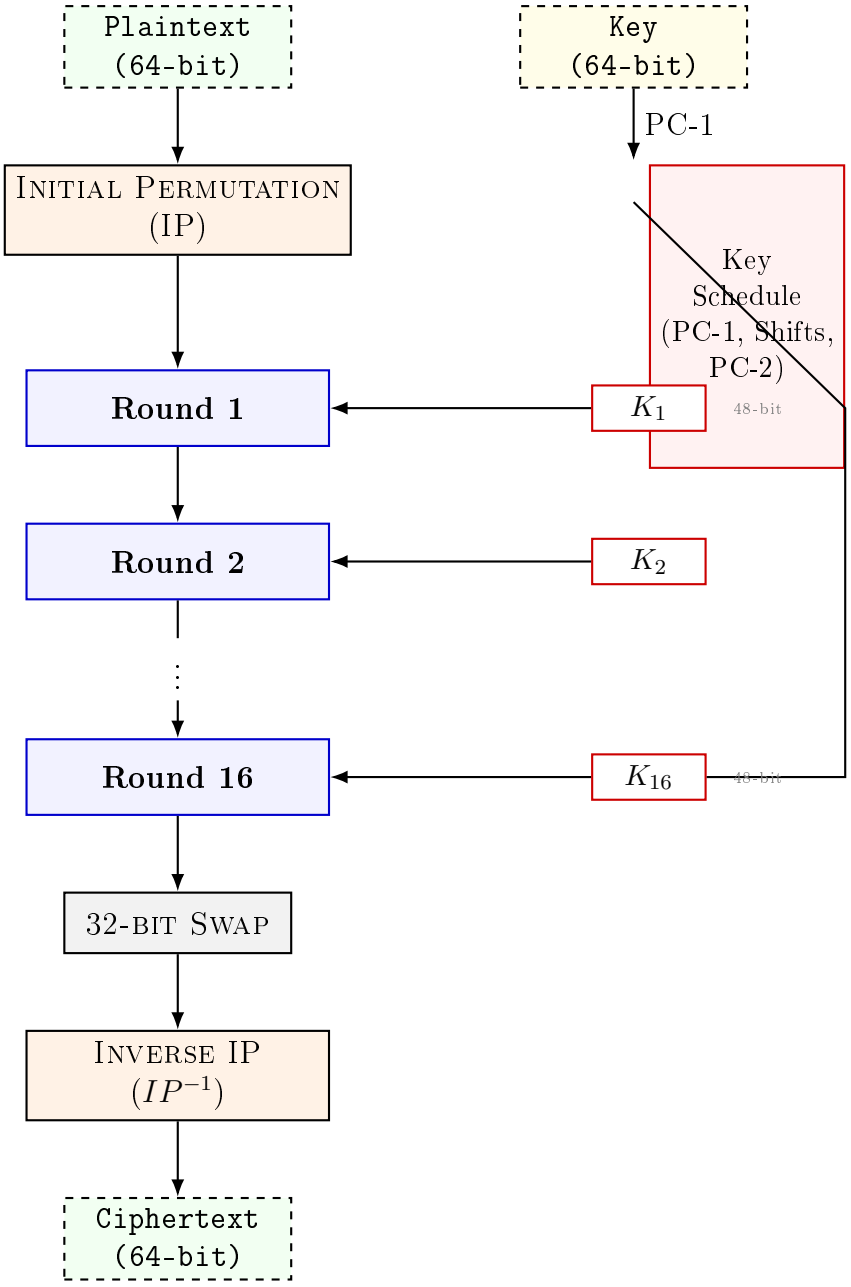

Figure 3: Overview of the Full DES Algorithm. The 64-bit Plaintext undergoes an Initial Permutation, 16 Rounds of processing (each using a unique 48-bit subkey derived from the main 64-bit Key), a final swap, and an Inverse Permutation.

4 Advanced Encryption Standard (AES)

- Block size: 128 bits
- Key Sizes: 128, 192, 256 bits
- Number of processing rounds depends on the key length:
  - 128: 10 rounds
  - 192: 12 rounds

– 256: 14 rounds

AES treats data blocks as 4x4 matrix of bytes → **State**.

4.1 1. SubBytes (Substitution)

This is the only non-linear transformation in the cipher. Each byte in the state is independently replaced by another byte from a fixed 8-bit Lookup Table (S-Box).

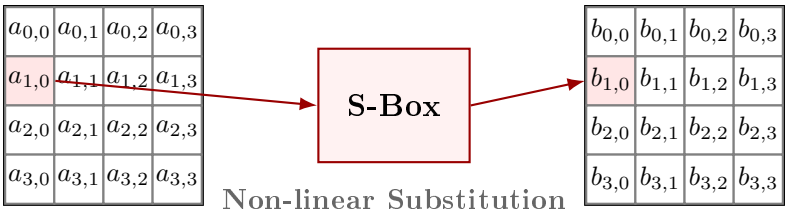

Figure 4: SubBytes: Each byte is replaced individually.

4.2 2. ShiftRows (Permutation)

This step provides diffusion among the columns. The rows of the state are shifted cyclically to the left.

- **Row 0:** Not shifted.
- **Row 1:** Shifted left by 1 byte.
- **Row 2:** Shifted left by 2 bytes.
- **Row 3:** Shifted left by 3 bytes.

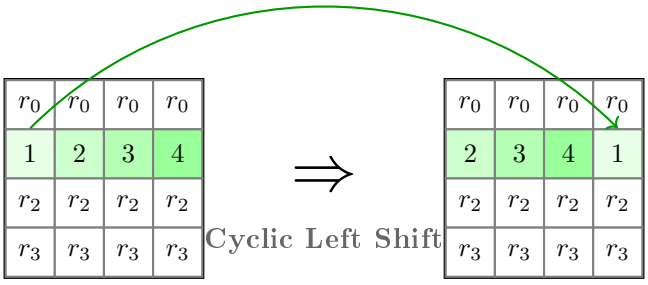

Figure 5: ShiftRows: Row 1 is shifted left by 1. Byte '1' wraps around to the end.

4.3 3. MixColumns (Diffusion)

This transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are multiplied by a fixed matrix over $GF(2^8)$. This step is omitted in the final round.

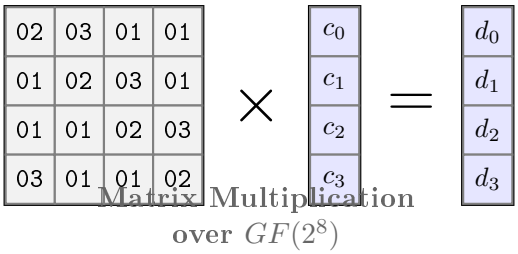

Figure 6: MixColumns: Each column is transformed linearly.

4.4 4. AddRoundKey (Key Mixing)

A Round Key is combined with the State by a simple bitwise XOR operation. This is the encryption step that injects the secret key into the state.

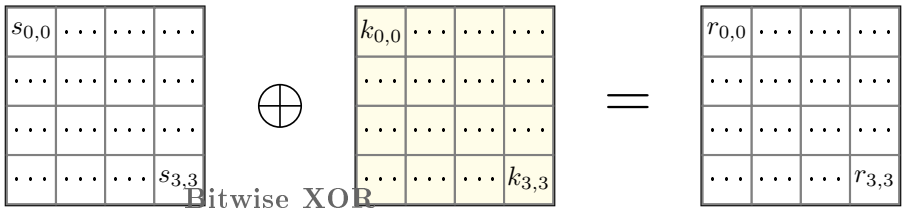

Figure 7: AddRoundKey: The internal State is XORed with the Round Key.

AES requires a separate 128-bit key for each round (plus one initial key). This means for AES-128, we need 11 keys ($4 \times 11 = 44$ words).

4.5 Expansion Algorithm

The expansion uses a recursive logic. Let W_i be the i-th 32-bit word of the key schedule:

1. If $i < 4$ (First 4 words): $W_i = \text{InputKeyWord}_i$.
2. If $i \geq 4$:
   - If i is a multiple of 4: $W_i = W_{i-4} \oplus g(W_{i-1})$.
   - Otherwise: $W_i = W_{i-4} \oplus W_{i-1}$.

4.6 The $g()$ Function

For every 4th word, a complex transformation $g()$ is applied to ensure non-linearity and break symmetries:

1. **RotWord:** A 1-byte left circular shift. $[b_0, b_1, b_2, b_3] \rightarrow [b_1, b_2, b_3, b_0]$.
2. **SubWord:** Apply the S-Box to each of the 4 bytes.
3. **Round Constant (Rcon):** XOR the first byte with $Rcon_j$ (where j is the round number).

Round (j)	1	2	3	4	5	6	7	8	9	10
Rcon$_j$ **(hex)**	01	02	04	08	10	20	40	80	1B	36

Table 1: Rcon values for each round (only affects the first byte of the word).

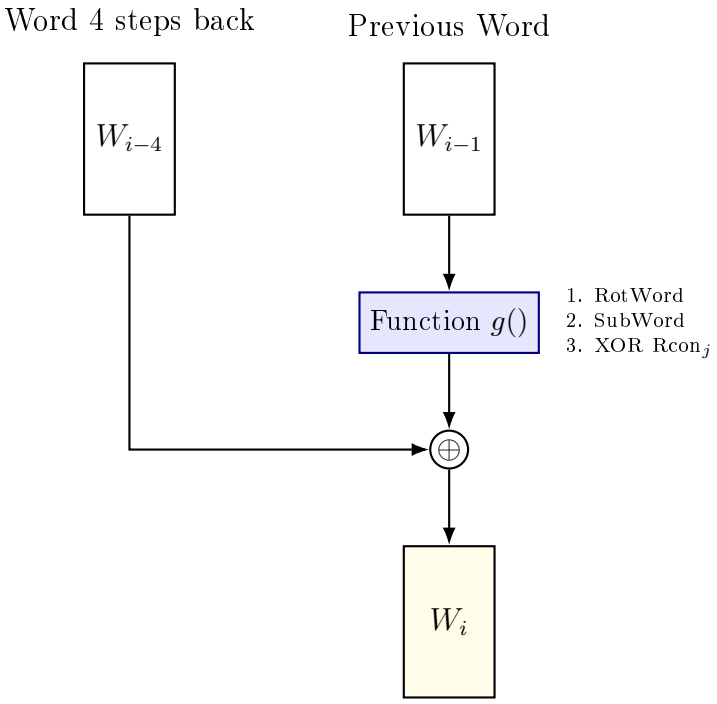

Figure 8: Key Expansion Logic (for indices i multiple of 4).

