Politechnika Poznańska

Cryptography and Cryptanalysis
Report

Quality assessment of an S-box, Nonlinearity and SAC

Student:
Thibault VILLEPREUX

Teacher :
Anna Grocholewska-Czuryło

March 3, 2024

Contents

List of Figures

1 Introduction

The code and analysis presented in this report are available here, with the code hosted on GitHub. The code is written in Python and has been tested to compile successfully under Python 3.10.12.

2 Balanced Functions

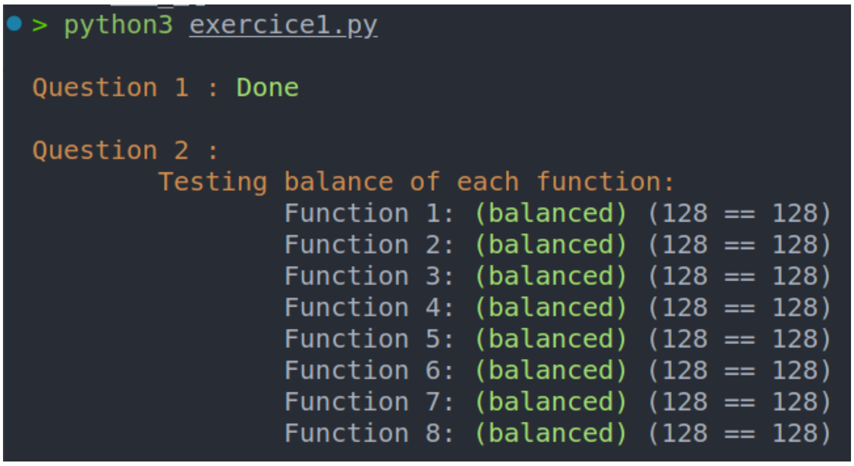
2.1 Reminder: Definition

A Boolean function f is considered balanced if its truth table has 2^{n-1} zeroes (or ones), where n is the number of variables or parameters the function takes. Mathematically, this is represented as $N_0[y_0, y_1, ..., y_{2^n-1}] = N_1[y_0, y_1, ..., y_{2^n-1}]$.

2.2 Functions in the S-box

In our case, the S-box file contains 256 data points. Therefore, if the Boolean functions derived from these data of the S-box are balanced, we would expect to have an equal distribution of zeros and ones in their truth tables. With 256 data points, this translates to 128 zeros and 128 ones.

This is indeed the case, as shown in Figure 1, which is the result of the compiled code (available at the GitHub link provided in the introduction) that tests if the functions are balanced.



```
● > python3 exercice1.py

  Question 1 : Done

  Question 2 :
          Testing balance of each function:
                  Function 1: (balanced) (128 == 128)
                  Function 2: (balanced) (128 == 128)
                  Function 3: (balanced) (128 == 128)
                  Function 4: (balanced) (128 == 128)
                  Function 5: (balanced) (128 == 128)
                  Function 6: (balanced) (128 == 128)
                  Function 7: (balanced) (128 == 128)
                  Function 8: (balanced) (128 == 128)
```

Figure 1: Results of the balanced functions

2.3 Importance of Balanced Functions in S-boxes

Balanced functions in S-boxes, as discussed in class and stipulated by the Common Cryptographic Criteria (CCC), play a crucial role in maintaining the security of cryptographic algorithms. They ensure that each bit in the output has an equal chance of appearing, preventing any biases that could provide attackers with information to exploit during cryptographic attacks.

3 Nonlinearity Analysis

3.1 Reminder: Definitions

3.1.1 Affine functions

Affine functions are Boolean functions that can be expressed in the form:

$$f(x_1, x_2, ..., x_n) = a_0 \oplus (a_1 \cdot x_1) \oplus (a_2 \cdot x_2) \oplus ... \oplus (a_n \cdot x_n)$$

where $a_0, a_1, ..., a_n$ are the coefficients of the function, and $x_1, x_2, ..., x_n$ are the input variables.

