# Hardware Acceleration of Authenticated Encryption with Associated Data via RISC-V Instruction Set Extensions in Low Power Embedded Systems

Carlos Gewehr*†, Nicolas Moura*†, Lucas Luza*, Eduardo Bernardon*,
Ney Calazans§, Rafael Garibotti*, Fernando Gehm Moraes*
*School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil
†EnSilica – Porto Alegre, Brazil
§PGMICRO, Federal University of Rio Grande do Sul – UFRGS – Porto Alegre, Brazil
{carlos.gewehr, nicolas.moura, lucas.luza, eduardo.bernardon}@edu.pucrs.br,
nlvcalazans@inf.ufrgs.br, rgaribotti@vectrading.com, fernando.moraes@pucrs.br

2024 IEEE 15th Latin America Symposium on Circuits and Systems (LASCAS) | 979-8-3503-8122-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/LASCAS60203.2024.10506132

***Abstract*—This work compares trade-offs to implement Authenticated Encryption with Associated Data (AEAD) in resource-constrained RISC-V-based embedded systems, both with and without hardware acceleration via Instruction Set Extensions (ISEs). AES-128 in CCM mode, Ascon and ChaCha20-Poly1305 algorithms are evaluated regarding suitable metrics for embedded systems, including clock cycle count, energy efficiency, memory usage, and die area cost for ISEs that accelerate each algorithm. Total clock cycles improve AES-128 in CCM mode, Ascon and ChaCha20-Poly1305, respectively by 19.66x, 2.45x and 1.05x, while energy efficiency enhances by 17.28x, 2.84x, 1.18x. Experiments employ a Zigbee network packet evaluation scenario and incur up to 9% area overhead. Static memory efficiency gains reach 1.56x for AES-128 in CCM mode and up to 1.41x for Ascon. Our RTL and software implementations are available at *https://github.com/cggewehr/RISCV-crypto*.**

***Index Terms*—RISC-V; Instruction Set Extensions; Lightweight Cryptography; Hardware Acceleration; Ascon; AES; AES-CCM; ChaCha20; ChaCha20-Poly1305.**

## I. Introduction

With the increase in the deployment of low-power embedded systems, secure communication becomes a core requirement for several applications. Such devices do not have expressive computing power and memory, and are often battery-powered or use energy harvesting. Authenticated Encryption with Associated Data (AEAD) ensure confidentiality and authenticity of sensitive data. Often, low-power embedded systems use AES-128 [1] with the Counter with Cipher Block Chaining-Message Authentication Code (CCM) operation mode [2] and ChaCha20-Poly1305 [3] as the AEAD algorithms of choice. These algorithms are supported in a wide range of internet protocols, notably TLS [4] and DTLS [5].

The lack of computing power in platforms such as low-cost microcontrollers increases the interest to develop new AEAD algorithms for resource-constrained devices. This branch of cryptography is called Lightweight Cryptography (LWC). As with AES, the National Institute of Standards and Technology (NIST) has organized a competition to standardize a single LWC algorithm. In February 2023, Ascon [6] was declared the winner of the LWC NIST competition.

This work was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES (Finance Code 001), CNPq (grant 309605/2020-2), and FAPERGS (grant 21/2551-0002047-4).



Hardware acceleration via Instruction Set Extensions (ISEs) can enhance the performance, memory footprint, and energy efficiency of cryptography. Compared to memory-mapped accelerators, extending a base instruction set with specialized instructions for a given algorithm provides benefits relevant to low-power embedded systems, including: ($i$) resource sharing among generic and specialized components, such as the register file and RAM interfaces; ($ii$) trivial data transfer among generic and specialized components via the register file, avoiding costly memory accesses; ($iii$) no added system complexity to e.g. the bus and interrupt controllers.

This work presents the implementation and evaluation of 3 AEAD algorithms (AES-128 in CCM mode, ChaCha20-Poly1305, and Ascon) in the context of the low-complexity Ibex RISC-V core [7]. The evaluation considers software and hardware-accelerated implementations via RISC-V ISEs in two forms: generic bit manipulation operations through the ***Zbkb*** extension [8]; and specialized algorithm-specific extensions, namely the standardized ***Zkne*** extension for AES and a custom Ascon extension proposal by Cheng et al. [9], ***Xascon***. The omission of specialized instructions for ChaCha20 is intentional, as ChaCha20 was designed to be efficient in pure software implementations and does not lend itself to efficient hardware acceleration [10].