4.7 Complete Process

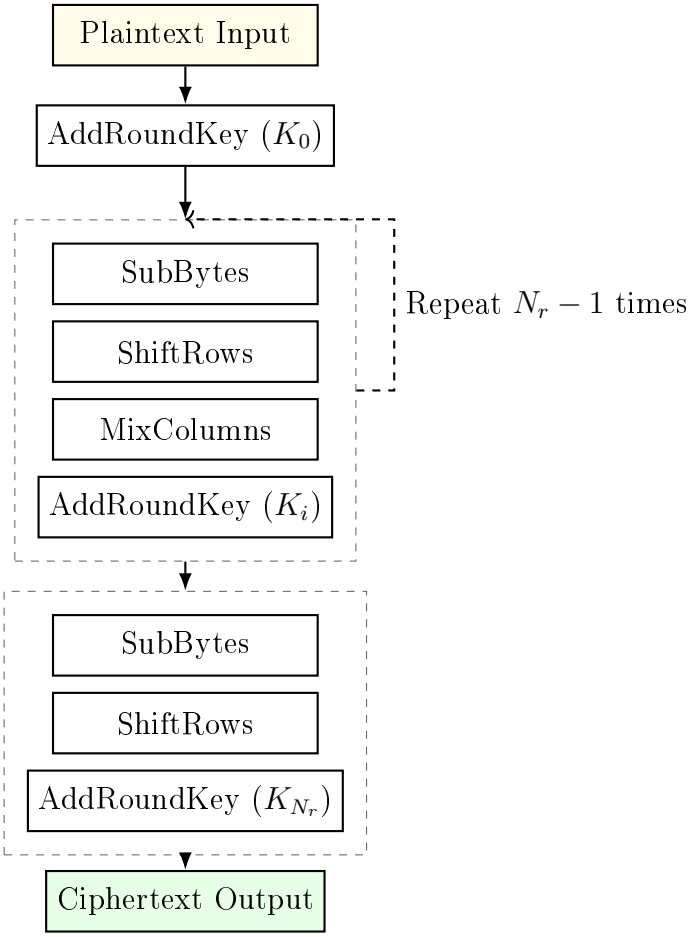

Figure 9: AES Execution Flow.

5 AES modes

5.1 ECB (Electronic Codebook)

The simplest mode. Each block is encrypted independently.

- **Weakness:** Identical plaintext blocks produce identical ciphertext blocks. This preserves patterns (e.g., the famous "Penguin" image remains visible).
- **Use Case:** Almost never recommended.

5.2 CBC (Cipher Block Chaining)

Each block of plaintext is XORed with the previous ciphertext block before being encrypted. This randomizes the output.

- **IV (Initialization Vector):** Required for the first block to ensure uniqueness.
- **Sequential:** Cannot parallelize encryption (must wait for previous block).

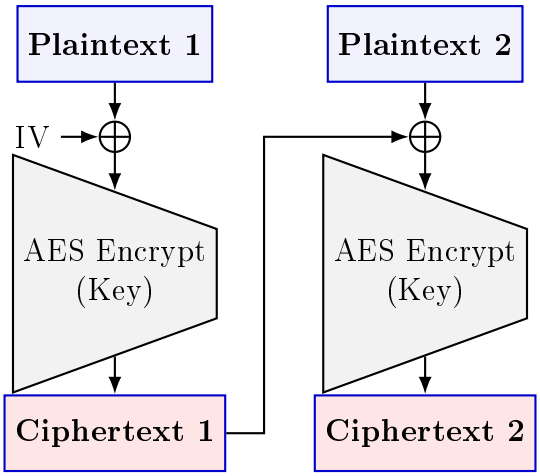

Figure 10: Cipher Block Chaining (CBC) Mode.

5.3 CTR (Counter Mode)

Turns the block cipher into a *Stream Cipher*. It generates a keystream by encrypting a counter, which is then XORed with the plaintext.

- **Parallelizable:** Blocks can be encrypted simultaneously.
- **No Padding:** The ciphertext is exactly the same length as the plaintext.

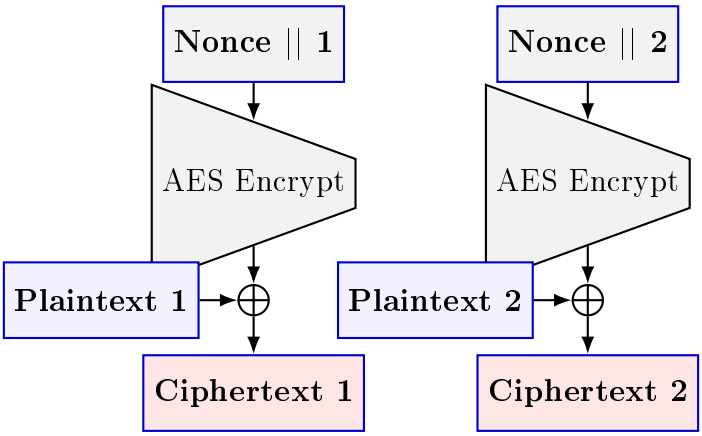

Figure 11: Counter (CTR) Mode.

5.4 GCM (Galois/Counter Mode)

Currently the most popular mode (used in HTTPS/TLS). It combines **CTR mode** for encryption with a Galois Field multiplication for **Authentication** (integrity).

- Provides both confidentiality and authenticity.
- Highly efficient (parallelizable).

6 Block ciphers vs Stream ciphers

6.1 Block Ciphers

A block cipher breaks the plaintext into fixed-size chunks called **blocks** (e.g., 64 bits for DES, 128 bits for AES). It then encrypts each block independently (or chained together) using the secret key.

Characteristics

- **Fixed Input Size:** Data must be padded if it doesn't fit the block size perfectly.
- **Complex Internals:** Uses "Confusion and Diffusion" (Substitution-Permutation Networks or Feistel Networks).
- **Modes of Operation:** Requires a mode (like CBC, GCM) to securely encrypt messages longer than one block.

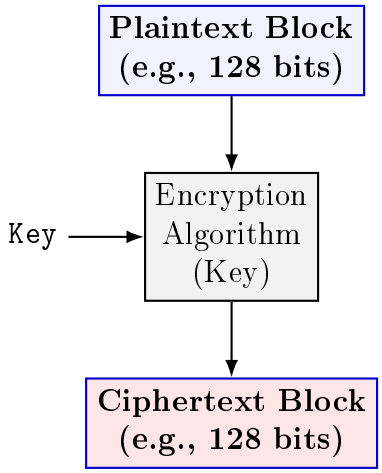

Figure 12: Block Cipher Concept: Transforming a whole chunk at once.

6.2 Stream Ciphers

A stream cipher processes data **one bit (or byte) at a time**. It functions by generating a pseudorandom stream of bits (called the *Keystream*) from the secret key and XORing it with the plaintext.

Characteristics

- **Continuous Flow:** No padding required; ciphertext is the same length as plaintext.
- **Speed:** Generally faster and simpler in hardware than block ciphers.
- **Simplicity:** Often just XOR operations combined with a state update function.
- **Error Propagation:** A bit-flip error in ciphertext affects only that specific bit in plaintext (unlike block ciphers where it ruins the whole block).

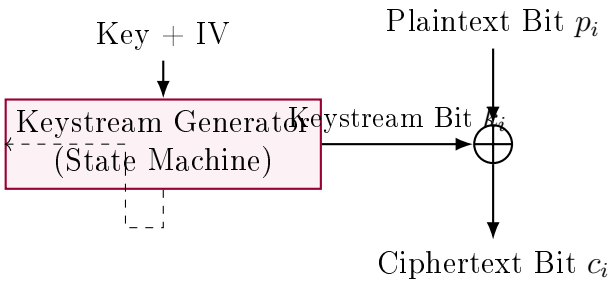

Figure 13: Stream Cipher Concept: XORing data with a running keystream.

6.3 Comparison Summary

Feature	Block Cipher	Stream Cipher
Unit of Processing	Large Blocks (64/128 bits)	Bits or Bytes (1 at a time)
Examples	AES, DES, 3DES, Blowfish	RC4, ChaCha20, Salsa20
Complexity	High (Complex transformations)	Low (XOR + State update)
Speed	Slower (unless hardware accelerated)	Very Fast
Error Propagation	One bit error corrupts whole block	One bit error affects only one bit
Padding	Often required	Not required
Primary Use	File storage, SSL/TLS, Database encryption	Real-time communications, Mobile, IoT

Table 2: Key differences between Block and Stream ciphers.

7 Assimetric Encryption

An assymetric encryption scheme is consists of three, polinomial time functions (Gen, Enc, Dec) so:

- Gen **Key Generation Algorithm**: Gen takes 1^n security parameter as input and outputs the keypair (pk, sk), whereas the pk is the public key and sk is the secret key. We assume that sk and pk has the size of at least n, and n can be defined from (pk, sk). $(pk, sk) \leftarrow Gen(1^n)$
- Enc **Encryption Algorithm**: Enc takes pk and $m \in \mathcal{M}$ as inputs, and outputs $c \in \mathcal{C}$ ciphertext. $c \leftarrow Enc_{pk}(m)$
- Dec **Decryption Algorithm**: Dec takes sk and $c \in \mathcal{C}$ as inputs, and outputs the original message m or returns error symbol $\perp$. $m \leftarrow Dec_{sk}(c)$

We require that $Dec_{sk}(Enc_{pk}(m)) = m$

7.1 The RSA Problem

The security of the RSA cryptosystem is based on the difficulty of taking e-th roots modulo a composite number n.

Definition

Given:

- A large composite integer $n = p \cdot q$ (where p and q are unknown large primes).
- A public exponent e such that $\gcd(e, \phi(n)) = 1$.
- A ciphertext $c \in \mathbb{Z}_n^*$.

The Problem: Find the integer m such that:

$$m^e \equiv c \pmod{n}$$

Relation to Factoring

The RSA assumption is closely related to the **Integer Factorization Problem**: Given n, find p and q.

