Cryptography - Homework 1 Theory

Samuel Breese

September 23, 2018

1 Differential cryptanalysis of simplified DES

Differential cryptanalysis works by exploiting nonuniformity in the cipher. This is done by analyzing the correspondence between differences in plaintext and differences in the corresponding ciphertext. In a cipher defined by a simple linear transformation (e.g. Caesar cipher, other substitution ciphers), this difference will be constant: a difference in the plaintext will correspond to the same difference in the ciphertext. It is easy to intuitively understand this by means of a geometric analogy, in that any two parallel lines have a constant distance along the perpendicular.

In the case of DES, however, nonlinearity is introduced through tables (i.e., the S-boxes). Thus, the relationship between a difference in plaintexts and a difference in ciphertexts is no longer constant, but instead based on the key, which we can treat as a random variable uniformly distributed over the key space. If an S-box is not uniformly distributed, we can build a probabilistic model of the way that a difference in two inputs to that S-box will affect the difference of the corresponding two outputs. This model allows us to reduce the search space for keys, since the input of the S-box is already a difference with the partial key for the round. Thus, with a known pair of inputs (pre-XOR with key) and the corresponding pair of outputs, we can deduce possible values for the key, since for any output difference y' and corresponding inputs i_1 and i_2 with difference i', we know that since $i' = x' = x_1 \oplus x_2$ we can find possible values for x_1 and x_2. Since the key $k = i \oplus x$, this reduces the possible keys to the number of possible inputs to the S-box that can have difference i' and output difference y' (that is, the values in the difference table for the S-box).

We will proceed with an example for how to reduce the key search space for the S-box S_0. First, we encode the S-box as a Python function for ease of computation (Listing 1).

```python
def S0(x):
    row = ((0b1000 & x) >> 2) | (0b0001 & x)
    column = (0b0110 & x) >> 1
    return [[1, 0, 3, 2],
            [3, 2, 1, 0],
            [0, 2, 1, 3],
            [3, 1, 3, 2]][row][column]
```

Listing 1: Python code to compute S_0

Next, we use this function to compute, for every pair of input values x_1 and x_2, the difference of inputs $x' = x_1 \oplus x_2$ and the difference of outputs $y' = S_0(x_1) \oplus S_0(x_2)$. This is trivial using a list comprehension (Listing 2).

```python
print([(x1, x2, x1 ^ x2, S_0(x1) ^ S_0(x2))
       for x1 in range(0x0, 0x10)
       for x2 in range(0x0, 0x10)])
```

Listing 2: Python code to compute input and output difference for each possible pair of inputs to S_0

The results are shown in the following (lengthy) table:

x_1	x_2	$x' = x_1 \oplus x_2$	$y' = S_0(x_1) \oplus S_0(x_2)$
0	0	0	0
1	0	1	2
2	0	2	1
3	0	3	3
4	0	4	2
5	0	5	0
6	0	6	3
7	0	7	1
8	0	8	1
9	0	9	2
A	0	A	3
B	0	B	0
C	0	C	0
D	0	D	2

Continued on next page

Continued from previous page

x_1	x_2	$x' = x_1 \oplus x_2$	$y' = S_0(x_1) \oplus S_0(x_2)$
E	0	E	2
F	0	F	3
0	1	1	2
1	1	0	0
2	1	3	3
3	1	2	1
4	1	5	0
5	1	4	2
6	1	7	1
7	1	6	3
8	1	9	3
9	1	8	0
A	1	B	1
B	1	A	2
C	1	D	2
D	1	C	0
E	1	F	0
F	1	E	1
0	2	2	1
1	2	3	3
2	2	0	0
3	2	1	2
4	2	6	3
5	2	7	1
6	2	4	2
7	2	5	0
8	2	A	0
9	2	B	3
A	2	8	2
B	2	9	1
C	2	E	1
D	2	F	3
E	2	C	3
F	2	D	2
0	3	3	3
1	3	2	1
2	3	1	2