The original contributions of this work include:

- an open source RTL implementation of the Ibex RISC-V processor with *Zbkb*, *Zkne*, and *Xascon* [9] extensions;
- a speed analysis of Associated Data and Plaintext processing in pure software and with ISEs implementations;
- an analysis of trade-offs in ISE implementations, evaluating performance, energy, memory usage, and die area.

## II. Related Work

Kane et al. [11] evaluate various combinations of three common microcontrollers (ATmega328, STM32F103C8T6, and ESP8266) and three encryption algorithms (AES, ChaCha, and Acorn), measuring relevant metrics such as power consumption, energy usage, execution time, and memory footprint. The authors evaluate only encryption, not authentication, from a software-only perspective. ChaCha20 is concluded to be the best encryption algorithm in pure software implementations.

Gewehr et al. [12] evaluate AES encryption and SHA-2 hashing with the *Zkne* and *Zknh* RISC-V extensions. The evaluation employs post-synthesis netlist simulations in a 28 nm technology. For AES, only the core block cipher operation is evaluated, modes of operation such as CCM are unexplored. Authors find that the extensions provide an acceleration of 42.87x in terms of clock cycle count to the AES-128 cipher.

Cheng et al. [9] evaluate Ascon and 8 other LWC algorithms in a RISC-V, making use of the standardized *Zbkb* extension. In the case of Ascon, they propose a new custom extension called *Xascon*. Authors evaluate their work in the context of the Rocket core [13], with 5 pipeline stages and floating point instructions, which is much more complex than expected for an IoT context. Hardware-specific metrics are evaluated in FPGAs. No mention exists to ASIC implementations.

Table I summarizes and compares the core contributions of the revised works. Deeper insights into AEAD algorithms suitable for low-complexity embedded systems in the context of RISC-V ISEs is clearly a gap in the revised literature.

TABLE I
RELATED WORK COMPARISON.

| Reference → Feature ↓ | [11] | [12] | [9] | Our Work |
|---|---|---|---|---|
| Open-source RTL | | X | | X |
| Open-source software | | X | X | X |
| ASIC evaluation | | X | | X |
| AES-128 in CCM mode | | | | X |
| ChaCha20-Poly1305 | X | | | X |
| Ascon | | | X | X |

## III. AEAD ALGORITHMS AND THEIR HARDWARE ACCELERATION VIA RISC-V ISES

### A. An Introduction to AEAD

AEAD algorithms seek to provide confidentiality and authentication for a given plaintext $P$ and authentication (but not confidentiality) of associated data $A$. An use case for AEAD is network packets, where headers ($A$) are visible to a router but payloads private. In addition to $A$ and $P$, the AEAD process $E$ (Equation (1)) expects a key $K$ and a unique nonce $N$. These additional values are used to set the initial state of the algorithm, in conjunction to, in the case of Ascon, a static initialization vector $IV$; or in the case of CCM mode, the message length, determining $CTR_0$. The outputs of the AEAD process are the ciphertext $C$ and an authentication tag $T$.

$$E(K, N, A, P) = (C, T) \tag{1}$$

### B. AES-128 in CCM mode

AES [1] is a block cipher standard widely used to confidentially transmit information. In embedded systems, the AES-128 variant is the most commonly employed. A common technique used in software implementations of AES is the use of T-Tables, in which an iteration in an AES cipher operation is reduced to 16 LUT accesses and 16 bitwise XORs, at a prohibitive cost of 4 KB of memory. The RISC-V *Zkne* extension defines the `aes32esmi` and `aes32esi` specialized instructions, which compute T-Table entries online in hardware and optionally XOR the current entry with previous entries for the same output column [12, 14].

The CCM operation mode [2] internally uses two simultaneous operation modes, CTR for encryption and CBC for generating authentication tags [15]. The ciphertext is computed by encrypting the plaintext using the CTR mode, and the authentication tag is computed by encrypting the associated data and plaintext using the CBC mode. This process can be visualized in Figure 1. The CCM mode of operation is interesting, since it uses the underlying block cipher both for encryption and authentication, unlike other AEAD modes of operation such as GCM [16], which employ other means for generating authentication tags. Hardware acceleration of the underlying block cipher (here, AES-128) improves the efficiency of both encryption/decryption and authentication. Note that one AES-128 operation is performed for each block of associated data, while two AES-128 operations are performed for each block of plaintext.

