CS641

Modern Cryptology
Indian Institute of Technology, Kanpur

Mid Semester Examination

Submission Deadline:
March 1, 2022, 23:55hrs

Group Name: Enciphered
Anindya Ganguly (21111261),
Gargi Sarkar (21111263),
Utkarsh Srivastava (21111063)

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

We will use differential cryptanalysis to break this 16-round DES with new S-Boxes.
We know that, to break r-round DES, we need r-2 round characteristics. Here, we need 14-round characteristic with as high probability as possible. We will recover k_r (key of the rth round) using this characteristic.
The algorithm to break this 16-round DES is as follows.

1. Predicting the XOR of output of round r-2 with as high probability as possible.
We know the values of L_r, R_r and R_{r-1}. We don't know the two values or their XOR at L_{r-1} which will be the same as the R_{r-2}. If we can somehow find the XOR of output of round r - 2 i.e. R_{r-2} with as high probability as possible, we can easily figure out the key

for rth round (K_r).
The new S-Boxes are non-linear in nature, that's why knowing the output XOR to an S-Box, one cannot comment on the input XOR values and thus finding the key is difficult. But if we observe the nature of these new S-boxes, we can see that if the XOR of the two inputs to any of the S-Boxes (since all are identical) is 000010, then out of 64 possible pairs which generate this input XOR, 32 pairs have the XOR of the output as 0000.
If we consider random input pairs that have input XOR to any one S-Box as 000010, then we expect that the XOR of the output will be 1110 with probability 32/64 = 1/2. If we ensure that the input XOR to remaining S-Boxes is all zeroes, then we can predict the XOR of the output of this S-Box with probability 1/2. Consider a 2-round DES with XOR at R_0 as 0000 0000 (Hexadecimal representation of 32-bit values), XOR at L_0 as 0000 0002. Then, XOR at L_1 will be 0000 0000 and XOR at R_1 will be 0000 0002 since, XOR of right half is all zeroes.
Now, the output XOR at R_1 (0000 0002) will be given to Expansion box which will produce the 48-value 0000 0000 0004 (in Hexadecimal). If we divide this value in 8 groups of size 6 each, we see that the first 7 S-boxes will get all zeroes and the last S-box (S8) will get value 000100 (binary). This S8 box will produce 0000 (binary) as the output with probability 1/2 and other remaining S-boxes will also produce 0000 (binary) as the output, which makes the final output after permutation as 0000 0000 at R_2 and the XOR at L_2 is 0000 0002.
This forms the basis for 2-round iterative 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.
We get the following 2-round characteristics.

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

The probability of the above 2- round characteristics is $\frac{1}{2}$.
We will concatenate this 7 times to get 7*2 i.e. 14-round characteristic. The probability of 14-round characteristic will be $(\frac{1}{2})^7 = \frac{1}{128}$ which is far better than the brute force. Thus, using this 2-round characteristic 7 times, we will get 14-round characteristic which can be used to find the key k_r.

2. Prediction of output XOR for S-boxes of 16th round

We know the XOR at R_{r-2} i.e. R_{14} with probability $\frac{1}{128}$ which is same as L_{15}. We also know the value at R_{16}, hence we know the output XOR at permutation box which in turn gives us the XOR of output of S-boxes for 16th round.

3. Extracting last round key (k_r)

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 probabilty $\geq \frac{14}{64}$, 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 (β,β') $\in X_i$.

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

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	0000	0000	0000	0000	0000	0000	0000	0100	0000	0000	0000	0010	0000	0001	1000	1111
01	0101	0101	0101	0101	0101	0101	0101	0001	0101	0101	0101	0010	0101	0100	1101	1010
10	1010	1010	1010	1010	1010	1010	1010	1110	1010	1010	1010	1000	1010	1011	0010	0101
11	1111	1111	1111	1111	1111	1111	1111	1011	1111	1111	1111	1100	1111	1110	0111	0000

We cannot use the method for three rounds here: If we compute another K_i' and take its intersection with K_i, $K_{(4,i)}$ may get dropped out since it is not garunteed 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

Basic Idea

We observe that the underlying assumption for the DH scheme over symmetric group is Discrete logarithm problem over symmetric group S_n. Suppose $\mathcal{O}$ denotes the oracle for computing the $h = g^\alpha$ for a given g and α, and $\mathcal{O}^{-1}$ denotes the oracle for computing the discrete log, that is for a given h and g, it will compute α.

