Zelong Li
24569650
Discussion 102
TA: Qi Zhong
CS161 Homework 4

1.
We start our meet-in-the-middle attack by assuming A = 0 and constructing a table 1 with 2^{56} entries of all possible K1 and corresponding $DES_{K1}^{-1}(0)$. Then, we calculate the possible 2^{56} entries of plaintext m and corresponding cipher-text $c = DES_{K1}(DES_{K2}^{-1}(DES_{K1}(m)))$. We find the entry in table 1 that has the same value of $DES_{K1}^{-1}(0)$ as one of the plaintext m. Then we record the corresponding K1 value. With this K1 and its corresponding cipher-text c, we find B' by $B' = DES_{K1}^{-1}(c)$. After that, we construct table 2 with 2^{56} entries of all possible K2 and corresponding $B = DES_{K2}^{-1}(0)$ (since we assume A = 0 and B = $DES_{K2}^{-1}(0)$). In this table, we find a value of B that is the same as the value of B' we find before. And the corresponding K2 is the K2 value we want. Thus, with the K1 we find before, we have found the correct value of (K1, K2) and we finish our meet in the middle attack.
(If we find multiple pairs of K1 and K2 that satisfy the meet in the middle attack, we simply use plaintext and cipher-text pairs (m, c) to check and find out the (K1, K2) that would work for the encryption algorithm).

2.

Suppose the cipher-text block that gets corrupted is C_i and the rest of the blocks are not corrupted. When we try to decrypt to get P_i, we have $P_i = D_K(C_i) \oplus C_{i-1}$. This P_i is not correct because the decryption of C_i is corrupted. Then, when we want to decrypt to get P_{i+1}, we have $P_{i+1} = D_K(C_{i+1}) \oplus C_i$. Although we have a correct C_{i+1}, P_{i+1} is still corrupted because C_i is still the error block we have. However, when we decrypt to get P_{i+2}, we have $P_{i+2} = D_K(C_{i+2}) \oplus C_{i+1}$. This time, both C_{i+2} and C_{i+1} are uncorrupted and thus P_{i+2} is uncorrupted. Therefore, if there is a transmission error in a block of cipher-text using CBC mode, the error propagates for two blocks in decryption and then recovers.

3.

Super one time pad encryption is not perfectly secure. If the starting bits in the beginning of the encryption key k we use were the same as the bits in the end (or in the extreme case the key is a Palindromic number), we would have leaked information of the contents of the plaintext because the reversal of the key k would have the same starting bits and ending bits and thus leak part of the plaintext. Also, this encryption method may leak the relationship between 1st and nth bits, 2^{nd} and (n-1)th bits,…, and so on since k xor k^R is symmetric.

Example:

If we have a message m = 11001011, k = 11101111,

Then the cipher-text would be

c = m xor k xor k^R = 11001011 xor 11101111 xor 11110111 = 00100100 xor 11110111 = 11010011

we can easily see that the cipher-text is really close to the plaintext.

In the extreme case where k is a Palindromic number:

c = m xor k xor k^R = m xor k xor k = m

Then, this time super one time pad encryption leaks the entire plaintext.

Therefore, super one time pad encryption is not perfectly secure.

4.

We know (a_L, a_R), (b_L, b_R), $F(a_L, a_R) = (a_R, a_L \text{ xor } f(a_R, K))$, $F(b_L, b_R) = (b_R, b_L \text{ xor } f(b_R, K))$, and $q = a_R \text{ xor } b_R$.

Then, we have:

$(c_L, c_R) = F(F(a_L, a_R)) = F(a_R, a_L \text{ xor } f(a_R, K)) = (a_L \text{ xor } f(a_R, K), a_R \text{ xor } f(a_L \text{ xor } f(a_R, K),K))$

$(d_L, d_R) = F(F(b_L, b_R)) = F(b_R, b_L \text{ xor } f(b_R, K)) = (b_L \text{ xor } f(b_R, K), b_R \text{ xor } f(b_L \text{ xor } f(b_R, K),K))$

if $c_L = d_L$, then:

$a_L \text{ xor } f(a_R, K) = b_L \text{ xor } f(b_R, K)$ and therefore $f(a_L \text{ xor } f(a_R, K),K) = f(b_L \text{ xor } f(b_R, K),K)$

$c_R \text{ xor } d_R = (a_R \text{ xor } f(a_L \text{ xor } f(a_R, K),K)) \text{ xor } (b_R \text{ xor } f(b_L \text{ xor } f(b_R, K),K))$

let $f(a_L \text{ xor } f(a_R, K),K) = p = f(b_L \text{ xor } f(b_R, K),K)$

then $c_R \text{ xor } d_R = (a_R \text{ xor } p) \text{ xor } (b_R \text{ xor } p) = (a_R \text{ xor } b_R) = q$

5.

a. In this program, we first enter the value of p, which is the modulus for P-256 in the reading. Then, we load the values of a and b and the order of the elliptic curve and create the elliptic curve. After that, we create a generator, which is a point P on the elliptic curve. Then, we enter our secret e = 123456 and calculate the encryption of it, Q = eP, using generator P and the elliptic curve. Finally, the program prints the corresponding x and y coordinates of our encryption Q = eP. The effect of publishing your point Q in the next edition of the NIST 800-90A standard is that I would be able to know the state of any system that uses this standard. It took less than one second for sage to run this program.

b. Qx = 7a926a19fbdc7aa3e2e6c1476c3b8f0819e1d7cfdc2904c1adaa2ce73299e7b8
   Qy = fc8929031165790a40adab6ce83e20786011473150e11a742ad46e68daaadf98

c. Qx =
   b89995a230041279c9cf06fa4eeaf7e95b10714dad42601038f1eaa8e63407a99a42204
   d2833b80df1c95bfad53d0fab
   Qy =
   50ea7c117720729baba003e9c14e606e30ab3cc29f5ffd681379031ffe464b110873ddab
   f8dc85037e580d3f5fde70c

d. Qx =
   f272381fd9b736ce6f9eb6810f98103919bafbd7b5538c3cbb785a9cc6dd75693851415
   c5b132c25831aebc22a2f71684c51b15e9f468d73d690dfdc437d997cc8
   Qy =
   9afecd5b35fdead12550fa9e99d1ec49c3ab79bd1a2eb7b25c81ca0de315e363a006de5
   db6c89421cbb8c59810f51756484583c2f758ddc15edc0be92d8f511629