Continued on next page

Continued from previous page

x_1	x_2	$x' = x_1 \oplus x_2$	$y' = S_0(x_1) \oplus S_0(x_2)$
3	3	0	0
4	3	7	1
5	3	6	3
6	3	5	0
7	3	4	2
8	3	B	2
9	3	A	1
A	3	9	0
B	3	8	3
C	3	F	3
D	3	E	1
E	3	D	1
F	3	C	0
0	4	4	2
1	4	5	0
2	4	6	3
3	4	7	1
4	4	0	0
5	4	1	2
6	4	2	1
7	4	3	3
8	4	C	3
9	4	D	0
A	4	E	1
B	4	F	2
C	4	8	2
D	4	9	0
E	4	A	0
F	4	B	1
0	5	5	0
1	5	4	2
2	5	7	1
3	5	6	3
4	5	1	2
5	5	0	0
6	5	3	3
7	5	2	1

Continued on next page

Continued from previous page

x_1	x_2	$x' = x_1 \oplus x_2$	$y' = S_0(x_1) \oplus S_0(x_2)$
8	5	D	1
9	5	C	2
A	5	F	3
B	5	E	0
C	5	9	0
D	5	8	2
E	5	B	2
F	5	A	3
0	6	6	3
1	6	7	1
2	6	4	2
3	6	5	0
4	6	2	1
5	6	3	3
6	6	0	0
7	6	1	2
8	6	E	2
9	6	F	1
A	6	C	0
B	6	D	3
C	6	A	3
D	6	B	1
E	6	8	1
F	6	9	0
0	7	7	1
1	7	6	3
2	7	5	0
3	7	4	2
4	7	3	3
5	7	2	1
6	7	1	2
7	7	0	0
8	7	F	0
9	7	E	3
A	7	D	2
B	7	C	1
C	7	B	1

Continued on next page

Continued from previous page

x_1	x_2	$x' = x_1 \oplus x_2$	$y' = S_0(x_1) \oplus S_0(x_2)$
D	7	A	3
E	7	9	3
F	7	8	2
0	8	8	1
1	8	9	3
2	8	A	0
3	8	B	2
4	8	C	3
5	8	D	1
6	8	E	2
7	8	F	0
8	8	0	0
9	8	1	3
A	8	2	2
B	8	3	1
C	8	4	1
D	8	5	3
E	8	6	3
F	8	7	2
0	9	9	2
1	9	8	0
2	9	B	3
3	9	A	1
4	9	D	0
5	9	C	2
6	9	F	1
7	9	E	3
8	9	1	3
9	9	0	0
A	9	3	1
B	9	2	2
C	9	5	2
D	9	4	0
E	9	7	0
F	9	6	1
0	A	A	3
1	A	B	1

Continued on next page

Continued from previous page

x_1	x_2	$x' = x_1 \oplus x_2$	$y' = S_0(x_1) \oplus S_0(x_2)$
2	A	8	2
3	A	9	0
4	A	E	1
5	A	F	3
6	A	C	0
7	A	D	2
8	A	2	2
9	A	3	1
A	A	0	0
B	A	1	3
C	A	6	3
D	A	7	1
E	A	4	1
F	A	5	0
0	B	B	0
1	B	A	2
2	B	9	1
3	B	8	3
4	B	F	2
5	B	E	0
6	B	D	3
7	B	C	1
8	B	3	1
9	B	2	2
A	B	1	3
B	B	0	0
C	B	7	0
D	B	6	2
E	B	5	2
F	B	4	3
0	C	C	0
1	C	D	2
2	C	E	1
3	C	F	3
4	C	8	2
5	C	9	0
6	C	A	3

