

CSI 4108
Carlisle Adams

Daniel Kinsey, 300256085
Nicolas Bérubé, 300239551

Cryptanalysis Project
Due November 8th

Code Analysis

Link to github repository: <https://github.com/kinseyda/CSI-4108-cryptanalysis-project>

The SPN

In order to create the 10,000 required plaintext / ciphertext pairs, a simple SPN implementation was created. It is modeled after the descriptions in the Heys paper, and so uses only one single S-Box. It is a very object oriented design, intended to be readable and potentially easily extensible to a “real” SPN which would have a matrix of S-Boxes.

The SPN works on “Blocks” of data, which are 16 bits. Behind the scenes, these are a tuple of four separate integers, as this allows for iterating and doing math easily. Operations are defined for the Block class such that they can be used like any integer, that wraps around to stay within 16 bits as expected. Additionally, several helper functions are defined to convert these blocks into different forms, such as a list of pure binary integers or as a single integer in the range 0 - 65535.

An S-Box class is also defined, simply storing a list of keys. The first in the list is the mapping of 0, the second is the mapping of 1, and so forth.

Finally, the SPN class itself is a class that contains one single S-Box (reused sixteen times) and a tuple of round keys (themselves blocks). The class contains a private function to perform cypher rounds, which utilizes a functional design to make going forwards and backwards both very readable. Substitution works by going forward / backwards through the SPN’s S-Box, permutation is done by computing the index as shown in the table in the Heys paper, and key mixing is done by a simple XOR. These steps are repeated for each key that is provided to the SPN upon construction.

Differential Characteristic

The java code has two parts to it, the first one is the DifferenceDistributionTable.java file, it was used to create the Difference Distribution Table using the default S-box, it was later implemented and added to the spn implementation. (Table 7)

The second part of the code was the differentialCharacteristic.java file, this file was used to calculate the route differential characteristic and the route a sample input would finish with. It worked by taking the DDT from the SPN implementation and then with a sample input it would go through 3 rounds and find the last output and probability of the difference pair (Figure 5). My understanding of how the difference pair was calculated was by using the most likely value to come out from the DDT in each row and if there was none, then it would take the first highest one (can be seen with image 1, red is highest value and blue is the first highest value)

S-Box Analysis

The S-Box

For the sake of simplicity, we chose to use the second DES S-Box, similar to the paper choosing the first. This can be found on Wikipedia

([https://en.wikipedia.org/wiki/DES_supplementary_material#Substitution_boxes_\(S-boxes\)](https://en.wikipedia.org/wiki/DES_supplementary_material#Substitution_boxes_(S-boxes)))

The S-Box is the following:

From	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
To	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8

Difference Distribution Table

Difference Distribution Table (DDT):																
$\Delta \setminus \Delta Y$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	2	0	0	0	2	8	0	0	2
2	0	0	0	4	0	0	2	2	0	0	2	2	0	4	0	0
3	0	4	0	0	2	0	2	0	2	0	2	0	0	0	0	4
4	0	0	0	6	0	2	0	0	0	0	2	0	0	2	2	2
5	0	2	2	0	2	0	2	0	2	2	0	0	0	0	0	4
6	0	0	0	0	0	2	2	0	0	4	0	0	0	2	6	0
7	0	2	6	0	0	4	0	0	0	2	2	0	0	0	0	0
8	0	0	0	0	0	0	0	4	0	4	2	2	0	0	2	2
9	0	0	2	0	0	4	2	0	2	0	0	4	2	0	0	0
A	0	0	2	0	6	0	0	0	0	0	2	0	2	4	0	0
B	0	4	0	2	0	0	2	0	4	0	0	2	0	0	2	0
C	0	2	0	0	6	0	0	0	0	2	4	0	2	0	0	0
D	0	0	0	0	0	2	4	2	4	0	0	0	0	2	0	2
E	0	2	2	2	0	0	0	6	0	2	0	0	0	0	2	0
F	0	0	2	0	0	2	0	0	2	0	0	4	2	2	2	0