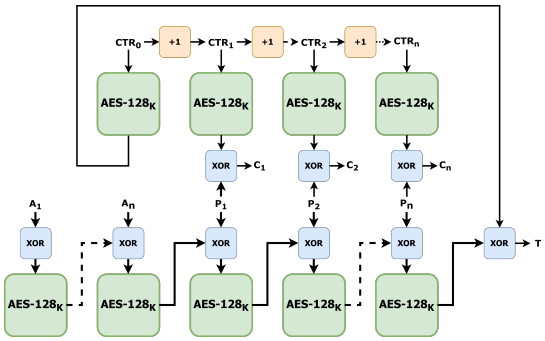


Fig. 1. The CCM mode of operation.

### C. ChaCha20-Poly1305

The ChaCha cipher and Poly1305 authenticator [3] were proposed as alternatives to AES due to its high performance in pure software implementations, especially in low-complexity microcontrollers. The ChaCha20 variant is the most commonly used. Internally, ChaCha uses three operations in 32-bit variables: addition, rotations by constant amounts, and XOR. It has an internal state of 512 bits, split into 16 32-bit variables. Four variables are processed simultaneously in the quarter-round function, depicted in Figure 2.

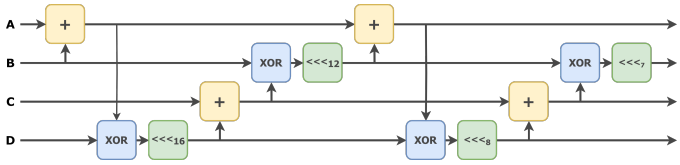


Fig. 2. ChaCha quarter-round function.

Hardware acceleration of ChaCha on 32-bit RISC-V cores via specialized algorithm-specific instructions has been briefly explored in [10]. Note that each group of Add-XOR-Rotate macro-operation has 3 inputs, and cannot be implemented directly in a single RISC-V instruction, since an instruction should read at most two inputs from the register file. Reading a third input would imply a profound change in the register file implementation, leading to a considerable area overhead. Merging two operations instead of three, would not result in expressive gains. Small gains are achievable via the *Zbkb* extension, noticing that the right-rotation operation consists in a single instruction (`rori`), instead of three RV32I instructions.

### D. Ascon

Unlike AES and ChaCha, Ascon [6] is based on a newer construction, called "sponge in duplex mode", depicted in Figure 3. At each stage of the sponge, a permutation of the internal state is performed ($p^a$ and $p^b$, which differ only in the number of rounds). Internally, the permutation has a state of 320 bits, split into five 64-bit variables $x_0, ..., x_4$. Figure 4 summarizes the Ascon permutation operation. Note that in Ascon, one permutation per data block absorbed is performed for both associated data and plaintext, unlike the CCM mode of operation, which computes two block cipher operations for each plaintext block. This work considers the Ascon-128 variant, with the sponge rate ($A_i$, $P_i$ and $C_i$ width) of 64 bits.

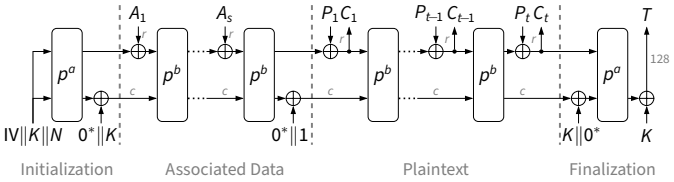
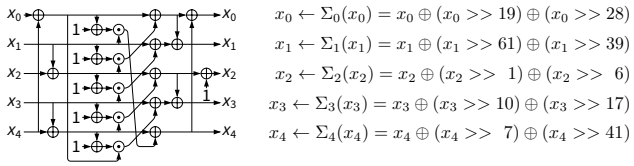



Fig. 3. Ascon duplex sponge [6].




(a) Ascon 5-bit SBOX S(x) (b) Ascon linear layer $\Sigma_i(x_i)$

Fig. 4. Ascon substitution later and linear diffusion layer [6].

