Breaking a Repeating Key XOR cipher

February 20, 2022

Abstract

The XOR cipher is the simpliest symmetric cipher, simply take a known key of some length that is shared between the two parties. When encrypting if the plain text to be encrypted is longer than the key, then simply repeat the key to match the length. The XOR operation and how we break serve as corner stones for future encyption algorithms and attacks. Thus here we go over the details of this cipher and the attack to break it.

1 Repeating Key XOR

Here we go over the details of a repreating key XOR. We define the plain text as a sequence of integers ranging from 0 to 255 i.e. $p = (p_0, ..., p_{n-1})$ such that $0 \leq p_i < 256$ i.e the plain text is brokein into bytes. We will make the assumption that the plain text is ASCII encoded english. Now the XOR operation operates on each bit of each byte such that the output is a logical 1 iff the input are not the same. So if both inputs are 0 or both inputs are 1 the output is 0, else the output is 1. We use the following notation to represent the XOR operation, $a \otimes b$. The inputs a and b can be integers of any size. The output is then defined as operating on each bit of the binary represemtation of the inputs.

Now we define a key to be again just be a sequence of bytes, $k = (k_0, ..., k_{m-1})$ such that $0 \leq k_i < 256$, shared between the two parties exchanging messages. Finally the cipher text is defined as the sequence of bytes $c = (c_0, ..., c_{n-1})$ where $0 \leq c_i < 256$ such that $c_i = p_i \otimes k_i$. Usually the length of the key is less than that of the text, so the key is just repeated and truncated to match the length of the input. The calculation of the cipher text is called encyption.

Given a cipher text, calculating the plaintext that produced the given cipher text is called decryption. To decrypt, we can simply use the fact that the XOR is its own inverse and 0 is the identity of the XOR operation. In other words, $a \otimes a = 0$ and $a \otimes 0 = a$. Using this and the defintion of the cipher text, $c_i = p_i \otimes k_i$, tells us that $p_i = c_i \otimes k_i$. Note that the XOR operation is commutative i.e. $a \otimes b = b \otimes a$.

As a final note, we would like to discuss why the XOR operation is used in encyption. First off, unlike other binary operations the XOR operation is

invertible i.e. if we know the key and one of the plain text or cipher text we can uniquely recover the other. Also it has the property of being "perfectly" balanced. Suppose we know only the cipher text and no nothing about the key. Now a given cipher text value of 1 or 0 tells us nothing about the input i.e. there is a 50% chance the input is 1 and 50% that it is a 0. Also the XOR operation is fast to compute. These are the main reasons that the XOR operation is a core operation in almost all encyption algorithms.

As an actual final note, if you have a truely secret key whose length is greater than that of the input and you never resuse that key, then the XOR cipher simply becomes a one time pad which is unbreakable in theory. Since we assume the input is much longer that of the key, the key will get reused repreatly. This combined with the fact that we have knowledge as to what the plain text is what allows this cipher to be broken fairly easily as we will see.

2 Finding the Key Size

Now the fun part, if Alice and Bob are communicating using repeating key XOR and we snoop their conversation can we, only knowing the cipher text, find out what the plain text is? The first step in doing this is determining the key length. First some notation, let A, B be two english letters encoded in ASCII. Now for this analysis we assume that A, B are random variables that follow the english letter character frequency. This distribution and all subsequent calculations are found in expected_hamming.py. Now let X, Y be two random bytes with a uniform distribution. Finally, we define the function $w(b)$ to be the number of 1's in the binary notation of the byte b. We call this the weight function. We define this function as the following where b_i is the binary representation of b.

$$w(b) = \sum_i b_i$$

Now consider the expected value of $w(X)$, that is the number of 1's in a random byte. This value is clearly 4, i.e. $\mathbb{E}[w(X)] = 4$. We claim that $\mathbb{E}[w(X \otimes C)]$ where C is a byte with any distribution is also 4. To prove this, we look at a single bit of each byte. Let X_i and C_i the ith bit with in these bytes. We know that ,

$$\mathbb{E}[w(X \otimes C)] = \mathbb{E}[\sum_i X_i \otimes C_i] = \sum_i \mathbb{E}[X_i \otimes C_i]$$

since XOR operates on a single bit independantly of the other bits and since both the sum and expected value are linear. Now C_i simply has some probability of being a 1 and some probability of being a 0, let these be p_0 and p_1. For X these are simply .5 since its uniformly random. So $\mathbb{E}[X_i \otimes C_i] = .5p_0 + .5p_1 = .5$. So we know $\mathbb{E}[w(X \otimes C)] = 4$. So we can make the claim that the expected number of 1s in a byte whose value is the XOR of a random byte with a byte from another distribution is 4.

Now we need one more fact before we can present a way of finding the key size. So consider $\mathbb{E}[w(A \otimes B)]$ again where A, B are two english letters encoded in ASCII. We compute this value in associated python script and find its value to be 2.36. So how does this help us? Let us first make the assumption that the key is composed of some number of uniformly random bytes. Now let $\widetilde{A} = A \otimes K$ where K is one of the random bytes in the key. Similarly $\widetilde{B} = B \otimes K$, so A, B are the plain text and $\widetilde{A}, \widetilde{B}$ are the cipher text. Suppose A, B are encoded using the same byte from the key. Then we have the follwoing:

$$\mathbb{E}[w(\widetilde{A} \otimes \widetilde{B})] = \mathbb{E}[w(A \otimes K \otimes B \otimes K)] = \mathbb{E}[w(X \otimes Y)] = 2.36$$

Now in contrats, suppose they were encoded using different random bytes from the key. Say these key bytes are K, K'. Then we can apply the fact that XOR-ing and taking the weight of uniformly random byte with another byte produces an expected value of 4. To see this let $C = A \otimes B$ and $X = K \otimes K'$. We know X is uniformly random and thus we have the following.

$$\mathbb{E}[w(\widetilde{A} \otimes \widetilde{B})] = \mathbb{E}[w(A \otimes K \otimes B \otimes K')] = \mathbb{E}[w(X \otimes C)] = 4$$

This gives us a fairly straight forward statistical test to determine the key size. We generate a list of reasonable key sizes. For each key size, we compute $\mathbb{E}[w(\widetilde{A} \otimes \widetilde{B})]$ over the entire cipher text where $\widetilde{B}$ is key size bytes away from $\widetilde{A}$. If the guessed key size is correct, then these two cipher text bytes were encoded with the same key byte and should yield an expected weight of 2.36. If not, then we should see an expected weight of 4. Thus we now have a way of computing the correct key size, this procedure is coded in python in the file repeat_key_xor.py

3 Breaking the Cipher

At this point we know the key length. Now to break the cipher is really straight forward. Take each byte in the cipher text that we know to be encoded with the same key byte put them in the same array or bin. For each bin, we brute force all possible key bytes by XORing that possible key byte with the cipher text characters in the bin. This produces a bin of possible plain text. The correct key byte should produce a bin whose possible plain text bin that roughly follows the english character letter distribution. Do this for all key bytes and we have broken the cipher.