- If an attacker can factor n, they can calculate $\phi(n) = (p-1)(q-1)$, compute the private key $d \equiv e^{-1} \pmod{\phi(n)}$, and trivially solve the RSA problem.
- It is an open question whether solving the RSA problem is *equivalent* to factoring, or potentially easier.

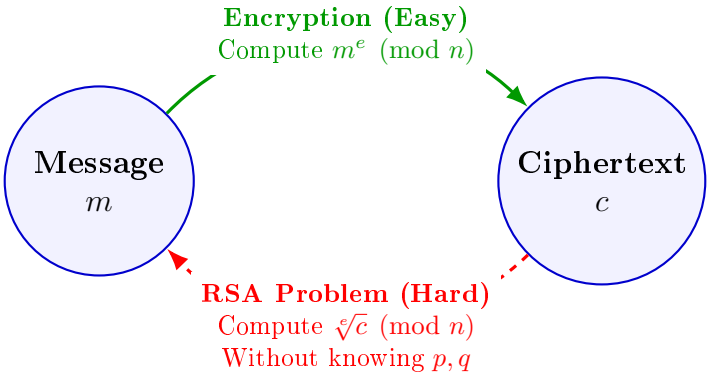

Figure 14: The asymmetry of the RSA function.

7.2 The Discrete Logarithm Problem (DLP)

The DLP is the foundation for Diffie-Hellman (DH), DSA, and ElGamal. It operates in a multiplicative cyclic group.

Definition

Given:

- A finite cyclic group G of order q (e.g., the multiplicative group $\mathbb{Z}_p^*$).
- A generator $g \in G$.
- An element $h \in G$ (where $h = g^x$).

The Problem: Find the integer x such that:

$$g^x \equiv h \pmod{p}$$

Visualizing the Hardness

In modular arithmetic, exponentiation "mixes" the number around the modulus circle chaotically. Reversing this mixing without exhaustive search (or algorithms like Baby-step Giant-step) is hard.

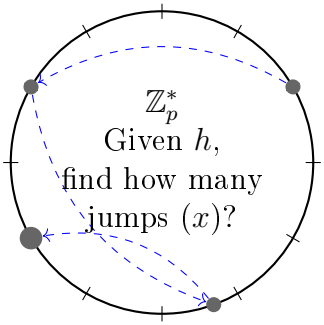

Figure 15: DLP behaves like determining the number of chaotic jumps on a clock face.

7.3 The Elliptic Curve Discrete Logarithm Problem (ECDLP)

This is the variant of the DLP used in ECDSA and ECDH. It operates on the algebraic structure of elliptic curves over finite fields.

Definition

Given:

- An elliptic curve E defined over a finite field $\mathbb{F}_q$.
- A base point $P \in E(\mathbb{F}_q)$ of order n.
- A point $Q \in E(\mathbb{F}_q)$ (where $Q = k \cdot P$).

The Problem: Find the scalar integer k such that:

$$Q = \underbrace{P + P + \cdots + P}_{k \text{ times}}$$

Why use Curves?

The best known algorithms for solving classical DLP (like Index Calculus) do not work on general Elliptic Curves. The only known attacks are generic algorithms (like Pollard's rho) which have exponential time complexity $\mathcal{O}(\sqrt{n})$. This allows ECDLP to achieve the same security level as RSA/DLP with **much smaller key sizes**.

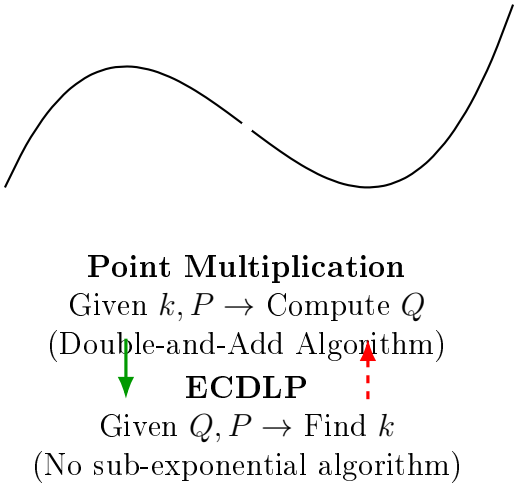

Figure 16: ECDLP involves finding how many times P was added to itself to reach Q.

7.4 Security Comparison

Approximate key sizes required for equivalent security levels (as per NIST recommendations).

Security Level (Bits of Security)	**Symmetric Key** (AES)	**RSA / DLP Key** (Integer Factoring)	**ECC Key** (ECDLP)
80	80 bits	1024 bits	160 bits
112	112 bits	2048 bits	224 bits
128	128 bits	3072 bits	256 bits
192	192 bits	7680 bits	384 bits
256	256 bits	15360 bits	521 bits

Table 3: Key Size Comparison. Notice how RSA keys must grow explosively to keep up, while ECC keys grow linearly.

7.5 RSA Encryption Scheme

RSA is one of the first public-key cryptosystems and is widely used for secure data transmission. Unlike symmetric systems (like AES) which use the same key for encryption and decryption, RSA uses a pair of keys:

- **Public Key**: Used to encrypt data. Can be shared openly.

- **Secret Key**: Used to decrypt data. Must be kept secret.

Key Sizes: 1024, 2048, 4092
RSA (Rivest-Shamir-Adleman) hardness is based on the prime factorization problem, since it is hard to factor the product of two large primes.

7.5.1 Key Generation

1. **Prime Selection:** Choose two distinct large random primes p, q.
2. **Modulus:** Compute $n = p \cdot q$.
3. **Exponents:** Compute $\phi(n) = (p-1)(q-1)$. Select $e \in \mathbb{Z}^*_{\phi(n)}$ such that $\gcd(e, \phi(n)) = 1$. Compute $d \equiv e^{-1} \pmod{\phi(n)}$.
4. **Output:** $pk = \langle n, e \rangle, \quad sk = \langle n, d \rangle$.

7.5.2 Encryption

The encryption algorithm is deterministic (in textbook RSA) or probabilistic (in RSA-OAEP). Here we define textbook RSA.

- **Input:** $pk = \langle n, e \rangle$, message $m \in \mathbb{Z}_n$.
- **Computation:** Compute ciphertext c:

$$c \equiv m^e \pmod{n}$$

- **Output:** $c \in \mathcal{C}$.

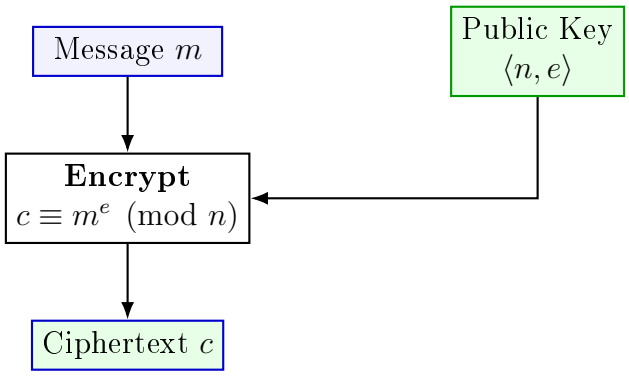

Figure 17: RSA Encryption $\mathsf{Enc}_{pk}(m)$.

7.5.3 Decryption

- **Input:** $sk = \langle n, d \rangle$, ciphertext $c \in \mathbb{Z}_n$.
- **Computation:** Recover message m':

$$m' \equiv c^d \pmod{n}$$

- **Output:** m'.

7.5.4 Correctness Proof

We must show that $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$. Substituting the definition of c:

$$c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$$

Since $ed \equiv 1 \pmod{\phi(n)}$, we have $ed = 1 + k\phi(n)$ for some integer k.

$$m^{ed} \equiv m^{1+k\phi(n)} \equiv m \cdot (m^{\phi(n)})^k \pmod{n}$$

Assuming $m \in \mathbb{Z}_n^*$, by Euler's Theorem $m^{\phi(n)} \equiv 1 \pmod{n}$, so:

$$m \cdot (1)^k \equiv m \pmod{n}$$

(Note: The proof also holds for m not in $\mathbb{Z}_n^*$ via the Chinese Remainder Theorem).

7.6 ElGamal Encryption

ElGamal is an asymmetric key encryption algorithm based on the **Diffie-Hellman Key Exchange**. Its security relies on the intractability of the **Discrete Logarithm Problem (DLP)**:

> Given g, p, and $h = g^x \pmod{p}$, it is computationally hard to find x.

A key feature of ElGamal is that it is **Probabilistic**: Encrypting the same message multiple times results in different ciphertexts.

7.6.1 Key Generation *Gen*

The algorithm takes a security parameter 1^λ.

1. **Group Generation:** Generate a cyclic group $\mathbb{G}$ of prime order q with generator g.
2. **Private Key:** Choose a random $x \leftarrow \mathbb{Z}_q$.
3. **Public Key:** Compute $h = g^x$.
4. **Output:**
$$pk = \langle \mathbb{G}, q, g, h \rangle, \quad sk = \langle \mathbb{G}, q, g, x \rangle$$

7.6.2 Encryption *Enc* (Alice)

ElGamal is probabilistic. To encrypt $m \in \mathbb{G}$:

- **Input:** pk, message m.
- **Randomness:** Choose an ephemeral key $y \leftarrow \mathbb{Z}_q$.
- **Computation:** Compute pair (c_1, c_2):
$$c_1 = g^y, \quad c_2 = m \cdot h^y$$
- **Output:** Ciphertext $c = \langle c_1, c_2 \rangle$.

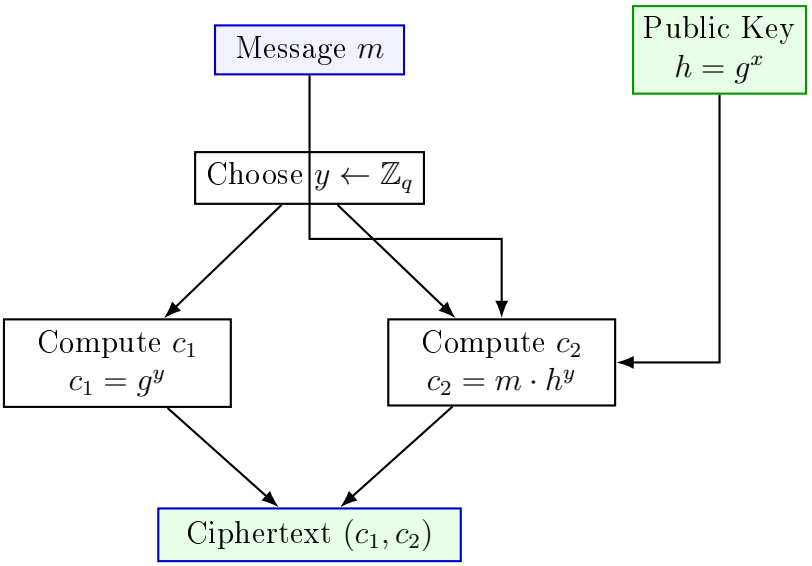

Figure 18: ElGamal Encryption $\mathsf{Enc}_{pk}(m)$.