Best Differential Characteristic

Best Characteristic - Overall: Probability: 3.12% Best input difference: 0003 Final round input difference: cc00 0->0 0->0 0->0 3->1 0->0 0->0 0->0 1->c 1->c 1->c 0->0 0->0	Best Characteristic - '1' in ΔP : Probability: 1.76% Best input difference: 0001 Final round input difference: 0c00 0->0 0->0 0->0 1->c 1->c 1->c 0->0 0->0 c->4 c->4 0->0 0->0	Best Characteristic - 2 S-Boxes affected: Probability: 3.12% Best input difference: 0003 Final round input difference: cc00 0->0 0->0 0->0 3->1 0->0 0->0 0->0 1->c 1->c 1->c 0->0 0->0
--	---	---

Attack Analysis - Daniel's Attack on Nicolas

The first thing I did was to examine Nicolas' code to create differential characteristics. I used this to learn how they work, and created my own version that worked on the Block classes previously designed. There are a number of helper functions defined but the important ones are:

- `differential_characteristic_path`
 - Given a differential distribution table, a plaintext delta (ΔP), and a number of rounds, this function traces through the rounds of the S-Boxes to determine the path, similar to figure 5 in the Heys paper. It returns a matrix of tuples, where each tuple is the input and output of the S-Box at that position, allowing us to visualize the path. Additionally, it returns the input to the final round (one step after what is in the matrix, ΔU_4 in the Heys paper), and the probability of the path occurring for the given input difference block.
- `best_differential_characteristic`
 - Repeatedly uses `differential_characteristic_path` to find the most probable path given some criteria. The set of possible criteria that can be provided includes which values of ΔP to use, whether to only consider paths for which good pairs exist in a given text, a list of which S-Boxes need to be affected in the final row, a number of S-Boxes that need to be affected, and more.

After creating these functions, to find a byte in the final subkey we simply need to find a differential characteristic that affects two S-Boxes in the final round, then apply it across the appropriate subkeys. Matching pairs in the plaintext files are found, and their corresponding ciphertexts partially decrypted. When the decryptions match the final round input difference (previously found by `best_differential_characteristic`), then we increment a count in a dictionary for that subkey. In the end, we have a dictionary of partial subkeys to their probabilities. Then we just find and return the maximal value. In this case, we get that the subkey is likely to be 3A__, and we can confirm this by checking Nicolas' key file.

```
-----Attacking S-Box (0 f 7 4 e 2 d 1 a 6 c b 9 5 3 8)-----
- Computing difference distribution table...
- Computing best differential characteristic...
- Computing subkey probabilities...
Top 10 subkey probabilities:
3a00: 5.44%
7a00: 3.70%
3e00: 3.37%
7e00: 2.44%
3300: 2.19%
2a00: 2.15%
fa00: 2.11%
3600: 2.04%
3b00: 1.93%
aa00: 1.85%
Last key will be: '3a__'
```

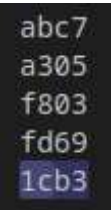
```
6ec5
a505
8e0b
69f6
3a98
```

Attack Analysis - Nicolas' Attack on Daniel

First thing that I did was to get the 10,000 16bit plaintext and resulting ciphertext from Daniel. After a quick analysis of the texts I discovered that we had a issue with our implementation of the algorithm and the resulting ciphertext and plaintext were not related, after a quick fix we were able to continue with the experiment and I received a new set of plaintext and ciphertext and we continued. My program was to make code that would get the best differential characteristic from an input and show the path that it would take which was then converted into python code to help with the key finding process. Then while running Daniel's key finding algorithm, I was able to follow the same procedure and the likely key byte was found

