A **monoalphabetic substitution cipher** is a type of encryption where each letter of the plaintext is replaced with another letter from the alphabet. The substitution remains **consistent** throughout the message, meaning each letter always maps to the same substitute letter.

The **Vigenère cipher** is a **polyalphabetic substitution cipher**, meaning it uses multiple Caesar ciphers based on a keyword. It was designed to overcome the weaknesses of monoalphabetic ciphers like the one you asked about earlier.

Key Idea:

The Vigenère cipher shifts each letter of the plaintext by a different amount depending on a repeating **keyword**.

How It Works:

1. **Choose a keyword** (e.g., `KEY`).
2. Repeat the keyword to match the length of the plaintext:
   - Plaintext: `HELLO`
   - Keyword: `KEYKE`
3. Use Caesar shift for each letter:
   - Convert each letter to its position in the alphabet (A=0, B=1, ..., Z=25).
   - Add the positions of the plaintext letter and the keyword letter (mod 26).
   - Convert the result back to a letter.

Example:

- **Plaintext:** `HELLO`
- **Keyword:** `KEYKE`

Plaintext	H	E	L	L	O
Keyword	K	E	Y	K	E
Shift	10	4	24	10	4
Ciphertext	R	I	J	V	S

Ciphertext: `RIJVS`

The **Vernam cipher** is a special type of **stream cipher** invented by Gilbert Vernam in 1917. It forms the basis of the **One-Time Pad**, which is the only proven **unbreakable cipher** when used correctly.

Example:

Plaintext: `HELLO`
One-Time Pad (Key): `XMCKL`

Step 1: Letter → Number Mapping

Letter	Plaintext (P)	Key (K)	P (num)	K (num)
H	H	X	7	23
E	E	M	4	12
L	L	C	11	2
L	L	K	11	10
O	O	L	14	11

Step 2: Add P + K (mod 26)

P + K	Sum	Mod 26	Cipher (number)	Cipher (letter)
H + X	30	4	4	**E**
E + M	16	16	16	**Q**
L + C	13	13	13	**N**
L + K	21	21	21	**V**
O + L	25	25	25	**Z**

Final Ciphertext: `EQNVZ`

The **Diffie-Hellman algorithm** is a **key exchange protocol** that allows two parties to securely share a **secret key** over an insecure channel (like the internet), **without actually transmitting the key itself**.

It's widely used in secure communications like HTTPS, VPNs, and messaging apps.

🔑 Key Concepts

- **Asymmetric key exchange** method (not encryption directly)
- Based on the **difficulty of computing discrete logarithms**
- Each party generates a private key and a corresponding public key
- Both parties combine their private key with the other's public key to compute the **same shared secret**

⚙️ How It Works (Step-by-Step)

Let's say Alice and Bob want to share a secret key.

Publicly agreed values (known to both parties):

- A large **prime number** `p`
- A **primitive root modulo p** (called the generator) `g`

These can be sent in the open.

🔐 Step 1: Key Generation

Each party chooses a **private key**:

- Alice picks a secret number `a`
- Bob picks a secret number `b`

Then they compute their **public keys**:

- Alice computes: `A = g^a mod p`
- Bob computes: `B = g^b mod p`

They exchange `A` and `B` over the insecure channel.

🔁 Step 2: Compute Shared Secret

Now each side computes the shared key:

- Alice computes: `s = B^a mod p`
- Bob computes: `s = A^b mod p`

Since:

```
B^a ≡ (g^b)^a ≡ g^(ab) mod p
A^b ≡ (g^a)^b ≡ g^(ab) mod p
```

✅ Both sides compute the same value `s`, the **shared secret key**, without ever sending `a` or `b`.

🔒 Example (Small Numbers for Simplicity)

- Public values: `p = 23`, `g = 5`

1. Alice picks private key `a = 6`
   → Computes public key `A = 5^6 mod 23 = 15625 mod 23 = 8`
2. Bob picks private key `b = 15`
   → Computes public key `B = 5^15 mod 23 = 759375 mod 23 = 2`
3. They exchange A and B
4. Alice computes: `s = B^a mod 23 = 2^6 mod 23 = 64 mod 23 = 18`
   Bob computes: `s = A^b mod 23 = 8^15 mod 23 = 18`

✅ Shared secret = **18**

The **RSA algorithm** is a **public-key cryptographic algorithm** used for secure data transmission. It is named after its inventors: **Rivest, Shamir, and Adleman**.

RSA is widely used for **encryption**, **digital signatures**, and **secure key exchange**.

🔐 Key Concepts

- **Asymmetric encryption**: uses a **public key** for encryption and a **private key** for decryption.
- Based on the difficulty of factoring large composite numbers.

