Hardware Security

Implementation of Differential Power Attack (DPA)

Fereshteh Baradaran

8th Azar, 1402

Abstract

This report presents an implementation of the Correlation Power Analysis (CPA) attack using Python. CPA is a side-channel attack method used in cryptanalysis to extract secret keys from cryptographic devices. By analyzing power consumption traces and correlating them with hypothetical data, the CPA attack aims to reveal the secret encryption key. This report details the methodology of the Python implementation, and the results obtained.

Introduction

Correlation Power Analysis (CPA) is a form of side-channel attack that exploits variations in power consumption during cryptographic operations to extract secret keys. Unlike brute-force attacks, CPA attacks leverage physical leakages from devices, making them more efficient and practical against vulnerable cryptographic systems. CPA attacks are based on the principle that the power consumption of a cryptographic device correlates with the data it processes. By observing the power consumption during encryption and correlating it with hypothesized operations (like S-Box outputs). The Hamming weight model, which associates power consumption with the number of binary ones in the data, is commonly used in CPA.

Implementation

Software and Tools Used:

- **Python** programming language
- **NumPy** library for numerical computations
- **Struct** and **Matplotlib** libraries for data processing and visualization

The complete source code of this assignment is available in a GitHub repository that can be accessed at the following link.

- helperFunctions.py:

  - calculateHammingWeight(number):

    Purpose: Computes the Hamming weight, an important metric in side-channel analysis, indicating the number of '1' bits in the binary representation of a number.

    Implementation: Converts the input number into its binary form and counts the occurrence of '1' bits.

  - calculateCorrelationTrace(vector, matrix):

    Purpose: Calculates the correlation coefficient between a given vector (representing a hypothesis) and each row of a matrix (representing power consumption data).

    Implementation: Uses NumPy to compute correlation coefficients. This is essential to identifying which key hypothesis aligns best with the observed power consumption data.

- calculateSboxOutput.py:

  - S-Box Array:

    Purpose: The S-Box (Substitution box) is a core component in many symmetric key algorithms like AES (Advanced Encryption Standard), used for introducing non-linearity in cryptographic operations.

    Implementation: Contains a predefined array representing the S-Box. This array is utilized to simulate the substitution step of the cryptographic algorithm under analysis.

  - calculateSboxOutpu(plainText, key):

    Purpose: Compute the output of the cryptographic S-Box given a specific input.

    Implementation: Returns $sbox(plainText[i] \oplus key[i])$

- getInput.py:

  - loadTrace(fileName, numberOfTraces, traceSize):

    Purpose: Loads power consumption traces from a binary file.

    Implementation: Reads the binary file and extracts the specified number of traces, each of a defined size. It processes the raw binary data into a list of byte values.

  - loadData(fileName):

    Purpose: Loads additional cryptographic data, which might include plaintexts or ciphertexts.

    Implementation: Parses a file containing cryptographic data and converting the contents into a 2D array of integers.

- Main.py:

  Initiates data loading for power traces and cryptographic data.

  Iterates over potential key values, using other modules to calculate expected outputs and correlations.

  Determines the most likely key by evaluating the correlation coefficients.

Results:

The implementation successfully executed the CPA attack, revealing the secret key.

Byte	Original Key	Obtained Key	Computation Time(hour)
0	0x00	0x00	1.85
1	0x11	0x11	1.90
2	0x22	0x22	1.88
3	0x33	0x33	1.87
4	0x44	0x44	1.85
5	0x55	0x55	1.88
6	0x66	0x66	1.90
7	0x77	0x77	1.91
8	0x88	0x88	1.89
9	0x99	0x99	1.90
10	0xaa	0xaa	1.88
11	0xbb	0xbb	1.86
12	0xcc	0xcc	1.89
13	0xdd	0xdd	1.86
14	0xee	0xee	1.93
15	0xff	0xff	1.93
Total Computation Time(hour)			**30.18**

Conclusion

In this assignment, we successfully implemented and demonstrated a Correlation Power Analysis (CPA) attack using Python, highlighting its effectiveness in extracting secret keys from cryptographic devices by exploiting power consumption variations.