Assignment 1: Implementation

Hardware and Embedded Systems Security

January 5, 2018

Introduction

The goal of this assignment is to implement, test, and benchmark (optimized) cryptographic algorithms on the MSP430 Microcontroller (μC). For each of the following tasks, you will be supplied with a template. Your code should always be put into the respective files `crypto.c` and `crypto.h` only. If you need to change `main.c` or `Makefile`, please add these files to the respective folder and mention this with a brief explanation in the report supplied with the submission (see below for exact tasks, length approx. 2–3 pages).

This assignment contributes 25% to your final grade.

The grade for each task is determined by: correctness of code, efficiency of code, code readability and comments, creativity and excellence, and quality of report. The pen-and-paper problems are graded for correctness and completeness (i.e. all intermediate values are given as required).

Comment your code!

Files to be submitted

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

`groupXX-assignment1.zip`

This archive should include the following files and folders:

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

present_ref/crypto.{c,h} Program code for Task 1.1

present_opt/crypto.{c,h} Program code for Task 1.2

present_bs/crypto.{c,h} Program code for Task 1.3

longnum_add/crypto.{c,h} Program code for Task 2

1 PRESENT (70%)

This task deals with the efficient implementation (mostly with respect to speed) of the block cipher PRESENT. You will receive three templates, one for the reference, one for the optimized, and one for the bitslicing implementation.

1.1 Reference Implementation (15%)

Write a working reference implementation of PRESENT by implementing the functions `add_round_key()`, `sbox_layer()`, and `pbox_layer()` in the provided template for present. Test the correctness of your implementation on the MSP430 by running `make program` and `make test` (or by executing `test_against_testvectors.py`).

Report: Give the average number of cycles per PRESENT execution and the throughput in cycles per bit.

1.2 Optimized Implementation (25%)

Write an optimized implementation of PRESENT by copying your code from Part 1.1 and applying the software optimzations described in the lecture. *Do not use bitslicing.* Test the correctness of your implementation on the MSP430 by running `make program` and `make test` (or by executing `test_against_testvectors.py`).

Report: Briefly describe which optimizations were used and their effect on the performance (before/after comparison). Give the average number of cycles per optimized PRESENT execution and the throughput in cycles per bit.

1.3 Bitslicing Implementation (30%)

Write a bitsliced implementation of PRESENT as discussed in the lecture. Use the supplied template, performing 16 PRESENT executions in parallel. The same key shall be used for all 16 parallel executions, i.e., the key schedule does *not* have to be bitsliced. 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[1] are given below. The expressions are in Algebraic Normal Form (ANF), i.e., + represents `XOR`, $\cdot$ is `AND`, and +1 is `NOT`. You are free to further minimize these expressions (also using `OR`, `AND`).

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

Test the correctness of your implementation on the MSP430 by running `make program` and `make test` (or by executing `test_against_testvectors.py`).

Report: Describe which optimizations you made (including changes to the Boolean expression for the S-Box) and their effect on the performance (before/after comparison). Give the average number of cycles per optimized PRESENT execution (keep in mind that 16 instances are computed in parallel) and the throughput in cycles per bit.

[1] based on `https://eprint.iacr.org/2012/587.pdf`

2 Long Numbers (15%)

This task deals with part of the implementation of long number arithmetic. You will receive one template. The template is defined for 1024-bit numbers (i.e., 128 16-bit words), however, your code should also work for other number sizes. The template already provides appropriate code for the PC communication and testing, similar to the one for PRESENT.

Implement the addition $a + b$ of two long numbers. Use the supplied template and test the correctness of your implementation.

Report: Give the average number of cycles per execution.

3 Pen-and-Paper Problems (15%)

Please include the solution to the following problems in the report. Note that similar questions may be asked in the exam, hence, avoid using any tool apart from pen & paper and a normal calculator.

3.1 ANF (5%)

Compute the ANF of the 3-to-1 Boolean function defined by the following table. Use the algorithm from the lecture. Please provide all intermediate results. It is sufficient to solve this task on paper and provide a *legible* scan/picture in the report.

x_0	x_1	x_2	$f(\cdot)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Table 1: Truth table of Boolean function

Simplify the resulting ANF by factoring the expression and/or using other common Boolean operators (e.g. `OR`). The simplified expression should require 7 or less operations.

3.2 Long Number Arithmetic (10%)

This task deals with long number arithmetic. We use the base $b = 10$, i.e., work in the decimal system. Note that a real processor would use bases between 2^8 and 2^64.

3.2.1 Addition

Compute the sum $a + b$ for $a = (12345)_{10}$ and $b = (54321)_{10}$, using the Schoolbook algorithm introduced in the lecture. Give all intermediate values. How many elementary base-b additions are required?

3.2.2 Multiplication

Compute the product $a \cdot b$ for $a = (1234)_{10}$ and $b = (12345)_{10}$, using the Schoolbook algorithm introduced in the lecture. Give all intermediate values. How many elementary base-b multiplications are required?