- `DifferenceDistributionTable`
 - Given a SBox, it would create a Distribution Table by going through all possible input differences and then testing all values of X1 with the ΔX , this value would find X2 and using this value we could then find Y1 and Y2 using the SBox. Then with Y1 and Y2 we could get ΔY which could then help us create the table
- `biggestTableValue`
 - Given a differential distribution table that we have already created previously, it would calculate the most likely characteristic that should be output
- `sBox`
 - Using the most likely value, we could then calculate the output from the input and calculate the probability of it occurring
- `main`
 - Using all functions previously described, we could then calculate what the likely output from the encryption could be and the probability of it happening. The result of the code can be seen in "Figure 1", it would show the path that was followed to get the output
- `PairFinder`
 - This code was used to find all pairs that have a specific ΔP input, this could then be used in the future when we would count the number of pairs to find the probability of occurrence of right pairs with the occurred partial subkey. This idea was later abandoned since the python would create plaintext pairs automatically instead of being found later

```
Top 10 subkey probabilities:
1c00: 6.00%
5c00: 3.68%
1800: 3.65%
1500: 2.57%
dc00: 2.46%
1000: 2.28%
0c00: 2.24%
5800: 2.20%
1d00: 2.10%
8c00: 2.06%
Last key will be: 1c
```



Figures and Tables

Image 1: Most likely value to come from the DDT

Difference Distribution Table (DDT):

$\Delta \setminus \Delta Y$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	0	0	0	2	0	0	0	2	8	0	0	2
2	0	0	0	4	0	0	2	2	0	0	2	2	0	4	0	0
3	0	4	0	0	2	0	2	0	2	0	2	0	0	0	0	4
4	0	0	0	6	0	2	0	0	0	0	2	0	0	2	2	2
5	0	2	2	0	2	0	2	0	2	2	0	0	0	0	0	4
6	0	0	0	0	0	2	2	0	0	4	0	0	0	2	6	0
7	0	2	6	0	0	4	0	0	0	2	2	0	0	0	0	0
8	0	0	0	0	0	0	0	4	0	4	2	2	0	0	2	2
9	0	0	2	0	0	4	2	0	2	0	0	4	2	0	0	0
A	0	0	2	0	6	0	0	0	0	0	2	0	2	4	0	0
B	0	4	0	2	0	0	2	0	4	0	0	2	0	0	2	0
C	0	2	0	0	6	0	0	0	0	2	4	0	2	0	0	0
D	0	0	0	0	0	2	4	2	4	0	0	0	0	2	0	2
E	0	2	2	2	0	0	0	6	0	2	0	0	0	0	2	0
F	0	0	2	0	0	2	0	0	2	0	0	4	2	2	2	0

Figure 1: Default Differential Characteristic and Probability with 0000,0000,1011,0000

```

Input binary value: 0000000010110000
Sbox0 will go from 0 to 0
Sbox1 will go from 0 to 0
Sbox2 will go from 11 to 2
Sbox3 will go from 0 to 0
Input value: 0000,0000,0010,0000
Input Array: [
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 1, 0],
  [0, 0, 0, 0],
]
Output Array after permutation: [
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 1, 0],
  [0, 0, 0, 0],
]
Sbox0 will go from 0 to 0
Sbox1 will go from 0 to 0
Sbox2 will go from 2 to 5
Sbox3 will go from 0 to 0
Input value: 0000,0000,0101,0000
Input Array: [
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 1, 0, 1],
  [0, 0, 0, 0],
]
Output Array after permutation: [
  [0, 0, 0, 0],
  [0, 0, 1, 0],
  [0, 0, 0, 0],
  [0, 0, 1, 0],
]
Sbox0 will go from 0 to 0
Sbox1 will go from 2 to 5
Sbox2 will go from 0 to 0
Sbox3 will go from 2 to 5
Input value: 0000,0101,0000,0101
Input Array: [
  [0, 0, 0, 0],
  [0, 1, 0, 1],
  [0, 0, 0, 0],
  [0, 1, 0, 1],
]
Output Array after permutation: [
  [0, 0, 0, 0],
  [0, 1, 0, 1],
  [0, 0, 0, 0],
  [0, 1, 0, 1],
]
Output binary value: 0000,0101,0000,0101
Result: 0.0263671875