Hardware acceleration of the Ascon permutation is done mainly in the SBOX (Figure 4(a)) and linear diffusion (Figure 4(b)) steps of the Ascon permutation. Several of the operations in the SBOX have one of the operands negated (represented graphically in Figure 4(a) as XOR with constant 1). The *Zbkb* extension provides the `andn` and `orn` instructions, which directly perform AND and OR bitwise operations with one negated operand. The `rev8` instruction also helps converting big-endian to litte-endian representations.

The linear diffusion step consists of rotations by constant amounts and bitwise XORs of these rotations. Variables $x_0, ..., x_4$ are 64-bit wide, requiring each two registers to store their value. Bit-interleaving can speed up these rotations, where instead of splitting a 64-bit variable in high and low registers, a register stores the even-indexed bits of the variable, while another stores its odd-indexed bits. A rotation by $2a$ bits in a 64-bit variable becomes two $a$ bit rotations, one in each 32-bit register [17]. *Zbkb* provides the `zip`, `unzip` and `pack` instructions for the efficient translation between both representations. *Zbkb* also provides the `rori` instruction, directly implementing a rotation, otherwise implemented with two shifts and one `or` RV32I instructions.

The linear diffusion step can be optimized even further with the use of the *Xascon* extension from [9]. With the *Xascon* extension, sigma functions can be computed directly with two instructions, much like the SHA-512 instructions from the standardized *Zknh* extension [8]. Note that with the *Xascon* extension, no conversions between ABI and bit-interleaved representations are necessary.

## IV. Experimental Evaluation

Comparisons are made between AES-128 in CCM mode, Ascon and ChaCha20-Poly1305 in unmodified and extended Ibex cores with *Zkne*, *Zbkb* and *Xascon* extensions. The *SecureIbex* parametrization is used, except for the costly ECC RAM and lockstep features. Also employed are parametrizations for latch-based register file and 3-cycle multiplier.

The following baseline software implementations from known libraries are considered: ($i$) TinyCrypt [18], for AES-128 in CCM mode; ($ii$) Ascon GitHub repository (employing "ascon128v12/asm_bi32_rv32b" and "ascon128v12/asm_rv32') [19]; and ($iii$) Libsodium [20], for ChaCha20-Poly1305. RISC-V optimized assembly implementations are not available for a fairer comparison. The Ibex data independent timing flag is set, such that no time side-channel is introduced. The evaluation herein considers two real-world scenarios originally proposed in [21], Zigbee and IPv6 packets. Compilation employs GCC version 12.2.0 with the `-Os` flag.

### A. Performance Evaluation in Real-World Scenarios

Table II presents key performance figures. White columns show values for the baseline, pure software implementations; yellow columns show data accelerated by the more generic *Zbkb* extension; blue columns show data on hardware accelerated by algorithm-specific extensions (*Zkne* and *Xascon*); finally, green columns show data on hardware accelerated by both *Zbkb* and algorithm-specific extensions (*Zkne* or *Xascon*).

Ascon is the fastest algorithm in pure software and with ISEs implementations in both scenarios (row "Clock cycles"). Comparing pure software implementations, ChaCha20-Poly1305 shows a much smaller difference to Ascon in the IPv6 scenario than in the Zigbee scenario. This suggests ChaCha20-Poly1305 performs better for messages longer than an IPv6 packet, but these may not correspond to real scenarios.

The Ibex processor requires two cycles for memory access operations, influencing the clock cycles per instruction (CPI) ratio. The Ascon implementation has much less memory operations than the AES and ChaCha20 implementations, leading to Ascon having a better CPI. Note that in the IPv6 scenario, software ChaCha20-Poly1305 (optimized for speed) executes fewer instructions than Ascon, but requires more clock cycles, due to the difference in the amount of memory operations.

### B. Theoretical Scalability Analysis

Follows an analysis of the maximum theoretical performance of Ascon and AES-128 in CCM mode, considering the core operations for each algorithm, the $p^b$ permutation and the AES-128 block cipher, but excluding auxiliary operations and data IO. The findings are summarized in Table III and Table IV. Based on the clock cycle per byte values from Table III, AES-128 in CCM mode with the *Zkne* extension should be the fastest AEAD algorithm for an extended Ibex core. This is not observed in the evaluation scenarios in Table II due to the fact that the AES-128 in CCM mode baseline C implementation is much less optimized than the baseline Ascon ASM implementation. The AES-128 in CCM mode implementation uses memory operations for internal temporary values, while in Ascon internals are kept inside the register file at all times.