7.6.3 Decryption

- **Input:** $sk = x$, ciphertext $c = \langle c_1, c_2 \rangle$.
- **Computation:** Compute shared secret $s = c_1^x$. Then compute:

$$m' = c_2 \cdot s^{-1}$$

- **Output:** m'.

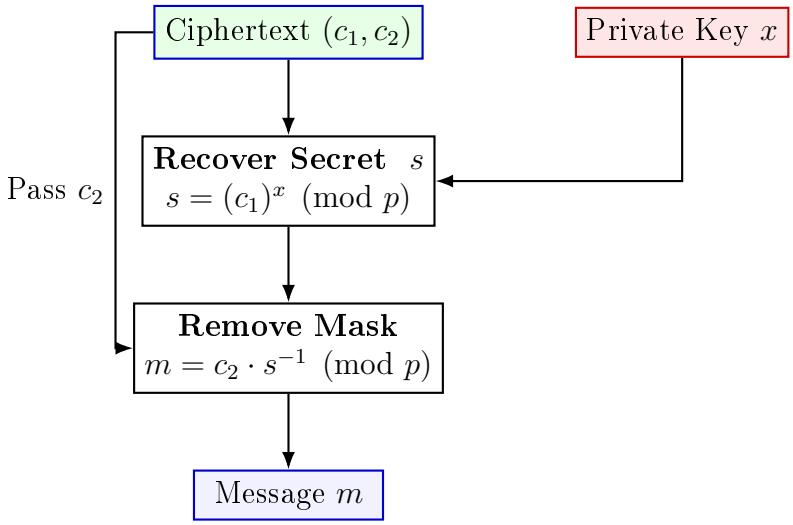

Figure 19: Decryption Process. The math works because $(c_1)^x = (g^k)^x = g^{kx} = (g^x)^k = h^k = s$.

7.6.4 Correctness Proof

We show that $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$. Substituting the values of c_1 and c_2:

$$m' = c_2 \cdot (c_1^x)^{-1} = (m \cdot h^y) \cdot (g^y)^{-x}$$

Since $h = g^x$:

$$m' = (m \cdot (g^x)^y) \cdot (g^{yx})^{-1}$$
$$m' = m \cdot g^{xy} \cdot g^{-xy}$$
$$m' = m \cdot 1 = m$$

Thus, the decryption correctly recovers the original message.

7.6.5 Example

Let's trace the values with small numbers.

1. **Setup:**
   - Prime $p = 23$, Generator $g = 5$.
   - Bob chooses private $x = 6$.
   - Bob computes $h = 5^6 \pmod{23} = 15625 \pmod{23} = 8$.
   - Public Key: $(p = 23, g = 5, h = 8)$.
2. **Encryption (Message $m = 10$):**
   - Alice chooses random $k = 3$.
   - Clue $c_1 = g^k = 5^3 = 125 \equiv 10 \pmod{23}$.
   - Secret $s = h^k = 8^3 = 512 \equiv 6 \pmod{23}$.
   - Mask $c_2 = m \cdot s = 10 \cdot 6 = 60 \equiv 14 \pmod{23}$.
   - Ciphertext: $(10, 14)$.
3. **Decryption:**
   - Bob receives $(10, 14)$.
   - Recover $s = c_1^x = 10^6 \pmod{23} = 1000000 \pmod{23} = 6$. (Matches!).
   - Find modular inverse of $6 \pmod{23}$. $6 \times 4 = 24 \equiv 1$. So $s^{-1} = 4$.
   - $m = c_2 \cdot s^{-1} = 14 \cdot 4 = 56 \equiv 10 \pmod{23}$.

8 Digital Signatures

Digital signatures guarantee the non-repudation, integrity and authenticity properties. The algorithm is usually done agains the hash of m message

A digital signature consists of three random, polinomial-time algorithms $(Gen, Sign, Vrfy)$, so:

- Gen **Key Generation Algorithm**: Gen takes 1^n security parameter as input, and outputs (pk, sk) keypair. We will refer to pk as the public key and sk as the secret key. We assume that pk and sk has a length of at least n, and n can be defined from (pk, sk) keypair. $(pk, sk) \leftarrow Gen(1^n)$
- $Sign$ **Signature Algorithm**: $Sign$ takes sk secret key and message m as inputs, and outputs signature σ. $\sigma \leftarrow Sign_{sk}(m)$

- $Vrfy$ **Verify Algorithm**: $Vrfy$ takes signature σ, public key pk and m as inputs, and outputs b bit. If $b = 1$ the signature is valid, if $b = 0$ the signature is invalid. $b \leftarrow Vrfy_{pk}(\sigma, m)$

For all valid $m \in \mathcal{M}$ messages the following condition must satisfy: $Vrfy_{pk}(m, Sign_{sk}(m)) = 1$.

8.1 RSA Signature Algorithm

The RSA signature scheme is defined as a tuple of three probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ over a message space $\mathcal{M}$ and a signature space $\mathcal{S}$. We utilize a cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_n$.

8.1.1 Key Generation

The key generation algorithm Gen takes a security parameter 1^λ as input and outputs a pair of keys (pk, sk).

1. **Prime Selection:** Choose two distinct large random primes p, q of equal bit-length.
2. **Modulus:** Compute $n = p \cdot q$. The group is defined over $\mathbb{Z}_n^*$.
3. **Euler's Totient:** Compute $\phi(n) = (p-1)(q-1)$.
4. **Public Exponent:** Select $e \in \mathbb{Z}_{\phi(n)}^*$ such that $\gcd(e, \phi(n)) = 1$.
5. **Private Exponent:** Compute $d \equiv e^{-1} \pmod{\phi(n)}$.
6. **Output:**
$$pk = \langle n, e \rangle, \quad sk = \langle n, d \rangle$$

8.1.2 Signing Algorithm

The signing algorithm takes the secret key sk and a message $m \in \mathcal{M}$.

- **Input:** $sk = \langle n, d \rangle$, message $m \in \{0,1\}^*$.
- **Hashing:** Compute the hash digest $h = H(m)$, where $h \in \mathbb{Z}_n$.
- **Computation:** Compute the signature σ using the RSA primitive:
$$\sigma \equiv h^d \pmod{n}$$
- **Output:** The signature $\sigma \in \mathcal{S}$.

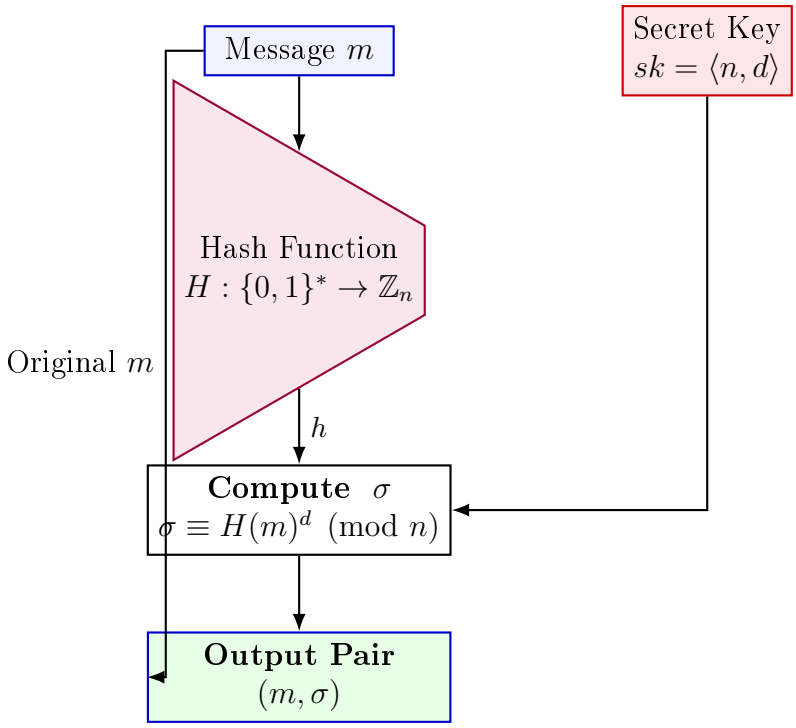

Figure 20: The Signing Algorithm $\mathsf{Sign}_{sk}(m)$.

8.1.3 Verification Algorithm

The verification algorithm is deterministic. It takes the public key pk, a message m, and a signature σ. It outputs a bit $b \in \{0,1\}$ (1 for valid, 0 for invalid).

- **Input:** $pk = \langle n, e \rangle$, message m, signature σ.
- **Hashing:** Compute $h = H(m)$.
- **Inversion:** Compute $h' \equiv \sigma^e \pmod{n}$.
- **Check:** Verify if $h' \stackrel{?}{=} h$.

$$\mathsf{Vrfy}_{pk}(m, \sigma) = \begin{cases} 1 & \text{if } \sigma^e \equiv H(m) \pmod{n} \\ 0 & \text{otherwise} \end{cases}$$

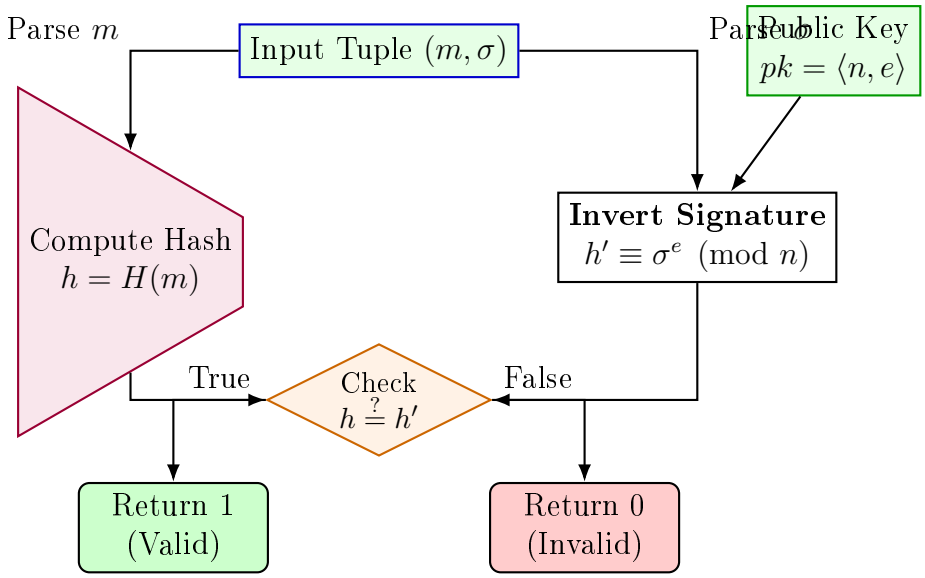

Figure 21: The Verification Algorithm $\mathsf{Vrfy}_{pk}(m, \sigma)$.

