Experimental Evaluation of Lightweight Encryption Algorithms on 16-bit Microcontrollers

Marino Miculan[a], Matteo Paier[b] and Jacopo Plozner
Dept. of Mathematics, Computer Science and Physics, University of Udine, Italy

Keywords: Lightweight Cryptography, IoT Security, Embedded Systems, MSP430, Energy Efficiency, Benchmarking.

Abstract: The Internet of Things (IoT) increasingly relies on resource-constrained devices that require efficient symmetric encryption. Lightweight cryptography (LWC) addresses this by providing algorithms optimized for minimal computational and energy overhead. However, the performance of any cryptographic primitive is highly architecture-dependent, and direct experimental data on microcontrollers remains scarce in the literature. In this paper we present a rigorous evaluation of ten lightweight encryption algorithms — seven block ciphers (AES, CLEFIA, LEA, PRINCE, QARMAv2, SPARX, SPECK) and three stream ciphers (Ascon, ChaCha20, Hummingbird-2) — running on the 16-bit Texas Instruments MSP430FR6989 microcontroller, a platform widely used in industrial IoT and smart metering. Unlike prior surveys that rely solely on software counters or emulation, our methodology employs professional-grade instrumentation for real-time current profiling at the hardware level. Each algorithm is characterised under two compiler optimisation profiles (speed-optimised and size-optimised) across three metrics: encryption throughput, code footprint, and charge consumption.
Our results show that ARX-based ciphers — particularly LEA and SPECK 32/64 — achieve the best energy efficiency, outperforming AES by up to 25% in charge consumption while maintaining a significantly smaller code footprint. Hardware-oriented designs (PRINCE, QARMAv2) perform poorly in software, confirming that hardware efficiency does not translate to software performance. Among stream ciphers, Ascon — the NIST LWC standard — offers the best balance of security and efficiency, whereas ChaCha20 proves unsuitable for heavily resource-constrained contexts. We provide concrete algorithm recommendations for developers targeting MSP430 and similar 16-bit platforms, and we openly release all benchmark implementations.

1 INTRODUCTION

The Internet of Things (IoT) encompasses a vast ecosystem of heterogeneous devices (Dhanda et al., 2020), ranging from high-performance gateways to RFID tags and application-specific integrated circuits (ASICs). In industrial contexts, such as smart metering and infrastructure monitoring, ensuring the confidentiality and integrity of transmitted data is a foundational security requirement (Rana et al., 2022). Because wireless transmissions are inherently susceptible to attacks, the unauthorized manipulation of operational data (e.g., charge consumption metrics) can lead to severe economic and physical consequences.

Symmetric encryption is the standard mechanism for providing confidentiality and integrity. However, established schemes designed for general-purpose hardware often impose computational and energy overheads that exceed the stringent resource budgets of low-end devices, such as low CPU clock frequencies (4–16 MHz), scarce memory, and the requirement for multi-year battery autonomy. These constraints have driven the emergence of *Lightweight Cryptography* (LWC) as a dedicated research field.

The existing literature on LWC is extensive. Design methodologies are surveyed in (Buchanan et al., 2017), while (Sevin and Mohammed, 2021) provide benchmarks for 50 block ciphers on 8-bit Atmel AVR platforms. (Dhanda et al., 2020) expanded this scope to include stream ciphers, hash functions, and Elliptic Curve Cryptography (ECC), and the FELICS framework (Dinu et al., 2015) introduced a standardized open-source environment for comparative benchmarking. This research effort culminated in 2023 with NIST's selection of the Ascon family as the official LWC standard (Turan et al., 2025).

