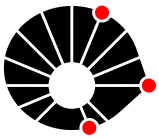

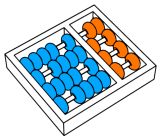

Exploring ASCON on 64-bit RISC-V

Paulo Pacitti *Julio López*

Technical Report - IC-PFG-23-41 - Relatório Técnico
December - 2023 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Exploring ASCON on 64-bit RISC-V

Paulo Pacitti* Julio López†

Abstract

Sed quis lorem magna. Sed sit amet ullamcorper massa, sit amet placerat lectus. Suspendisse pulvinar ipsum sed enim commodo, ac malesuada lectus finibus. Aliquam eu eros eleifend, interdum nisi faucibus, viverra sapien. Vivamus lobortis a lectus eu rutrum. Quisque in est sit amet libero sollicitudin ornare a sed ipsum. Suspendisse potenti. Aliquam sit amet nisi sed nulla tincidunt imperdiet. Pellentesque elementum lacus eget dolor gravida lobortis. Sed placerat lacinia nisi, sed varius turpis facilisis ac.

1 Introduction

Inspired by the works of the UNICAMP's Laboratory of Security and Cryptography in the optimization of cryptographic algorithms for the ARM architecture [1], the NIST Lightweight Cryptography competition winner algorithm [2], and the RISC-V open architecture, this research aims to explore the Ascon family of algorithms [3] on the RISC-V 64-bit architecture and wether it's possible to optmize it for this architecture.

The approach was to analyze the Ascon algorithm design and 3 different implementations. All the implementations tested are written in C. The first implementation `ref` is the reference implementation of Ascon, written by Ascon team [4]. The second one is `opt64`, and optmized implementation for a generic 64-bit architecture system, also developed by the Ascon team. The third implementation was the main objective of this research, named `ascon-v` [5], this implementation is focused on producing a optmized version for the RISC-V 64-bit architecture. The research was focused on trying to improve the basic blocks of the Ascon family of algorithms. Because of that, the analysis, optimizations and results are focused on the Ascon-128, which is the *de facto* AEAD standard of the Ascon family.

2 Ascon

Ascon is a family of algorithms for lighweight cryptography, designed to be used in constrained environments, like embedding computing. Designed by cryptographers from Graz University of Technology, Infineon Technologies, Intel Labs, and Radboud University, Ascon has been selected as the new standard for lightweight cryptography in the 2019–2023 NIST Lightweight Cryptography competition. The Ascon family is mainly composed by 4 algorithms: Ascon-128, Ascon-128a, Ascon-Hash and Ascon-Hasha. There's also variants Ascon-80pq, Ascon-Xof, Ascon-Xofa, where the first it's a version of AEAD with an increased key size of 160 bits and the latter two are versions of the hash algorithm but they produce hash outputs of arbitrary length, just changing the number of rounds necessary for it.

*Computer Engineering Undergraduate, Institute of Computing, UNICAMP. p185447@dac.unicamp.br
†Associate Professor, Institute of Computing, UNICAMP. jlopez@ic.unicamp.br

insert table with Ascon parameters

Ascon lightweight properties comes from using the simple bitwise operations that majority of microcontrollers have, like XOR, AND, OR, NOT, and bitwise rotations. The algorithm is based on a sponge construction, which is a cryptographic primitive that can be used to build cryptographic hash functions, pseudorandom functions, and authenticated encryption schemes, like the SHA-3 (also know as "Keccak") [6] algorithm. The sponge construct consists in keeping a finite internal state that takes input streams (absorb) to update the state and output streams (squeeze) to produce the output from the internal state, as it's displayed in Figure 1.

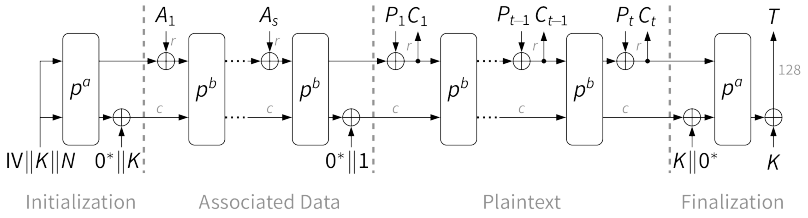

Figure 1: Sponge construct for AEAD encryption with ASCON.

The Ascon state is composed by 5 64-bit words, alson named as Ascon words, resulting in a 320-bit internal state. This internal state is then manipulated using the Ascon permutation procedure.

2.1 Permutation

asdasd

2.2 Encryption

2.3 Decryption

3 RISC-V

4 Implementation

The device used for this research is the MangoPi MQ-Pro, a SBC powered with a Allwinner D1 chip and 1GB DDR3 of RAM, with Wi-Fi, Bluetooth and HDMI video output. The Allwinner D1 chip contains a T-Head Xuantie C906 core, a RISC-V 64-bit 1GHz CPU supporting RV64GC ISA. The board runs Ubuntu Server 23.04, running the 6.2.0-36-generic version of the Linux kernel. For compiling the implementation in C, it was used the RISC-V GNU Compiler Collection (GCC) version 12.2.0 [7] through cross-compilation with Newlib, using a MacBook Pro with an Apple M1 chip.