8.1.4 Correctness Proof

We must show that for any valid signature generated by Sign, Vrfy returns 1.

Given $\sigma \equiv h^d \pmod{n}$ where $h = H(m)$, the verifier computes $\sigma^e \pmod{n}$. Using Euler's Theorem, since $ed \equiv 1 \pmod{\phi(n)}$, we have $ed = k\phi(n) + 1$ for some integer k.

$$\sigma^e \equiv (h^d)^e \equiv h^{de} \equiv h^{k\phi(n)+1} \equiv (h^{\phi(n)})^k \cdot h \pmod{n}$$

Since $h \in \mathbb{Z}_n^*$, by Euler's Theorem $h^{\phi(n)} \equiv 1 \pmod{n}$. Thus:

$$1^k \cdot h \equiv h \pmod{n}$$

Therefore, $\sigma^e \equiv H(m) \pmod{n}$, and the verification succeeds.

8.2 DSA Signature Scheme

The Digital Signature Algorithm (DSA) is a FIPS standard (FIPS 186) based on the discrete logarithm problem. It is defined as a tuple $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ utilizing a cryptographic hash function $H : \{0,1\}^* \to \mathbb{Z}_q$.

8.2.1 Key Generation (Gen)

The key generation algorithm takes security parameters L and N (e.g., $L = 2048, N = 256$).

1. **Parameter Generation:**
   - Choose a prime q of length N bits.
   - Choose a prime p of length L bits such that $p - 1$ is a multiple of q.
   - Choose an integer h $(1 < h < p - 1)$ such that $g = h^{(p-1)/q} \bmod p > 1$.
   - (p, q, g) are public domain parameters.

2. **Private Key:** Choose a random integer x such that $0 < x < q$.
3. **Public Key:** Compute $y = g^x \bmod p$.
4. **Output:**

$$pk = \langle p, q, g, y \rangle, \quad sk = \langle p, q, g, x \rangle$$

8.2.2 Signing Algorithm (Sign)

The signing algorithm is probabilistic. It takes the secret key sk and a message m.

- **Input:** sk, message m.
- **Nonce Generation:** Choose a random per-message integer k such that $0 < k < q$.
- **Computation:**
  1. Compute $r = (g^k \bmod p) \bmod q$.
  2. Compute $z = H(m)$ (taking the leftmost $\min(N, \text{outlen})$ bits).
  3. Compute $s = k^{-1}(z + x \cdot r) \bmod q$.
- **Retry:** If $r = 0$ or $s = 0$, restart with a new k.
- **Output:** Signature $\sigma = (r, s)$.

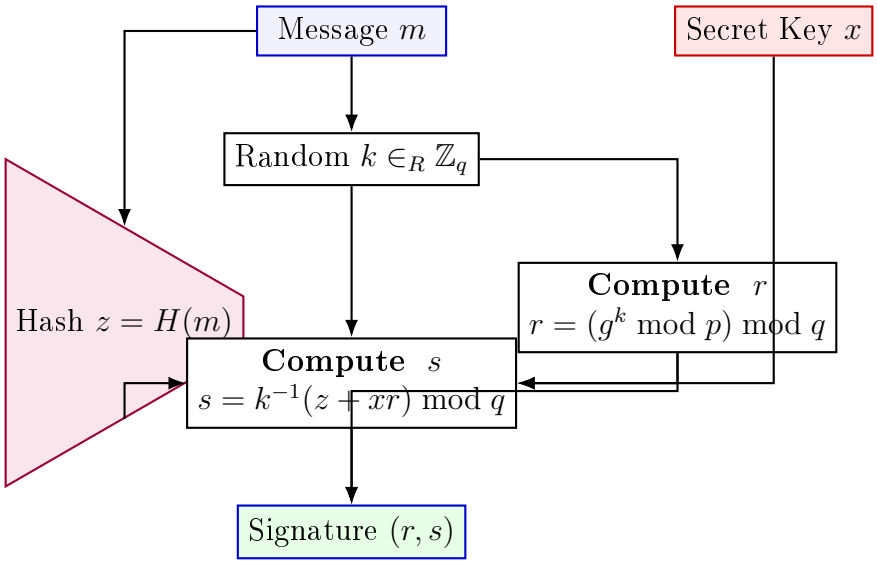

Figure 22: DSA Signing Process $\mathsf{Sign}_{sk}(m)$.

8.2.3 Verification Algorithm (Vrfy)

The verification algorithm takes the public key pk, message m, and signature $\sigma = (r, s)$.

- **Check:** Verify $0 < r < q$ and $0 < s < q$. If not, return Invalid.
- **Computation:**
  1. Compute $w = s^{-1} \bmod q$.

2. Compute hash $z = H(m)$.
3. Compute $u_1 = z \cdot w \bmod q$.
4. Compute $u_2 = r \cdot w \bmod q$.
5. Compute $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.

- **Output:** Valid if $v = r$, otherwise Invalid.

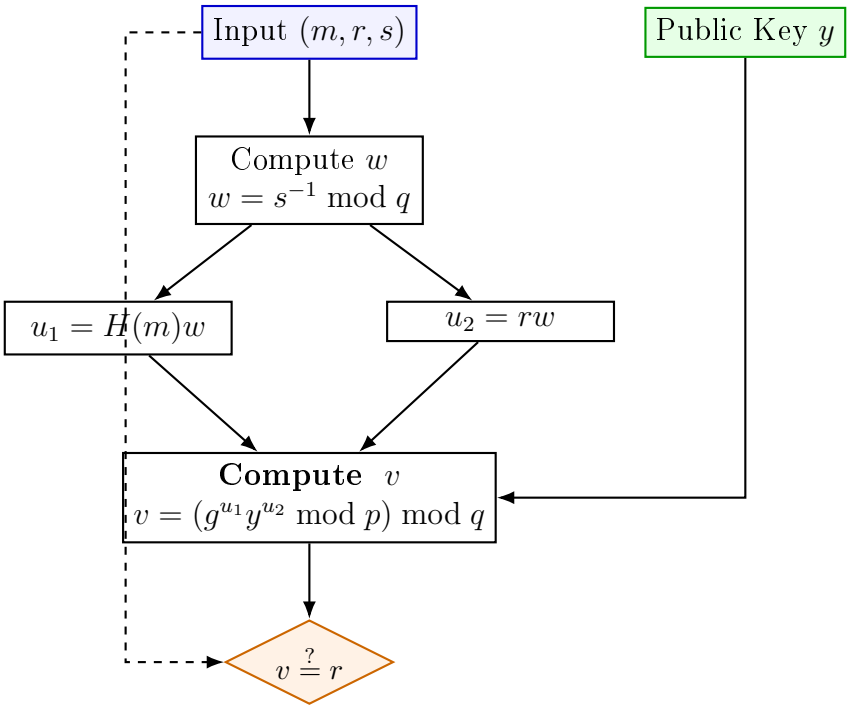

Figure 23: DSA Verification Process $\mathsf{Vrfy}_{pk}(m, \sigma)$.

8.2.4 Correctness Proof

We show that if the signature is valid, then $v = r$. Recall $s \equiv k^{-1}(z + xr) \pmod q$. Thus $k \equiv s^{-1}(z + xr) \pmod q$. Substituting $w = s^{-1}$:

$$k \equiv w(z + xr) \equiv zw + xrw \equiv u_1 + xu_2 \pmod q$$

Therefore, we can write $k = u_1 + xu_2 + nq$ for some integer n. Now compute v:

$$v = (g^{u_1} y^{u_2} \bmod p) \bmod q$$

Substitute $y = g^x$:

$$v = (g^{u_1}(g^x)^{u_2} \bmod p) \bmod q = (g^{u_1 + xu_2} \bmod p) \bmod q$$

Since g has order q modulo p (i.e., $g^q \equiv 1 \pmod p$), the exponent works modulo q:

$$g^{u_1 + xu_2} \equiv g^{k - nq} \equiv g^k \cdot (g^q)^{-n} \equiv g^k \cdot 1^{-n} \equiv g^k \pmod p$$

Thus:

$$v = (g^k \bmod p) \bmod q$$

By definition of the signing algorithm, $r = (g^k \bmod p) \bmod q$. Therefore, $v = r$.

9 Elliptik Curve Digital Signature ALgorithm Scheme (ECDSA)

10 Formal Definition

ECDSA is an ANSI, FIPS, and IEEE standard. It operates in an elliptic curve group $\mathbb{E}(\mathbb{F}_q)$ rather than the multiplicative group of integers used in DSA. It is defined as a tuple $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

11 1. Key Generation (Gen)

The algorithm takes a security parameter defining the curve parameters.

1. **Domain Parameters:**
   - An elliptic curve E defined over a finite field $\mathbb{F}_q$.
   - A base point $G \in E(\mathbb{F}_q)$ of large prime order n.
   - The order n (where $n \times G = \mathcal{O}$, the point at infinity).
2. **Private Key:** Choose a random integer d such that $1 \leq d \leq n-1$.
3. **Public Key:** Compute the point $Q = d \times G$ (Scalar Multiplication).
4. **Output:**
$$pk = \langle E, G, n, Q \rangle, \quad sk = \langle E, G, n, d \rangle$$

11.0.1 Signing Algorithm (Sign)

To sign a message m:

- **Input:** sk, message m.
- **Nonce:** Select a random integer $k \in_R [1, n-1]$.
- **Computation:**
  1. Compute point $R = k \times G = (x_1, y_1)$.
  2. Compute $r = x_1 \pmod{n}$. (If $r = 0$, try new k).
  3. Compute hash $z = H(m)$ (truncated to bit-length of n).
  4. Compute $s = k^{-1}(z + r \cdot d) \pmod{n}$. (If $s = 0$, try new k).
- **Output:** Signature $\sigma = (r, s)$.