Continued on next page

Continued from previous page

x_1	x_2	$x' = x_1 \oplus x_2$	$y' = S_0(x_1) \oplus S_0(x_2)$
7	C	B	1
8	C	4	1
9	C	5	2
A	C	6	3
B	C	7	0
C	C	0	0
D	C	1	2
E	C	2	2
F	C	3	3
0	D	D	2
1	D	C	0
2	D	F	3
3	D	E	1
4	D	9	0
5	D	8	2
6	D	B	1
7	D	A	3
8	D	5	3
9	D	4	0
A	D	7	1
B	D	6	2
C	D	1	2
D	D	0	0
E	D	3	0
F	D	2	1
0	E	E	2
1	E	F	0
2	E	C	3
3	E	D	1
4	E	A	0
5	E	B	2
6	E	8	1
7	E	9	3
8	E	6	3
9	E	7	0
A	E	4	1
B	E	5	2

Continued on next page

Continued from previous page

x_1	x_2	$x' = x_1 \oplus x_2$	$y' = S_0(x_1) \oplus S_0(x_2)$
C	E	2	2
D	E	3	0
E	E	0	0
F	E	1	1
0	F	F	3
1	F	E	1
2	F	D	2
3	F	C	0
4	F	B	1
5	F	A	3
6	F	9	0
7	F	8	2
8	F	7	2
9	F	6	1
A	F	5	0
B	F	4	3
C	F	3	3
D	F	2	1
E	F	1	1
F	F	0	0

From here, we can aggregate the above data (see Listing 3) to produce the input values x that are part of an input pair x_1, x_2 where x' and y' are equal to particular values. Given this, we can determine the possible values for x given inputs i_1 and i_2 with difference $i' = x'$ and output difference y'.

```python
print([(xp, yp, [x1
                 for x1 in range(0x0, 0x10)
                 for x2 in range(0x0, 0x10)
                 if x1 ^ x2 == xp and S0(x1) ^ S0(x2) == yp])
       for xp in range(0x0, 0x10)
       for yp in range(0, 4)])
```

Listing 3: Python code to compute inputs producing input-output difference combinations for S_0

The results are collected in the following table:

x'	y'	possible x
0	0	0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15
1	0	
2	0	
3	0	14, 13
4	0	13, 9
5	0	5, 4, 7, 6, 1, 0, 3, 2, 15, 10
6	0	
7	0	14, 12, 11, 9
8	0	9, 1
9	0	10, 13, 12, 15, 3, 5, 4, 6
A	0	8, 14, 2, 4
B	0	11, 0
C	0	12, 13, 15, 10, 6, 0, 1, 3
D	0	9, 4
E	0	11, 5
F	0	14, 8, 7, 1
0	1	
1	1	15, 14
2	1	2, 3, 0, 1, 6, 7, 4, 5, 15, 13
3	1	11, 10, 9, 8
4	1	12, 14, 8, 10
5	1	
6	1	15, 9
7	1	7, 6, 5, 4, 3, 2, 1, 0, 13, 10
8	1	8, 14, 0, 6
9	1	11, 2
A	1	9, 3
B	1	10, 15, 13, 12, 1, 7, 6, 4
C	1	11, 7
D	1	14, 8, 5, 3
E	1	15, 12, 13, 10, 4, 2, 3, 1
F	1	9, 6
0	2	
1	2	1, 0, 3, 2, 5, 4, 7, 6, 13, 12
2	2	10, 11, 8, 9, 14, 12
3	2	
4	2	4, 5, 6, 7, 0, 1, 2, 3