Experimental data targeting 16-bit architectures remain sparse in the literature. Prior studies such

[a] https://orcid.org/0000-0003-0755-3444
[b] https://orcid.org/0009-0000-7588-7169

Miculan, M., Paier, M. and Plozner, J.
Experimental Evaluation of Lightweight Encryption Algorithms on 16-bit Microcontrollers.
DOI: 10.5220/0015189200004103

In *Proceedings of the 23rd International Conference on Security and Cryptography (SECRYPT 2026) - Volume 1*, pages 228-239
ISBN: 978-989-758-858-7; ISSN: 2184-7711

as (Buhrow et al., 2014) and (Cazorla et al., 2015) have benchmarked lightweight cryptographic primitives on the TI MSP430 platform; our work extends these efforts through a more rigorous and comprehensive methodology, measuring cycle-accurate execution timing and capturing more precise energy consumption metrics. Additionally, we include newer lightweight ciphers, such as QARMAv2, which were not covered in earlier studies. This allows us to offer updated insights into the performance trade-offs of modern LWC primitives on 16-bit architectures, complementing and building upon the existing literature.

This paper presents the following contributions:

1. **Comprehensive Benchmarking:** We evaluate ten algorithms, including both block and stream ciphers, using the TI MSP430FR6989 as a reference 16-bit microcontroller.
2. **High-Precision Methodology:** We employ a dual-instrument measurement approach, utilizing a 100 MHz oscilloscope for cycle-accurate execution timing and a hardware-level current profiler, thereby eliminating the software jitter inherent in timer-based measurements.
3. **Multi-Objective Optimization:** Each algorithm is analyzed under two distinct compilation profiles—speed-optimized and size-optimized—to provide developers with a granular view of performance trade-offs.
4. **Architectural Analysis:** We provide concrete algorithm recommendations based on a rigorous security-to-performance trade-off analysis tailored for the MSP430 architecture.
5. **Open-Source Availability:** Our experimental suite and C implementations are available at: https://github.com/cysecud/crypto-lw.

The rest of the paper is structured as follows. Section 2 provides some background on lightweight ciphers. Section 3 introduces the ten benchmarked algorithms. Section 4 describes the experimental setup and measurement methodology. Section 5 presents and analyses the results. Section 6 concludes with recommendations and future directions.

2 SYMMETRIC ENCRYPTION AND LIGHTWEIGHT CRYPTOGRAPHY

A symmetric cipher defines encryption $E_K(P) = C$ and decryption $D_K(C) = P$ algorithms, both parameterised by a secret key K.

Hundreds ciphers have been proposed, but they all fall into two categories: *block ciphers* and *stream ciphers*. Block ciphers operate on fixed-length blocks of b bits and can be modelled as a keyed permutation $F : \{0,1\}^k \times \{0,1\}^b \to \{0,1\}^b$, where k is the key length. The main architectural families are *Substitution-Permutation Networks* (SPN), which apply alternating substitution (S-boxes) and permutation layers (Katz and Lindell, 2015); *Feistel networks*, which split the block into two halves and apply a potentially non-invertible round function (Feistel, 1973); and *Add-Rotate-XOR* (ARX) designs, which use only modular addition, bitwise rotation, and XOR (Beaulieu et al., 2013).

On the other hand, stream ciphers encrypt data bit-by-bit (or byte-by-byte) by XORing the plaintext with a pseudorandom keystream generated from the key and a nonce. They are typically based on Linear Feedback Shift Registers (LFSRs) or *sponge* constructions (Katz and Lindell, 2015).

Lightweight Cryptography (LWC) is designed for systems with extremely limited resources, like sensors networks or embedded systems (White, 2024). Their key constraints are: limited RAM and non-volatile storage (ROM/Flash/FRAM); slow processors (4–16 MHz); battery-powered operation requiring minimal current draw; and real-time guarantees. For these reasons, LWC trades reduced security margins for drastically lower resource consumption (Buchanan et al., 2017). Typical design strategies include: smaller block and key sizes; fewer rounds; use of only ARX operations; and hardware-friendly bit-sliced S-boxes. Importantly, the notion of "lightweight" is context-dependent: a cipher efficient in hardware (measured in Gate Equivalents, GE) may be inefficient in software (measured in operations), and vice versa (Buchanan et al., 2017).

