Genetic Mutation Cipher (GMC) - Simplified Explanation

The Genetic Mutation Cipher (GMC) is a custom encryption method that works on both text and files. It mimics genetic mutations by applying a combination of substitution, noise insertion, and transposition. These transformations obfuscate the original message and make it hard to reverse without the correct key.

# Text Encryption Process

Step 1: Substitution

Each character is XORed with the number 5. This means we modify the character's binary value by flipping certain bits to create a different character.

Step 2: Transposition

We reverse the entire substituted string. This adds further confusion by changing the character order.

Step 3: Noise Insertion

We insert a '\*' symbol every few characters. The exact frequency is determined by the key length. This makes it harder for attackers to distinguish original characters from noise.

## Example Code for Text Encryption

plaintext.replace(/[a-z]/gi, c => String.fromCharCode(c.charCodeAt(0) ^ 5));  
  
text.split('').reverse().join('');  
  
...map((c, i) => (i + 1) % ((key.length % 3) + 2) === 0 ? c + '\*' : c).join('');  
  
base64 encoded final string

# Text Decryption Process

Step 1: Base64 Decode

We decode the base64 ciphertext to get the raw encrypted string.

Step 2: Noise Removal

We remove all '\*' characters to restore the clean transposed string.

Step 3: Reverse Transposition

We reverse the string back to its original character order.

Step 4: Reverse Substitution

We apply XOR with 5 again to restore the original characters.

# File Encryption & Decryption

Files are encrypted by XORing each byte with a character from the key. Every few bytes, a noise byte is inserted.

Encryption Function (simplified):

byte = buffer[i] ^ key.charCodeAt(i % key.length);  
if ((i + 1) % interval === 0) add noise byte

Decryption Function (simplified):

if ((i + 1) % (interval + 1) === 0) skip;  
byte = buffer[i] ^ key.charCodeAt(skip % key.length)

# Key Importance in GMC

The key is central to GMC. It determines:

- How characters or bytes are XORed.

- How frequently noise is added (interval = (key.length % 3) + 2).

A longer key results in a more complex XOR pattern and less predictable encryption.