WIKIPEDIA
The Free Encyclopedia

XOR cipher

In cryptography, the **simple XOR cipher** is a type of *additive cipher*,[1] an encryption algorithm that operates according to the principles:

$A \oplus 0 = A$,
$A \oplus A = 0$,
$A \oplus B = B \oplus A$,
$(A \oplus B) \oplus C = A \oplus (B \oplus C)$,
$(B \oplus A) \oplus A = B \oplus 0 = B$,

For example where $\oplus$ denotes the exclusive disjunction (XOR) operation.[2] This operation is sometimes called modulus 2 addition (or subtraction, which is identical).[3] With this logic, a string of text can be encrypted by applying the bitwise XOR operator to every character using a given key. To decrypt the output, merely reapplying the XOR function with the key will remove the cipher.

Example

The string "Wiki" (`01010111 01101001 01101011 01101001` in 8-bit ASCII) can be encrypted with the repeating key `11110011` as follows:

```
  01010111 01101001 01101011 01101001
⊕ 11110011 11110011 11110011 11110011
= 10100100 10011010 10011000 10011010
```

And conversely, for decryption:

```
  10100100 10011010 10011000 10011010
⊕ 11110011 11110011 11110011 11110011
= 01010111 01101001 01101011 01101001
```

Use and security

The XOR operator is extremely common as a component in more complex ciphers. By itself, using a constant repeating key, a simple XOR cipher can trivially be broken using frequency analysis. If the content of any message can be guessed or otherwise known then the key can be revealed. Its primary merit is that it is simple to implement, and that the XOR operation is computationally inexpensive. A simple repeating XOR (i.e. using the same key for xor operation on the whole data) cipher is therefore sometimes used for hiding information in cases where no particular security is required. The XOR cipher is often used in computer malware to make reverse engineering more difficult.

If the key is random and is at least as long as the message, the XOR cipher is much more secure than when there is key repetition within a message.[4] When the keystream is generated by a pseudo-random number generator, the result is a stream cipher. With a key that is truly random, the result is a one-time pad, which is unbreakable in theory.

The XOR operator in any of these ciphers is vulnerable to a known-plaintext attack, since *plaintext* ⊕ *ciphertext* = *key*. It is also trivial to flip arbitrary bits in the decrypted plaintext by manipulating the ciphertext. This is called malleability.

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)[a]

Example implementation

Example using the Python programming language.[b]

```
from os import urandom


def genkey(length: int) -> bytes:
    """Generate key."""
    return urandom(length)


def xor_strings(s, t) -> bytes:
    """Concate xor two strings together."""
    if isinstance(s, str):
        # Text strings contain single characters
        return "".join(chr(ord(a) ^ b) for a, b in zip(s, t)).encode('utf8')
    else:
        # Bytes objects contain integer values in the range 0-255
        return bytes([a ^ b for a, b in zip(s, t)])


message = 'This is a secret message'
print('Message:', message)

key = genkey(len(message))
print('Key:', key)

cipherText = xor_strings(message.encode('utf8'), key)
print('cipherText:', cipherText)
print('decrypted:', xor_strings(cipherText, key).decode('utf8'))

# Verify
if xor_strings(cipherText, key).decode('utf8') == message:
```

```
    print('Unit test passed')
else:
    print('Unit test failed')
```

See also

- Block cipher
- Vernam cipher
- Vigenère cipher

References

Notes

a. There are 3 ways of getting a (ciphertext) output bit of 0 from an AND operation:
Plaintext=0, key=0;
Plaintext=0, key=1;
Plaintext=1, key=0.
Therefore, if we know that the ciphertext bit is a 0, there is a 2/3 chance that the plaintext bit was also a 0 for a truly random key. For XOR, there are exactly 2 ways, so the chance is 1/2 (i.e. equally likely, so we cannot learn anything from this information)

b. This was inspired by Richter 2012

Citations

1. Tutte 1998, p. 3
2. Lewin 2012, pp. 14–19.
3. Churchhouse 2002, p. 11
4. Churchhouse 2002, p. 68
5. Paar & Pelzl 2009, pp. 32–34.

Sources

- Budiman, MA; Tarigan, JT; Winata, AS (2020). "Arduino UNO and Android Based Digital Lock Using Combination of Vigenere Cipher and XOR Cipher" (https://doi.org/10.1088%2F1742-6596%2F1566%2F1%2F012074). *Journal of Physics: Conference Series*. IOP Publishing. **1566** (1): 012074. Bibcode:2020JPhCS1566a2074B (https://ui.adsabs.harvard.edu/abs/2020JPhCS1566a2074B). doi:10.1088/1742-6596/1566/1/012074 (https://doi.org/10.1088%2F1742-6596%2F1566%2F1%2F012074). ISSN 1742-6588 (https://www.worldcat.org/issn/1742-6588).
- Churchhouse, Robert (2002), *Codes and Ciphers: Julius Caesar, the Enigma and the Internet* (https://archive.org/details/codesciphersjuli0000chur), Cambridge: Cambridge University Press, ISBN 978-0-521-00890-7
- Garg, Satish Kumar (2017). "Cryptography Using Xor Cipher". *Research Journal of Science and Technology*. A and V Publications. **9** (1): 25. doi:10.5958/2349-2988.2017.00004.3 (https://doi.org/10.5958%2F2349-2988.2017.00004.3). ISSN 0975-4393 (https://www.worldcat.org/issn/0975-4393).

- Gödel, Kurt (December 1931). "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I". *Monatshefte für Mathematik und Physik* (in German). 38–38 (1): 173–198. doi:10.1007/BF01700692 (https://doi.org/10.1007%2FBF01700692). ISSN 0026-9255 (https://www.worldcat.org/issn/0026-9255). S2CID 197663120 (https://api.semanticscholar.org/CorpusID:197663120).
- Lewin, Michael (June 2012). "All About XOR" (https://accu.org/journals/overload/20/109/lewin_1915/). *Overload*. **2** ((109): 14–19. Retrieved 29 August 2021.
- Paar, Christof; Pelzl, Jan (2009). *Understanding cryptography : a textbook for students and practitioners* (http://site.ebrary.com/id/10355821). Springer. ISBN 978-3-642-04101-3. OCLC 567365751 (https://www.worldcat.org/oclc/567365751).
- Richter, Wolfgang (August 3, 2012), "Unbreakable Cryptography in 5 Minutes" (http://xrds.acm.org/blog/2012/08/unbreakable-cryptography-in-5-minutes/), *Crossroads: The ACM Magazine for Students*, Association for Computing Machinery
- Tutte, W. T. (19 June 1998), *Fish and I* (https://uwaterloo.ca/combinatorics-and-optimization/sites/ca.combinatorics-and-optimization/files/uploads/files/corr98-39.pdf) (PDF), retrieved 11 January 2020 Transcript of a lecture given by Prof. Tutte at the University of Waterloo

---