ISO/IEC 29192 provides international LWC standards: part 2 covers block ciphers (including CLEFIA and LEA) (ISO, 2019). In 2023, NIST selected the Ascon family as the US LWC standard after a multi-year evaluation process (Turan et al., 2025).

3 BENCHMARKED ALGORITHMS

We selected ten algorithms that represent the primary LWC design paradigms and current standardization landscape (Table 1). All implementations were written in C, adhering strictly to the pseudocode provided in the original publications (reproduced in Appendix 6). Our analysis focuses on *encryption* performance, as this typically constitutes the primary bot-

Table 1: Benchmarked algorithms summary.

Algorithm	Type	Structure	Block/Key (bits)	Standard
AES-128	Block	SPN	128/128	FIPS/NIST
CLEFIA-128	Block	Feistel	128/128	ISO 29192-2
LEA-128	Block	ARX	128/128	ISO 29192-2
SPARX (2 variants)	Block	ARX (LTS)	64/128,128/128	—
SPECK (3 variants)	Block	ARX	32/64,64/128,128/128	—
PRINCE	Block	SPN (α-refl.)	64/128	—
QARMAv2-64	Block	SPN (α-refl.)	64/128	—
Ascon-128	Stream	Sponge	(128)/128	NIST LWC
ChaCha20	Stream	ARX	—/256	RFC 8439
Hummingbird-2	Stream	Hybrid	—/128	—

tleneck in embedded sensing scenarios.

AES (National Institute of Standards and Technology (US), 2023) is the NIST/FIPS block cipher standard based on an SPN with SubBytes, ShiftRows, MixColumns, and AddRoundKey transformations over 10 rounds (128-bit key). It represents our baseline for security and performance comparison.

CLEFIA (Shirai et al., 2007) is a Sony-designed 128-bit block cipher using a 4-branch generalised Feistel network with a Diffusion Switching Mechanism and two distinct S-boxes for enhanced resistance against differential and linear attacks. It is standardised in ISO/IEC 29192-2 (ISO, 2019).

LEA (Hong et al., 2014) (Lightweight Encryption Algorithm) is an ARX cipher operating on 32-bit words with 24 rounds. Its simple structure (XOR, modular addition, and rotation) targets efficient software implementation on common processors and is also standardised in ISO/IEC 29192-2 (ISO, 2019).

SPARX (Dinu et al., 2016) is an ARX cipher family that achieves *provable* resistance against differential and linear cryptanalysis via the Long Trail Strategy (LTS), a design paradigm introduced by its authors as a dual to the Wide Trail Strategy, that is at the basis of many S-box based ciphers, including AES. We evaluate different block sizes: both SPARX-64/128 and SPARX-128/128.

SPECK (Beaulieu et al., 2013) is an NSA-designed ARX cipher with multiple block/key size variants. We evaluate three variants: 32/64, 64/128, and 128/128. SPECK is explicitly optimised for software performance.

PRINCE (Borghoff et al., 2012) is a 64-bit hardware-optimised block cipher designed for minimum latency using an α-*reflection* construction: decryption is equivalent to encryption with a related key, enabling a shared circuit for both operations.

QARMAv2 (Avanzi et al., 2023) is a tweakable block cipher based on the same reflection principle as PRINCE, designed for memory encryption with configurable round count ($r \in \{4,6,7,9\}$), allowing a security/performance trade-off.

Ascon (Turan et al., 2025) is an *Authenticated Encryption with Associated Data* (AEAD) scheme selected as the NIST LWC standard. We evaluate both the bare encryption mode and the full AEAD variant.

ChaCha20 (Nir and Langley, 2018) is a 256-bit ARX stream cipher widely deployed in TLS, SSH, and IPsec. Its 64-bit internal state and 32-bit operations explicitly target 32-bit and 64-bit processors.

Hummingbird-2 (Engels et al., 2012) is a hybrid stream/block cipher targeting 16-bit word sizes, in principle well suited to simpler microcontroller architectures. Also here we evaluate both bare encryption and the AEAD variant.

