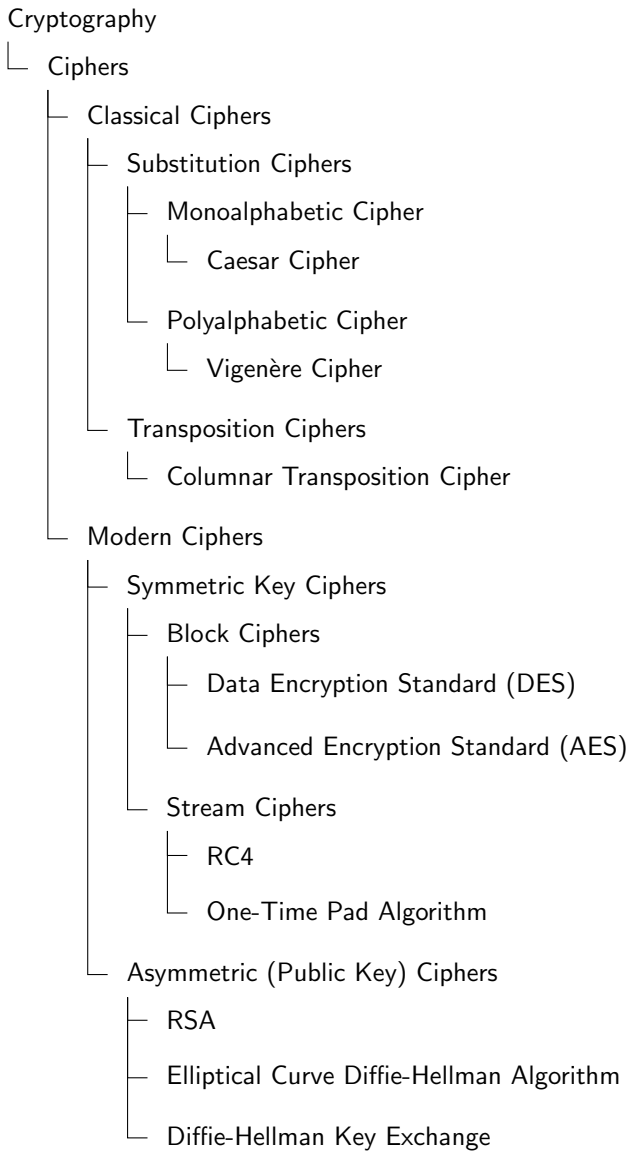
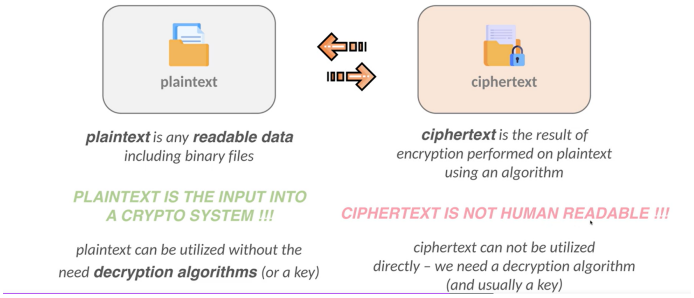
Contents

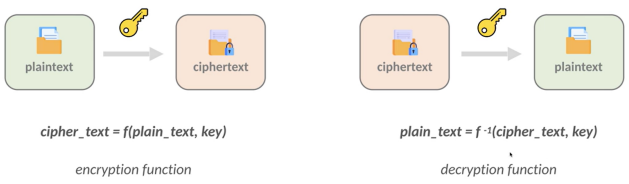

Cryptography
Ciphers
Classical Ciphers
Substitution Ciphers
Monoalphabetic Cipher
Caesar Cipher
Polyalphabetic Cipher
Vigenère Cipher
Transposition Ciphers
Columnar Transposition Cipher
Modern Ciphers
Symmetric Key Ciphers
Block Ciphers
Data Encryption Standard (DES)
Advanced Encryption Standard (AES)
Stream Ciphers
RC4
One-Time Pad Algorithm
Asymmetric (Public Key) Ciphers
RSA
Elliptical Curve Diffie-Hellman Algorithm
Diffie-Hellman Key Exchange

1 What is cryptography?

- **Cryptography** is the practice and study of techniques for secure communication in the presence of third parties.
- There are cases when we want to make sure that a given message is read by the reader and the receiver exclusively.
- Cryptography is used in order to make sure that the users information are not going to be stolen.
- **Plaintext** is any readable data, including binary files.
- **Ciphertext** is the result of encryption performed on plain text using an encryption algorithm.



- **Encryption** is the process of encoding a given message in a way that only the authorized parties can access it.
- **Decryption** is the process of decoding a given message. So essentially this is the inverse operation of encryption.
- **Key** :This is a sequence that is needed both for encryption and for decryption as well.



2 Types of cryptosystem

There are two types of cryptosystems:

1. **Classical Cryptography:** It represents early, historical encryption techniques relying on simple substitution and transposition methods.

   Types of Classical Cryptography:

   - **Substitution cipher :**

     Definition: A substitution cipher involves replacing each plaintext unit (typically a letter or group of letters) with another unit (often a letter) according to a regular system.
     It can be further classified into monoalphabetic and polyalphabetic ciphers.

     Monoalphabetic Substitution Cipher: In a monoalphabetic substitution cipher, each letter in the plaintext is consistently replaced by another letter of the alphabet.

     Polyalphabetic Substitution Cipher: In a polyalphabetic substitution cipher, different alphabets (or shifts) are used for different plaintext units at different positions.

     Ex-caesar cipher, vigenere cipher

   - **Transposition cipher:**

     Definition: A transposition cipher involves rearranging the order of the letters in the plaintext to form the ciphertext, based on a systematic method or key.

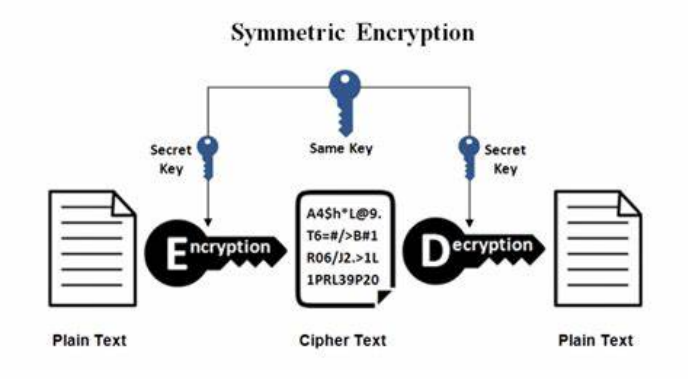
     Columnar Transposition Cipher: In a columnar transposition cipher, the plaintext is written in rows of a fixed length and then rearranged to form the ciphertext based on a specific key or permutation of column order.

2. **Modern Cryptography**: It employs sophisticated mathematical algorithms and protocols to achieve robust security and addresses the complexities and challenges of contemporary digital environments.

