CS641
Modern Cryptology
Indian Institute of Technology, Kanpur

Mid Semester Examination

Group Name: Noobs
Divyansh Bisht (21111027), Manu Shukla (21111040), Rishabh Lakhwani (18817611)

Submission Deadline:
March 3, 2022, 23:55hrs

Question 1

Consider a variant of DES algorithm in which all the S-boxes are replaced. The new S-boxes are all identical and defined as follows.

> Let $b_1, b_2, \cdots, b_6$ represent the six input bits to an S-box. Its output is $b_1 \oplus (b_2 \cdot b_3 \cdot b_4), (b_3 \cdot b_4 \cdot b_5) \oplus b_6, b_1 \oplus (b_4 \cdot b_5 \cdot b_2), (b_5 \cdot b_2 \cdot b_3) \oplus b_6$.

Here '$\oplus$' is bitwise XOR operation, and '$\cdot$' is bitwise multiplication. Design an algorithm to break 16-round DES with new S-boxes as efficiently as possible.

Solution

[Aga22]
We will break this 16-round DES with new S-boxes using differential cryptanalysis.
We know that r-2 round features are required to break r-round DES. We need a 14-round characteristic with the highest probability feasible in this case. This characteristic will be used to recover k_r (key of the rth round).
The following is the final algorithm for breaking this 16-round DES:

1. We will find the XOR output of r-2 round with as high probability as feasible
The values of L_r, R_r, and R_{r-1} are known. At L_{r-1}, which will be the same as R_{r-2}, we do not know the two values or their XOR. We may figure out the key for the rth round (K_r) if we can determine the XOR of the output of round r - 2, i.e., R_{r-2} with as high a probability as possible.
Because the new S-Boxes are non-linear, knowing the output XOR to an S-Box is problematic. One can simply not comment on the input XOR values, making obtaining the key

challenging. However, observing the structure of these new S-boxes, we can see that if the XOR of the two inputs to either of the S-Boxes (since they are all identical) is 000010, the XOR of the output is 0000 in 32 of the 64 possible pairs that yield this input XOR.
If we analyze random input pairings with 000010 as the input XOR to anyone S-Box, we may anticipate the output XOR to be 1110 with a chance of $32/64 = 1/2$. Thus, we can anticipate the XOR of the output of this S-Box with a probability of $1/2$ if we make sure that the input XOR to the remaining S-Boxes is all zeroes. Let us consider a two-round DES with XOR as 0000 0000 at R_0 and 0000 0002 at L_0 (Hexadecimal representation). Then, because XOR of the right half is all zeroes, XOR at L_1 will be 0000 0000, and XOR at R_1 will be 0000 0002.
The expansion box will now receive the output XOR at R_1 (0000 0002), yielding the 48-bit value 0000 0000 0004. We can see that if we divide this value into eight groups of size 6, the first 7 S-boxes will have all zeros, and the last S-box (S8) will have the binary value 000100. With probability $1/2$, the S8 box will generate binary value 0000 as the output, and the subsequent S-boxes will also generate binary value 0000, resulting in the final output after permutation being 0000 0000 at R_2 and the XOR at L_2 being 0000 0002.

This is the foundation for iterative 2-round characteristics with
x_0 = 0000 0002,
y_0 = 0000 0000,
$p_1 = 1$,
x_1 = 0000 0000,
y_1 = 0000 0002,
$p_2 = 1/2$,
x_2 = 0000 0002 and
y_2 = 0000 0000.

The 2-round characteristics are as follows.

$$(\bar{0}0002, \bar{0}\bar{0}, 1, \bar{0}\bar{0}, \bar{0}0002, \frac{1}{2}, \bar{0}0002, \bar{0}\bar{0})$$

The probability of the above 2-round characteristics is $\frac{1}{2}$.
We will multiply this by 7 to achieve 7*2, or a 14-round characteristic. The probability of a 14-round characteristic is $(\frac{1}{2})^7 = \frac{1}{128}$, which is significantly better than brute force.

As a result, by repeating this 2-round characteristic seven times, we acquire a 14-round characteristic that may be utilized to find the key k_r.

2. Finding output of XOR for S-boxes of sixteenth round and extracting last round key (k_r)

We know the XOR at R_14, with a probability of $\frac{1}{128}$, which is the same as L_15. We also know the value at R_16; thus, we know the permutation box output XOR, which gives us the XOR of S-box output for the 16th round.

