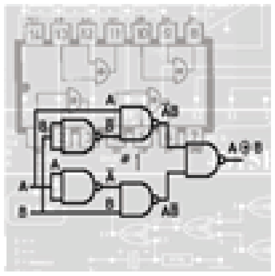
Crossroads
The ACM Student Magazine



Obfuscation of The Standard XOR Encryption Algorithm

by ***Zachary A. Kissel***

Introduction

XOR Encryption is a popular encryption algorithm that is used in many browsers and it is blatantly simpleand fairly secure. The XOR Encryption algorithm is an example of a **Symmetric Encryption** algorithm. This means that the same key is used for both encryption and decryption [7]. In the case of XOR Encryption, this is true because XOR is a **two-way function** which means that the function can easily be undone [6]. In the following paper the standard XOR Encryption algorithm will be introduced along with a modification. The modification comes in the form of creating random permutations of the key.

XOR Encryption

The classical XOR encryption algorithm is derived from Boolean Algebra. The **XOR** function, here on expressed as `XOR(a,b)` where *a* and *b* are binary valued variables, is defined by the followingtruth table (**Table 1**):

a	b	XOR(a,b)

0	0	0
0	1	1
1	0	1
1	1	0

Table 1: The truth table for the XOR function.

Another way to state the XOR function is to say that the function returns true when the values of the two arguments are different. How does one apply this function to the art of encryption? In the most basic sense one must generate a **key**. A key is a password of sorts that the algorithm hinges on. For our purposes let *k* be some key value represented in binary, for now let usjust use a byte (eight bits). Let *m* be a binary representation of the message one byte in length. To obtain the **cipher text** ,which is also known as the encrypted text, one simply applies the XOR function to generate the cipher text `c(c = XOR(m,k))` [**3**].

We know that not every message which we wish to encrypt is one byte long. In fact, very rarely do we talk of bytes when we speak of encryption, more oftenwe speak of bits. The above instance of the XOR algorithm is known as the 8-bit XOR Encryption algorithm. We can generalize the algorithm to be of the*n*-bit form by creating an *n*-bit key.

In practice the message *m* may be broken into chunks of the same length, in bits, as the key. For our purposes this is sensible considering the definition of the XOR function. The cipher text is then generated in a similar way (`c = XOR(p,k)`, *p* is a chunk of the message with the same length in bits as *k*).In general, the algorithm is as follows:

```
while not end_of_message p = get_chunk(m) c = c + XOR(p,k) // where + is
the concatenation operator.end while
```

In the event that the message length, in bits, is not divisible by *n*, padding is added to the message. This algorithm is good, but how does one get the key between peers in a peer-to-peer communication model?

XOR Encryption and Peer-to-Peer Communication

XOR Encryption does not do much good if you cannot use it to communicate between peers in real time. One common way in which peer-to-peer communication is established is through the use of a **Key Distribution Center** (KDC). A Key Distribution Center is responsible for creating keys in peer-to-peer communication. The keys which a KDC provides are known as **session keys**. A session key is a key that is only valid for the duration of the communication [7].

The use of a KDC works as follows: consider peer-A and peer-B who wish to communicate using encrypted data. Peer-A starts by sending a request to connect to peer-B. The request is buffered and the KDC generates a session key. The KDC then sends the session key to both peer-A and peer-B, the request is then sent. Now both peer-A and peer-B can communicate using encrypted messages [7] (**Figure 1**).