```

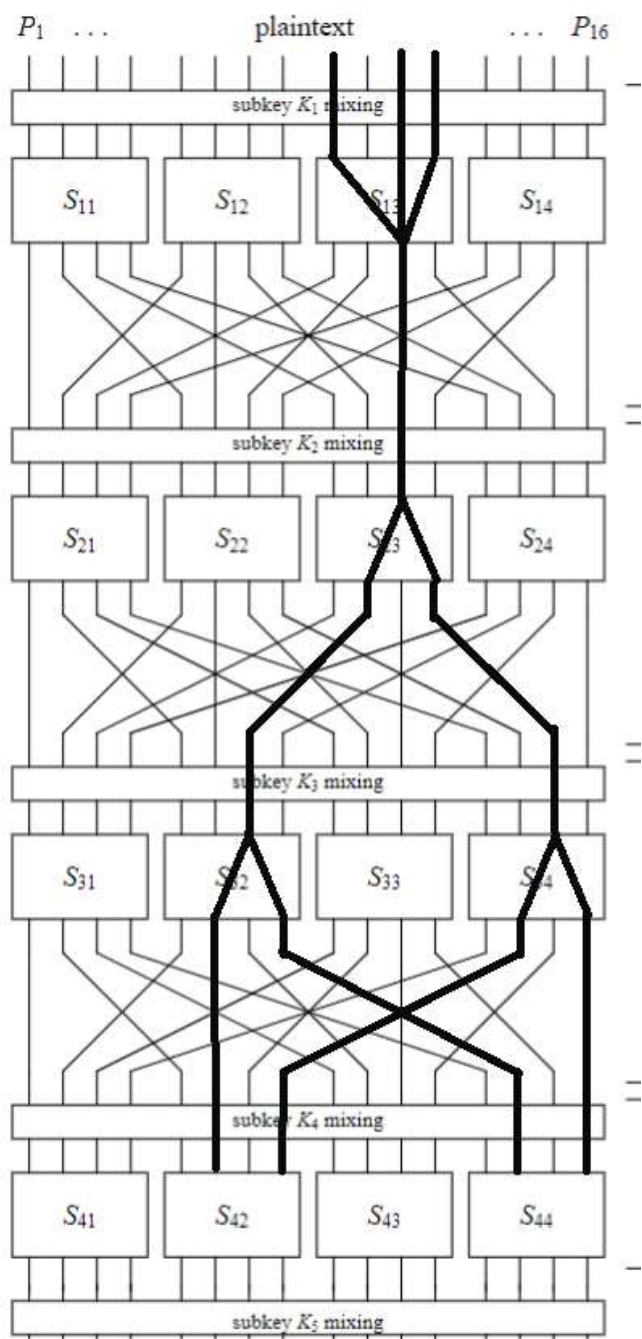


Figure 2: Differential Characteristic and Probability with new Sbox with 0000,0000,1011,0000

```

Input binary value: 0000000010110000
Sbox0 will go from 0 to 0
Sbox1 will go from 0 to 0
Sbox2 will go from 11 to 1
Sbox3 will go from 0 to 0
Input value: 0000,0000,0001,0000
Input Array: [
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 0, 1],
  [0, 0, 0, 0],
]
Output Array after permutation: [
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 1, 0],
]
Sbox0 will go from 0 to 0
Sbox1 will go from 0 to 0
Sbox2 will go from 0 to 0
Sbox3 will go from 2 to 3
Input value: 0000,0000,0000,0011
Input Array: [
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 1, 1],
]
Output Array after permutation: [
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [0, 0, 0, 1],
  [0, 0, 0, 1],
]
Sbox0 will go from 0 to 0
Sbox1 will go from 0 to 0
Sbox2 will go from 1 to 12
Sbox3 will go from 1 to 12
Input value: 0000,0000,1100,1100
Input Array: [
  [0, 0, 0, 0],
  [0, 0, 0, 0],
  [1, 1, 0, 0],
  [1, 1, 0, 0],
]
Output Array after permutation: [
  [0, 0, 1, 1],
  [0, 0, 1, 1],
  [0, 0, 0, 0],
  [0, 0, 0, 0],
]
Output binary value: 0011,0011,0000,0000
Result: 0.015625

```

