XOR Cipher

VHDL/FPGA

Directed by : Mr. Moumni

Made by :

Anas Sabbar

Ilyass Mebsout

Marwane El-arif

Imad Chagri

Introduction

The XOR cipher is a simple, yet powerful cryptographic method for securing data. In its simplest form, only a secret key is necessary to perform both encryption and decryption using the bitwise exclusive OR operation, often denoted with a circle plus, $\oplus$

To encrypt, we simply XOR a plaintext message M with our secret key K so that $M \oplus K = E$. To decrypt we simply XOR the encrypted message E with the same key, $E \oplus K = M$. Conveniently, this means the same operation (or program, in our case) can be used to both encipher and decipher (or encrypt and decrypt).

If a fixed-length key is shorter than the message and used repeatedly to encrypt pieces of the message, then a frequency analysis could be used to break the encryption of text messages (in the same way you or your parents may have broken cryptogram puzzles in the newspaper, which rely on the fact that certain letters and combinations appear more often than others).

However, if the key is as long as the message, the only way to break the encryption is to try every possible key. This is known as a brute-force attack and is implausible in practice because an N-bit key has 2^N possibilities.

Securely sharing a long key is just as difficult as the original problem of sharing a long message. Moreover, repeated use of even a long key for multiple messages is also subject to frequency-based attacks. One way around these problems is to use a pseudo-random number generator (***RNG***) to generate an unpredictable but repeatable stream of keys. Then, only the (much shorter) initial ***seed*** for the generator needs to be shared (assuming both parties are using the same generator). Each block of the message is then encrypted using subsequent keys from the generator.

Usefulness in cryptography

The primary reason XOR is so useful in cryptography is because it is "perfectly balanced"; for a given plaintext input 0 or 1, the ciphertext result is equally likely to be either 0 or 1 for a truly random key bit.[5]

The table below shows all four possible pairs of plaintext and key bits. It is clear that if nothing is known about the key or plaintext, nothing can be determined from the ciphertext alone.[5]
XOR Cipher Trace Table

Plaintext	**Key**	**Ciphertext**
0	0	0
0	1	1
1	0	1
1	1	0

Other logical operations such and AND or OR do not have such a mapping (for example, AND would produce three 0's and one 1, so knowing that a given ciphertext bit is a 0 implies that there is a 2/3 chance that the original plaintext bit was a 0, as opposed to the ideal 1/2 chance in the case of XOR)

Detailed Explanation

- Encryption Module:

The XOR_Encrypt entity takes plaintext and key as 8-bit input vectors and produces ciphertext as an 8-bit output vector.

Inside the architecture, the plaintext is XORed with the key using the xor operator, and the result is assigned to ciphertext.

- Plaintext: The original data that you want to encrypt.
- Key: A secret key known only to the sender and receiver.
- Ciphertext: The encrypted data.
- Process: Each bit of the plaintext is XORed with the corresponding bit of the key to produce the ciphertext.
- Formula: $Ci=Pi\oplus Ki$Ci =Pi $\oplus$Ki

Where CiCi is the ith bit of the ciphertext, PiPi is the ith bit of the plaintext, and KiKi is the ith bit of the key.

- ## Decryption Module:

The XOR_Decrypt entity takes ciphertext and key as 8-bit input vectors and produces plaintext as an 8-bit output vector.

Inside the architecture, the ciphertext is XORed with the key using the xor operator, and the result is assigned to plaintext.

- Ciphertext: The encrypted data.
- Key: The same secret key used for encryption.
- Plaintext: The original data recovered from the ciphertext.
- Process: Each bit of the ciphertext is XORed with the corresponding bit of the key to retrieve the original plaintext.
- Formula: $Pi=Ci\oplus Ki$Pi =Ci $\oplus$Ki

Where PiPi is the ith bit of the plaintext, CiCi is the ith bit of the ciphertext, and KiKi is the ith bit of the key.

(CODE IN GITHUB)