4 EXPERIMENTAL SETUP

4.1 Hardware Platform

To ensure practical relevance, we conducted our benchmarks on the TI MSP430FR6989 (Texas Instruments, 2018; Texas Instruments, 2020). This 16-bit RISC microcontroller is a staple of the IIoT and smart metering sectors, making it an ideal candidate for evaluating lightweight cryptographic primitives.

The MSP430 is characterized by its ultra-low-power profile and specific memory constraints:

CPU: 16-bit RISC processor operating at a maximum frequency of 16 MHz.

Memory: 2 kB of RAM and 128 kB of Ferroelectric RAM (FRAM). FRAM is a non-volatile memory technology that offers read/write speeds and endurance comparable to SRAM, distinguishing it from traditional Flash.

Power Consumption: The device exhibits exceptional energy efficiency, drawing approximately 100 μA/MHz in *Active Mode* and dropping to 1 μA in *Low Power Mode 3* (LPM3).

The platform's architecture imposes significant constraints on cryptographic implementations. While it natively supports 8-bit and 16-bit operations, 32-bit and 64-bit calculations require software emulation, leading to substantial computational overhead (Texas Instruments, 1998).

A critical bottleneck for ARX ciphers is the lack of a hardware *barrel shifter*. Consequently, bit-rotation operations (which are typically $O(1)$ on modern desktop CPUs) require multiple single-bit shift instructions on this platform. This significantly penalizes ciphers optimized for 32-bit or 64-bit word sizes.

For our experiments, the CPU main clock (MCLK) was fixed at 4 MHz using the internal Digitally Controlled Oscillator (DCO) with a clock divider of 1. The stack size was limited to 160 bytes to simulate a strictly constrained operating environment.

Code was *cross-compiled* on a development host and deployed via the TI MSP-FET Flash Emulation Tool using a JTAG interface. The development toolchain was restricted to C/C++ with minimal library support to ensure deterministic performance and a small binary footprint.

4.2 Measurement Instrumentation

The execution time was measured using a SIGLENT SDS1104X-E digital oscilloscope (100 MHz bandwidth, 4 channels). A dedicated GPIO test pin is toggled by the firmware after each (repeated) cipher invocation, producing a square wave whose pulse width directly corresponds to the execution time of each encryption operation. This approach eliminates timer interrupt overhead and software timestamping artefacts entirely, since any latency introduced by the GPIO toggle is constant and negligible.

Current consumption was measured using a Nordic Semiconductor Power Profiler Kit II (PPK2), configured as an ammeter in series between a 3.6 V battery and the MSP430 target. Real-time current samples are streamed to the nRF Connect Power Profiler application, reporting the mean current consumption in 1 second increments during the benchmark window. To prevent PPK2 measurements from being influenced by the debugger supply, the MSP-FET is physically disconnected during measurements.

Memory footprint (FRAM and RAM) is reported by TI Code Composer Studio after linking and loading, subtracting the overhead of the bare test harness.

4.3 Compilation and Test Conditions

All programs were compiled using the TI Optimising C/C++ Compiler (Texas Instruments, 2021) with the maximum optimisation level (`-O4`: Whole Program Optimisation with link-time optimisation).

A critical design parameter is the trade-off between code size and execution speed, controlled by the `--opt_for_speed` flag (Texas Instruments, 2021): level 5 aggressively inlines and unrolls loops (favouring speed at the expense of FRAM); level 0 minimises code size (favouring a smaller footprint at the cost of some slowdown). We report the results for both settings, leaving to practitioners the choice of determining the best option for whichever constraint is tighter.