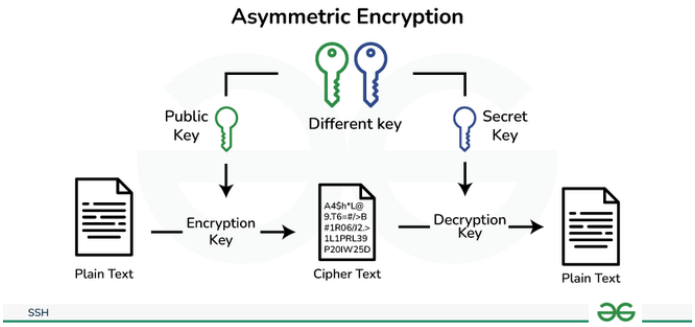
   Types of modern cryptography:

   - **Symmetric cryptosystem:** They use just a single key, both for encryption and for decryption as well. This is why it is called private

key cryptosystem or symmetric Cryptosystem. The main problem is that this private key that we use for encryption and for decryption must be exchanged.
Ex: DES, AES, One Time Pad,...etc



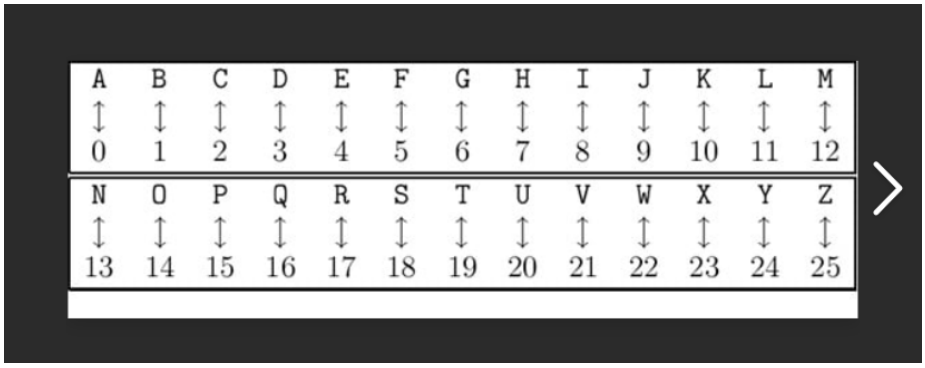
- **Asymmetric cryptosystem:** They use different keys, a public key and a private key for encryption and for decryption. This is why it is called public key cryptosystem or asymmetric cryptosystem. We must keep the private key, private and the public key is known for anyone.
Ex: RSA, ECDSA..etc



3 Classical ciphers:

3.1 Caesar cipher:

It is monoalphabetic cipher.
First we convert all the given text to capital letters.And then assgin each letter to a integer as shown here



The Caesar cipher encryption function $e(x)$, where x is the character we are encrypting and k is the key (the shift) applied to each letter, is represented as:

$$e(x) = (x + k) \mod 26$$

After applying this function, the result is a number which must then be translated back into a letter.

The decryption function $d(x)$, where x is the encrypted character, is:

$$d(x) = (x - k) \mod 26$$

$$c_i = E(p_i) = p_i + 3$$

A full translation chart of the Caesar cipher is shown here.

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

Using this encryption, the message

TREATY IMPOSSIBLE

would be encoded as

T	R	E	A	T	Y		I	M	P	O	S	S	I	B	L	E
w	u	h	d	w	b		l	p	s	r	v	v	l	e	o	h

Figure 1: caesar cipher example

3.2 Vigenere cipher:

It is a polyalphabetic cryptosystem.
The Vigenère cipher algorithm involves using a keyword to shift each letter of the plaintext by different amounts based on the corresponding letters of the keyword.

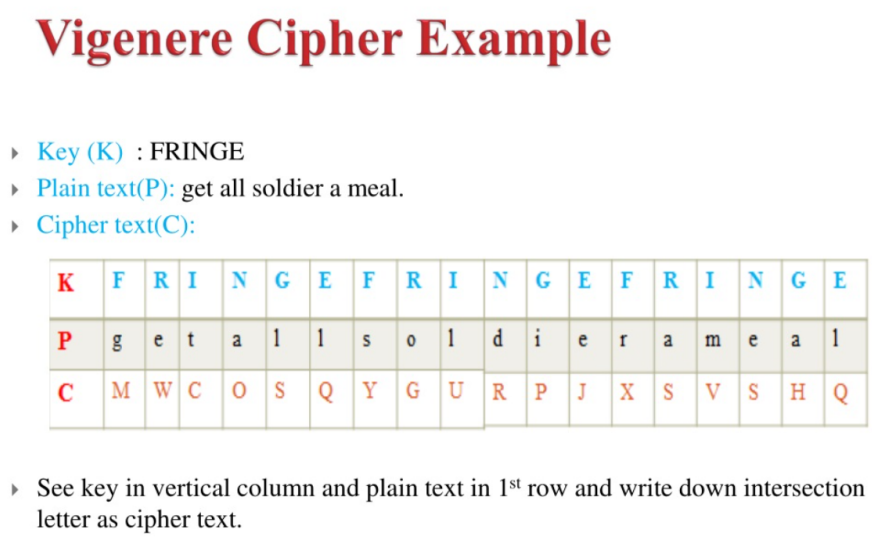


Figure 2: vigenere cipher example

3.3 Columnar Transposition Cipher:

Example of Columnar Transposition Cipher Encryption:

To encrypt the plaintext message "defend the east wall of the castle" using the keyword **GERMAN**, follow these steps:

Step 1: Write the plaintext in rows based on the keyword GERMAN:

G	E	R	M	A	N
d	e	f	e	n	d
t	h	e	e	a	s
t	w	a	l	l	o
f	t	h	e	c	a
s	t	l	e	x	x

Step 2: Reorder the columns based on the alphabetical order of the keyword letters:

A	E	G	M	N	R
n	e	d	e	d	f
a	h	t	e	s	e
l	w	t	l	o	a
c	t	f	e	a	h
x	t	s	e	x	l

Step 3: Read off the ciphertext along the columns:
The ciphertext is **nalcxehwttdttfseeleedsoaxfeahl**.

4 Symmetric key ciphers:

4.1 Block ciphers:

- A block cipher is a method of encrypting data in blocks to produce ciphertext using a cryptographic key and algorithm. The block cipher processes fixed-size blocks simultaneously, as opposed to a stream cipher, which encrypts data one bit at a time.
- Examples: i.DES, ii.AES

4.1.1 DES

:

- Data Encryption Standard is a symmetric- key algorithm
- it is a block cipher
- hybrid of substitution cipher and permutation cipher
- DES has a so-called Feistal structure

    1. we have to split the plain text into 64 bits long blocks
    2. there ate 16 rounds (input for every iteration is a 64 bits private key
    3. every round needs a different keys (called subkeys)
    4. its encryption and decryption operations are very similar
    5. Block size : 64 bits
    6. Key size : 64 bits (56 relevant bits are only used in the algorithm)
    7. No of rounds : 16
    8. No of subkeys: 16
    9. Cipher text size : 64 bits

Initial Permutation (IP)

The Initial Permutation (IP) is the first step in the DES algorithm. It rearranges the bits of the input plaintext according to a fixed permutation table(8 * 8 matrix).
So we are going to shuffle the values of the input bits based on the values of the permutation table.
These table values define the location of the given input bits.

$$\text{IP}(P) = \text{Output of the Initial Permutation for Plaintext } P$$
$$\text{input: } 64 \text{ bits} \rightarrow \text{output: } 64 \text{ bits}$$

Round Function (Round #1)

In each round of DES (there are 16 rounds in total), the round function involves several operations:

- Expansion Permutation (E): Expands the 32-bit input from the previous round into a 48-bit value.
- Key Mixing (XOR with Round Key): XOR the expanded input with a subkey derived from the main key.
- S-Boxes (Substitution Boxes): Sixteen 4x16 S-boxes substitute the 48-bit input into a 32-bit output.
- Permutation (P): Permutes the 32-bit output from the S-boxes according to a fixed permutation table.