Continued on next page

Continued from previous page

x'	y'	possible x
5	2	12, 14, 9, 11
6	2	13, 11
7	2	15, 8
8	2	10, 12, 13, 15, 2, 4, 5, 7
9	2	9, 0
A	2	11, 1
B	2	8, 14, 3, 5
C	2	9, 5
D	2	13, 12, 15, 10, 7, 1, 0, 2
E	2	14, 8, 6, 0
F	2	11, 4
0	3	
1	3	9, 8, 11, 10
2	3	
3	3	3, 2, 1, 0, 7, 6, 5, 4, 15, 12
4	3	15, 11
5	3	13, 8
6	3	6, 7, 4, 5, 2, 3, 0, 1, 14, 12, 10, 8
7	3	
8	3	11, 3
9	3	8, 14, 1, 7
A	3	10, 15, 12, 13, 0, 6, 7, 5
B	3	9, 2
C	3	14, 8, 4, 2
D	3	11, 6
E	3	9, 7
F	3	15, 13, 12, 10, 5, 3, 2, 0

Finally, we can simplify the above by simply noting the number of unique inputs x that can contribute to some particular input and output difference (Listing 4). This is useful when considering multiple S-boxes, as it allows us to calculate just how much of a reduction in the key search space the differential cryptanalysis yields.

```
print([(xp,
        len([x1
             for x1 in range(0x0, 0x10)
             for x2 in range(0x0, 0x10)
             if x1 ^ x2 == xp and S0(x1) ^ S0(x2) == 0]),
        len([x1
             for x1 in range(0x0, 0x10)
             for x2 in range(0x0, 0x10)
             if x1 ^ x2 == xp and S0(x1) ^ S0(x2) == 1]),
        len([x1
             for x1 in range(0x0, 0x10)
             for x2 in range(0x0, 0x10)
             if x1 ^ x2 == xp and S0(x1) ^ S0(x2) == 2]),
        len([x1
             for x1 in range(0x0, 0x10)
             for x2 in range(0x0, 0x10)
             if x1 ^ x2 == xp and S0(x1) ^ S0(x2) == 3]))
       for xp in range(0x0, 0x10)])
```

Listing 4: Python code to compute frequency of output differences of S_0 given input differences

This is summarized in the following table:

x'	$y' = 0$	$y' = 1$	$y' = 2$	$y' = 3$
0	16	0	0	0
1	0	2	10	4
2	0	10	6	0
3	2	4	0	10
4	2	4	8	2
5	10	0	4	2
6	0	2	2	12
7	4	10	2	0
8	2	4	8	2
9	8	2	2	4
A	4	2	2	8
B	2	8	4	2
C	8	2	2	4
D	2	4	8	2
E	2	8	4	2
F	4	2	2	8

Using the above tables, we can determine possible keys using known input/output pairs. For example, let's say that we know that two inputs (pre-XOR with the key k), $i_1 = 4$ and $i_2 = 7$, and the we know that the corresponding outputs are $S_0(i_1 \oplus k) = 3$ and $S_0(i_2 \oplus k) = 2$. Since $i_1 \oplus i_2 = (i_1 \oplus k) \oplus (i_2 \oplus k) = 3$, and $S_0(i_1 \oplus k) \oplus S_0(i_2 \oplus k) = 1$, we know that the possible values for $i_1 \oplus k$ and $i_2 \oplus k$ are B, A, 9, and 8 Since $k = i_1 \oplus x_1$ and $k = i_2 \oplus x_2$, we know that k must be one of $4 \oplus B = 7 \oplus 8 = F$, $4 \oplus A = 7 \oplus 9 = E$, $4 \oplus 9 = 7 \oplus A = D$, or $4 \oplus 8 = 7 \oplus B = C$.

Already this has substantially limited the search space for k - there are only 2^2 possible keys, down from 2^4. We can continue to reduce the key space with access to more input/output pairs. Say we know another pair of inputs, $i_1 = A$ and $i_2 = 2$. The corresponding outputs using the same key are $S_0(i_1 \oplus k) = 0$ and $S_0(i_2 \oplus k) = 2$. Because $i_1 \oplus i_2 = 8$ and $S_0(i_1 \oplus k) \oplus S_0(i_2 \oplus k) = 2$, we know that the possible values for $i_1 \oplus k$ and $i_2 \oplus k$ are A, C, D, F, 2, 4, 5, and 7. Thus, we know that k must be one of $A \oplus A = 2 \oplus 2 = 0$, $A \oplus C = 2 \oplus 4 = 6$, $A \oplus D = 2 \oplus 5 = 7$, $A \oplus F = 2^7 = 5$, $A \oplus 2 = 2 \oplus A = 8$, $A \oplus 4 = 2 \oplus C = E$, $A \oplus 5 = 2 \oplus D = F$, and $A \oplus 7 = 2 \oplus F = D$.