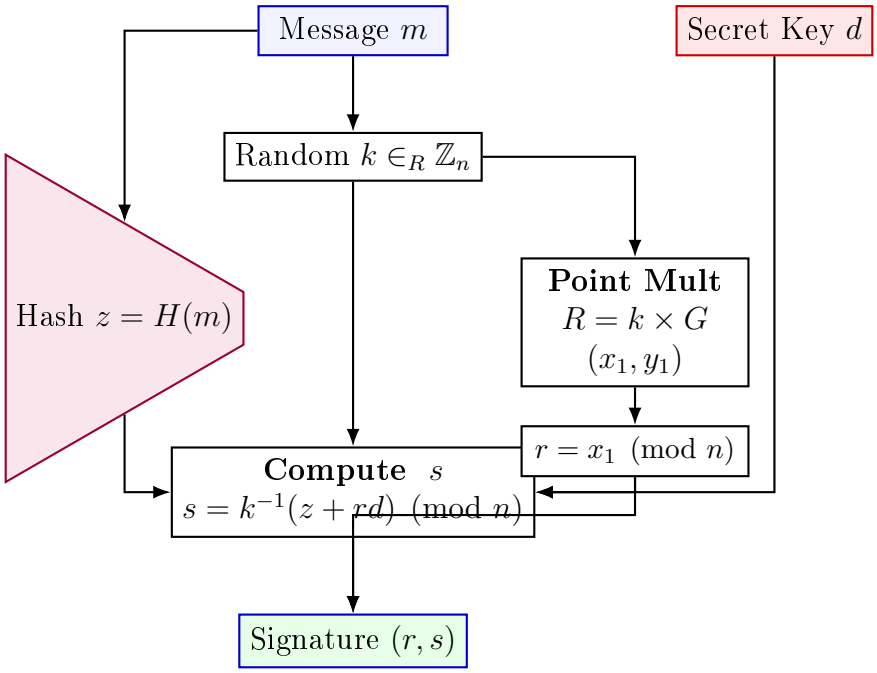

Figure 24: ECDSA Signing Process $\mathsf{Sign}_{sk}(m)$.

11.0.2 Verification Algorithm (Vrfy)

To verify signature $\sigma = (r, s)$ for message m:

- **Check:** Verify $r, s \in [1, n-1]$.
- **Computation:**
  1. Compute $w = s^{-1} \pmod n$.
  2. Compute hash $z = H(m)$.
  3. Compute $u_1 = z \cdot w \pmod n$.
  4. Compute $u_2 = r \cdot w \pmod n$.
  5. Compute point $P = u_1 \times G + u_2 \times Q$. (Check if $P = \mathcal{O}$).
  6. Extract $P = (x_P, y_P)$.
  7. Compute $v = x_P \pmod n$.
- **Output:** Valid if $v = r$, otherwise Invalid.

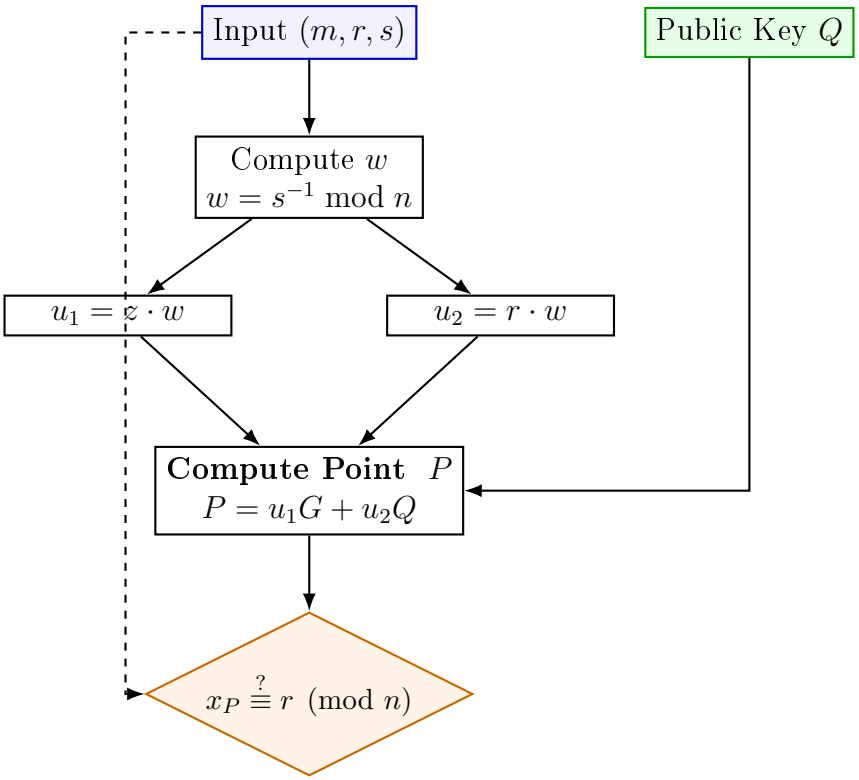

Figure 25: ECDSA Verification Process $\mathsf{Vrfy}_{pk}(m, \sigma)$.

11.0.3 Correctness Proof

We verify that the calculated point P essentially reconstructs the random point R used during signing.

Recall $s \equiv k^{-1}(z+rd) \pmod n$. Thus $k \equiv s^{-1}(z+rd) \pmod n$. Substituting $w = s^{-1}$:

$$k \equiv w(z + rd) \equiv zw + rdw \equiv u_1 + u_2 d \pmod n$$

Therefore:

$$k \times G = (u_1 + u_2 d) \times G = u_1 \times G + u_2 \times (d \times G)$$

Since $Q = d \times G$, we have:

$$k \times G = u_1 \times G + u_2 \times Q$$

Thus, the verification point P is exactly the original point R (where $R = k \times G$). Consequently, $x_P \equiv x_1 \equiv r \pmod n$.

12 Key excahnge algorithms

12.1 RSA Key Transport (Key Exchange)

Unlike Diffie-Hellman, RSA Key Transport is asymmetric: one party defines the key, encrypts it, and sends it to the other.

12.1.1 Setup

- **Bob (Receiver):** Generates RSA keys.

- Public Key: (n, e).
- Private Key: (n, d).

12.1.2 Protocol Flow

1. **Bob:** Sends Public Key (n, e) to Alice.
2. **Alice (Initiator):**
   - Generates a random session key K (e.g., 256 bits).
   - Converts K to integer m (with padding, e.g., PKCS#1 v1.5 or OAEP).
   - Computes ciphertext $c \equiv m^e \pmod{n}$.
   - Sends c to Bob.
3. **Bob:**
   - Decrypts $m \equiv c^d \pmod{n}$.
   - Extracts K from the padded message m.

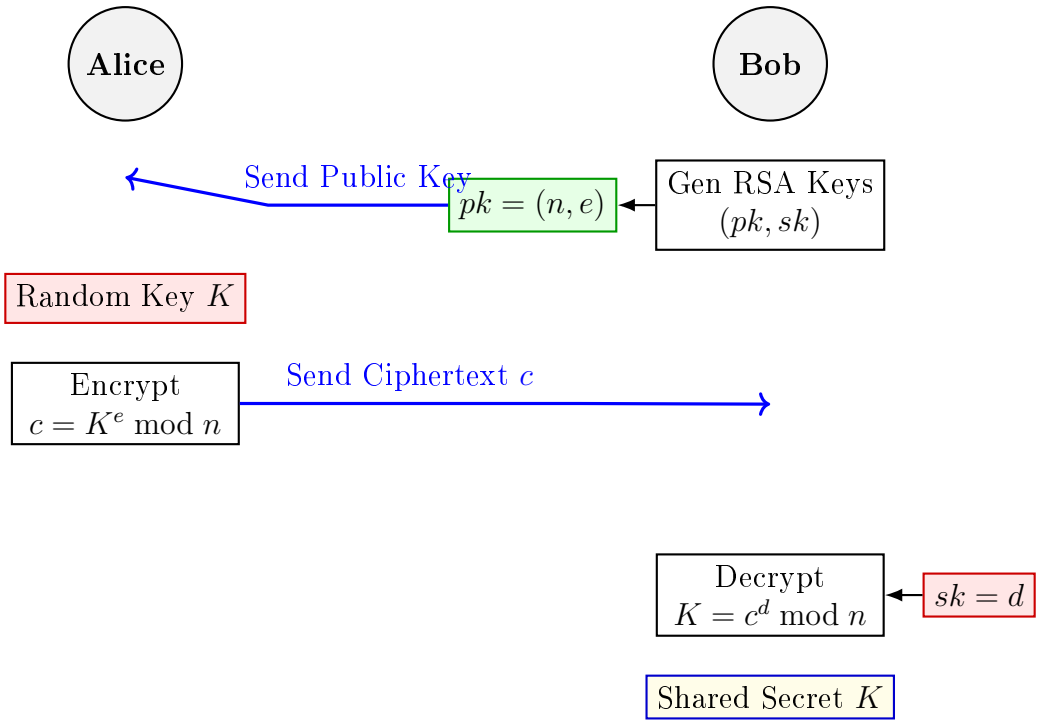

Figure 26: RSA Key Transport Protocol.

12.2 Diffie-Hellman Key Exchange (DHKE)

A protocol allowing two parties to jointly establish a shared secret over an insecure channel.

12.2.1 Setup

- **Parameters:** A large prime p and a generator g of the multiplicative group $\mathbb{Z}_p^*$.

12.2.2 Protocol Flow

1. **Alice:** Chooses secret integer $a \in_R [1, p-2]$. Computes $A = g^a \pmod{p}$. Sends A to Bob.
2. **Bob:** Chooses secret integer $b \in_R [1, p-2]$. Computes $B = g^b \pmod{p}$. Sends B to Alice.
3. **Key Computation:**
   - Alice computes $S_A = B^a \pmod{p}$.
   - Bob computes $S_B = A^b \pmod{p}$.
4. **Shared Secret:** $K = S_A = S_B$.