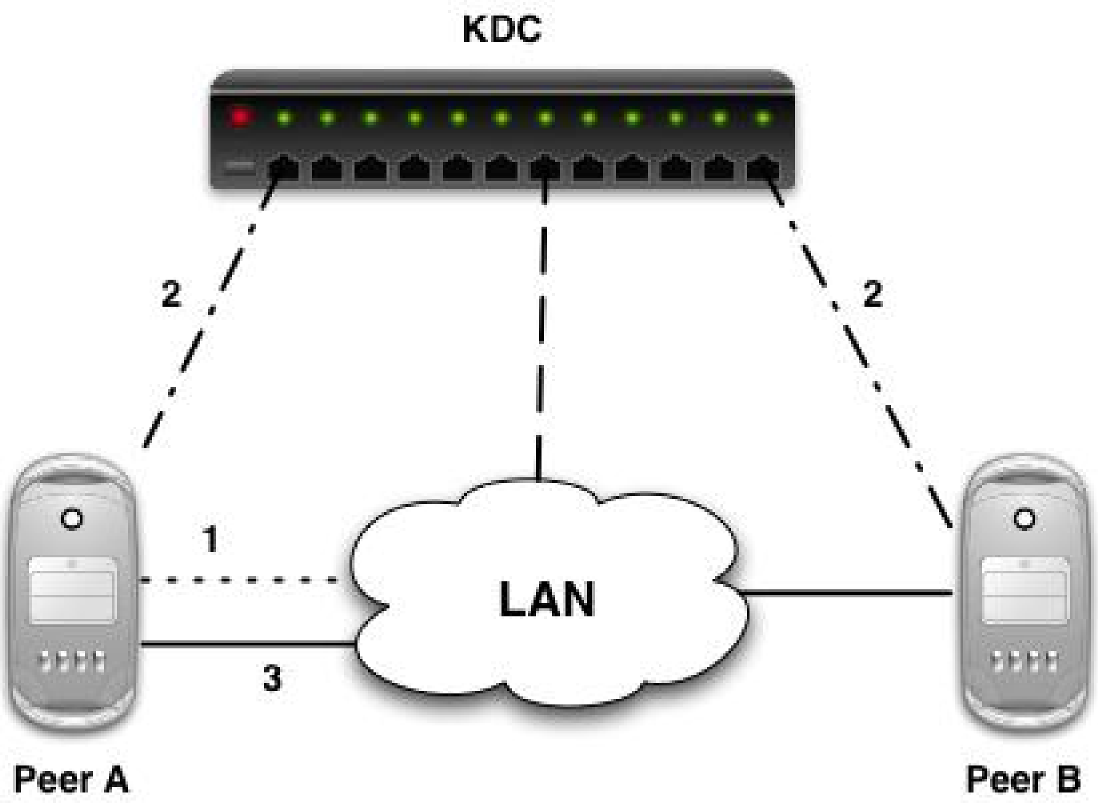


Figure 1: The key generation process: (1) Request key from KDC. (2) KDC distributes key to peers. (3) Peers communicate.

The use of a KDC is the most secure method of key generation and distribution, because the key is generated from a third party location. This fact makes the communication resistant to cracking. However, because of the nature of XOR Encryption, an attack on the KDC would destroy the integrity of the cipher.

Modification to the XOR Encryption Algorithm

In order to strengthen the XOR Encryption algorithm, principles from the **Data Encryption Standard** (DES) are borrowed. The Data Encryption Standard isa symmetric cipher considered to be a strong cipher not easily broken. Like most ciphers DES has been broken; yet, is still consideredsecure enough for most applications [4].

The concept that is being borrowed from DES is the use of rotating bits in the key, also known as a cyclic shift. Cyclic shifts introduce **transposition**--the replacing of one character in a message for another. To further elaborate, bit rotation has two forms: right bit rotations and left bit rotations. A single bit rotation can be performed simply. For the right bit rotations, take the rightmost bit and put it in front of the leftmost bit. For left bit rotations, take the leftmost bit and put it in front of the rightmost bit (**Figure 2**). It should be noted that in order to rotate more than one bit the process described above is applied the number of times that one wishes to rotate the string of *n* bits.

