Information Security A.Y. 2024/2025

Lab 1: Implementation and linear cryptanalysis of a simplified AES-like cipher

Abdalaziz Zainelabedeen Abdalaziz Abdo (2141828)
Enkeleda Bardhi (2070874) Laura M. Schulze (2122311)
Aitegin Zhamgyrchieva (2144165)

May 26, 2025

In the following report we provide a brief description of how Tasks 1-8 were implemented, explaining the choices we made in terms of algorithms, matrices, and any other critical implementation decisions. Furthermore, we list our results and guesses for the cryptanalyses we conducted.

Task 1 and 2: Encryption and Decryption

We implemented the three core primitives: substitution, transposition, and linear transformation, along with their inverses for decryption.

Substitution:

$$\texttt{substitution}(v) = (2 \cdot v) \bmod 11, \quad \texttt{substitutionInv}(v) = (6 \cdot v) \bmod 11$$

Transposition: reverses the second half of an 8-element vector. It is its own inverse:

$$\texttt{transposition}(y) = y[:4] \parallel y[4:][::-1]$$

Linear Transformation:

$$\texttt{linear}(z) = (A \cdot \text{reshape}(z, 2 \times 4)) \bmod 11, \quad \texttt{linearInv}(w) = (A^{-1} \cdot W) \bmod 11$$

We verified the encryption of `u = [1, 0, ..., 0]` using the key `k = [1, 0, ..., 0]`, which produced the ciphertext `x = [4, 0, 0, 9, 7, 0, 0, 3]`. Decryption was also tested with multiple random plaintext-key pairs and the original messages were correctly recovered.

Task 3 – Linear Approximation of the Cipher

We approximated the cipher linearly using the equation $x = A \cdot k + B \cdot u \mod p$. To obtain matrices A and B, we used a black-box approach: we fixed $u = 0$ and varied $k = e_j$ to get columns of A, and fixed $k = 0$ and varied $u = e_j$ to get columns of B. The cipher function `encryptA()` was used. This allowed us to reconstruct the linear behavior of the cipher for each input position.

Recovered Matrices:

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

These matrices were later used for key recovery in Task 4 and linear approximations in Task 6.

Task 4 – Key Recovery Using Linear Approximation

Using the equation from Task 3 $x = A \cdot k + B \cdot u \mod p$, we isolated k using one known pair (u, x): $k = A^{-1} \cdot (x - B \cdot u) \mod p$.

Recovered Key Guess $\hat{k}$

$$\hat{k} = [7, 2, 3, 2, 4, 6, 6, 0]$$

This key was tested on all available (u_i, x_i) pairs using the same equation $x_i = A \cdot \hat{k} + B \cdot u_i \mod p$, and in every case the computed ciphertext matched the actual ciphertext. **Conclusion:** The key $\hat{k}$ was successfully recovered and verified.

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

In Task 5, a variation of the original cipher using a different substitution function is implemented. The new substitution function is described by table **??**, and is simply implemented using a `NumPy array`. Besides the new substitution function, the encryptor is implemented using the same structure and the same functions for subkey generation, transposition, and linear transformation as in Task 1.

The correctness of the implementation is verified with the given test message and key. Furthermore, the corresponding decryptor is implemented and again tested with 100 random test messages and keys. The inverse of the substitution can simply be implemented with the table as well.

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

We attack the nearly linear cipher by approximating it as

$$A\,k + B\,u + C\,x \equiv 0 \pmod{11},$$

with $C = I$ and using the matrices A and B from Task 3. For each known plaintext/ciphertext pair (u, x) we computed an approximate key

$$\hat{k} = A^{-1}(x - B\,u) \pmod{11}.$$

Since the nonlinearity introduces noise, these estimates vary; we aggregate them by taking the mode for each coordinate, yielding The approximate keys obtained were:

$$\hat{k}_1 \approx [1\,8\,6\,7\,1\,9\,7\,4], \quad \hat{k}_2 \approx [6\,10\,8\,6\,7\,1\,1\,1], \quad \hat{k}_3 \approx [7\,8\,6\,4\,8\,0\,2\,0],$$

$$\hat{k}_4 \approx [1\,0\,1\,6\,10\,0\,0\,5], \quad \hat{k}_5 \approx [3\,10\,10\,8\,0\,10\,7\,2].$$

Aggregating by taking the mode for each coordinate gives

$$\hat{k}_{\text{agg}} = [1\,8\,6\,6\,1\,0\,7\,4].$$

We then performed an exhaustive local search in the Hamming neighborhood (up to distance 4). The best candidate found was

$$\hat{k} = [4\,1\,3\,5\,1\,0\,7\,4],$$

Task 7

In task 7, a third variation of the AES-like cipher is implemented. It uses a lower key length of $l_k = 4$, and thus a different subkey generation pattern. Furthermore, the substitution function is changed to

$$y_i(j) = 2v_i(j)^{-1} \mod p, \quad j \in \{1, \ldots, 8\}$$

where $v_i(j)^{-1}$ is the multiplicative inverse of $v_i(j)$ on the field $\mathbb{F} = \text{GF}(p)$. For the sake of convenience, the multiplicative inverse on GF(11) is precomputed and implemented as a table. The encryptor is again verified with the given test message and key. A corresponding decryptor is implemented as well, and tested with random vectors similarly to the previous implementations.

Task 8

In Task 8, we implemented a *meet-in-the-middle* attack to recover the two 4-digit keys (k', k'') used in a double encryption scheme. The idea is to match intermediate values by encrypting plaintexts with k' and decrypting ciphertexts with k'', both padded to 8 digits.

We generated up to 80,000 random key candidates for k' and k''. For each k', we encrypted all known plaintexts and stored the outputs. Then, for each k'', we decrypted the ciphertexts and searched for a matching intermediate result. When a match was found, the corresponding (k', k'') was returned.

Our implementation correctly recovered the key pair using 5 known plaintext-ciphertext pairs and the functions `encryptC` and `decryptC`.

Recovered Keys Guess k', k''

$$k' : [0, 3, 10, 10] \quad k'' : [1, 9, 8, 0]$$