Assembly or Optimized C for Lightweight Cryptography on RISC-V?

CANS 2020

19th International Conference on Cryptology and Network Security

Fabio Campos[1], Lars Jellema[2], Mauk Lemmen[2], Lars Müller[1], Daan Sprenkels[2], Benoit Viguier[2]

[1]RheinMain University of Applied Sciences, Germany
[2]Radboud University, The Netherlands

Basic idea

- analysis of strategies for optimizing lightweight cryptography
- several (round 2) candidates of NIST's lightweight cryptography standardization project
- on a RISC-V architecture
- assembly or Optimized C?

RISC-V

- a new open reduced instruction set architecture (ISA)
- research project that started at UC Berkely in 2010
- a serious competitor to ARM?
- frozen 32-bit base ISA (RV32I) and 64-bit (RV64I)
- general implementation strategies are derived and discussed

RV32I

- 32 32-bit registers x0–x31 (i.e. a lot)
- Basic three-operand arithmetic, bitwise, basic shift & load/store instructions
- No: rotate instructions, carry flag, nice bit operation instruction
- Compensated by extensions: M, A, F, D, B, V, C, ...

Environment

SiFive HiFive1 Board

- 5-stage single-issue in-order pipelined RV32IMAC E31 CPU
- $<$ 384 MHz, 64 KiB RAM, 16 MiB flash
- 16 KiB instruction cache

riscvOVPsim simulator

- Just-in-Time Code Morphing and execution on Linux host
- enables extensions such as Bit manipulations
- simulates neither pipeline nor cache

Optimized Algorithms

NIST lightweight round 2 candidate

- Gimli (Gimli-Cipher & Gimli-Hash)
- Sparkle (Schwaemm & Esch)
- Saturnin
- Ascon
- Elephant
- Xoodyak

NIST Standards:

- AES
- Keccak

Optimization techniques

- careful scheduling of instructions
- loop unrolling
- renaming variable to avoid move operations
- reduce the number of required instructions in the S-Box
- avoid loading of round constants
- bitslicing (in & out blocks)
- parallelization through bit interleaving
- lane complementing: reduce number of NOT instructions

Optimized C

register = `uint32_t`

- optimized implementation usually written directly in assembly
- considering the small size of the RISC-V ISA
- ensuring no branch on secret data, code mimics assembly instructions
- translation of an assembly implementation back into C
- compiler further optimize and take care of register allocation

Physical vs. Simulators

Algorithm	OVP	SiFive	
Gimli	37596	38530	(+2%)
Schwaemm256-128	20842	72286	(+247%)
Saturnin	55367	152803	(+176%)
Ascon	41228	42562	(+3%)
Delirium	110171	765235	(+595%)
Xoodyak	18852	64869	(+244%)

Table 1: Implementations by Weatherley[1], cycle counts for AEAD mode on the SiFive and riscvOVPsim platform, compiled with Clang-10 `-O3`, for encrypting 128 bytes of message and 128 bytes of associated data.

[1] `https://github.com/rweather/lightweight-crypto`, commit 52c8281

Comparison

Algorithm	Weatherley[2]	our results
Gimli	38530	35853 (−7%)
Schwaemm256-128	72286	43877 (−40%)
Saturnin	152803	59368 (−61%)
Ascon	42562	27271 (−36%)
Delirium	765235	145936 (−81%)
Xoodyak	64869	26246 (−60%)

Table 2: Cycle counts for AEAD mode on SiFive (128 bytes of message and 128 bytes of associated data)

[2] `https://github.com/rweather/lightweight-crypto`, commit 52c8281

Clang-10 vs. GCC

Algorithm	GCC	Clang-10
Schwaemm256-128	42634	43877
Esch256	33331	34664
Saturnin-Hash bs32	30321	28199
Saturnin-Cipher bs32	60817	59368
Saturnin-Cipher bs32x	5210541	68792
Saturnin-Cipher bs32x	138187 (`-Os`)	68792
Delirium	113031	145936
Xoodyak	23238	26246

Table 3: cycle counts on the SiFive, compilation with -O3 for encrypting 128 bytes of message and 128 bytes of associated data or for hashing 128 bytes of data.

Conclusion

Conclusion

- translating assembly implementation back into C leads to further speed-ups
- approach applicable to existing code bases: AES & Keccak assembly implementations
- fully unrolled loops may fail on physical devices (instruction cache)
- applying the bit manipulation extension[3] (B) leads to a reduction of instructions up to 66%

[3] `https://github.com/riscv/riscv-bitmanip`, commit a05231d

Thank you for your attention!

Paper: https://ia.cr/2020/836

Code: https://github.com/AsmOptC-RiscV/Assembly-Optimized-C-RiscV