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

Outline

- Preliminaries
- Optimized Algorithms
- Comparison & Benchmark
- Conclusion

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?

Preliminaries

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
- 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

Optimized C

- 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

Optimized Algorithms

- NIST lightweight round 2 candidate
- lightweight scheme (Gimli-Hash & Gimli-Cipher)
- 384-bit (3 x 4 matrix of 32-bit words) permutation

- careful scheduling of instructions
- saving only 4 callee into the stack
- unrolling in C over 8 rounds
- renaming variable to avoid move operations
- speed-up up to 19%

type	C-ref	Assembly	Optimized C
Gimli	2178	2092 (−4%)	1900 (−13%)
Gimli-Hash	23120	20812 (−10%)	18678 (−19%)
Gimli-Cipher	44423	39583 (−10%)	35853 (−19%)

Table 1: Cycle counts on the SiFive board

- NIST lightweight round 2 candidate
- family of permutations (Sparkle) based on the block cipher Sparx
- Schwaemm (AEAD) and Esch (hash function)
- Schwaemm works on 384 bits (Sparkle384)
- Esch works on 256 bits (Sparkle256)

- loop unrolling
- avoid loading of round constants
- speed-up up to 42%

mode	C-Ref.	Optimized C
Esch256	58193	33331 (−42%)
Schwaemm256-128	71271	42634 (−40%)

Table 2: Cycle counts for Esch256 and Schwaemm256-128

- NIST lightweight round 2 candidate
- based on a 256-bit block cipher with a 256-bit key
- Saturnin-Hash (hash function) and Saturnin-Cipher (AEAD)

- two bitsliced variants analyzed:
  - **bs32** bitslices inside of block
  - **bs32x** bitslices across blocks
- speed-up the loading of constants
- greedy unrolling and inlining

mode	C-Ref.	bs32	bs32x
Saturnin-Cipher	121651	59368 (−51%)	68792 (−43%)
Saturnin-Hash	49433	28199 (−43%)	-

Table 3: Cycle counts for Saturnin-Cipher (128 AD bytes and 128 message bytes) and Saturnin-Hash

- NIST lightweight round 2 candidate
- 320-bit state
- Ascon-Hash based on eXtendable output function Ascon-Xof
- Ascon (AEAD) based on duplex construction

- reduce the number of required instructions in the S-Box
- improved S-box in a 6-round unrolled permutation
- speed-up up to 15%

Elephant

- NIST lightweight round 2 candidate
- family of lightweight authenticated encryption schemes
- Delirium = Elephant-Keccak-f[200]
- 200-bit state

- parallelization through bit interleaving
- combining 4 blocks into 1 block of 4-byte elements
- state 25 32-bit words (5-by-5-by-32) with size of 800 bits

message/data length	C-ref	bit interleaved
16/16	66541	73989 (+11%)
64/64	143181	74890 (−47%)
128/128	241975	145936 (−53%)

Table 4: Cycle counts on the SiFive board

- NIST lightweight round 2 candidate
- scheme suitable for several symmetric-key function
- hashing, encryption, MAC computation and authenticated encryption
- 384-bit state
- based on the Xoodoo permutation

- lane Complementing: reduce number of NOT instructions
- loop unrolling

mode	Ref	unrolled & lane comp.
hash	82741	17963 (−79%)
AEAD	103522	23238 (−18%)

Table 5: Cycle counts for Xoodyak on the SiFive board

AES

- based on RISC-V available assembly implementation[1]
- combined multiple steps of the round function in a lookup table (T-table)
- "safe", since none of our benchmarking platforms have data cache
- using a bitsliced approach, multiple blocks processed in parallel
- speed-up up to 4% compared to the assembly version

[1] `https://github.com/Ko-/riscvcrypto`

Comparison & Benchmark

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 6: 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

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`