Padua (PD)
7th April 2025

Università degli Studi di Padova

INFORMATION SECURITY LAB.1
A.Y 2024/2025

Lab 1 - Information Security

Cyberducks

Andrea Andreozzi (2163406)
Riccardo Scalco (2155352)
Sergio Cibecchini (2155353)
Luca Ferrari (2166294)

---

1 Task 1 - Simplified AES-like cipher: Encryption

Task 1 required the implementation of a simplified AES cipher based on the (S, T, L) iterated ciphers. First we created the `subkey_generation(k)` function, that creates the various subkeys from the key $\mathcal{K}$ according to the matrix detailed in the instructions. Then we created the various implementations for the required operations in the encryption algorithm:

- `subkey_sum(w,ki)` for the implementation of $v_i = w_{i-1} + [k_i, k_i] \mod p$
- `substitution(v)` which applied a substitution by multiplying for 2 inside the modular space
- `transposition(y)` which applied the matrix transposition
- `linear(z)` which transforms z_i into a 2×4 matrix and multiplies by the provided matrix.

We then repeat the block succession four times, and in the last one, we don't apply the linear block.

2 Task 2 - Simplified AES-like cipher: Decryption

To approach the decryption, we simply had to create, as suggested in the instructions, the inverse of each block that we created in task 1 and apply them in reverse order. In the implementation of `decryption(x, k)`, we invoked respectively:

- `z = inverse_linear(w)`
- `y = inverse_transposition(z)`
- `v = inverse_substitution(y)`
- `w = inverse_subkey_sum(v,subkey[i])`

During the implementation, we paid close attention to the fact that we were working in a positive modular space.

3 Task 3 - Identify the cipher vulnerability

Given that the original cipher created in Task 1 is a linear cipher, we know that there exist two matrices A and B s.t this condition is true: $x = E(k, u) = Ak + Bu \mod p$. To find these matrices, we gave as input the standard orthonormal basis first for the key and later for the plaintext, while setting the other parameter to zero. Meaning that:

- Input $k = e_j, u = 0$ allows us to find the $j - th$ column of A
- Input $k = 0, u = e_j$ allows us to find the $j - th$ column of B

By iterating this method, we were able to find:

$$A = \begin{bmatrix} 9 & 0 & 1 & 6 & 0 & 0 & 1 & 10 \\ 0 & 8 & 6 & 2 & 2 & 9 & 0 & 0 \\ 0 & 6 & 0 & 8 & 3 & 10 & 0 & 0 \\ 6 & 0 & 0 & 8 & 0 & 1 & 6 & 6 \\ 2 & 0 & 1 & 10 & 0 & 0 & 1 & 3 \\ 0 & 1 & 8 & 4 & 9 & 6 & 0 & 0 \\ 0 & 10 & 0 & 5 & 7 & 6 & 0 & 0 \\ 3 & 0 & 0 & 1 & 0 & 1 & 4 & 8 \end{bmatrix} \quad B = \begin{bmatrix} 6 & 0 & 0 & 3 & 3 & 0 & 0 & 0 \\ 0 & 6 & 3 & 0 & 0 & 3 & 0 & 0 \\ 0 & 3 & 6 & 0 & 0 & 0 & 3 & 0 \\ 3 & 0 & 0 & 6 & 0 & 0 & 0 & 3 \\ 5 & 0 & 0 & 0 & 4 & 0 & 0 & 8 \\ 0 & 5 & 0 & 0 & 0 & 4 & 8 & 0 \\ 0 & 0 & 5 & 0 & 0 & 8 & 4 & 0 \\ 0 & 0 & 0 & 5 & 8 & 0 & 0 & 4 \end{bmatrix}$$

To verify the found matrices we created a function `validate_matrixs(A, B, u, k)` that compares `encryption(u, k)` with `(A @ k + B @ u) % p` and verifies that they are equal.