All ciphers were tested under identical conditions on the same physical device. No dynamic memory allocation is used in any implementation. Only the encryption operation was benchmarked, as this is the dominant workload in ultra-constrained IoT sensor nodes: these devices, having minimal computational resources, limited memory, and strict power budgets, are typically deployed in unidirectional data-flow scenarios. In such contexts, data is transmitted upward to a better-resourced gateway, with no requirement for decryption, rekeying, or over-the-air updates. Furthermore, nowadays many block ciphers are deployed in stream-like modes such as CTR (Counter Mode) or GCM (Galois Counter Mode), where the encryption process itself generates a keystream that is XORed with the plaintext. In these modes, decryption is functionally identical to encryption: the same keystream is regenerated and XORed with the ciphertext to recover the plaintext.

The benchmarks in this study were conducted exclusively on the TI MSP430FR6989, which utilizes FRAM for non-volatile storage. FRAM offers lower write latency and energy overhead compared to traditional flash memory, as it does not require charge pumps or erase-before-write operations. In practice, this means that charge consumption measured in our experiments may not directly translate to MSP430 variants that use flash memory instead.

4.4 Metrics

Three primary metrics are reported:

- **Encryption rate** (kB/s): normalises throughput across ciphers with different block sizes.

  $$\text{Rate} = \text{Block size [B]}/\text{Encryption time [ms]}$$

- **FRAM footprint** (bytes): code memory consumed by the cipher implementation (test harness overhead subtracted).
- **Charge consumption** (nAh/kB): a single scalar summarising the electric charge cost per unit of encrypted data, combining throughput and current

draw.

$$\text{Charge Consumption} = \frac{\text{Current } [\mu\text{A}]}{3.6 \cdot \text{Rate } [\text{kB/s}]}$$

RAM consumption is not tabulated separately as all implementations use only the static activation stack (160 bytes for all ciphers, no dynamic allocation).

5 RESULTS AND ANALYSIS

The source code used for the experimental evaluation is at https://github.com/cysecud/crypto-lw.

5.1 Block Ciphers

Table 2 presents detailed measurements for all block ciphers. Figure 1 visualises FRAM footprint and charge consumption for software-oriented and hardware-oriented ciphers.

We provide a side-by-side comparisons across different security levels to allow practitioners to choose cryptographic primitives that best align with their specific threat models and operational constraints. This approach ensures that one can make informed decisions based on their unique priorities: for example, when evaluating SPECK-64/128, the data reveals that SPECK-128/128 offers nearly identical performance, making it a valuable choice for applications where higher security margins are desired without sacrificing efficiency. Conversely, in scenarios where security requirements are less strict and performance is critical, such as in resource-constrained sensor nodes, our results immediately shows that SPECK-32/64 delivers significantly higher throughput. By presenting these trade-offs explicitly, we enable system designers to balance security, performance, and resource usage in a way that best suits their demands.

AES. AES performs quite well on the MSP430 for a non-lightweight cipher, with throughput reaching 5.56 kB/s. Size-optimization proves highly effective, slashing FRAM usage by 59% while maintaining nearly identical throughput (+0.7%) and actually lowering current consumption by 14%. This efficiency gain suggests that more compact code may be improving the execution efficiency of the RISC pipeline. These results, combined with AES's unparalleled standardization, confirm its status as a high-security baseline for constrained 16-bit systems.

CLEFIA. CLEFIA's dual S-boxes (requiring large lookup tables) and matrix multiplications over GF(2^8) result in the largest FRAM footprint of all software-oriented ciphers (9580 B speed-optimised) and the lowest throughput (2.33 kB/s), yielding the worst charge consumption among standardised algorithms (54.35 nAh/kB). Size-optimised compilation halves the FRAM cost (3610 B) but further degrades throughput. Despite ISO standardisation (ISO, 2019) and solid cryptanalytic confidence (Biryukov and Nikolic, 2019), CLEFIA is not recommended for memory-constrained MSP430 applications.

LEA. LEA performs well among block ciphers: its ARX operations on 32-bit words deliver a charge consumption of only 21.02–21.35 nAh/kB (a 25% improvement over AES) with a FRAM footprint 66% smaller than AES under speed-optimised compilation (994 vs. 2875 B). Size-optimised LEA reaches a minimal 954 B of FRAM and the lowest current draw of all algorithms (354 μA). The slight penalty from 32-bit rotation operations (emulated as multi-instruction sequences on the 16-bit MSP430) is offset by the simplicity of the overall structure. ISO/IEC standardisation (ISO, 2019) provides strong security assurances.