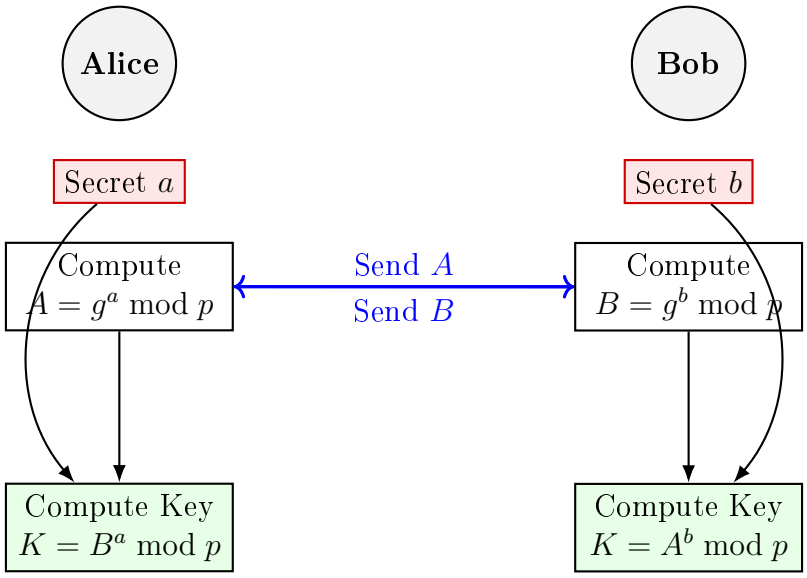

Figure 27: Diffie-Hellman Key Exchange.

12.2.3 Correctness Proof

$$S_A \equiv B^a \equiv (g^b)^a \equiv g^{ba} \pmod{p}$$
$$S_B \equiv A^b \equiv (g^a)^b \equiv g^{ab} \pmod{p}$$

Since multiplication in the exponent is commutative ($ab = ba$), $S_A = S_B$.

12.3 Elliptic Curve Diffie-Hellman (ECDH)

An variant of DHKE using additive elliptic curve groups, offering smaller key sizes for equivalent security.

12.3.1 Setup

- **Parameters:** Elliptic curve $E(\mathbb{F}_q)$, base point G of order n.

12.3.2 Protocol Flow

1. **Alice:** Chooses private $d_A \in_R [1, n-1]$. Computes public point $Q_A = d_A \times G$.
2. **Bob:** Chooses private $d_B \in_R [1, n-1]$. Computes public point $Q_B = d_B \times G$.
3. **Exchange:** Parties exchange Q_A and Q_B.
4. **Key Derivation:**
   - Alice computes point $S = d_A \times Q_B$.
   - Bob computes point $S = d_B \times Q_A$.
5. **Shared Secret:** The x-coordinate of S, denoted x_S, is the shared secret.

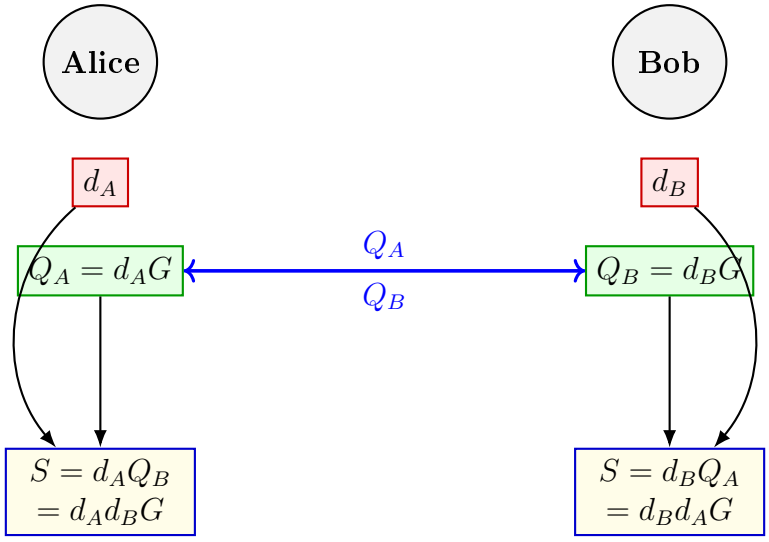

Figure 28: ECDH Protocol Flow.

12.3.3 Correctness Proof

$$S_{Alice} = d_A \times Q_B = d_A \times (d_B \times G) = (d_A \cdot d_B) \times G$$
$$S_{Bob} = d_B \times Q_A = d_B \times (d_A \times G) = (d_B \cdot d_A) \times G$$

Since scalar multiplication is associative and commutative over scalars, the points are identical.

13 Cryptographic Hash Functions

13.1 Formal Definition

A cryptographic hash function is a deterministic algorithm $H : \{0,1\}^* \to \{0,1\}^n$ that maps an arbitrary length input m to a fixed length output $h = H(m)$. It must satisfy three core security properties:

1. **Pre-image Resistance:** Given h, it is computationally infeasible to find m such that $H(m) = h$.

2. **Second Pre-image Resistance:** Given m_1, it is infeasible to find $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$.

3. **Collision Resistance:** It is infeasible to find any pair $m_1 \neq m_2$ such that $H(m_1) = H(m_2)$.

13.2 The Merkle-Damgård Construction (SHA-2)

Most traditional hash functions (MD5, SHA-1, SHA-2) are built upon the Merkle-Damgård construction. It iterates a collision-resistant compression function f over fixed-size message blocks.

13.2.1 Construction Specifications

- **Padding:** Input M is padded to a multiple of the block length.
- **Initialization:** A fixed Initial Vector (IV) initializes the internal state H_0.
- **Iteration:** For each message block m_i:

$$H_i = f(H_{i-1}, m_i)$$

- **Finalization:** The final hash is the last state H_t (sometimes processed by a finalization function).

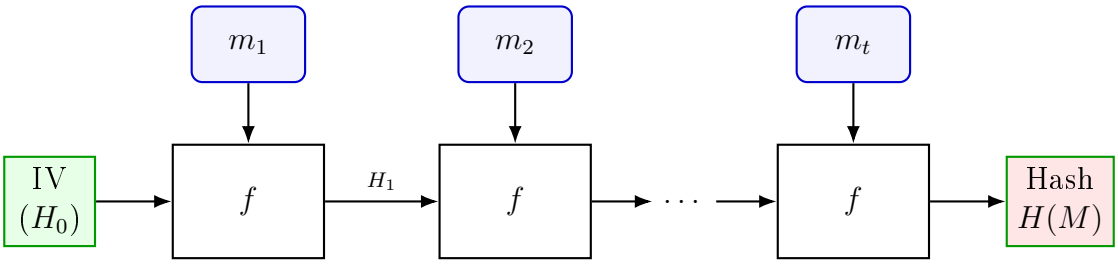

Figure 29: Merkle-Damgård Construction. The chaining variable H_i passes the state to the next block.

13.3 SHA-256 (Specifics)

SHA-256 is a Merkle-Damgård hash function.

- **Block Size:** 512 bits.
- **Output Size:** 256 bits.
- **State:** 8 x 32-bit words (A, B, C, D, E, F, G, H).
- **Rounds:** 64 rounds per block using the Davies-Meyer structure (Feed-forward).

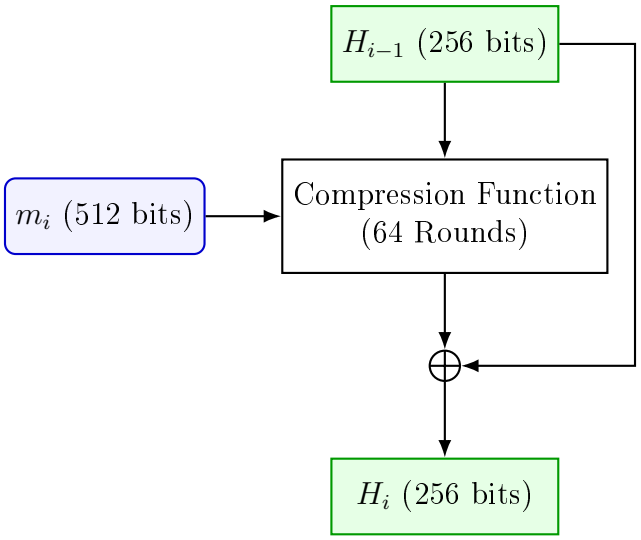

Figure 30: Davies-Meyer structure used inside SHA-2.

13.4 The Sponge Construction (SHA-3 / Keccak)

SHA-3 uses the **Sponge construction**, which is fundamentally different from Merkle-Damgård. It handles arbitrary input and output lengths naturally.

13.4.1 Construction Specifications

The state S is divided into two parts:

1. **Bitrate (r):** The part of the state that is touched by message blocks.
2. **Capacity (c):** The part of the state that is never directly touched by input/output (ensures security).

Total State Width $b = r + c$ (1600 bits for Keccak-f[1600]).

13.4.2 Phases

1. **Absorb Phase:** Message blocks m_i (size r) are XORed into the bitrate part of the state, followed by a permutation function f.
2. **Squeeze Phase:** Output blocks z_i (size r) are read from the bitrate part, interleaved with permutations f.

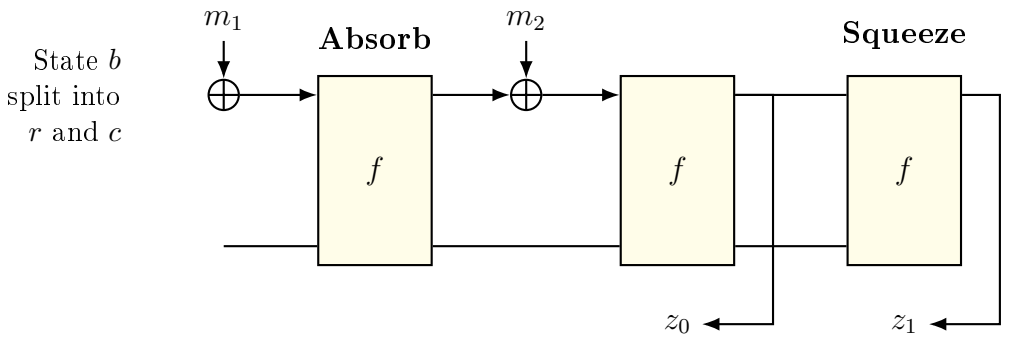

Figure 31: The Sponge Construction. r determines speed, c determines security.