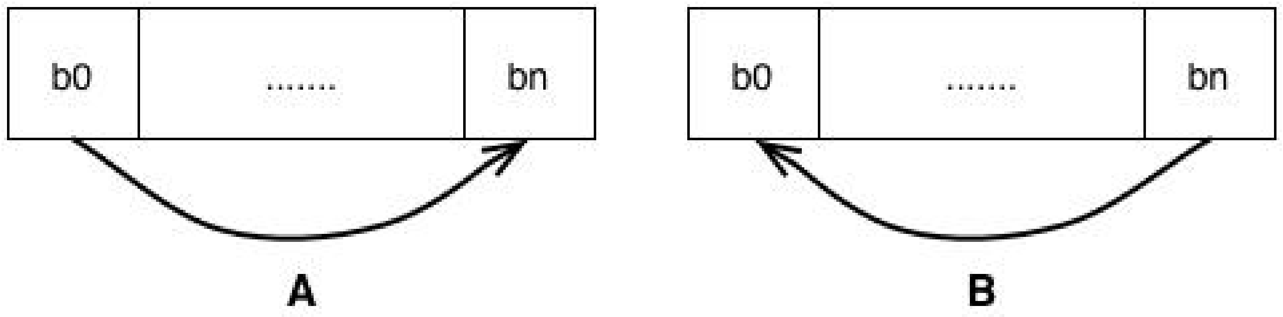


Figure 2: Left (A) and right (B) bit rotations, the place the bit indicated by the tail of the arrow in front of the bit pointed to by the arrow.

For our purposes let the rotate function be defined as `rot(v,d,b)` where *v* is the binary variable, *d* is the direction of the shift, and *b* is the number bits to shift such that $0 < b < n$. The modified algorithm can now be fully described. Let the length of the key, *k*, in bits be 128; this implies, for our simplified purposes, that the message chunks will also be 128 bits in length. Assuming we have a valid session key from the KDC, the algorithm proceeds as follows:

1. Generate the rotation direction *d.*
2. Generate, from random, the number of bits to rotate, *b*, such that $0 < b < n.$
3. Rotate the key *b* times in the direction of *d* (`rot(k,d,b)`).
4. Preform the encryption (`c = XOR(k,m)`), where *m* is a 128-bit chunk of the message.

5. Send the encrypted message *c* to the peer. Also, in the packet send the rotation direction and number of bits to rotate.
6. Repeat steps 2-5 for every 128-bit chunk of the message.

In the event that the message is not divisible by 128, padding is added to the end of the message. The padding character should be something that is not used often in the data of the packet and must be agreed upon by both the sender and receiver. A good choice for a padding character would be the null zero. The given improvements to the standard XOR Encryption algorithm should complicate things if an attacker were able to intercept the key from the KDC. Probabilistically, the key will never be the same for at least two contiguous packets without deciphering each packet by hand, recalculating the new key each time the attacker would not be able to penetrate the cipher. The algorithm as presented is akin to Shannon's one time pad algorithm except Shannon's one time pad only uses a key only once [1].In the described algorithm a key is probabilistically never used twice consecutively, but a keywill be used again eventually. Executing this kind of process offers a level of obfuscation.

How would one create the packet for the new, **Random Rotating XOR** (RRX), encryption algorithm? The data segment of the packet should be 136-bits in length. The first bit will specify the rotation direction (0 = Left, 1 = Right), *d*. The next 7 bits, which in implementation should be longer, will be representative of the number of bits to rotate, *b*. The final 128 bits will hold the encrypted message (**Figure 3**).

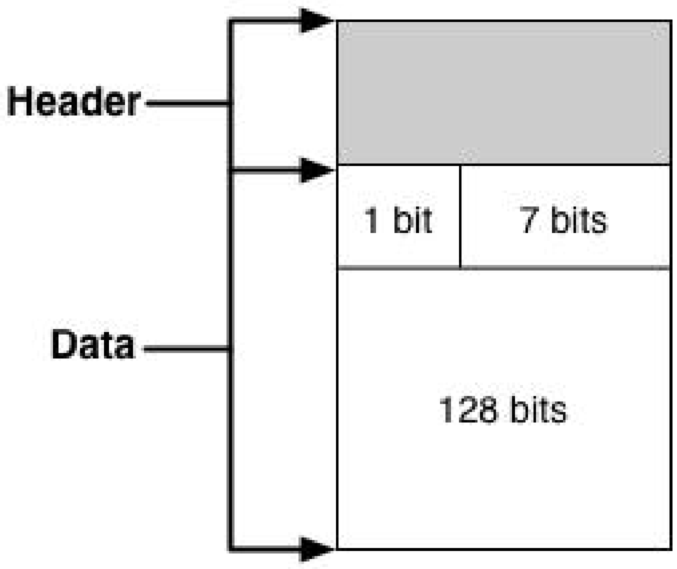


Figure 3: A sample RRX packet structure.

How Good is RRX?