3.1.2 Nonlinearity

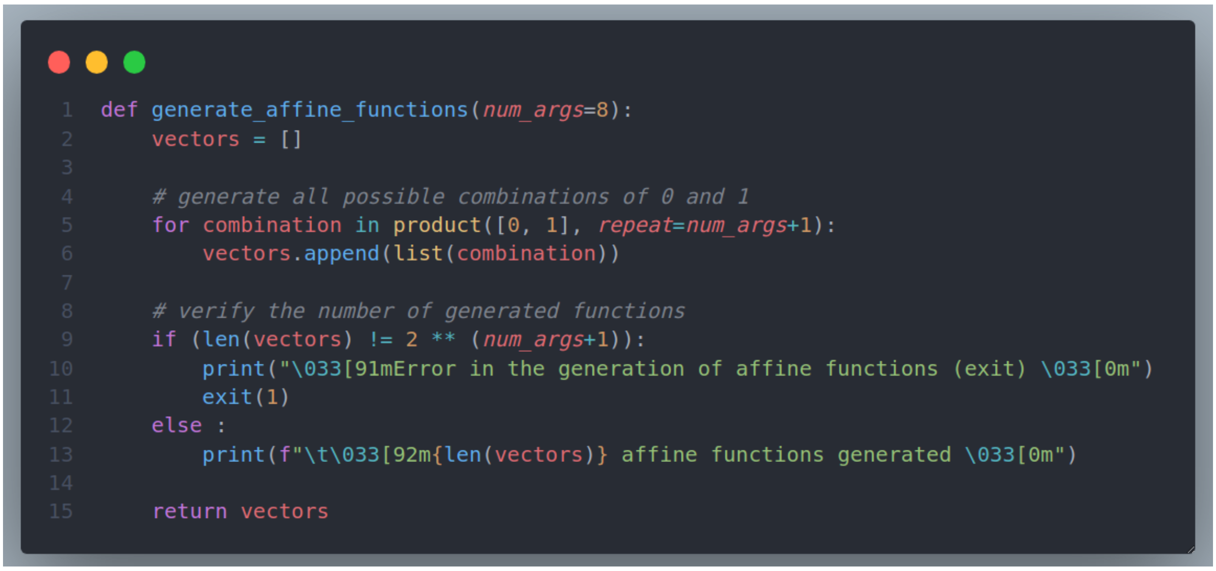
The nonlinearity of a Boolean function is defined as the Hamming distance between the function and the set of all affine functions.

The nonlinearity function is bounded by $2^{n-1} - 2^{n/2-1}$, where n is the number of variables in the function.

3.2 Generation of All Affine Functions

Considering that we are dealing with Boolean affine functions, where a_0 to a_8 belong to the set $\{0, 1\}$, the size of the set of possible affine functions is $2^9 = 512$.

Thus, it is sufficient to generate all possible 9-bit vectors, as demonstrated in the code below:



```python
def generate_affine_functions(num_args=8):
    vectors = []

    # generate all possible combinations of 0 and 1
    for combination in product([0, 1], repeat=num_args+1):
        vectors.append(list(combination))

    # verify the number of generated functions
    if (len(vectors) != 2 ** (num_args+1)):
        print("\033[91mError in the generation of affine functions (exit) \033[0m")
        exit(1)
    else :
        print(f"\t\033[92m{len(vectors)} affine functions generated \033[0m")

    return vectors
```

Figure 2: Code snippet for generating affine functions

3.3 Calculation of Nonlinearity

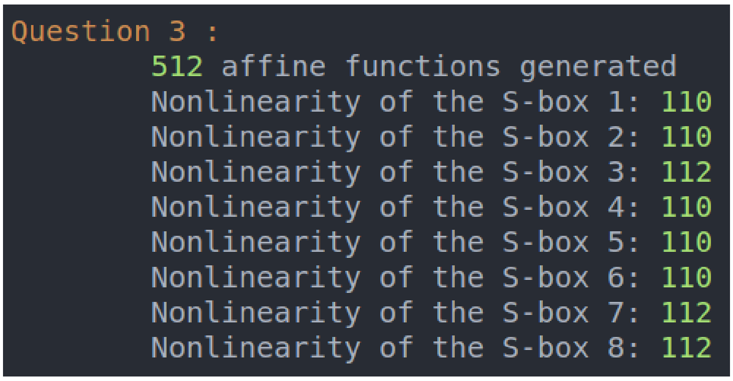
As it is not explicitly requested, and for the sake of simplicity, I will not detail the code but only the main steps that allowed me to obtain a result.

Once all the affine functions are generated, it is necessary to compute their truth tables. There are thus 512 functions to evaluate 256 times (since we vary x_1 to x_8, i.e., 2^8).

Then, we retrieve the truth table of the 8 functions from the S-box. For example, for the first function, for the parameter number 0, we look at the last bit of the first byte in the file. If it ends with 1, we note 1 in its truth table; otherwise, we note 0. We repeat this process for the 256 integers in the file and the 8 functions.