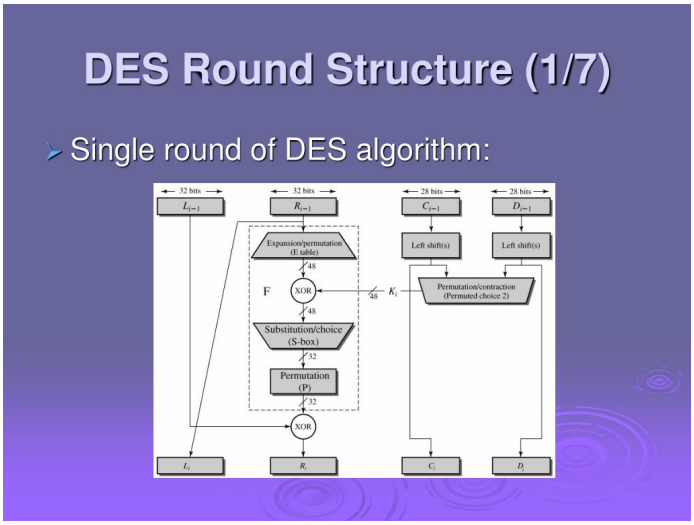


Figure 3: Round key operation

$$\text{input: 64 bits} \rightarrow \text{output: 64 bits}$$

Permuted Choice 1 (PC-1)

Permuted Choice 1 (PC-1) is the first step in deriving the subkeys for each round. It permutes the 64-bit main key according to a fixed permutation table.

$$\text{PC-1}(K) = \text{Output of the Permuted Choice 1 for Key } K$$

$$\text{input: 64 bits} \rightarrow \text{output: 56 bits}$$

Left Circular Shift (LCS)

During key schedule generation, after PC-1, the key is split into two 28-bit halves. Each half undergoes a left circular shift (LCS), where the bits are shifted to the left by a certain number of positions.

$$\text{LCS}(C_i, D_i) = (C_{i+1}, D_{i+1}) \quad \text{where } C_i \text{ and } D_i \text{ are the left and right halves of the key in round } i$$

$$\text{input: 56 bits} \rightarrow \text{output: 56 bits}$$

Permuted Choice 2 (PC-2)

After the Permuted Choice 1 (PC-1) step, PC-2 further transforms and selects specific bits from the 56-bit key to generate the round keys.

- **Input:** PC-1 generates two 28-bit halves from the original 64-bit key:
  - C_0 (28 bits): Left half of the key after PC-1.
  - D_0 (28 bits): Right half of the key after PC-1.
- **Round Key Generation:** PC-2 generates 16 round keys, one for each round of DES encryption.
- **Process:**
  - PC-2 takes C_i and D_i (where i ranges from 0 to 15 for each round) as input.
  - Each round key K_i is derived by selecting 48 bits from C_i and D_i, and then permuting these selected bits according to a fixed permutation table.
- **Selection and Permutation:**
  - From C_i and D_i, 48 bits are selected based on the positions specified in the PC-2 permutation table.

– The selected bits are then permuted to generate K_i, the round key for round i.

•

input: 56 bits → output: 48 bits

Inverse Permutation (IP^{-1})

At the end of the 16 rounds, the output of the final round undergoes the Inverse Permutation (IP^{-1}). This step is the inverse of the Initial Permutation and rearranges the bits of the ciphertext to produce the final encrypted output.

$$IP^{-1}(C) = \text{Output of the Inverse Permutation for Ciphertext } C$$

input: 64 bits → output: 64 bits

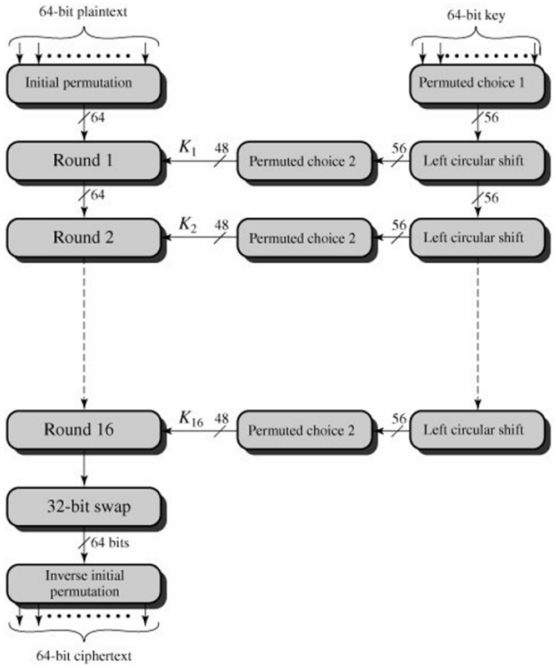


Figure 4: DES algorithm

4.1.2 AES

- AES : Advance Encryptioin Algorithm

- It is a private key cryptosystem with three different keylengths -128,192,257.
- It is a block cipher, but it has nothing to do with the Feistel structure
- It stores the values (plaintext,key,ciphertext) in the matrix form.
- Plaintext block: 128 bits (4 words, as 1word = 32 bits)
- Key size = 128 bits (4 words)
- No of subkeys : 10 subkeys (or 12 or 14)
- No of rounds 10 rounds (or 12 or 14) .
  In each round we use 1 subkey + we have to use the original 128 bits long key before applying the round functioin
- Ciphertext block : 128 bits

Operations in AES

- **Add Round Key:** This operation XORs the 128-bit plaintext with the initial key K_0.
- **Shift Rows Operation:**
  1. Bytes in each row of the 4x4 state matrix are cyclically shifted left:
  2. First row remains unchanged.
  3. Second row shifts left by one position.
  4. Third row shifts left by two positions.
  5. Fourth row shifts left by three positions.
- **Mix Columns Operation:**
  1. This operation transforms columns of the state matrix using a matrix-vector multiplication.
  2. Binary sequences are multiplied with numbers using left shifts and additions through XOR operations.
- **Key Expansion (Generation of Subkeys):**
  1. The 128-bit private key matrix is used to generate subkeys on a word-by-word basis.
  2. Each word undergoes rotation, substitution using an S-box, and XOR with previous words and values from an Rcon table.