This reduces our search space even further, down to three possible keys, since we now know that $k = C$ is not possible (only $k = D$, $k = E$, and

$k = F$). This same principle applies as we add more and more known inputs and outputs, allowing us to winnow the search space down to a size amenable to brute force.

2 Entropy of sample cryptosystem

The cryptosystem:

- $P = \{a, b, c\}$ with $P_P(a) = \frac{1}{3}$, $P_P(b) = \frac{1}{6}$, and $P_P(c) = \frac{1}{2}$.
- $K = \{k_1, k_2, k_3\}$ with $P_K(k_1) = \frac{1}{2}$, $P_K(k_2) = \frac{1}{4}$, and $P_K(k_3) = \frac{1}{4}$.
- $C = \{1, 2, 3, 4\}$
- $e_{k_1}(a) = 1,\ e_{k_1}(b) = 2,\ e_{k_1}(c) = 2$
- $e_{k_2}(a) = 2,\ e_{k_2}(b) = 3,\ e_{k_2}(c) = 1$
- $e_{k_3}(a) = 3,\ e_{k_3}(b) = 4,\ e_{k_3}(c) = 4$

We must find $H(K|C) = H(K) + H(P) - H(C)$.

Assuming X is a random variable which takes on a finite set of n values according to some distribution $p(X)$, then

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Therefore,

$$H(P) = -\left(\frac{\log_2 \frac{1}{3}}{3} + \frac{\log_2 \frac{1}{6}}{6} + \frac{\log_2 \frac{1}{2}}{2}\right) \approx 1.45915$$

and

$$H(K) = -\left(\frac{\log_2 \frac{1}{2}}{2} + \frac{\log_2 \frac{1}{4}}{4} + \frac{\log_2 \frac{1}{4}}{4}\right) \approx 1.5$$

Computing $H(C)$ requires us to find a probability distribution P_C for the ciphertext, which is slightly more involved. To do this, we look at values of K and P that can lead to a given ciphertext C:

$$P_C(1) = P_P(a)P_K(k_1) + P_P(c)P_K(k_2) = \frac{1}{3} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{4} = \frac{7}{24}$$

$$P_C(2) = P_P(a)P_K(k_2) + P_P(b)P_K(k_1) + P_P(c)P_K(k_1) = \frac{1}{3} \times \frac{1}{4} + \frac{1}{6} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = \frac{5}{12} = \frac{10}{24}$$

$$P_C(3) = P_P(a)P_K(k_3) + P_P(b)P_K(k_2) = \frac{1}{3} \times \frac{1}{4} + \frac{1}{6} \times \frac{1}{4} = \frac{1}{8} = \frac{3}{24}$$

$$P_C(4) = P_P(b)P_K(k_3) + P_P(c)P_K(k_3) = \frac{1}{6} \times \frac{1}{4} + \frac{1}{2} \times \frac{1}{4} = 1/6 = \frac{4}{24}$$

From here, it is simple to compute

$$H(C) = -\left(\frac{7 \log_2 \frac{7}{24}}{24} + \frac{10 \log_2 \frac{10}{24}}{24} + \frac{3 \log_2 \frac{3}{24}}{24} + \frac{4 \log_2 \frac{4}{24}}{24} \right) \approx 1.85056$$

Thus, $H(K|C) \approx 1.5 + 1.45915 - 1.85056 \approx 1.10859$.