⚙️ RSA Algorithm Steps

▶ Step 1: Key Generation

1. Choose two **distinct large prime numbers**, `p` and `q`
2. Compute:
   `n = p * q`
   `φ(n) = (p - 1) * (q - 1)` (Euler's totient)
3. Choose an integer `e` such that:
   `1 < e < φ(n)` and `gcd(e, φ(n)) = 1`
   (e is the **public exponent**)
4. Compute `d` as the **modular inverse** of `e` modulo `φ(n)`
   → `d ≡ e⁻¹ mod φ(n)`

- **Public Key** = `(e, n)`
- **Private Key** = `(d, n)`

▶ Step 2: Encryption

To encrypt a message `M`:

- Convert plaintext to an integer `m` such that `0 < m < n`
- Ciphertext `c = m^e mod n`

▶ Step 3: Decryption

To decrypt ciphertext `c`:

- Plaintext `m = c^d mod n`

📌 Example (using small numbers for clarity)

🔢 Key Generation:

- `p = 61`, `q = 53`
- `n = p * q = 61 * 53 = 3233`
- `φ(n) = (p - 1)(q - 1) = 60 * 52 = 3120`

Choose `e = 17` (a common public exponent)

Find `d` such that `d * e ≡ 1 mod 3120`
→ `d = 2753`

- Public Key: `(e=17, n=3233)`
- Private Key: `(d=2753, n=3233)`

🔐 Encrypt message `M = 65`:

`c = 65^17 mod 3233 = 2790`

🔓 Decrypt ciphertext `c = 2790`:

`m = 2790^2753 mod 3233 = 65`
✅ Original message retrieved

The **Caesar Cipher** is one of the **simplest and oldest** encryption techniques. It's a type of **substitution cipher** in which each letter in the plaintext is **shifted a fixed number of places** down the alphabet.

🔐 Caesar Cipher Basics

- Invented by **Julius Caesar**
- It uses a single key: the **shift amount** (usually from 1 to 25)
- For example, with a shift of **+3**:
  - A → D
  - B → E
  - C → F
  - ...
  - X → A
  - Y → B
  - Z → C

🔁 Encryption Formula:

pgsql
CopyEdit
```
Encrypted letter = (plaintext_letter + shift) mod 26
```
Assuming:

- `A=0`, `B=1`, ..., `Z=25`

🔓 Decryption Formula:

bash
CopyEdit
```
Decrypted letter = (cipher_letter - shift + 26) mod 26
```

🔤 Example

▶ Plaintext: `HELLO`

▶ Shift: `3`

Letter	Position	3	Cipher
H	7	10	K
E	4	7	H
L	11	14	O
L	11	14	O
O	14	17	R

✅ **Ciphertext**: `KHOOR`

The **DES (Data Encryption Standard)** is a symmetric-key block cipher developed in the 1970s. It was once the gold standard for secure data encryption but is now considered insecure due to its short key length.

🔐 DES at a Glance

Feature	Detail
Algorithm type	Symmetric block cipher
Block size	64 bits (8 bytes)

Key size	56 bits (plus 8 parity bits)
Rounds	16 rounds
Encryption method	Feistel structure (split & process)
Status	Obsolete (replaced by AES)

⚙️ How DES Works

1. Initial Permutation (IP)

- The 64-bit plaintext block undergoes an initial permutation.

2. 16 Rounds of Processing

Each round involves:

- Splitting the block into **Left (L)** and **Right (R)** halves.
- Using a **round key** (generated from the main key).
- Applying a **function f(R, K)** that:
  - Expands R from 32 to 48 bits.
  - XORs it with the round key.
  - Applies **S-box substitution** and **permutation**.
- Swapping and recombining L and R after each round.

3. Final Permutation (IP^{-1})

- After 16 rounds, the halves are recombined and a final permutation is applied.

The **AES (Advanced Encryption Standard)** is the current **U.S. federal standard** for encrypting data, and it's one of the most widely used **symmetric encryption algorithms** in the world today.

🔐 AES at a Glance

Feature	Detail
Type	Symmetric-key block cipher
Block size	128 bits (16 bytes)
Key sizes	128, 192, or 256 bits

Rounds	10 (128-bit), 12 (192-bit), 14 (256-bit)
Structure	Substitution–Permutation Network (SPN)
Standardized by	NIST in 2001
Replaces	DES

⚙️ AES Encryption Steps (128-bit key)

AES works on a **4×4 matrix of bytes** called the **state**.

🔁 Each round includes:

1. **SubBytes** – Each byte is replaced using an S-box (substitution)
2. **ShiftRows** – Rows of the matrix are cyclically shifted
3. **MixColumns** – Columns are mixed using matrix multiplication over GF(2^8)
4. **AddRoundKey** – Each byte is XORed with a round key

The **first round** starts with AddRoundKey.
The **last round** skips MixColumns.

🗝️ Key Expansion

- The input key is expanded into multiple **round keys** using a **key schedule**
- Number of rounds depends on key size:
  - 128-bit key → 10 rounds
  - 192-bit key → 12 rounds
  - 256-bit key → 14 rounds

📦 Example (Conceptual)

Let's say you want to encrypt:
Plaintext: `"HelloWorld123456"` (16 bytes)
Key: `"ThatsMyAESKey123"` (16 bytes = 128 bits)

AES will:

1. Convert the plaintext and key into byte matrices

2. Apply 10 rounds of transformations as described above
3. Output the ciphertext (also 16 bytes)

Note: Actual encrypted text will look like unreadable binary or hex — not regular characters.

✅ AES Advantages

- Fast and secure even on limited-resource devices
- Resistant to all known practical attacks (when implemented correctly)
- Supports multiple key lengths