Define

$$X_i = \{(\beta, \beta') | \beta \oplus \beta' = \beta_i \oplus \beta_i' and S_i(\beta) \oplus S_i(\beta') = \gamma\}$$

pair $(\beta_i, \beta_i') \in X_i$ whenever our guess for $\gamma_i \oplus \gamma_i' = \gamma$ is correct, which happens with probability $\frac{14}{64}$

Define

$$K_i = \{k | \alpha_i \oplus k = \beta and (\gamma, \gamma') \in X_i for some \beta'\}$$

Since, $(\beta_i, \beta_i') \in X_i$ with probability $\geq \frac{14}{64}$, we have $k_{(4,i)} \in K_i$ with probability $\geq \frac{14}{64}$.

Let $E(R_3) = \alpha_1\alpha_2...\alpha_8$ and $E(R_3')) = \alpha_1'\alpha_2'...\alpha_8'$ with $|\alpha_i| = 6 = |\alpha_i'|$

R_3 and R_3' are right-halves of output of third round on the plaintexts L_0R_0 and

$L_0'R_0'$ = $L_0'R_0$

Let $\beta_i = \alpha_i \oplus k_{(4,i)}$ and $\beta_i' = \alpha_i' \oplus k_{(4,i)}$, $|\beta_i| = 6 = |\beta_i'|$

$k_4 = k_{(4,1)}k_{(4,2)}...k_{(4,8)}$.

Let $\gamma_i = S_i(\beta_i)$ and $\gamma_i' = S_i(\beta_i')$,$|\gamma_i| = 4 = |\gamma_i'|$.

we know α_i, α_i' and $\beta_i \oplus \beta_i' = \alpha_i \oplus \alpha_i'$.

we also know a value γ such $\gamma_i \oplus \gamma_i' = \gamma$ with probability $\frac{14}{64}$

We have $|K_i| = |X_i|$ since α_i and $\beta \oplus \beta'$ is fixed for $(\beta,\beta') \in X_i$.

Therefore, $|K_i| \leq 16$ as per property of S-boxes.

Here is why we cannot utilize the strategy for three rounds; $K_{(4,i)}$ may be dropped out if we compute another K_i' and take its intersection with K_i because it is not guaranteed to be present in both.

Question 2

Suppose Anubha and Braj decide to do key-exchange using Diffie-Hellman scheme except for the choice of group used. Instead of using F_p^* as in Diffie-Hellman, they use S_n, the group of permutations of numbers in the range $[1, n]$. It is well-known that $|S| = n!$ and therefore, even for $n = 100$, the group has very large size. The key-exchange happens as follows:

> An element $g \in S_n$ is chosen such that g has large order, say l. Anubha randomly chooses a random number $c \in [1, l-1]$, and sends g^c to Braj. Braj choses another random number $d \in [1, l-1]$ and sends g^d to Anubha. Anubha computes $k = (g^d)^c$ and Braj computes $k = (g^c)^d$.

Show that an attacker Ela can compute the key k efficiently.

Solution

[Aga22, VRS18]

Overview

In this problem, Anubha and Braj decide to do key-exchange using Diffie-Hellman scheme except for the choice of group used. The default assumption for key exchange using Diffie-hellman scheme over symmetric group is the discrete logarithm problem over symmetric group S_n. Assume the oracle for calculating the $h = g^\alpha$ be $\mathcal{O}$ for a given g and α. Therefore, $\mathcal{O}^{-1}$ will denote the oracle for computing the discrete log. It means that for a given h and g, $\mathcal{O}^{-1}$ will compute α. Anubha and Braj need to compute g^c and g^d. So, they will run the oracle $\mathcal{O}$ privately to compute g^c and g^d respectively. In order to do so, we'll need to build an oracle $\mathcal{O}^{-1}$ which should run in polynomial time. Also, we need to build $\mathcal{O}^{-1}$ in such a way that it must be computationally feasible to retrieve c and d respectively. After c and d have been computed, we will find $g^{c \cdot d}$. It will be done by passing $((g, c \cdot d))$ to the oracle $\mathcal{O}$. As a result, we will be able to answer this problem. The idea for building $\mathcal{O}^{-1}$ will be explained in detail in the next sub-section. Later, a complexity study will be performed. All our claims will be mathematically justified at the end of the solution. So, being done with the basic overview, let's head towards the next subsection which explains basic designing of the oracle $\mathcal{O}^{-1}$.

Cryptanalysis phase

We need to evaluate α now. The values of g and h are publically available as per our knowledge. We know that any entity $e \in S_n$ is expressible through a cycle notation or a list of images represented as follows: $[e(1), e(2), \cdots, e(n)]$. Now, we'll move on to study the various stages of this cryptanalysis phase.

Stage: 1 Disjoint cycles are formed by decomposing g and h as shown below:

$$g = g_1 \circ g_2 \circ \cdots \circ g_r$$

$$h = h_1 \circ h_2 \circ \cdots \circ h_s$$

Here $\circ$ represents the composition operation. We'll include length-one cycles here if required, so that each $i \in \{1, 2, \cdots, n\}$ occurs in exactly 1 cycle.

