Cryptography Exercise Sheet 1

Andrew Parker Talbot 1434667
Luke Edward Alden 1423268

June 2, 2018

Please note, on the sheet, it says for only one person to hand it in, however on canvas it says for both to hand in. WE HAVE BOTH HANDED IN (to be safe).

1 Question 1

We consider a Feistel Block Cipher with two rounds $i = 0$, and $i = 1$. The two round Feistel keys are defined by $K_i = [K + 75 \times i] \mod 256$, for $i = 0, 1$. Our Feistel Function is defined by $f(K_i, R_i) = [127 \times (K_i \times R_i)] \mod 256$, again for $i = 0, 1$. We encrypt the message block $M = \{V, S\}$, with key $K = Y$, and use ASCII encoding of capital letters, where A is 65.

In ASCII, we see that

$$\begin{aligned} V = 86 &\implies V = 01010110 \text{ in binary.} \\ S = 83 &\implies S = 01010011 \text{ in binary.} \\ Y = 89 &\implies Y = 01011001 \text{ in binary.} \end{aligned}$$

Thus, $M = (86, 83)$. First off we must calculate the key for each round:

$$\begin{aligned} K_0 &= (89 + 75 \times 0) \mod 256 \\ &\equiv 89 \mod 256 \\ \text{and } K_1 &= (89 + 75 \times 1) \mod 256 \\ &\equiv 164 \mod 256 \end{aligned}$$

So, our Key for the first round is $K_0 = 89$ and for the second round is $K_1 = 164$. We now compute the first round of the Feistel block cipher, with $L_0 = 86$ and $R_0 = 83$. We use the following formula $L_1 = R_0 = 83$ and $R_1 = L_0 \oplus f(K_0, R_0)$.

$$\begin{aligned} R_1 &= L_0 \oplus f(89, 83) \\ &= L_0 \oplus (127 \times (89 + 83)) \mod 256 \\ &= L_0 \oplus 21844 \mod 256 \\ &= L_0 \oplus 84 \mod 256 \\ &= 01010110 \oplus 01010100 \text{ (in binary.)} \\ &= 00000010 \\ &= 2 \text{ (in decimal.)} \end{aligned}$$

So, after the first round, we have $M_0 = (83, 2)$. We now perform the second round. Where we calculate $L_2 = R_1 = 2$ and $R_2 = L_1 \oplus f(K_1, R_1)$.

$$\begin{aligned}
R_2 &= L_1 \oplus f(164, 2) \\
&= L_1 \oplus (127 \times (164 + 2)) \mod 256 \\
&= L_1 \oplus 21082 \mod 256 \\
&= L_1 \oplus 90 \mod 256 \\
&= 01010011 \oplus 01011010 \text{ in binary. (base 2)} \\
&= 00001001 \\
&= 9 \text{ (in decimal.)}
\end{aligned}$$

So, we now have that after our 2nd and final round we yield $M_1 = (2, 9)$, however, in the final round of a Fesitel Cipher, we do not swap the L_i and R_i values, so in total we only swap once, but our formula automatially swaps, and thus our **actual final answer is** $C = (9, 2)$.

2 Question 2

2.1 2a (Insecure)

We are checking if the cipher $E_1(k, m) = k \oplus m$ is a secure pseudorandom permutation, where the key and message are 128 bits, note that the key is constant, and that E_1 is one to one.

We check whether E_1 is a secure pseudorandom permutation by playing the game defined on slide 39 of the cryptography notes.

Our game is between an attacker and a challenger. Our challenger chooses a random bit $b \in \{0, 1\}$. If the challenger chooses the $b = 0$, then the challenger chooses a random key $k \in K$ where K is the set of keys. If he chooses $b = 1$, then he chooses a random permutation f on $X = \{1, 0\}^n$.

