BITS F463: Cryptography
Programming Assignment 1

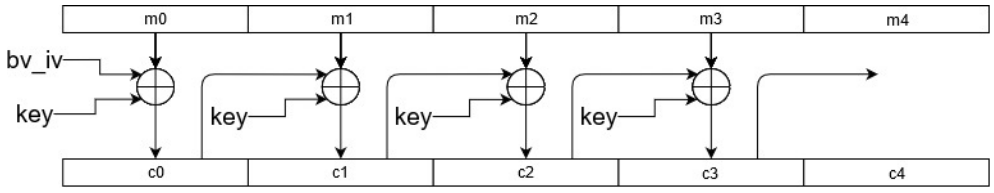
October 25, 2020

1 Brief overview of the encryption algorithm

1. This is a block cipher of block size 64 bits, or 8 bytes.
2. Every block of the plaintext is XORed with two different blocks - one that comes from the key, and one that comes from the previous block of ciphertext.
3. For the initial block of plaintext only, a predefined block is used in the place of the previous block for the XOR operation. This predefined block is generated from a pass phrase, where each successive 8-byte slice of the pass phrase is XORed together.

2 Detailed view of the encryption algorithm

2.1 Visual explanation



$$
\begin{aligned}
c_0 &= m_0 \oplus bv\_iv \oplus key \\
c_1 &= m_1 \oplus c_0 \oplus key \\
c_2 &= m_2 \oplus c_1 \oplus key \\
&\vdots \\
c_n &= m_n \oplus c_{n-1} \oplus key
\end{aligned}
$$

Each operand in the above equations is 8 bytes.

2.2 Generation of Key

An initial key is entered.
If it is greater than 8 bytes, it is reduced to 8 bytes by successively applying the XOR operation to 8-byte slices of the key string.

key = [ ]
FOR every number from 0 to $\lfloor$*number of bytes in the key string* / 8$\rfloor$
    keyblock $\leftarrow$ 8 *byte slice of* key string *corresponding to iteration number*
    key $\leftarrow$ key $\oplus$ keyblock

2.3 Generation of the initial BitVector: bv_iv

A custom passphrase is used.
If it is greater than 8 bytes, it is reduced to 8 bytes by successively applying the XOR operation to 8-byte slices of the passphrase.

bv_iv = [ ]
FOR every number from 0 to $\lfloor$*number of bytes in the passphrase* / 8$\rfloor$
    textstr $\leftarrow$ 8 *byte slice of* passphrase *corresponding to iteration number*
    bv_iv $\leftarrow$ bv_iv $\oplus$ textstr

3 Programming details

The code is written in Python, using the BitVector package which is a library that expands operations on bit representations of data.

4 Cracking the cipher

XORing each block of the ciphertext successively, pairwise, turns the problem into the MTP. If

$$c_n = m_n \oplus c_{n-1} \oplus key$$

then

$$c_n \oplus c_{n-1} = m_n \oplus key = C_n$$

and doing this for every block gives us a set of plaintext blocks all XORed with the same 8-byte key. It is clear that this is the same form as an MTP.

Now, we need to try to guess the key based on the ciphertext blocks. First, there is an important property of ASCII values to notice:

$$A - Z \oplus space = a - z$$
$$a - z \oplus space = A - Z$$

where '*space*' refers to the ASCII value of the space character.

If we can identify which byte of a ciphertext block refers to a space, then performing XOR of that byte (in ciphertext) with the ASCII value of space gives the corresponding byte of the key. How do we identify that? Using the principle above.

Now, if we perform the XOR of the first block, say c_0, with c_1, c_2...c_n, that would be the same as taking one block of plaintext, say m_0, and performing

XOR with $m_1, m_2 ... m_n$. Let's say we notice that one byte position seems to consistently be an alphabet (either A-Z or a-z). What does that mean? If we were looking at just two operands, that means either one of them had a space at that position.

But if we notice an alphabet at byte position i for all the XORs - $m_0 \oplus m_1$, $m_0 \oplus m_2$, $m_0 \oplus m_3$ ... $m_0 \oplus m_n$, that means that byte position i of m_0 has a space. Which means $c_0 \oplus space$ gives us the corresponding byte of the key at that position (i). And so each byte of the key is guessed based on the frequency of occurrence of an alphabet at that byte position.

By Kerckhoff's principle, we assume that the initial passphrase is public knowledge. So once we have the key, we can run it through the regular decryption algorithm to end up with our recovered plaintext.

Empirically, it seems that 70% of the total number of ciphertexts is a good estimate for the number of times a byte position must yield an alphabet (while doing pairwise XORs) for it to be considered a space in the original plaintext.