Caesar cipher :

Definition: The Caesar Cipher is one of the oldest and simplest encryption techniques, named after Julius Caesar, who reportedly used it to communicate with his generals securely.

⚙️ Working Principle : It is a substitution cipher, meaning each letter in the plaintext is replaced by another letter a fixed number of positions down the alphabet.

The number of positions shifted is called the key or shift value.

Limitations :

- Very weak encryption. Since there are only 25 possible shifts, it’s easy to break using brute force.

- Does not change letter frequencies — so it’s also vulnerable to frequency analysis attacks.

Advantages :

- Simple to implement and understand.

- Useful for teaching basic cryptography concepts.

def caesar(text,shift):

out = ""

for c in text:

if c.isalpha():

base = ord('A') if c.isupper() else ord('a') if letter is uppercase -> base is 65 else 97

out+=chr((ord(c)-base+shift)%26+base) ord ->converts from char to int, - base to give position in alphabet, shift for cipher, +base to convert to ascii, chr to int->char

else:

out+=c

return out

text = input("Enter text to encrypt: ")

shift = int(input("enter your shift: "))

encrypted = caesar(text,shift)

print("Encrypted text is : ",encrypted)

Atbash cipher :

The Atbash Cipher is a monoalphabetic substitution cipher that replaces each letter of the alphabet with its opposite letter — meaning A ↔ Z, B ↔ Y, C ↔ X, and so on.

It is one of the oldest known ciphers, originally used in the Hebrew language for encoding the alphabet.

Working Principle:

- The cipher uses a fixed substitution pattern — there is no key or shift value like in Caesar Cipher.

- It simply reverses the alphabet.

Advantages :

Very simple and requires no key.

Easy to implement manually.

Limitations :

Extremely weak encryption, as the substitution pattern is fixed and known.

Can be easily broken through simple analysis or even by observation.

def atbash(text):

out=""

for c in text:

if c.isalpha():

base = ord('A') if c.isupper() else ord('a')

out += chr(25 - (ord(c) - base)+base)

else:

out+=c

return out

text = input("Enter your text")

encrypted = atbash(text)

print("Encrpted text: " + encrypted)

ROT13 cipher :

Definition: The ROT13 Cipher (short for “Rotate by 13 places”) is a special case of the Caesar Cipher, where each letter in the plaintext is replaced by the letter 13 positions ahead in the alphabet. It is a simple substitution cipher that is often used to obscure text rather than truly encrypt it.

Working Principle:

ROT13 uses a fixed shift value of 13.

The alphabet has 26 letters, so applying ROT13 twice returns the original text — making it self-inverse.

Advantages:

- Simple and fast to apply.

- No key needed.

- Useful for hiding spoilers, jokes, or sensitive text on the internet.

Limitations:

- Offers no real security — anyone can easily decode it.

- Not suitable for modern encryption or data protection.

def rot13(text):

out = ""

for c in text:

if c.isalpha():

base = ord('A') if c.isupper() else ord('a')

out += chr((ord(c) - base + 13)%26 + base)

else:

out+=c

return out

text = input("Enter text: ")

encrypted = rot13(text)

print("Encrypted msg : " ,encrypted)

Rail fence cipher: The **Rail Fence Cipher** is a **transposition cipher**, meaning **it doesn’t change the letters**, but **rearranges their order** to encrypt the message.

It’s called “rail fence” because the letters are written in a **zigzag pattern** (like rails on a fence), and then read row by row to form the ciphertext.

**Example**

Let’s take a simple message:

**Message:** ATTACK AT DAWN  
(remove spaces → ATTACKATDAWN)

and suppose we use **2 rails (rows)**.

**Step 1: Write in a zigzag pattern**

We write the letters diagonally down and up across the rails.

A T A A W

T A K T D N

**Step 2: Read row by row**

Now, read **row 1**, then **row 2**:

**Ciphertext:**  
→ Rail 1: ATA AW → ATAAW  
→ Rail 2: TAKTDN → TAKTDN

**Final Ciphertext:** **ATAAWTAKTDN**

**Step 3: Decryption**

To decrypt, you reverse the process:

1. Write the ciphertext in **zigzag pattern** again (knowing the number of rails).
2. Read the message **diagonally down and up** to reconstruct the original text.

**Advantages:**

* Simple and easy to implement.
* No special symbols or math.

**Disadvantages:**

* Very weak — easily broken by inspection.
* Only rearranges letters; doesn’t disguise them.

def encrypt\_2rail(plaintext: str) -> str:

# remove spaces (optional) — comment out if you want to keep spaces

pt = plaintext.replace(" ", "")

even = pt[0::2] # characters at indices 0,2,4,...

odd = pt[1::2] # characters at indices 1,3,5,...

return even + odd

def decrypt\_2rail(ciphertext: str) -> str:

n = len(ciphertext)

first\_len = (n + 1) // 2 # rail 0 length (ceil(n/2))

first = ciphertext[:first\_len] # rail 0 chars

second = ciphertext[first\_len:] # rail 1 chars

pt\_chars = []

i0 = 0

i1 = 0

for i in range(n):

if i % 2 == 0:

pt\_chars.append(first[i0]); i0 += 1

else:

pt\_chars.append(second[i1]); i1 += 1

return "".join(pt\_chars)

# Example

plain = "ATTACK AT DAWN"

ct = encrypt\_2rail(plain)

print("Ciphertext:", ct)

print("Decrypted :", decrypt\_2rail(ct))

Diffie Hellmann :

Definition: The Diffie–Hellman (DH) key exchange is a cryptographic method that allows two parties to generate a shared secret key over an insecure communication channel without sending the key directly. This shared key can later be used for symmetric encryption.