4 Task 4 - linear cryptanalysis KPA

Given that in Task 3 we were able to find the correct A and B that approximate our cryptosystem, we now will be trying to find the key k give some known plaintexts and ciphertexts pairs. Specifically, given u and x we applied:

$$k = A^{-1}(x - Bu) \mod p$$

We implemented a `recover_key(u, x, A, B)` function that performed the operation described in the formula above.
The recovered key is:

$$k = [10, 4, 10, 2, 6, 7, 8, 2]$$

We then checked if the recovered key we found was correct by using the function `validateKey(plaintexts, cipherte` that compared the known ciphertext x with the encryption with the found key k of the known plaintext u.
Thus for example, give the known pair:

$$u = [5, 6, 2, 9, 6, 10, 8, 4] \quad x = [6, 1, 6, 4, 2, 3, 3, 9]$$

We verified that:

```
x == encryption(u, k)
```

5 Task 5 - "Nearly linear" simplified AES-like cipher

In this task we were asked to refactor our previous AES-like cipher from task 1 by editing the previous substitution block to behave according to the given table. To achieve this, we defined a `substitution_map` as follows:

```
substitution_map = {0: 0, 1: 2, 2: 4, 3: 8, 4: 6, 5: 10, 6: 1, 7: 3, 8: 5, 9: 7, 10: 9}
```

After having defined the map, we simply applied the substitution to each element of the array v:

```
v[i] = substitution_map[v[i]]
```

We verified that the new cipher behaved as expected by checking with the provided plaintext - ciphertext pair with the function `test(x, x_test)`.

6 Task 6 - Linear cryptanalysis of a "nearly linear" cipher

In this task, we were asked to find the key of a "nearly linear" cipher given some pairs of plaintext - ciphertext. So we will analyze this formula:

$$k = A^{-1}(-Cx - Bu) \mod p$$

Since the cipher is nearly linear, we have chosen to use the same matrices A and B found in the previous point, and C as the identity matrix. Since we have five pairs of plaintext - ciphertext, we are able to find five different values of the key. We have chosen to use the average of these keys as the best approximation of the key. We found:

`k' = [5, 4, 4, 8, 2, 6, 3, 4]`

We check the probability in this way:

- we generate a random plaintext
- we compute the ciphertext starting from the plaintext and `k'`
- then we compute `Ak+Bu+Cx` and we compare it to zero

We repeat the step above for 100k times and count how many times it is equal to zero. Then we compute the probability in this way:

$$P = \frac{\text{Number of success}}{\text{Total test}}$$

and finally we compare P with $\frac{1}{p^{l_x}} = \frac{1}{11^8} \approx 4.66 \times 10^{-9}$. We obtain $P = 0.00039$ that is $P \gg \frac{1}{p^{l_x}}$. Then we start a brute force cycle from it to find the real key. First, we explore the sixteen keys with a distance of one from `k'`. Then we increment the distance until we find the real key. We found (with distance thirteen):

`k = [5, 10, 4, 9, 2, 3, 3, 8]`

The brute force tests about seven million keys and it requires 15-20 minutes in Google Colab. It is a lot less expensive than testing all the possible keys ($\sim$ 215 million keys and more than 6 hours of execution).

7 Task 7 - Non linear simplified AES-like cipher

In this task, we want to implement a non-linear cipher. We take the base function of Task 1, and update the `subkey_generation()` and the `substitution()` functions as described in the instructions. We need an inverse of a vector in module `p`, so we choose to implement it in this way:

- if gcd between the number and `p` is one then we can calculate the inverse
- otherwise we put the inverse equal to zero

Then we implement the encryptor and the decryptor as in Task 1 (but with the overloaded methods). Finally, we check our results.

8 Task 8 - Meet in the middle attack

In the final task, we have to implement the "Meet in the middle attack". To find the two keys, we create two dictionaries (both with about 50000 generated random guesses). Then we check a match between the two dictionaries and finally, we find:

`k' = [7, 7, 3, 8]` and `k'' = [0, 0, 10, 1]`

We check the keys with all the pairs given in the file. Sometimes, since the keys are randomly generated, it may not find the correct key. To resolve this problem, it is enough to re-run the script or increase the number of guesses.