- **Time Complexity Analysis:** The time required for this decomposition technique takes $O(n)$ time. Without sacrificing the generality, we assume that g acts on $1, 2, \cdots, n$. We start from $i = 1$. Then, we perform a look-up and determine the image of i under g. Then, whenever the image is equal to i, we stop the cycle and increment i to $i + 1$. If the image is not equal to i, we'll append $g(i)$ at the end of the cycle and repeat this process for index i. So, n look ups are required at max. After completion of this stage, we move on to the next stage.

Stage: 2 Computing and maintaining arrays G and H.

- The i-th index of $G[i]$ must store
  - the index j of the cycle g_j containing i
  - the place of i within this cycle $(1 \leq i \leq n)$

  It means that the tuple $G[i] = (j, p(i))$ showcases that the element i appears in cycle g_j at location $p(i)$.
- $H[i]$ will be built in a similar sort of method. Like previousy, $H[i] = (k, p(i))$ means that element i appears in cycle h_k at location $p(i)$. So, $H[i]$ consists:
  - the index k of the cycle h_k having i
  - the position of i within this cycle $(1 \leq i \leq n)$

- **Time Complexity Analysis:** Both the arrays G and H are having $2n$ integers. So, we can easily figure out that the construction of G and H requires $O(n)$ time. Now, we'll move on to the next stage.

Stage: 3 We'll be maintaining two arrays named $X[k]$, $Y[k]$, where $X[k]$ contains the initial element of each cycle h_k of h. Also, $Y[k]$ contains the second element of each cycle h_k of h. It's an important to note that $X[k] = Y[k]$ holds for length-one cycles. Our earlier built array G assists to find the cycle of g containing $X[k]$ and $Y[k]$ each $k \in [n]$. We'll use the array $Z[k]$ to preserve the difference between their locations. It means $Z[k] = p(Y[k]) - p(X[k])$ for all $k \in [n]$. Then we'll compute the length of the cycle containing the element i. Finally, we'll put it in the array W.

- **Time Complexity Analysis:** We know that g^α has at most n-cycles. So, the size of $X[k]$ and $Y[k]$ is at most n. Clearly $X[k]$ and $Y[k]$ lies to the same cycle $g_{k'}$ of g for some k'. In order to perform this, we need a look up in array G to identify that in which cycle of g the value $X[k]$ lies. Hence, it takes $O(n)$ for look up. We then look up the location numbers of $Y[i]$ and $X[i]$, and calculate their difference. It requires $O(n)$ operations and $O(n)$ look up. We move on to the next stage then.

Stage: 4 We need to calculate the value of α. In order to do so, we require to call Chinese Remainder Theorem. We need to do so a right now we got:

$$\alpha \equiv Z[i] \mod W[i] \text{ for } 1 \leq i \leq |Z|.$$

So we'll have $|Z|$-linear equations at max, where for any i, j $\gcd(W_i, W_j) \neq 1$

- **Time Complexity Analysis:** The time complexity of computing n-modular linear equations is analysed now. Let's have a look at the first two linear congruences:

$$\alpha \equiv Z[1] \mod W[1]$$

$$\alpha \equiv Z[2] \mod W[2].$$

Let α_1 be the solution of the two linear equations. It means that

$$\alpha \equiv \alpha_1 \mod (lcm(W[1], W[2])).$$

For solving these two linear equations, we use extended Euclidean algorithm. It

requires $O(\log(W[1]) \cdot \log(W[2]))$ time. It means that the best time complexity is $O(\log^2 n)$.

Now again

$$\alpha \equiv \alpha_1 \mod (lcm(W[1], W[2]))$$

$$\alpha \equiv Z[3] \mod W[3].$$

Let the solution of the above two linear equations be represented by α_2. Like previously, computing α_2 will require $O(\log^2 n)$ time. Therefore, we need to perform extended Euclidean algorithm $(n-1)$ times for solving $|Z| \approx O(n)$ equations. Thus the time complexity is

$$O(\sum_{k=1}^{n-1} k \cdot \log^2 n) = O(n^2 \log^2 n).$$

Therefore, the time complexity for computing α is $O(n^2 \log^2 n)$.

References

[Aga22] M. Agarwal. CS 641A Course Lecture Slides (4 to 7), 2022.

[VRS18] Maria Vasco, Angela Robinson, and Rainer Steinwandt. Cryptanalysis of a Proposal Based on the Discrete Logarithm Problem Inside Sn. *Cryptography*, 2:16, 07 2018.