TABLE II

AEAD PERFORMANCE FIGURES (WHITE: BASELINE SOFTWARE; YELLOW: *Zbkb*; BLUE: [*Zkne*, *Xascon*]; GREEN: *Zbkb* AND [*Zkne*, *Xascon*]).

| Performance Counters | Zigbee packet (A = 25 bytes, P = 86 bytes) | | | | | | | | IPv6 packet (A = 40 bytes, P = 1224 bytes) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AES-128 CCM | | Ascon | | | | ChaCha20-Poly1305 | | AES-128 CCM | | Ascon | | | | ChaCha20-Poly1305 | |
| **Clock cycles** | **218,582** | **11,119** | **14,385** | **12,097** | **6,825** | **5,881** | **21,913** | **20,953** | **2,171,290** | **107,176** | **128,471** | **107,921** | **60,011** | **50,979** | **129,660** | **122,940** |
| **Instructions retired** | **133,834** | **7,414** | **13,734** | **8,939** | **6,174** | **5,230** | **16,615** | **14,695** | **1,325,726** | **73,752** | **123,381** | **79,887** | **54,921** | **45,889** | **100,666** | **87,226** |
| **Instructions fetched (KB)** | **309.541** | **25.568** | **53.086** | **34.326** | **23.555** | **19.867** | **54.049** | **47.250** | **4,112.371** | **258.230** | **478.639** | **308.428** | **211.217** | **175.936** | **330.445** | **282.865** |
| **Loads from memory (KB)** | **50.142** | **3.698** | **0.222** | **0.448** | **0.222** | **0.222** | **4.392** | **4.392** | **493.120** | **34.993** | **1.342** | **3.409** | **1.342** | **1.342** | **20.497** | **720.497** |
| **Stores to memory (KB)** | **30.818** | **0.818** | **0.190** | **0.190** | **0.190** | **0.190** | **4.204** | **4.204** | **303.554** | **5.429** | **1.295** | **1.295** | **1.295** | **1.295** | **16.341** | **16.341** |

TABLE III

ASSOCIATED DATA & PLAINTEXT PERFORMANCE (SMALLER IS BETTER).

| | | Ascon | | | | AES |
|---|---|---|---|---|---|---|
| **Metric** | | **SW** | **Zbkb** | **Xascon** | **Zbkb+Xascon** | **AES-128 Zkne** |
| AD | Cycles/byte | 89.875 | 75.375 | 37.375 | 34.375 | **18.563** |
| | Instr./byte | 88.125 | 56.875 | 35.625 | 32.625 | **14.5625** |
| P | Cycles/byte | 89.875 | 75.375 | 37.375 | **34.375** | 37.125 |
| | Instr./byte | 88.125 | 56.875 | 35.625 | 32.625 | **29.125** |

TABLE IV

CORE OPERATION PROFILING – (CLOCK CYCLES, INSTR. RETIRED), CPI.

| | Ascon | | | | AES |
|---|---|---|---|---|---|
| **Step** | **SW** | **Zbkb** | **Xascon** | **Zbkb+Xascon** | **AES-128 Zkne** |
| SBOX | (204, 204), 1 | (180, 180), 1 | (204, 204), 1 | (180, 180), 1 | - |
| Linear | (480, 480), 1 | (365, 238), 1.53 | (60, 60), 1 | (60, 60), 1 | - |
| Other | (35, 21), 1.66 | (58, 37), 1.57 | (35, 21), 1.67 | (35, 21), 1.67 | - |
| Total | (719, 705), 1.02 | (603, 455), 1.32 | (299, 285), 1.05 | (275, 261), 1.05 | (297, 233), 1.27 |

The TinyCrypt AES in CCM mode implementation spends roughly 5600 cycles outside of AES-128, amounting to nearly half the clock cycles in the Zigbee scenario considering the *Zkne* extension. An ASM implementation of AES-128 with the *Zkne* extension in CCM mode akin to the Ascon ASM implementation would provide a fairer comparison. Again note the CPI difference due to memory operations in Table IV. AES-128 must load the round key at each iteration, while Ascon permutations do not employ memory operations, except in the *Zbkb* case. Here, due to the bit-interleaved representation, round constants cannot be determined through subtractions.

### C. Memory Usage Evaluation

The GCC `fstack-usage` flag is used to obtain stack usage and GNU `nm` for functions and static data sizes. Results for both encryption and decryption are depicted in Figure 5.