Next, for each of the 8 functions from the S-box, we calculate the Hamming distance with all the affine functions, and we retrieve the minimum.

In the end, we obtain the following result:



```
Question 3 :
        512 affine functions generated
        Nonlinearity of the S-box 1: 110
        Nonlinearity of the S-box 2: 110
        Nonlinearity of the S-box 3: 112
        Nonlinearity of the S-box 4: 110
        Nonlinearity of the S-box 5: 110
        Nonlinearity of the S-box 6: 110
        Nonlinearity of the S-box 7: 112
        Nonlinearity of the S-box 8: 112
```

Figure 3: Results of the nonlinearity functions

This bound aligns with the expected nonlinearity range for Boolean functions, which is from 0 to $2^{n-1} - 2^{n/2-1}$ for functions with n variables. In our case, with $n = 8$, this translates to a maximum nonlinearity of 120.

4 Strict Avalanche Criterion (SAC) Evaluation

4.1 Reminder: Definition

Strict Avalanche Criterion or SAC. Informally, an S-box satisfies SAC if a single bit change on the input results in changes on a half of the output bits.

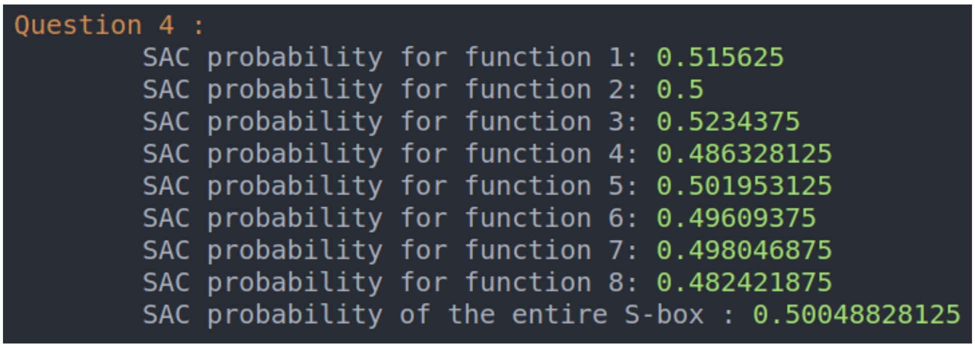
4.2 Calculation of SAC

I will briefly summarize the operation of my code for calculating the SAC.

I start by creating the set of all possible pairs that have a Hamming distance of one bit (i.e., 7 bits in common and 1 change). To achieve this, I use a Python dictionary data structure. For each number (key), I associate it with an array of values representing the set of numbers at a Hamming distance of 1 bit. There are thus 256 keys in the dictionary, each having 8 values, resulting in a total of 2048 pairs. I did not optimize the pairs; they are redundant, but this is accounted for in the subsequent calculations, affecting only the program's complexity and not the SAC probability.

Once all pairs are generated, I iterate over each function:
If the truth values differ for the key and its values, it implies a change in the output, thereby increasing the SAC probability.

I perform this process for all functions, and here are the results:



```
Question 4 :
        SAC probability for function 1: 0.515625
        SAC probability for function 2: 0.5
        SAC probability for function 3: 0.5234375
        SAC probability for function 4: 0.486328125
        SAC probability for function 5: 0.501953125
        SAC probability for function 6: 0.49609375
        SAC probability for function 7: 0.498046875
        SAC probability for function 8: 0.482421875
        SAC probability of the entire S-box : 0.50048828125
```

Figure 4: Results of the SAC functions

To determine the overall probability of change in the output for the entire block, I calculated the average SAC probability across all 8 functions of the S-box.

From this analysis, it can be inferred that the SAC is satisfied for each individual function, with a small margin of approximation, and it is also upheld for the entire block.

5 Summary

As discussed in class, designing the S-box is often the most critical aspect of cryptographic system design. There exist several common cryptographic criteria for evaluating S-boxes, including high nonlinearity, low XOR profile value, complex algebraic description, balancedness, satisfaction of the SAC, absence of affine equivalence, and absence of cycles.

In the proposed exercise, we focused on determining whether our S-box satisfied the criteria of being balanced, having high nonlinearity, and meeting the SAC. Our analysis revealed that the S-box was perfectly balanced, exhibited high nonlinearity, although not perfect, and satisfied the SAC.

However, further analysis of our S-box could be performed by evaluating the other criteria to ensure the robustness of the S-box.