Assignment 1: Implementation

Hardware and Embedded Systems Security

January 22, 2019

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/size of code (where applicable), 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_bs/crypto.{c,h} Program code for Task 1.2

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

1 PRESENT vs AES (85%)

This task deals with the efficient implementation (with respect to speed) of the block cipher PRESENT, and a comparison to the MSP430's hardware AES module. You will receive templates for each task.

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:

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 (50%)

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 (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 MSP430 by running `make program` and `make test` (or by executing `test_against_testvectors.py`).

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 16 instances are computed in parallel) and the throughput in cycles per bit of the final version.

1.3 Hardware AES (20%)

Write an implementation that uses the built-in hardware AES engine of the MSP430 to encrypt a 16-byte block using a 128-bit key. For a basic solution, you can use the library functions in `driverlib/aes256.h`, but this will give at most 70% of the grade in this task (if well-commented and all below points are addressed in the report). For up to 100%, you have to write completely new code to save the overhead of the driver calls. Test the correctness of your implementation on the MSP430 by running `make program` and `make test` (or by executing `test_against_testvectors.py`).

Report:

1. Give the average number of cycles per full AES encryption of one block and the throughput in cycles per bit.
2. Compare your results to the results from Task 1.1 and Task 1.2.
3. Describe the low-level steps executed by the firmware to control the AES block: provide a table that shows the sequence of register writes/reads, with a short explanation what each read/write does and which bits it affects.
4. Briefly discuss under which circumstances it might make sense to use a software implementation instead of the AES hardware implementation.

2 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. You can solve the tasks on paper and embed a scan/photo in the report, but please make sure it is *legible*.

2.1 ANF (5%)

Compute the ANF of the 3-to-1 Boolean function defined by the following table. Use the algorithm from the lecture. Show all intermediate steps and results.

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.

2.2 Long Number Arithmetic (10%)

This task deals with long number arithmetic. We use the base $b = 10$, i.e., work in the decimal system to simplify your paper calculations. But keep in mind that real processors would use bases between 2^8 and 2^{64} — small microcontrollers are 8-bit architectures, while PC-level processors use 64 bit.

2.2.1 Addition (5%)

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?

2.2.2 Multiplication (5%)

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?