Ascon words, used to mantain the state in the sponge construct, are big endian. The reference implementation merges data to these words by loading and storing bytes using big-endianess, requiring operations to fill the right-side with zeros. However, RISC-V, as like most of other ISAs, is little endian, making bitwise operations slower than it could be if the architecture had the same endianess than the algorithm. It's possible to implement an opmitization considering this issue by handling the data as little-endian in the implementation and reversing the endianess when merging data to the Ascon words [8]. This turns out to be way more effective than operating in data in big

```c
// Ascon state with 5 64-bit words.
typedef struct {
    uint64_t x[5];
} ascon_state_t;

// Bitwise rotation to the right.
static inline uint64_t ROR(uint64_t x, int n) {
    return x >> n | x << (-n & 63);
}

// Ascon permutation round function.
static inline void ROUND(ascon_state_t *s, const uint8_t C) {
    ascon_state_t t;
    /* round constant layer */
    s->x[2] ^= C;
    /* substitution layer */
    s->x[0] ^= s->x[4];
    s->x[4] ^= s->x[3];
    s->x[2] ^= s->x[1];
    t.x[0] = s->x[0] ^ (~s->x[1] & s->x[2]);
    t.x[1] = s->x[1] ^ (~s->x[2] & s->x[3]);
    t.x[2] = s->x[2] ^ (~s->x[3] & s->x[4]);
    t.x[3] = s->x[3] ^ (~s->x[4] & s->x[0]);
    t.x[4] = s->x[4] ^ (~s->x[0] & s->x[1]);
    t.x[1] ^= t.x[0];
    t.x[0] ^= t.x[4];
    t.x[3] ^= t.x[2];
    t.x[2] = ~t.x[2];
    /* linear diffusion layer */
    s->x[0] = t.x[0] ^ ROR(t.x[0], 19) ^ ROR(t.x[0], 28);
    s->x[1] = t.x[1] ^ ROR(t.x[1], 61) ^ ROR(t.x[1], 39);
    s->x[2] = t.x[2] ^ ROR(t.x[2], 1) ^ ROR(t.x[2], 6);
    s->x[3] = t.x[3] ^ ROR(t.x[3], 10) ^ ROR(t.x[3], 17);
    s->x[4] = t.x[4] ^ ROR(t.x[4], 7) ^ ROR(t.x[4], 41);
}
```

Listing 1: Ascon permutation used in `ref` implementation.

endinaness since loading bytes and other biwise operations does not need to fill the right-side of the bistring with zeros, as it is in big-endinaness. That way, the cost of reversing the endianess of little-endian 64-bit bistring is lower than the cost of loading data in big-endianess.

5 Results

Considering t the elapsed time to run encryption/decryption of a plaintext/ciphertext, the resolution R of the timer used to measure the time of the C906 core to be 45 nanoseconds [9], F the CPU frequency, the number of clock cycles used in encryption/decryption can be calculated

Equation (1):

$$C = t \times R \times F \times \frac{10^9}{60} \tag{1}$$

6 Future

As we can see, the RV64GC instructions do not allow great optimizations from the architecture itself since it doesn't have any special instructions to accelerate operations of the Ascon128. However, RISC-V does have instructions extensions in development, and even ratfied, that could improve the performance of Ascon. Such cryptographic specialized instruction extenstions are divided in scalar and vectorial.

The Scalar Cryptography set of extensions (Zbkb, Zbkc, Zbkx, Zknd, Zkne, Zknh, Zksed, Zksh, Zkn, Zks, Zkt, Zk, Zkr) [10] provide instructions that could accelerate operations of the Ascon permutation. The Zbkb extension provides bitmanipulation instructions for cryptographic operations such as bit rotations operations (`rori`) and bitwise logical AND operation between a value a and the bitwise inversion of a value b (`andn`), that could accelerate the Ascon permutation as seen in Listing 1. This same extension also provides a byte-reverse register instruction (`rev8`) that could be used to reverse the endianess of the Ascon words, making the work with little endinaness data loading first and then reversing to big endinaness way faster. The Zkn extension provides an entropy source in a CSR register that could be used to generate random numbers for the nonce and the key, improving the security of the algorithm.

7 Conclusions

References

[1] H. Fujii and D. F. Aranha. "Curve25519 for the Cortex-M4 and beyond". In: June 2017. URL: http://www.cs.haifa.ac.il/~orrd/LC17/paper39.pdf.

[2] Meltem Sönmez Turan et al. "Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process". In: (2023).

[3] Christoph Dobraunig et al. *Ascon v1.2*. Submission to Round 1 of the NIST Lightweight Cryptography project. 2019. URL: https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/ascon-spec.pdf.

[4] Ascon Team. *ascon-c: Reference and optimized implementations of Ascon*. https://github.com/ascon/ascon-c. 2023.

[5] Paulo Pacitti. *ascon-v: Ascon lightweight cryptographic algorithm implementation for improved performance on RISC-V*. https://github.com/paulopacitti/ascon-v. 2023.

[6] Guido Bertoni et al. "Keccak". In: *Cryptology ePrint Archive* (2015).

[7] RISC-V Collaboration. *riscv-gnu-toolchain: GNU toolchain for RISC-V, including GCC*. https://github.com/riscv-collab/riscv-gnu-toolchain. 2023.

[8] Lars Jellema. "Optimizing ascon on RISC-V". In: *Bachelor Thesis, Radboud University* (2019).

[9] Lukas Gerlach et al. "A Security RISC: Microarchitectural Attacks on Hardware RISC-V CPUs". In: *2023 IEEE Symposium on Security and Privacy (SP)*. 2023, pp. 2321–2338. DOI: 10.1109/SP46215.2023.10179399.

[10] RISC-V Foundation. *RISC-V Cryptography Extensions Volume: Scalar & Entropy Source Instructions (v1.0.1)*. `https://github.com/riscv/riscv-crypto/releases/tag/v1.0.1-scalar`. 2023.