13.5 Comparison

Feature	SHA-2 (Merkle-Damgård)	SHA-3 (Sponge)
Structure	Iterative Compression ($H_i = f(H_{i-1}, m_i)$)	Permutation-based state update
Operations	Add (mod $2^{32}/2^{64}$), Rotate, XOR, Shift	XOR, AND, NOT, Rotate (No arithmetic add)
Vulnerabilities	Susceptible to Length Extension Attacks (without HMAC)	Resistant to Length Extension Attacks
Flexibility	Fixed output length per variant	Extendable Output Function (XOF)

Table 4: Comparison of modern hash standards.

14 SSL/TLS Protocol

14.1 Formal Definition

The TLS protocol (successor to SSL) provides privacy and data integrity between two communicating applications. It is composed of two layers:

1. **The TLS Record Protocol:** Layered on top of a reliable transport (TCP), it ensures that the connection is private (using symmetric encryption) and reliable (using message authentication).
2. **The TLS Handshake Protocol:** Allows the client and server to authenticate each other and negotiate encryption algorithms and cryptographic keys before the application protocol transmits or receives its first byte of data.

14.2 The TLS Handshake Protocol

The handshake is the most complex part of TLS. Its goal is to establish a **Master Secret** from which session keys are derived.

14.3 Protocol Flow (TLS 1.2 Full Handshake)

We visualize the exchange involving a Diffie-Hellman Key Exchange (DHE) for Forward Secrecy.

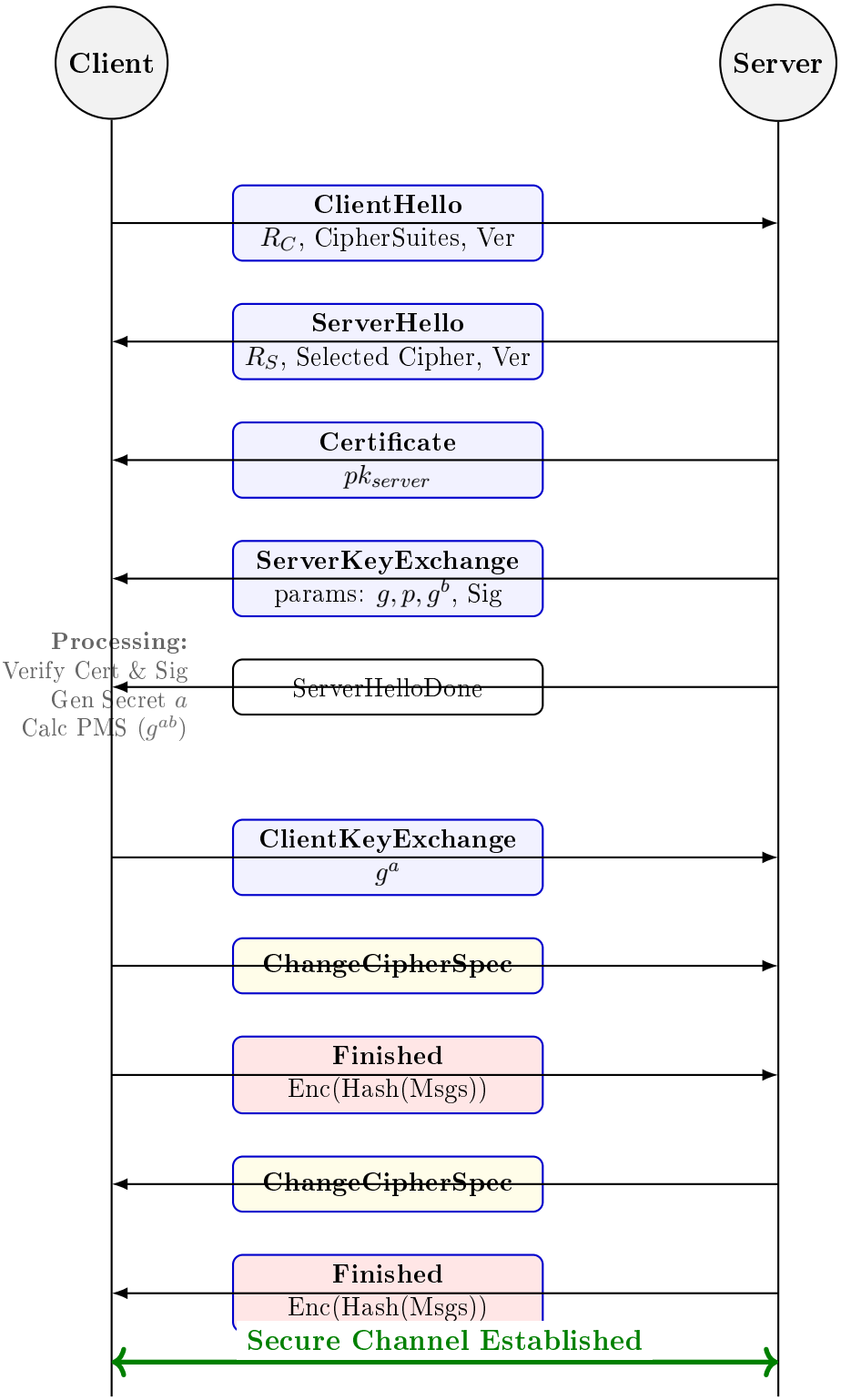

Figure 32: TLS 1.2 Full Handshake with Diffie-Hellman Key Exchange.

14.4 Key Derivation

The security of TLS relies on deriving symmetric keys from the material exchanged in the handshake.

14.5 The Pre-Master Secret (PMS)

Depending on the key exchange method:

- **RSA:** Client generates a 48-byte random string and encrypts it with Server's public key.
- **Diffie-Hellman:** Both sides compute $g^{ab} \pmod{p}$.

14.5.1 The Master Secret (MS)

The MS is a 48-byte value calculated using a Pseudorandom Function (PRF):

$$MS = \mathsf{PRF}(PMS, \text{"master secret"}, R_C + R_S)$$

Where R_C and R_S are the client and server random nonces sent in the Hello messages.

14.5.2 Session Keys

The Master Secret is expanded to generate the **Key Block**, which is sliced into:

- Client Write MAC Key
- Server Write MAC Key
- Client Write Encryption Key
- Server Write Encryption Key
- (IVs for block ciphers)

14.6 The TLS Record Protocol

Once the handshake is complete, the Record Protocol takes application data and secures it.

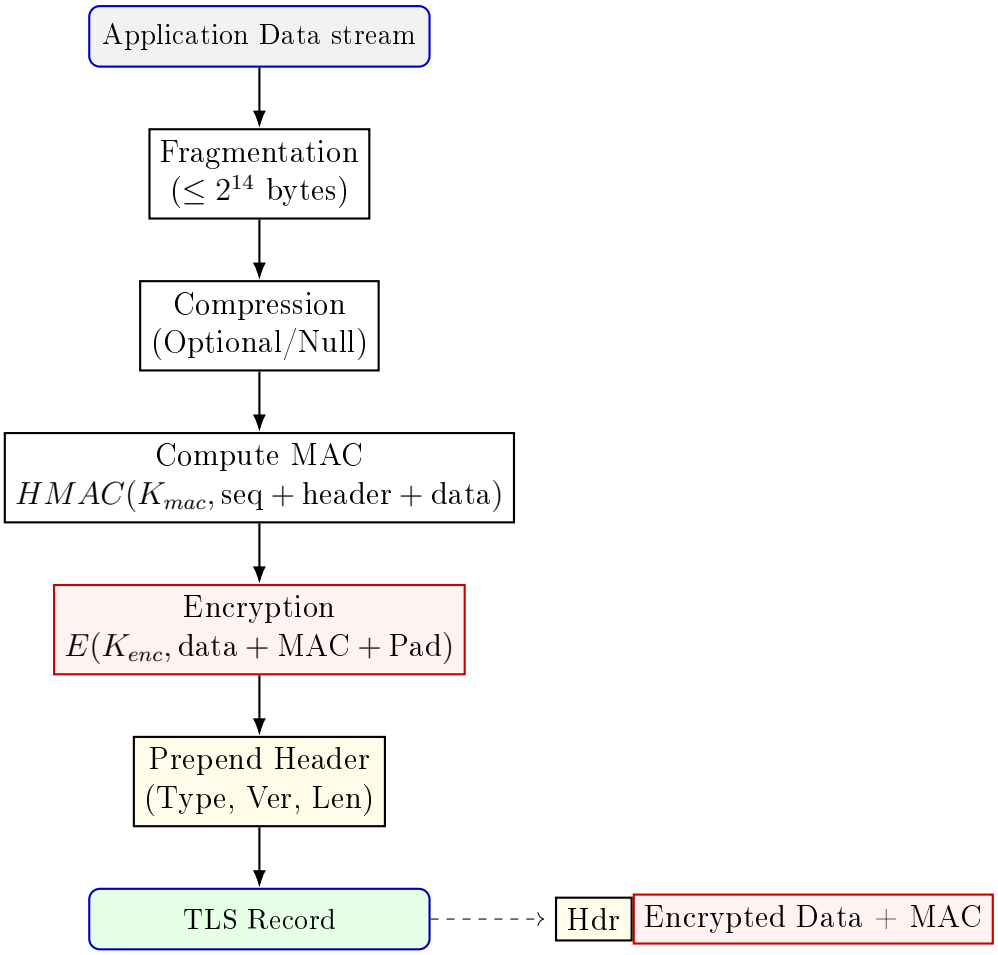

Figure 33: The TLS Record Protocol Processing Stack.

14.7 Security Services Provided

- **Confidentiality:** Achieved via symmetric encryption (AES, ChaCha20) using keys derived from the Master Secret.
- **Integrity:** Achieved via HMAC or AEAD (Authenticated Encryption with Associated Data).
- **Authentication:** Achieved via Certificates (X.509) and Digital Signatures (RSA/ECDSA) during the handshake.
- **Replay Protection:** Implicitly provided by the sequence numbers used in the MAC calculation.