Cryptographic Engineering 2019 – Software Assignment

January 28, 2019

Submission Deadline: Apr. 5, 23:59 (Amsterdam time)
Submission via Brightspace only!
Submission in **groups of two**!

Assignment

Download C reference implementations of

- ChaCha20 (`http://cr.yp.to/papers.html#chacha`),
- Poly1303 (`http://cr.yp.to/papers.html#poly1305`), and
- ECDH key exchange on Curve25519 in Edwards form

from `https://cryptojedi.org/peter/teaching/ce2019/ce2019-sw-assignment.tar.bz2` and optimize all three primitives for the ARM Cortex-M4 microcontroller.

Code requirements

In order to get a passing grade (i.e., ≥ 6) for Part I of the Cryptographic Engineering course, the code you submit must fulfill the following minimal requirements:

1. It must not have any secret-dependent branches or access to memory at secret-dependent locations.
2. The submitted software must offer the same functionality as the C reference implementations, i.e., regression tests must pass.
3. All three primitives must be substantially (i.e., at least 20%) faster than the C reference implementations compiled with `gcc-5.4.1 -O3`, i.e., substantially faster than
   - 44747 cycles for ChaCha20,
   - 164231 cycles for Poly1305, and
   - 47493242 cycles for ECDH `scalarmult_base` and 44308530 cycles for ECDH `scalarmult`.
4. The `crypto_core_chacha20` function must be rewritten in assembly.

Submission requirements

Each submission must consist of two parts:

1. A `tar.bz` archive containing your code. This should be simply the original code package with additional files and modified Makefiles as required for your optimized versions.

2. A `pdf` document briefly describing the optimization techniques you applied for each of the three primitives.

Additionally, please keep the following in mind when submitting:

- Only one student of each group of two should submit; make sure to mention both names and both S-numbers in the submission documents.
- Do not modify the files called `test.c` and `speed.c` from the original code package.

Hints for optimizing the three primitives

Optimizing ChaCha20

The natural way to reimplement the core of ChaCha20 in assembly are the following steps:

- Write `quarterround` function in assembly.
- Merge 4 `quarterround` functions into a full round.
- Implement loop over 20 rounds in assembly.

For further optimization you might want to also write the loop over the message length in assembly.

Optimizing Poly1305

The reference implementation is using a very small radix of 2^8 for arithmetic in $\mathbb{F}_p$ with $p = 2^{130} - 5$. An obvious strategy for optimizing is to rewrite the field arithmetic using a larger radix, for example 2^{26}.

Optimizing ECDH

Before optimizing the ECDH sofware, it makes sense to investigate for possible timing leaks (secret-dependent branches and/or memory addresses) and eliminate those.

There are many ways how the ECDH software can be sped up, for example:

- more efficient scalar multiplication,
- specialized base-point scalar multiplication,
- optimized group arithmetic, and
- optimized field arithmtic (using larger radix).