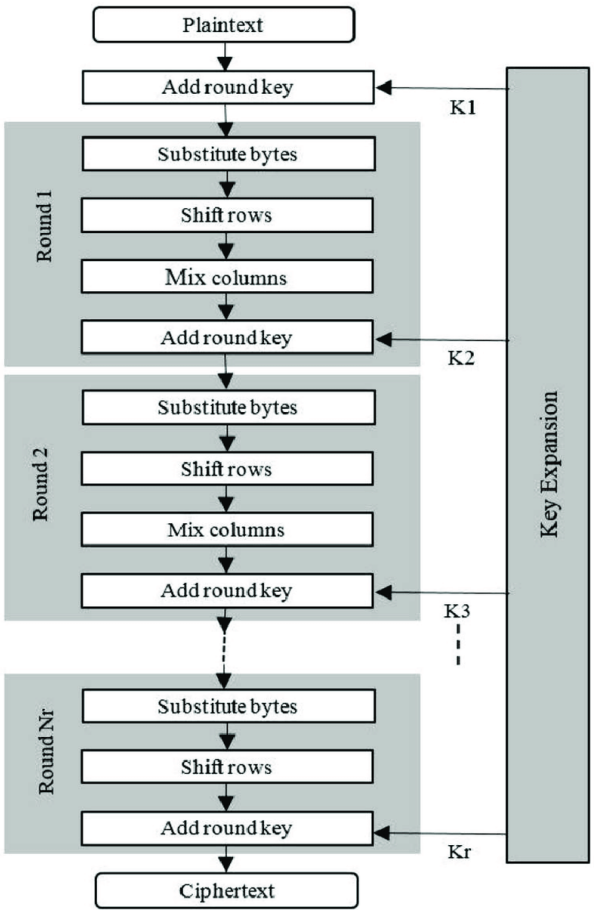


Figure 5: AES algorithm

4.1.3 Block cipher mode of operation:

A block cipher encrypts one block of plaintext at a time. The protocol for encrypting multiple blocks of plaintext is dictated by the mode of operation.

- **ECB mode:**
  - Direct encryption of each block of input plaintext where the output is in the form of blocks of encrypted ciphertext of b bits.
  - Generally, if a message is larger than b bits in size, it can be broken down into a bunch of blocks and the procedure is repeated.
  - Parallel encryption of bits of blocks is possible, making it the fastest block cipher. However, there is a direct relationship between plaintext and ciphertext, making ECB the weakest block cipher.

- **CBC mode:**
  - In CBC mode, the previous ciphertext block is given as input to the next encryption algorithm after XORing with the corresponding plaintext block.
  - Hence, each ciphertext block is produced by encrypting an XOR output of the previous ciphertext block and the present plaintext block.
  - The first plaintext block, not having a previous ciphertext block, is instead XORed with an Initialization Vector (IV).

The padding problem: If the plaintext is not divisible by 128 bits, then we append some extra bits to the plaintext.

Padding modes:

- We can add extra bits.
- We can add white spaces to the plaintext.
- We can use CMS (cryptographic message syntax).

Pad with bytes all of the same value as the number of padding bytes:

1. Padding byte needed: 0x01 (0x represents hexadecimal representation)
2. Padding byte needed: 0x02
3. Padding byte needed: 0x03
4. Padding byte needed: 0x04

4.2 Stream cipher:

4.2.1 Definition:

A stream cipher is a type of encryption algorithm that encrypts plaintext one bit or byte at a time using a key stream. Here are the key characteristics and workings of a stream cipher:

1. **Encryption Process**:
   - The plaintext is combined with a pseudorandom or random key stream (bit sequence) using a bitwise operation (usually XOR).
   - Each bit or byte of the plaintext is encrypted separately.
2. **Key Stream Generation**:

- The key stream is typically generated by a keystream generator based on a cryptographic key.
- The key stream generator can produce a sequence of bits that appear random (pseudorandom) or truly random.

  Mathematically stream cipher is defined as:A cipher defined over (k, m, c) is a pair of efficient algorithms (E, D) where

  $$E : k \times m \to c$$

  $$D : k \times c \to m$$

  such that for all m, k, $D(k, E(k, m)) = m$. E is often randomized. D is always deterministic. In stream cipher, one byte is encrypted at a time. instead of random key they produce pseudo random keys

3. Example: One-Time-Pad algorithm

4.2.2 One Time Pad Algorithm:

- generate a truly random sequence with as many random numbers as the number of letters in the plaintext.
  We cannot reuse the same randomly generated sequence over and over again.
- shift the letters in the plaintext with the random numbers in the exact same manner as we have seen with Vigenere cipher or with Caesar cipher.
- There is no information leaking because every letter in the cipher text is equally likely.
  Random numbers can eliminate information leaking.
  And this is the final conclusion that is extremely important in symmetric cryptosystems that random numbers can eliminate information leaking. here , down + represents XOR operation
- E(k,m) = k+m = c, D(k,c) = c + m =k
  Proof: k + c = k + (k+m) = (k + k) + m = 0 + m = m The length of key is same as the length of the plaintext.

4.2.3 Perfect Secrecy

- A cipher (E, D) over (K, M, C) has perfect secrecy if for all $m_0, m_1 \in M$,

  $$\Pr[E(k, m_0) = c] = \Pr[E(k, m_1) = c]$$

- OTP has perfect secrecy.
- Perfect secrecy $\to |K| \geq |M|$.

5 Mathematics in cryptography:

5.1 Discrete Probability:

- **U** (Universe): finite set
- **Probability distribution**: over U is a function $P : U \to [0, 1]$ such that $\sum_{x \in U} P(x) = 1$.
- **Random variable**: A random variable $X : U \to V$.
- **The uniform random variable**: Let U be some set. For all $a \in U$: $\Pr[r = a] = \frac{1}{|U|}$.
- **Deterministic algorithms:** $y \leftarrow A(m)$.
  Same output is generated every time for the same input.
- **Randomized algorithms:** $y \leftarrow A(m, r)$, where $r \leftarrow \{0, 1\}^n$.
  We get different output every time , with the same input.
- **Independence**:
  events A and B are independent if Pr[A and B] = Pr[A].Pr[B]
  Random variables X, Y taking values in V are independent if for all $a, b \in V$:
  $$\Pr[X = a \text{ and } Y = b] = \Pr[X = a] \cdot \Pr[Y = b].$$

5.2 Modular arithmetic:

- Congruence: Two integers a and b are said to be congruent modulo m, denoted as $a \equiv b \pmod{m}$, if they have the same remainder when divided by m. Congruence is a relation, not an operation.
- Fermat's little theorem:
  Let p be a prime number. For any integer a (where a is not divisible by p), the expression $a^{p-1} \equiv 1 \pmod{p}$ holds true, which means $a^{p-1} - 1$ is an integer multiple of p.
- (a + b) mod n = (a mod n + b mod n) mod n
- (a b) mod n = (a + n b) mod n
- (a b) mod n = ((a mod n) (b mod n)) mod n