Working :

Step 1: Public parameters

Both parties agree on:

p → a large prime number.

g → a primitive root modulo p (generator).

Step 2: Private keys

Each party selects a private key secretly:

Alice chooses a

Bob chooses b

Step 3: Compute public keys

Alice computes: A = g^a mod p and sends it to Bob

Bob computes: B = g^b mod p and sends it to Alice

Step 4: Compute shared secret key

Alice computes: secret = B^a mod p

Bob computes: secret = A^b mod p

Both values are equal

Step 5: Use the shared secret

The resulting secret key can now be used for symmetric encryption (e.g., AES) to secure communication.

Advantages:

Allows secure key exchange over an insecure channel

Only public values are exchanged, private keys remain secret

Basis for many cryptographic protocols (TLS, VPNs)

Limitations :

Vulnerable to man-in-the-middle attacks if authentication is not used

Security depends on the size of p — small primes are insecure

Does not encrypt messages by itself — only generates a shared key

p = int(input("Prime p: "))

g = int(input("Primitive root g: "))

a = int(input("Alice's private key: "))

b = int(input("Bob's private key: "))

A = pow(g,a,p)

B = pow(g,b,p)

print("Alice's public key: ",A)

print("Bob's public key: ",B)

a\_sec = pow(B,a,p)

b\_sec = pow(A,b,p)

print("Alice ssecret key: ",a\_sec)

print("Bob secret key: ",b\_sec)

RSA algorithm :

Definition: The RSA algorithm is a widely used public-key cryptosystem that allows secure communication over insecure channels. It uses two keys:

Public key (for encryption)

Private key (for decryption)

Working Principle:

Security relies on large prime numbers and the hardness of factoring the product of two primes.

RSA is an asymmetric encryption algorithm:

Anyone can encrypt a message using the public key.

Only the owner of the private key can decrypt it.

Step 1: Key Generation

Choose two distinct large prime numbers p and q.

Compute n = p \* q → modulus for public and private keys.

Compute Euler’s totient: phi = (p-1)\*(q-1)

Choose public exponent e such that 1 < e < phi and gcd(e, phi) = 1.

Compute private exponent d such that (d \* e) % phi = 1 (modular inverse).

Public key: (n, e)

Private key: (n, d)

Step 2: Encryption

Plaintext message M must satisfy 0 < M < n.

Ciphertext C is calculated as: C = M^e mod n

Step 3: Decryption

Receiver computes the original message M using private key d: M = C^d mod n

Advantages:

Secure key exchange without sharing private key.

Basis for digital signatures and secure online communication (HTTPS, SSL/TLS).

Limitations:

Slow compared to symmetric algorithms (e.g., AES).

Requires large primes and careful key generation.

Not suitable for encrypting large amounts of data directly — usually used to encrypt symmetric keys.

p = int(input("Prime p : "))

q = int(input("Prime q: "))

m = int(input("Msg m : "))

n = p\*q

phi = (p-1)\*(q-1)

e = 13

while phi%e==0:

e+=2

d = 1

while (e\*d)%phi != 1:

d+=1

print("Public key: ",n,e)

print("Private key: ",n,d)

c = pow(m,e,n)

print("Encrypted msg: ",c)

decry = pow(c,d,n)

print("Decrypt msg: ",decry)

1. PING (Packet Internet Groper)

To check connectivity between your computer and another device (like a website or IP address).

What it does:

Sends ICMP Echo Request packets to a target.

Waits for Echo Reply.

Measures round-trip time (RTT) and packet loss.

ping google.com

Output shows:

Target IP address

Response time (in milliseconds)

Number of packets sent, received, lost

Why used:

To test if a host is reachable

To check network latency or connection problems

2. TRACEROUTE (Windows: tracert)

To trace the path (or route) packets take from your computer to a destination host.

What it does:

Sends packets with increasing TTL (Time To Live) values.

Each router that handles the packet reduces TTL by 1.

When TTL reaches 0, router replies with its IP — showing the “hop”.

tracert google.com

Output shows:

IP address or domain name of each hop (router)

Time taken for each hop

Why used:

To detect where the delay or failure occurs in the network path.

3. NSLOOKUP (Name Server Lookup)

Purpose:

To find DNS details — i.e., translate domain name ↔ IP address.

What it does:

Queries the DNS server to get IP address of a domain (or vice versa).

nslookup google.com

Output shows:

Domain name

IP address(es)

Name of the DNS server used

Why used:

To verify DNS resolution

To check if DNS records are correct or misconfigured

4. WHOIS

Purpose:

To get registration details about a domain name or IP address.

What it does:

Queries a WHOIS database to get information about:

Domain owner / organization

Registration & expiry dates

Registrar (company handling the domain)

Contact info

whois google.com

Output shows:

Domain Name

Registrar and Registrant info

Creation, Expiry, Update Dates

Name Servers

Why used:

For domain investigation, security research, or troubleshooting website ownership issues.

5. NMAP (Network Mapper)

Purpose:

A powerful network scanning tool used for security auditing and network discovery.

What it does:

Scans IPs, networks, or hosts to find:

Open ports

Running services

Operating system

Potential vulnerabilities

nmap 192.168.1.1

nmap -p 80,443 google.com

nmap -A google.com

Output shows:

Open ports and the services running on them

Host status (up/down)

OS and version (if detected)

Why used:

To map a network

To find security holes or unauthorized devices