The strength of any encryption algorithm cannot always be accurately analyzed in the laboratory. That being said there are two obvious flaws with the RRX algorithm. The first major flaw is if the key were intercepted from the KDC by an attacker. The attacker could then decrypt the messages for the rest of that session. This, however, is complicated by the fact that the key is constantly being operated upon and therefore dynamic. The fact that the key is dynamic does not add to the strength of the cipher in a natural way, because it does not change the **entropy**. The cipher does, however, offer a layer of obfuscation which presents a hurdle for an attacker. Entropy is defined as a measure of randomness in the cipher. In order for an attacker to decrypt a communication, the attacker needs to intercept all of the messages between the peers and decrypt each packet so that the key is not operated on wrong. The second obvious flaw is in the fact that RRX is a symmetric cipher. This means that the cipher hinges mainly on the protection of the key. In the case of RRX this is slightly relaxed, but still necessary.

Placing the weaknesses aside, RRX does offer some protection that is not available in most XOR based encryption methods. RRX offers the protection of a dynamic key; this dynamic key aides in preventing an attacker, who cannot intercept messages from a well protected KDC, from applying **frequency analysis** as easily, across the collected sub-messages. Frequency Analysis is the process of determining the percentage of the occurrence of a certain pattern in a message. These percentages, or frequencies, are then compared against a known list of frequencies and the attacker can guess at what the message says without knowing the key [2]. Since, a packet of data is so small, there does not exist a sufficient sample size to accurately use frequency analysis. However, the algorithm can be cracked if every packet was saved and a user was able to XOR appropriate packets together to obtain the proper key for a given pair of packets. A rectification to this problem would be to request a new key for the session from the KDC after a given amount of time or a statistical event becomes highly likely.

Knowing the algorithm for RRX does not allow an attacker to easily decipher the communications because of the random nature of the key operations. If the results of the key operations were predictable the attacker would only need to know the given datum's placement in the sequence, provided the key had also been intercepted

A final strength that RRX has, as much as the other XOR based encryption ciphers, is that RRX can be implemented in both hardware and software effectively. This allows the actual hardware that supports RRX to be implemented directly on the Network

Interface Card (NIC). As far as RRX implemented as a software solution, the program could be easily written as a tiny segment of well tuned assembly code (to improve performance).

Conclusion

RRX is a viable solution to encryption on small LANs, such as a college or small university campuses. An alternative XOR based encryption algorithm, viable for large networks, is Blowfish [5]. Blowfish is one of the better known XOR based encryption standards that are commonly investigated. A particularly interesting avenue would beto compare the effectiveness of Blowfish with that of RRX on a small to medium sized LAN.

The algorithm presented in this paper is a small, tidy, and quick algorithm that I believe will keep a small to mid sized LAN secure for a long time. Above that, it is based on the beauty and flaws of a symmetric encryption algorithm--so, pains must be taken to ensure secure implementation.

References

1

Collins, G. P. (2002). *Claude E. Shannon: Founder of Information Theory*. Scientific American. <**http://cispom.boisestate.edu/murli/links/ShannonSciAm.htm**>.

2

Gaines, H. F. (1956). *Cryptanalysis: A Study of Ciphers and Their Solution*. Dover Publications, New York, NY, pp. 74-75.

3

Irvine, K. R. (2003). *Assembly Language for Intel-Based Computers*. Prentice Hall,Upper Saddle River, NJ, pp. 195-198.

4

Loudon, K. (1999). *Mastering Algorithms with C*. O'Reilly Press, Sebastopol, CA, pp. 425-432.

5

Schneier B. (1994). *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*.Springer-Verlag, UK, pp. 191-204. <**http://www.schneier.com/paper-blowfish-fse.html**>.

6

Singh, S. (2000). *The Code Book: The Science of Secrecy from Ancient Egypt to*

Quantum Cryptography. Anchor Books, NY, pp. 260-261

7 Stallings, W. (2004). *Data and Computer Communications*. Pearson Prentice Hall, Upper Saddle River, NJ, pp. 706-740.

Biography

Zach Kissel (**kisselz@merrimack.edu**) is a Computer Science and Mathematics major at Merrimack College. His main Computer Science interests include artificial intelligence (specifically in the field of Bio-Computation), large scale computing, and data communications. In his spare time he enjoys mountain biking and running.