5.2.1 Finding Prime Numbers

There are several approaches to find prime numbers:

1. **Naive Algorithm:** Consider all the numbers between 2 and $\sqrt{N}$. If a given number N divides any number in this range, then N is not a prime.
   Code Part for naive approach:

```
def is_prime(num):
    if num < 2:
        return False
    # 2 is the only even prime number
    if num == 2:
        return True
    for n in range(3, math.isqrt(num) + 1, 2):
        if num % n == 0:
            return False
    return True
```

2. **Fermat's algorithm:**
   We can use fermat's little theorm to check whether a given N number is prime or not.
   $a^{N-1} \equiv 1 \pmod{N}$
   Code for Fermat's algorithm:

```
def is_prime(n,k=20):
if n<=1:
    return False
for  _ in range(k):
    a = random.randint(2,n) -1
    if pow(a,n-1,n) != 1:
        return False
return True
```

5.2.2 Integer factorization:

- **Fundamental theorm of Arithmetic:**This theorm states that every positive integer can be written uniquely as a product of prime numbers. Prime factorization is a trapdoor-function.
- code for finding factors:

```
 def get_factors(num):
factors = []
limit = sqrt(num)
for n in range(1,floor(limit)+1):
    if num % n == 0 :
        factors.append([n, num/n])
return factors
```

5.2.3 Discrete Loagarithm Problem:

- The equation $a \equiv b^c(\pmod{m})$ represents modular exponentiation, where $a = b^c \mod m$.

If we know b, c, m, computing a is straightforward.
However, if we know a, b, m, the problem of finding c such that $a \equiv b^c \pmod{m}$ is called the discrete logarithm problem (DLP).

Solving the discrete logarithm problem is generally considered difficult, especially for large prime numbers m.

- it is a trapdoor function.
- Code of discrete logarithm problem:

```
def discrete_logarithm(a,b,m):
    c=1
    while pow(b,c) % m != 0:
        c+=1
    return c
```

5.3 Euler's Totient Function

Euler's Totient function $\Phi(n)$: For any n, $\Phi(n) = \#\{x \in \{1, 2, \ldots, n-1\} \mid (x, n) = 1\}$ where (a, b) denotes the Greatest Common Divisor (GCD) of numbers a and b, and $\#$ means the cardinality of the set.

Examples:

$$\Phi(5) = 4, \quad \Phi(2) = 1$$

Some specific values of Euler's Totient function:

- $\Phi(p) = p - 1$ if p is Prime.
- $\Phi(p^n) = p^n - p^{n-1}$ if p is Prime.
- $\Phi(ab) = \Phi(a)\Phi(b)$ if $(a, b) = 1$.

In general, if $n = \prod_{i=1}^{r} p_i^{k_i}$ where p_i is a prime number, then:

$$\Phi(n) = n \prod_{i=1}^{r} \left(1 - \frac{1}{p_i}\right)$$

5.4 Chinese Remainder Theorem (CRT)

Before moving directly to the Chinese Remainder Theorem, we need to introduce some symbols.

- $\mathbb{Z}_n$ represents the set of whole numbers less than n, i.e., the set $\{0, 1, 2, \ldots, n-1\}$. This set is called the multiplicative group of integers modulo n.
- $\mathbb{Z}_n^*$ represents the set $\{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$. Note that the cardinality of this group is $\varphi(n)$, where φ is Euler's totient function.
- An integer a is said to be congruent to b modulo n if n divides $a - b$. This is represented as $a \equiv b \pmod{n}$.

Theorem (Chinese Remainder Theorem): For every element $x \in \mathbb{Z}_{pq}$, where p and q are coprime integers, there exists a unique pair ($x \mod p, x \mod q$) in $\mathbb{Z}_p \times \mathbb{Z}_q$. Conversely, for every pair (r, s) in $\mathbb{Z}_p \times \mathbb{Z}_q$, there exists a unique $x \in \mathbb{Z}_{pq}$ such that $r = x \mod p$ and $s = x \mod q$.

Definition: An integer a is said to be the modular inverse of another integer b modulo n if $ab \equiv 1 \pmod{n}$. One can find the modular inverse using the Extended Euclidean Algorithm.

6 Asymmetrical cryptography

6.1 *Problems with private key cryptosystems:

- Somehow we have to share the private keys.
- Lack of authentication.
- We have to use a huge number of private keys in an organization.

6.2 *Advantages of asymmetrical cryptography:

- we have authentication. we know who is the sender and we can verify it easily.
- the scale of usage of no of keys is less comparitively.

6.3 Definition:

- The users have two types of keys :
  public key for encryption , private key for decryption
- We can calculate private key from the public key.
- It is about modular arithmetic and prime numbers.
- Trapdoor function: is a function that is easy to compute in one direction yet difficult to compute in the other direction (finding its inverse).
- Examples of asymmetric cryptosystem : Diffie-Hellman key exchange, RSA algorithm, Elliptic curve cryptography.

6.4 1.Diffie-Hellman Key Exchange

- Here, we are able to exhange private keys over a public channel.
- We are not sharing the information during the key exchange..
- **Primitive root**: g is the primitive root of n if $g \mod n, g^2 \mod n, \ldots, g^{(n-1)} \mod n$ generates all the integers within the range $[1, n-1]$.
- We have to make sure that size of keyspace is large, so that it is hard to predict.
- Size of keyspace is proportional to n. Therefore, n must be large.
- if n is composite, we can crack by Chinese Remainder Theorem.So, it is advisible to use prime number only.

- $a \equiv b^c \pmod{m}$
  If we know b,c,m then this is called modular exponentiation, which is not hard to solve.
  If we know a,b,m then this is called discrete logarithm problem which is a very difficult problem to solve.
- It lacks authentication which can be solved by SHA256 hashes for authentication.

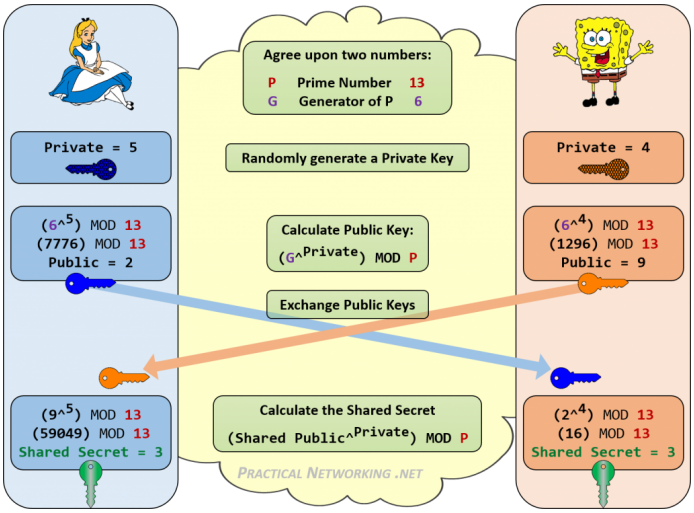


Figure 6: Diffie-Hellman algorithm

6.5 2.RSA cryptosystem

- It is a public key cryptosystem.
- RSA is secure because of integer factorization problem.
- Public Key: (e, n)
  Private Key: (d, n)
  $\rightarrow$ First, we have to transform the plaintext into blocks, where every block is smaller than n.
  $\rightarrow$ Ciphertext block: $ciphertext\_block = plaintext\_block^e \mod n$
  Plaintext block: $plaintext\_block = ciphertext\_block^d \mod n$

6.5.1 RSA algorithm:

- Select two prime numbers, say p and q.
- Calculate $n = pq$. Hence, $\Phi(n) = (p-1)(q-1)$.