In the problem Anubha (A) and Braj (B) run the oracle $\mathcal{O}$ privately to compute g^c and g^d respectively. Our task is to design a oracle $\mathcal{O}^{-1}$which will runs in polynomial time and computationally feasible to retrieve c and d. After computing c and d, we will invoke $((g, c \cdot d))$ to the oracle $\mathcal{O}$ for computing $g^{c \cdot d}$. Hence, we able to solve the question.

The next section roughly sketched the idea for constructing $\mathcal{O}^{-1}$. Later complexity analysis has been carried out. The discussion ends with a mathematical justification of our claims.

Cryptanalysis

As we know that g and h are publicly available and goal is to compute α. Any elements $e \in S_n$ can represented via cycle notation or a list of images $[e(1), e(2), \cdots, e(n)]$.

Phase: 1 Suppose g and h can be decomposed into disjoint cycles

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

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

where $\circ$ denotes the composition operation. Note that each every $i \in \{1, 2, \cdots, n\}$ lies in exactly one cycle.

- **Time Complexity:** The decomposition techniques requires $O(n)$-time. Without loss of any generality we are assuming that g acts on $1, 2, \cdots, n$. Let us start from $i = 1$, do a look-up and compute the image of i under g. Now, when image is equal to i, stop the cycle and do $i + 1$, and for image not equal to i, append $g(i)$ at the end of the cycle and repeat the process for index i. So at most n look up is required.

Phase: 2 Maintaining arrays G and H.

- The i-th index of $G[i]$ contains
  1. the index j of the cycle g_j having i
  2. the location of i within this cycle $(1 \leq i \leq n)$

  Basically $G[i]$ can be consider as a tuple $(j, p(i))$ which illustrates that element i appears in cycle g_j at a position $p(i)$.
- In similar fashion $H[i]$ will be constructed. Like above $H[i] = (k, p(i))$ means that element i appears in cycle h_k at location $p(i)$. So $H[i]$ contains:
  1. the index k of the cycle h_k having i
  2. the location of i within this cycle $(1 \leq i \leq n)$

- **Time Complexity:** The arrays G and H each of them have $2n$ integers. Clearly construction of G and H require $O(n)$ time.

Phase: 3 Again maintain two array called $X[k]$, $Y[k]$, where $X[k]$ has the first element of each cycle h_k of h and $Y[k]$ has the second element of each cycle h_k of h. Note that $X[k] =$

$Y[k]$ holds for length-one cycles. Our previously constructed array G helps to find the cycle of g containing $X[k]$ and $Y[k]$ each $k \in [n]$. Use array $Z[k]$ to maintain the difference between their location that means $Z[k] = p(Y[k]) - p(X[k])$ for all $k \in [n]$. Then we calculate the length of the cycle containing the element i and put it in the array W.

- **Time Complexity:**Since g^α has at most n-cycle, so 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'. To perform this, needs a look up in array G to identify which cycle of g the value $X[k]$ lies. Thus it requires $O(n)$ look up. Look up the location numbers of $Y[i]$ and $X[i]$ and subtract. This needs $O(n)$ operations and $O(n)$ look up.

Phase: 4 To obtain the value of α, we need to call CRT. Because, right now we got

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

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

- **Time Complexity:** Here we analyze the time complexity for computing n-modular linear equation. Let us consider first two linear congruences

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

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

Suppose α_1 be the solution of two linear equations. That means $\alpha \equiv \alpha_1 \mod (lcm(W[1], W[2]))$. Solving these two linear equations use extended Euclidean algorithm, which need $O(\log(W[1]) \cdot \log(W[2]))$ time. That is the best time complexity is $O(\log^2 n)$

Now again

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

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

Suppose α_2 denote the solution of above two linear equations. Like above, computing α_2 needs $O(\log^2 n)$ time. Thus to solve $|Z| \approx O(n)$ equations, we need to perform $(n-1)$ times extended Euclidean algorithm. Thus the time complexity

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

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

Correctness