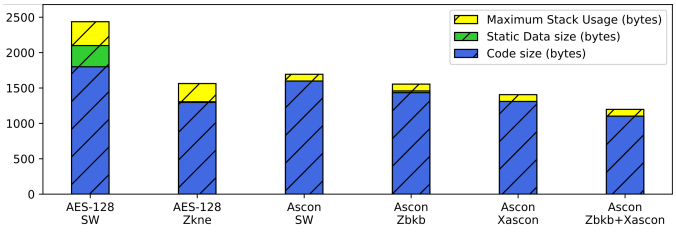



Fig. 5. Memory Usage for AES-128 in CCM mode and Ascon.

Due to the convoluted internal structure of Libsodium ChaCha20-Poly1305 precise memory usage values could not be obtained, but are more than twice the size of TinyCrypt AES-128 CCM and do not decrease significantly with *Zbkb*. Figure 5 shows AES-128 in CCM mode generates a more pronounced decrease in memory usage compared to Ascon.

### D. Die Area Cost Evaluation

The baseline and extended cores were synthesized targeting a high-density 8-track cell library for a 28 nm FDSOI process from ST Microelectronics. Table V shows the synthesis results achieved with Cadence Genus Version 2112, using the PLE flow, and considering a timing worst-case PVT corner of a slow process (0.75 V at 125 C). All netlists show a timing slack of 0 ns, leading to a fair comparison. From Table V, it is clear that accelerating AES-128 in CCM mode has a smaller overhead than accelerating Ascon and ChaCha20-Poly1305.

TABLE V

SYNTHESIS RESULTS FOR BASELINE AND EXTENDED IBEX CORES.

| **Synthesis results** | **Baseline** | **Zbkb** | **Xascon** | **Xascon+Zbkb** | **Zkne** |
|---|---|---|---|---|---|
| Cell Area ($\mu m^2$) | 11,238 | 11,307 | 12,210 | 12,148 | 11,709 |
| Net Area ($\mu m^2$) | 6,992 | 5,142 | 8,181 | 5,957 | 6,261 |
| Total Area ($\mu m^2$) | 18,230 | 16,449 | 20,391 | 18,105 | 17,970 |
| **Cell Instance Count** | **10,289** | **11,769** | **11,010** | **12,256** | **11,458** |
| **Equiv. NAND2 gates** | **34,433** | **34,642** | **37,408** | **37,219** | **35,873** |

### E. Energy Consumption Evaluation

CACTI [22] and gate-level simulations allow computing energy consumption. Figures come from a nominal PVT corner of a typical process (0.9 V at 25 C). For a 16KB dual-port SRAM with low-power bit cells in 28 nm technology, CACTI reports 63.362 fJ/bit for reads and 41.436 fJ/bit for writes. Energy results are shown in Figure 6. Software AES-128-CCM figures are omitted, due to their high energy cost.

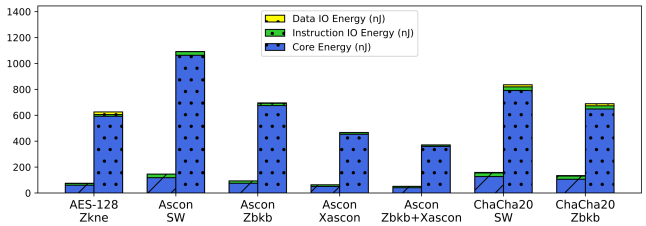



Fig. 6. Energy Consumption Evaluation - Zigbee - left, IPv6 - right.

Clearly, energy consumption is dominated by the RISC-V core rather than memory I/O. With ISEs, the most efficient implementation is Ascon with *Zbkb* and *Xascon* for both scenarios. In pure software, Ascon is slightly more efficient in the Zigbee scenario, but ChaCha20-Poly1305 is much more efficient in the IPv6 scenario. This is due to the CPI difference discussed in Section IV-A, causing ChaCha20-Poly1305 to dissipate less power for a similar runtime.