- Choose a value e such that $1 < e < \Phi(n)$ and $\gcd(e, \Phi(n)) = 1$, where gcd denotes the Greatest Common Divisor.
- Calculate the modular inverse of e modulo $\Phi(n)$, denoted as d, such that $ed \equiv 1 \pmod{\Phi(n)}$.
- Hence, we have the public key (e, n) and private key (d, n).

Now, suppose we want to encrypt a message. It usually consists of characters, so we need to convert it to an integer to perform mathematical operations on it. Let us understand this through an example:

Suppose we want to convert the message "the" to an integer. The ASCII bytes array is [116, 104, 101]. From this array, we convert each integer to hexadecimal and then concatenate them. In this case, the number would be 0x746865 (obtained by joining the list ['74', '68', '65']). Finally, convert the obtained number to base 10:

$$(746865)_{16} = (7628901)_{10}$$

Now that we have obtained the number M from plaintext, we can continue with the encryption scheme. The ciphertext C is calculated as follows:

$$C = M^e \mod n$$

Once the correct receiver receives the ciphertext, they recover M using the private key d with the following relation:

$$M = C^d \mod n$$

6.5.2 Problems of RSA:

- Factorization is the trapdoor function in RSA, but it has neven been proven that factorization is hard.
- General Number Fiels Sieve (GNFS) algortihm is the fastest known algorithm for prime factorization.
- quantum computing will make RSA obsolete.

The code for RSA algorithm is :

```
RANDOM_START = 1e3
RANDOM_END = 1e5

def is_prime(num):
    if num < 2:
        return False
    if num == 2:
        return False
```

```
    if num % 2 == 0:
        return False
    for i in range(3, int(num**0.5) + 1, 2):
        if num % i == 0:
            return False
    return True

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def modular_inverse(a, b):
    if a == 0:
        return b, 0, 1
    div, x1, y1 = modular_inverse(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return div, x, y

def generate_large_prime(start=RANDOM_START, end=RANDOM_END):
    num = random.randint(start, end)
    while not is_prime(num):
        num = random.randint(start, end)
    return num

def generate_rsa_keys():
    p = generate_large_prime()
    q = generate_large_prime()
    n = p * q
    phi = (p - 1) * (q - 1)
    e = random.randrange(1, phi)
    while gcd(e, phi) != 1:
        e = random.randrange(1, phi)
    d = modular_inverse(e, phi)[1]
    return (d, n), (e, n)

def encrypt(public_key, plain_text):
    e, n = public_key
    cipher_text = []
    for char in plain_text:
        a = ord(char)
        cipher_text.append(pow(a, e, n))
    return cipher_text

def decrypt(private_key, cipher_text):
```

```
d, n = private_key
plain_text = ''
for num in cipher_text:
    a = pow(num, d, n)
    plain_text += chr(a)
return plain_text
```

6.6 3.Elliptic Curve Cryptography

6.6.1 Definition:

- It is public key cryptosystem.
- Bitcoin uses ECC.
- Elliptic Curve: Let $a, b \in \mathbb{R}$ be constants such that $4a^3 + 27b^2 \neq 0$. A non-singular elliptic curve E is the set of solutions $(x, y) \in \mathbb{R} \times \mathbb{R}$ to the equation:
$$y^2 = x^3 + ax + b.$$
They are symmetric about x axis.
They are non singular.
- Point addition operation:
$$P(x_1, y_1) + Q(x_2, y_2) = R(x_3, y_3)$$
- Point doubling operation:
$$P(x_1, y_1) + P(x_1, y_1) = 2P(x_1, y_1) = P(x_2, y_2)$$
- **Double and Add algorithm:**
Scalar multiplication involves computing kP, where k is an integer (the scalar) and P is a point on the elliptic curve. Direct computation of kP by repeated addition is computationally expensive, particularly for large values of k, which are common in cryptographic applications. The double-and-add algorithm provides a more efficient method by leveraging the binary representation of the scalar k.

Double-and-Add Algorithm for Point Multiplication
Input: elliptic curve E together with an elliptic curve point P
a scalar $d = \sum_{i=0}^{t} d_i 2^i$ with $d_i \in 0, 1$ and $d_t = 1$
Output: $T = dP$
Initialization:
$T = P$
Algorithm:
1 FOR $i = t - 1$ DOWNTO 0
1.1 $T = T + T \bmod n$
IF $d_i = 1$
1.2 $T = T + P \bmod n$
2 RETURN (T)

Figure 7: Double and Add algorithm

- **Elliptic curve discrete logarithm problem (ECDLP):**
  Given an elliptic curve E defined by the equation $y^2 = x^3 + ax + b$ and order $|E| = n$, and points $P = (x_1, y_1)$ and $R = (x_2, y_2)$ on the curve, the aim is to find $1 \leq x \leq n$ such that $xP = R$.
  This problem serves as a trapdoor function.

6.6.2 Elliptic Curve Diffie-Hellman Algorithm:

The following are the domain parameters specified:

- E — the elliptic curve itself
- G — a point on E that is set as the base point

Breakdown:

1. I generate a random integer a as my private key
2. I generate my public key A by computing aG
3. You generate a random integer b as your private key
4. You generate your public key B by computing bG
5. We exchange public keys
6. I calculate K as aB
7. You calculate K as bA