The attacker can choose messages to send for the challenger to respond with an output for each message. Say the attacker chooses two bitstrings that are only one bit different, then the challenger will respond two bitstrings that are again only one bit different, in the same spot. Note the keys remains constant.

Another arguement is that, if an attacker sends a message m_1 to the challenger, the challenger will respond with either a random permutation or $E_1(k, m_1) = k \oplus m = c_1$, say that the attacker then sends c_1 to the challenger, then the attacker will perform $E_1(k, c_1) = k \oplus c_1$. This will return the original message m_1. The attacker might now think that $b' = 0$, however there is still the possibility this happened by chance. The challenger can resend many messages, and then again resend their ciphertexts, if he keeps getting the original messages back, the probability that the attacker chooses $b = 0$ correctly will get higher and higher. Thus, we deduce that it is insecure, as this strategy allows for the attacker to deduce whether it is pseudorandom or not with a probability of greater than 0.5.

We conclude that E_1 is insecure because if $b = 0$, then the attacker will be able to spot patterns in the outputs of similar messages ($b' = 0$). If $b = 1$, then the attacker will guess that it is a random permutation ($b' = 1$). Thus the attacker will correctly guess, and we get $b = b'$.

2.2 2b (Insecure)

We again play a game, that is defined exactly the same as above in 2a. However, we are now dealing with the function $E_2(k, m) = (m - k \mod 2^{128}) \oplus k$.

We note that E_2 is similar to E_1 in that the output is the result of an XOR operation with the key. However, a preceding $m - k \mod 2^{128}$ term results in a linear transformation on the message m. Because of this, the ciphertext received from a message will not necessarily return the original message when itself is encrypted by E_2, which differs from the behaviour of the block cipher E_1. However, due to the nature of this problem, it is still possible to distinguish E_2 from a truly random permutation as patterns and similarities will eventually arise from multiple outputs of various messages sent by the attacker. This is possible due to a similar arguement in the part 2a, in that if we submit two bitstrings of length 128 that are all the same except for one bit, say the j-th bit, so patterns will still arise for certain values of m, and again the attacker can guess with a probability of over 0.5 as to whether E_2 is pseudorandom or not.

So, as in Q2a we run the game, and see that E_2 is in fact insecure, because of the arguements from 2a. i.e the attacker will correctly guess that $b = b'$ with a probability of over 0.5.

3 Question 3

3.1 3a

We compute,

$$\begin{aligned}
&(1234567890 \times 5678901234) \mod 7890123456 \\
&= 7010989113977776260 \mod 7890123456 \\
&\equiv 3628010628 \mod 7890123456 \\
&= 3628010628
\end{aligned}$$

Thus, for this exercise our answer is 3628010628.
This was calculated using python from the following code:

```
answer = (1234567890 * 5678901234) % 7890123456
print(answer)
```

3.2 3b

We calculate $(x^2+1)(x^3+x+1) \mod x^4+x+1$
To do this we first multiply out the brackets.

$$\begin{aligned}
(x^2+1)(x^3+x+1) &= x^5+x^3+x^2+x^3+x+1 \\
&= x^5+2x^3+x^2+x+1
\end{aligned}$$

But our coefficients are calculated module 2. Thus,

$$(x^2+1)(x^3+x+1) = x^5+x^2+x+1.$$

Thus, our initial problem can be simplified to the calculation of:

$$x^5+x^2+x+1 \mod x^4+x+1$$

We perform polynomial long division (we are struggling to show our working neatly in latex), and get that

$$\frac{x^5+x^2+x+1}{x^4+x+1} = x + \frac{1}{x^4+x+1}$$

We multiply by the denominator x^4+x+1 to get that

$$\begin{aligned}
x^5+x^2+x+1 &= x(x^4+x+1)+1 \\
&\equiv 1 \mod (x^4+x+1) \\
&= 1
\end{aligned}$$