## V. CONCLUSIONS AND FURTHER WORK

This work demonstrated the acceleration of AES-128 in CCM mode and Ascon in the low-complexity RISC-V Ibex core by using the *Zbkb*, *Zkne* and *Xascon* extensions, with a smaller improvement for ChaCha20-Poly1305. Results show a performance gain of 19.66x, 2.45x and 1.05x; and an increase in energy efficiency of 17.28x, 2.84x, 1.18x, respectively. Area overheads are up to 9%. Improvements in memory usage (considering code size, static data size and stack usage) of 1.56x (AES in CCM mode) and up to 1.41x (Ascon) have also been measured. The software as well as new RTL implementations are available at *https://github.com/cggewehr/RISCV-crypto*.

An analysis of the core operations in AES-128 in CCM mode and Ascon shows that AES-128 in CCM mode should be slightly faster than Ascon considering algorithm-specific extensions. This was not observed due to the difference in the optimization levels of the baseline software implementations. As future work, assembly implementations of AES-128 in CCM mode and ChaCha20-Poly1305 optimized for code size enabling a fairer comparison to Ascon are suggested.

## REFERENCES


[1] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," Tech. Rep., Nov. 2001. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.197

[2] ——, "Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality," Tech. Rep., May 2007. [Online]. Available: https://doi.org/10.6028/nist.sp.800-38c

[3] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 7539, May 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7539

[4] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[5] E. Rescorla, H. Tschofenig, and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," RFC 9147, Apr. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9147

[6] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Lightweight Authenticated Encryption and Hashing," *Journal of Cryptology*, vol. 34, no. 3, Jun. 2021. [Online]. Available: https://doi.org/10.1007/s00145-021-09398-9

[7] lowRISC, "Ibex RISC-V Core," 2018. [Online]. Available: https://github.com/lowRISC/ibex

[8] RISC-V Foundation, "RISC-V Cryptography Extensions Volume I: Scalar & Entropy Source Instructions, Document Version v1.0.1," February 2022. [Online]. Available: https://github.com/riscv/riscv-crypto/releases/tag/v1.0.1-scalar

[9] H. Cheng, J. Großschädl, B. Marshall, D. Page, and T. Pham, "RISC-V Instruction Set Extensions for Lightweight Symmetric Cryptography," 2022. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9951

[10] B. Marshall, D. Page, and T. H. Pham, "A Lightweight ISE for ChaCha on RISC-V," Cryptology ePrint Archive, Paper 2021/1030, 2021. [Online]. Available: https://eprint.iacr.org/2021/1030

[11] L. E. Kane, J. J. Chen, R. Thomas, V. Liu, and M. Mckague, "Security and Performance in IoT: A Balancing Act," *IEEE Access*, vol. 8, pp. 121 969–121 986, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3007536

[12] C. Gewehr and F. Moraes, "Improving the Efficiency of Cryptography Algorithms on Resource-Constrained Embedded Systems via RISC-V Instruction Set Extensions," in *2023 36th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, Aug. 2023. [Online]. Available: https://doi.org/10.1109/sbcci60457.2023.10261964

[13] chipsalliance, "Rocket Core," 2023. [Online]. Available: https://github.com/chipsalliance/rocket

[14] M.-J. O. Saarinen, "A Lightweight ISA Extension for AES and SM4," 2020. [Online]. Available: https://arxiv.org/abs/2002.07041

[15] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques," Tech. Rep., 2001. [Online]. Available: https://doi.org/10.6028/nist.sp.800-38a

[16] ——, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," Tech. Rep., Nov. 2007. [Online]. Available: https://doi.org/10.6028/nist.sp.800-38d

[17] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, and R. V. keer, "Keccak implementation overview," Keccak Team, Tech. Rep., 2012. [Online]. Available: https://keccak.team/files/Keccak-implementation-3.2.pdf

[18] Intel, "TinyCrypt Cryptographic Library," 2017. [Online]. Available: https://github.com/intel/tinycrypt

[19] Ascon Team, "Reference, Highly Optimized, Masked C and ASM Implementations of Ascon," 2017. [Online]. Available: https://github.com/ascon/ascon-c

[20] F. Denis, "Libsodium." [Online]. Available: https://github.com/jedisct1/libsodium

[21] L. C. dos Santos, J. Großschädl, and A. Biryukov, "FELICS-AEAD: Benchmarking of Lightweight Authenticated Encryption Algorithms," in *Smart Card Research and Advanced Applications*. Springer International Publishing, 2020, pp. 216–233. [Online]. Available: https://doi.org/10.1007/978-3-030-42068-0_13

[22] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017. [Online]. Available: https://doi.org/10.1145/3085572