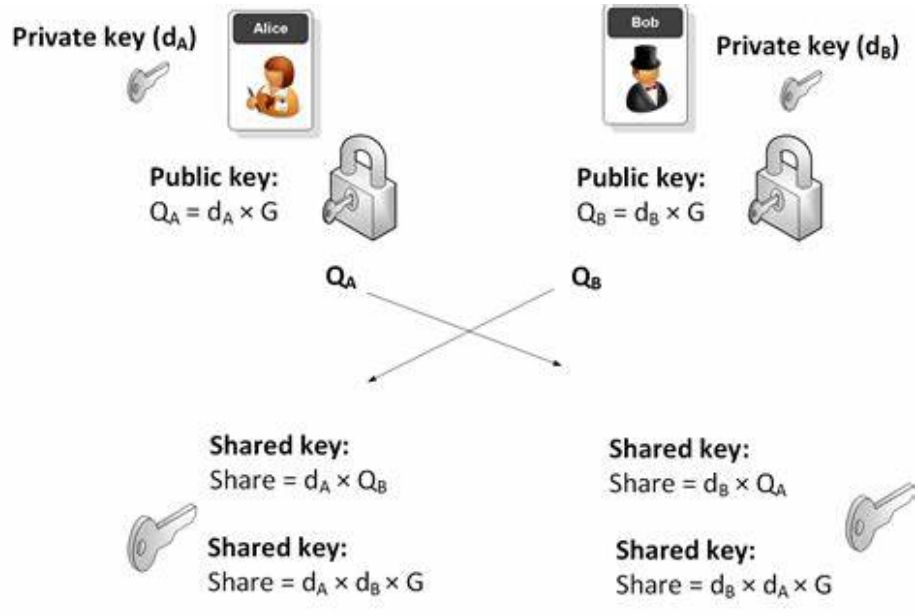


Figure 8: ECDHA

7 Digital Signatures

- How can the recipient of a digital message be sure of who the sender was?" Here is where digital signatures come into play.
- A digital signature is an electronic, encrypted, stamp of authentication on digital information such as email messages, macros, or electronic documents.
- A digital signature is an electronic equivalent of a handwritten signature that allows a receiver to persuade a third party that the message was indeed sent by the sender.
- It should be difficult to forge, and it should be difficult to remove from one document and attach it to some other document.

7.1 Components of Digital Verification Scheme:

- K^{Pri}: A private signing key.
- K^{Pub}: A public verification key.
- **Sign**: A signing algorithm that takes as input a digital document D and a private key K^{Pri}, and returns a signature D^{sig} for D.
- **Verify**: A verification algorithm that takes as input a digital document D, a signature D^{sig}, and a public key K^{Pub}. The algorithm returns **True** if D^{sig} is a valid signature for D associated with the private key K^{Pri}, and **False** otherwise.
  Importantly, the verification algorithm only has access to the public key K^{Pub} and does not know K^{Pri}.

Today's digital signature methods can be categorized based on a mathematical issue that provides the foundation for their security:

- **Integer Factorization (IF) Schemes:** They rely their security on the integer factorization problem's intractability. RSA Signature Schemes are one example.
- **Discrete Logarithm (DL) Schemes:** Their security is based on the intractable nature of the discrete logarithm challenge in a finite field.
- **Elliptic Curve (EC) Schemes:** They rely their security on the elliptic curve discrete logarithm problem's intractability. The Elliptic Curve Digital Signature Algorithm, for example, is being used in this investigation and is without a doubt the most recent of the many designs.

7.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

- **Private key:** A secret number (typically generated using SHA256).
  - We can sign a given message with the private key.
- **Public key:** Generated from the private key. The public key is a 2D point coordinate on an elliptic curve.
  - We can verify the message (that has the signature) with the help of the public key.

Public components:

1. The elliptic curve $y^2 = x^3 + ax + b \mod n$ is public.
2. A generator point $R(x_r, y_r)$ on the elliptic curve, with the order $|E| = n$ that is the size of the elliptic curve space.

Signature Generation: Alice wants to send a signed message m to Bob.

1. Alice's private key is a randomly chosen integer d_A in the range $[1, n-1]$.
2. Her public key is $Q_A = d_A \cdot R(x, y)$.
3. Compute $z = \text{SHA}(m)$ and transform it into an integer.
4. Generate a random integer k in the range $[1, n-1]$.
5. Compute the point $(x, y) = k \cdot R(x, y)$.
6. Calculate $r = x \mod n$.
7. Compute $s = k^{-1} \cdot (z + r \cdot d_A) \mod n$. If r or s is zero, perform the random generation step again.
8. The signature is the pair (r, s).

Signature Verification: Bob wants to verify that a signed message belongs to Alice. Then:

1. Verify that Alice's public key Q_A is valid and that r and s are within the range $[1, n-1]$.
2. Calculate $z = \text{SHA}(m)$.
3. Compute $u_1 = z \cdot s^{-1} \mod n$ and $u_2 = r \cdot s^{-1} \mod n$.
4. Calculate the point $(x, y) = u_1 \cdot R(x_r, y_r) + u_2 \cdot Q_A$.
5. The signature is valid if and only if $r = x \mod n$.

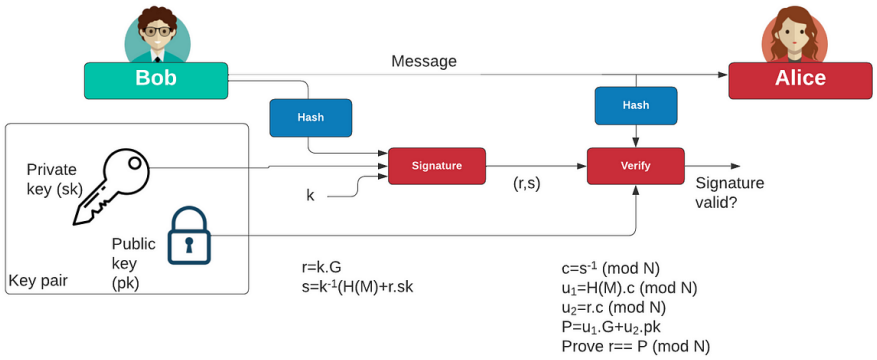


Figure 9: ECDSA

7.3 RSA Digital Signature:

The user chooses two large secret primes p and q and she publishes their product $N = pq$ and a public verification exponent v. The user uses their knowledge of the factorization of N to solve the congruence

$$sv \equiv 1 \pmod{(p-1)(q-1)}$$

In the present setup, s is their signing exponent and v is their verification exponent. In order to sign a digital document D, which we assume to be an integer in the range $1 < D < N$, The user computes

$$S \equiv D^s \pmod{N}$$

Our verifier, verifies the validity of the signature S on D by computing S^v mod N and checking that it is equal to D.

This process works because Euler's formula tells us that

$$S^v \equiv D^{sv} \equiv D \pmod{N}$$

***INEFFICIENCY OF DIGITAL SIGNATURES FOR LARGE DOCUMENTS**:
The natural capability of most digital signature schemes is to sign only a small amount of data, say b bits, where b is between 80 and 1000.
The standard solution to this problem is to use a hash function, which is an easily computable function

$$\text{Hash} : (\text{arbitrary-size-documents}) \to \{0,1\}^k$$

that is very hard to invert. Then, rather than signing their document D, The user computes and signs the hash Hash(D).