Thus, we see that **our final answer is 1**. ie

$$\begin{aligned}
&(x^2+1)(x^3+x+1) \mod x^4+x+1 \\
&\equiv 1 \mod x^4+x+1
\end{aligned}$$

4 Question 4

4.1 4a

The encryption by k of $m_1||m_2||...||m_n$ is $c_0||c_1||c_2||...||c_n$ where c_0 is a randomly chosen "initialisation vector" and $c_i = E(k,m) \oplus c_{i-1}$, for $i > 0$ (this is comparable to CBC except the XOR with the IV occurs after the encryption). Using the indistinguishability under chosen-plaintext-game we can see that this form of encryption is insecure. If the attacker were to supply one of his two plaintexts as a series of blocks with identical messages (m_0 = m_1 ... = m_n) to be encrypted by the challenger, patterns would eventually arise due to the nature of this mode of operation. This is because the initial output (prior to XORing) of each identical message will result in the same initial output to be XORed with c_{i-1}. After the initial XOR by c_0, the output of each block will alternate between two certain sequences such that $c_n = c_{n+2}$ (thus for even values of n, we get identical ciphertext, and likewise for odd values of n, so long as all the plaintext blocks are equal).

This results in the attacker being able to correctly guess the value of b with a greater probability than 0.5, giving him a non-negligible advantage rendering the cipher insecure.

4.2 4b

The encryption by k of $m_1||m_2||...||m_n$ is $N||c_1||c_2||...||c_n$, where N is a randomly chosen nonce of appropriate size, and $c_i = E(k, N \oplus m_i)$. By using the indistinguishability under chosen-plaintext-game between an attacker and challenger, it can be seen that this specific form of encryption is not secure. The encryption used can be viewed as CTR except that there is only a nonce and no counter is used.

Given that the attacker sends his two messages m_0 and m_1, he will be capable of telling whether the ciphertext he receives corresponds to the first or second message. He can confirm this by asking the challenger for the encryption of one or both of his two messages during the computation phase of the game. As, in this encryption, the nonce remains constant for each computation of c_i from m_i, each plaintext will return the same ciphertext and he will be able to clearly see which of his two messages was returned to him as ciphertext. This results in the attacker being able to correctly guess the value of b with a greater probability than 0.5, giving him a non-negligible advantage rendering the cipher insecure.

(However, if a new nonce is generated for each message (i.e. series of blocks) sent, then the proposed solution outlined above will fail. However, if a message is sent with all blocks being equal, then the ciphertext will show a repeating pattern in the outputted blocks. This allows distinguishability between a repeating message and a non repeating message and again renders the cipher insecure.)

5 Question 5

Here, we are told to find the first 20 bytes of the output of the RC4 algorithm, when run with the key $K = [1, 2, 3]$. I have calculated these first 20 bytes using the pseudo code from the slide 103 from the cryptography notes. I used Java and ended up with the following program:

```
public class RC4 {

    public static void main(String[] args) {

        int[] k = {1, 2, 3};

        int[] s = new int[256];

        for (int i = 0; i < 256; i++) {
            s[i] = i;
        }

        int j = 0;
        int holder;

        for (int i = 0; i < 256; i++) {
         j = (j + s[i] + k[i % k.length]) % 256;
            holder = s[i];
            s[i] = s[j];
            s[j] = holder;
        }

        int i = 0;
        j = 0;

        for (int c = 0; c < 20; c++) {
         i = (i+1) % 256;
            j = (j + s[i]) % 256;
            holder = s[i];
            s[i] = s[j];
            s[j] = holder;
            System.out.print(s[(s[i] + s[j]) % 256] + " ");
        }
    }
}
```

This program printed the first 20 bytes of the RC4 stream. The first 20 bytes are listed here in decimal:

$$151, 54, 143, 104, 66, 149, 196, 180, 153, 173, 143, 8, 56, 140, 194, 176, 113, 31, 98, 221$$

6 Question 6

We have both uploaded a picture of ourselves onto our canvas.