Assignment 2: Implementation

Secure Software and Hardware Systems

March 24, 2023

Introduction

The goal of this assignment is to implement, test, and benchmark (optimized) cryptographic algorithms on the Pi Pico Microcontroller (μC). For each of the following tasks, you will be supplied with a template. Your code should always be put into the file `crypto.c` only. There should be no need to change any other file. You also need to supply a brief report with the submission (see below for exact tasks, length approx. 1–2 pages).

This assignment contributes 25% to your final grade.

The grade for each task is determined by these main factors: correctness of code, efficiency of code (where applicable), code readability and comments, creativity and excellence, and quality of report.

Comment your code!

Files to be submitted

Please submit your solution via Canvas in a `.zip` archive named

`groupXX-assignment2.zip`

This archive should include exactly the following files and folders (ensure names and folders are exactly like this) and nothing more:

report.pdf Your report / solutions for the non-programming questions (see below)

present_ref/crypto.c Program code for Task 1.1

present_bs/crypto.c Program code for Task 1.2

1 PRESENT implementation and optimisation (100 pts)

This task deals with the efficient implementation (with respect to speed) of the block cipher PRESENT. You have received templates for each task.

1.1 Reference Implementation (25 pts)

Write a reference implementation of PRESENT by implementing `add_round_key()`, `sbox_layer()`, and `pbox_layer()` in the provided template. Test the correctness of your implementation on the Pi Pico by executing `test_against_testvectors.py` after flashing.

Report:

1. Give the average number of cycles per PRESENT execution and the throughput in cycles per bit.
2. Describe any special techniques or optimizations you used (if applicable).

1.2 Bitslicing Implementation (75 pts)

Write a bitsliced implementation of PRESENT as discussed in the lecture. Use the supplied template, performing 32 PRESENT executions in parallel. The same key shall be used for all 32 parallel executions, i.e., the key schedule does not have to be bitsliced (but you are not forbidden to optimize/bitslice that part as well). The template already provides appropriate code for the PC communication and includes the key schedule. Your task consists of:

1. Implement the functions `enslice()` and `unslice()` to bring a "normal" array of 16 plaintexts into bitsliced representation.
2. Implement the bitsliced PRESENT in `crypto_func()`.
3. Further optimize the code for minimal runtime.

As a starting point, Boolean expressions for the S-Box (based on `https://eprint.iacr.org/2012/587.pdf`) are given below. The expressions are in Algebraic Normal Form (ANF), i.e., + represents `XOR`, · is `AND`, and +1 is `NOT`. You are free to further minimize these expressions (also using `OR`, `AND`).

$$
\begin{aligned}
y_0 &= x_0 + x_1 \cdot x_2 + x_2 + x_3 \\
y_1 &= x_0 \cdot x_2 \cdot x_1 + x_0 \cdot x_3 \cdot x_1 + x_3 \cdot x_1 + x_1 + x_0 \cdot x_2 \cdot x_3 + x_2 \cdot x_3 + x_3 \\
y_2 &= x_0 \cdot x_1 + x_0 \cdot x_3 \cdot x_1 + x_3 \cdot x_1 + x_2 + x_0 \cdot x_3 + x_0 \cdot x_2 \cdot x_3 + x_3 + 1 \\
y_3 &= x_1 \cdot x_2 \cdot x_0 + x_1 \cdot x_3 \cdot x_0 + x_2 \cdot x_3 \cdot x_0 + x_0 + x_1 + x_1 \cdot x_2 + x_3 + 1
\end{aligned}
$$

To give a few ideas of other things that can be optimised:

- Unroll loops,
- Trade off increased memory footprint for speed,
- Minimize the boolean expressions for the S-Box,
- Optimize the enslice and unslice operations,
- Represent and update the key in ensliced form,
- Write parts of the code in (inline) assembly (if you know what you are doing),
- ...

Test the correctness of your implementation on the Pi Pico by executing `test_against_testvectors.py` after flashing.

Report:

1. Describe any optimizations you made (including changes to the Boolean expression for the S-Box) and their effect on the performance (before/after comparison for each optimization step).
2. Give the average number of cycles per optimized PRESENT execution (keep in mind that 32 instances are computed in parallel) and the throughput in cycles per bit of the final version.