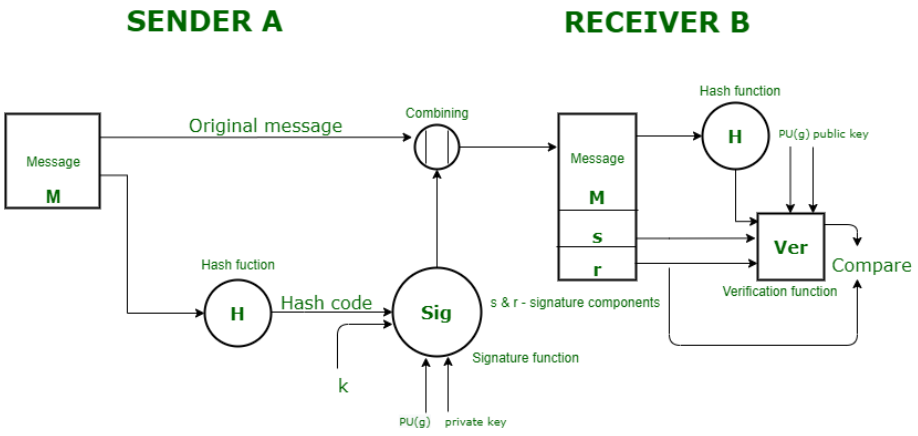


Figure 10: DSS

8 Hashing:

- A cryptographic hash function is a mathematical algorithm that maps data of arbitrary size to a bit of array of a fixed size.
- message(arbitary length) -¿ message digest or hash(fixed size)
  $h : \{0,1\}^* \rightarrow \{0,1\}^d$
- for MD5: d is 128 bits, SHA1: d is 160 bits, SHA2: d is 256 bits.
- all operations are public.
- There are no hash functions.
- It is deterministic and random(pseudo-random).

8.1 Properties of Hash function:

- **Pre-image resistance:** It is exponentially hard (computationally infeasible) to determine the message m from the hash value $h(m)$.
- **Second pre-image resistance:** Given m_1, it is infeasible to find m_2 such that $h(m_1) = h(m_2)$, ensuring that the hashes are identical.
- **Collision resistance:** It is infeasible to find any two messages m_1 and m_2 such that $h(m_1) = h(m_2)$. Breaking collision resistance is typically easier than breaking second pre-image resistance.
- Hash function should be deterministic(output must be same for the same input every time), one-way, collision-free, avalanche effect(a little change in the input results in a completely different output hash.).

9 Message aunthentication code:

The message authentication code, also known as a digital authenticator, is used as an integrity check that uses a secret key held by two parties to validate information sent between them.
It is supported by using a cryptographic hash or symmetric encryption technique.

9.1 HMAC(Hash based Message Authentication Code)

- It is a type of MAC(message authentication code) that is aquired by executing a hash function on the data and a secret shared key.
- Digital signatures are nearly similar to HMACs i.e they both employ a hash function and a shared key. The difference lies in the keys i.e HMAC use symmetric key(same copy) while Signatures use asymmetric (two different keys).
- The client makes a unique hash (HMAC) for every request. When the client requests the server, it hashes the requested data with a private key and sends it as a part of the request. Both the message and key are hashed in separate steps making it secure. When the server receives the request, it makes its own HMAC. Both the HMACS are compared and if both are equal, the client is considered legitimate.
- The formula for HMAC:
  HMAC = hashFunc(secret key + message)

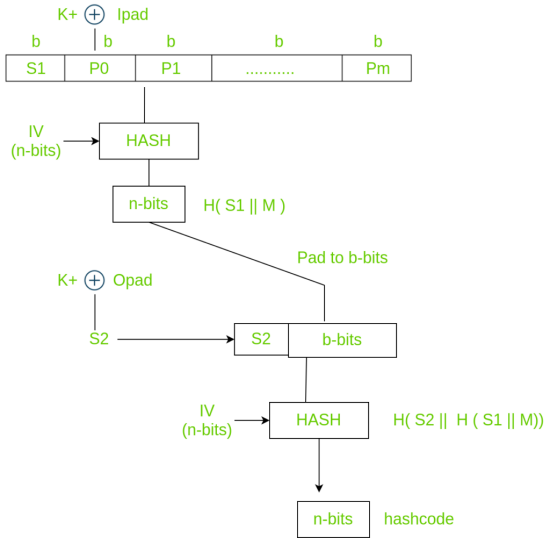


Figure 11: HMAC

- There are three types of authentication functions. They are message encryption, message authentication code, and hash functions.
- Before applying hash(HMAC), we have to compute S bits and then append it to plain text and after that apply the hash function. For generating those S bits we make use of a key that is shared between the sender and receiver.

9.2 CBC-MAC

- A cipher block chaining message authentication code (CBC-MAC) is a technique for constructing a message authentication code (MAC) from a block cipher.
- The message is encrypted with some block cipher algorithm in cipher block chaining (CBC) mode to create a chain of blocks such that each block depends on the proper encryption of the previous block.

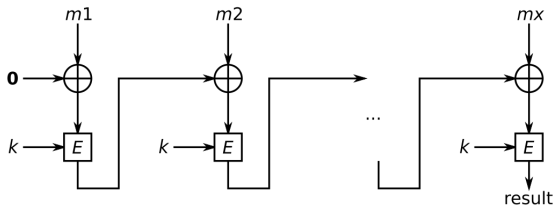


Figure 12: CBC-MAC

10 Collision Resistance

10.1 Merkle-Damgard Scheme in Cryptography

- MD scheme is used to build collision-resistant cryptographic hash functions from collision-resistant one-way compression functions. It is used in algorithms like SHA-1, SHA-256, etc.

1. **Stage-1:** Design a fixed-length, collision-resistant compression function.
2. **Stage 2:**Design a CRHF H for arbitrary length messages, using 'h'.

- Encode the input M (with length L) for HMD to make the encoded message a multiple of l bits. If L is already a multiple of l bits, then add an additional dummy block.

Original Message||Padding Length

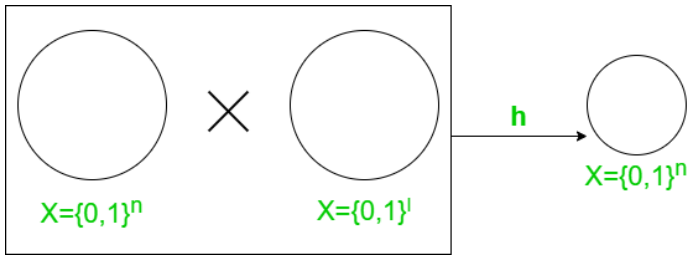
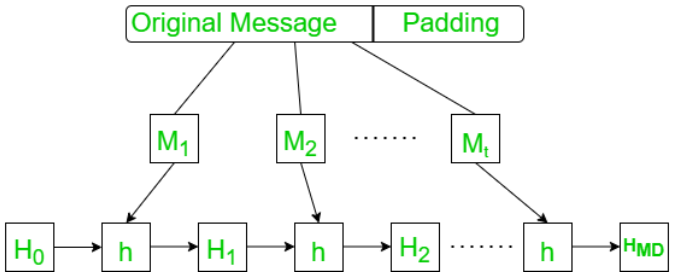


Figure 13: Collision resistant compression function



- The message is then considered as t-blocks each of n bits, i.e., $M1, M2, \ldots, Mt$. Apply function h iteratively over the blocks of M and the previous outcome of h (i.e., $H1, H2, \ldots, HMD$).

$$F(H_{i-1}, M_i) = H_i$$

- Before starting iteration, an initial vector ($H0$) is used.
- The digest HMD created after the tth iteration is the compressed hash value of the original message.

11

References

[1] *Udemy*, Available at: link

[2] *Coursera*, Available at: link

[3] *Geeks for Geeks*, Available at: link