SPARX. SPARX-64/128 achieves the highest throughput among all block ciphers (7.49 kB/s), benefiting from its ARX structure on 16-bit words that directly maps to the MSP430's native word size. Consumption (21.32 nAh/kB) is comparable to LEA. However, current draw is above average (≈575 μA). The LTS-based provable security against differential and linear cryptanalysis (Dinu et al., 2016) is a significant advantage over empirically analysed ARX ciphers. SPARX-64/128 is a strong alternative to LEA when throughput is the primary constraint.

SPECK. SPECK-32/64 achieves the best absolute throughput (11.43–11.56 kB/s) and the lowest charge consumption of all block ciphers (13.95–14.20 nAh/kB), with a minimal FRAM footprint (332 B size-optimised). These figures reflect the direct alignment of SPECK's 16-bit word operations with the MSP430 architecture. However, SPECK's NSA provenance and the controversy surrounding the Dual_EC_DRBG backdoor affair have led to its rejection for ISO standardisation and removal from the Linux kernel (v. 4.20). SPECK-32/64 is recommended only in contexts where performance is the most important feature and the NSA-provenance risk is considered acceptable. The 64/128 and 128/128 variants, operating on 32-bit words, are slower on the MSP430, while offering larger security margins.

PRINCE and QARMAv2. These hardware-oriented ciphers deliver very poor performance when implemented in software. PRINCE encrypts at only 0.17–0.33 kB/s with a charge consumption of 365–689 nAh/kB — 16–27× worse than LEA. QARMAv2 is even slower, reaching 1602 nAh/kB at r=9. The

Table 2: Block cipher benchmark results on MSP430FR6989 at 4 MHz. Top row of each entry: `--opt_for_speed=5`; bottom row: `--opt_for_speed=0`. Green: best in class; Red: worst in class.

Algorithm		**Init.** (ms)	**Exec.** (ms)	**FRAM** (B)	**Current** (μA)	**Rate** (kB/s)	**Charge** (nAh/kB)
AES-128	speed	1.44	2.88	2875	568	5.56	28.40
	size	1.46	2.98	1169	489	5.37	25.30
CLEFIA-128	speed	8.12	6.88	**9580**	455	2.33	54.35
	size	9.88	10.24	3610	461	1.56	81.96
LEA-128	speed	6.32	3.36	994	366	4.76	21.35
	size	7.00	3.42	954	**354**	4.68	21.02
SPARX-64/128	speed	0.87	1.07	870	575	7.49	21.32
	size	0.62	1.21	594	575	6.60	24.20
SPARX-128/128	speed	3.10	2.62	1356	565	6.11	25.70
	size	2.23	3.33	856	576	4.80	33.30
SPECK-32/64	speed	0.32	0.35	546	574	11.43	**13.95**
	size	0.37	0.35	**332**	**591**	**11.56**	14.20
SPECK-64/128	speed	1.60	1.61	1186	365	4.97	20.40
	size	1.74	1.62	402	385	4.94	21.66
SPECK-128/128	speed	3.86	3.98	1436	389	4.02	26.88
	size	3.94	3.90	644	389	4.10	26.34
PRINCE	speed	0.00	24.00	5144	439	0.33	365.83
	size	0.00	47.40	1586	419	0.17	689.60
QARMAv2 (r=4)	speed	0.00	41.60	5086	404	0.19	583.56
	size	0.00	52.00	2276	411	0.15	742.08
QARMAv2 (r=6)	speed	0.00	60.80	5164	406	0.13	857.11
	size	0.00	76.40	2278	408	0.10	1082.33
QARMAv2 (r=7)	speed	0.00	70.80	5200	399	0.11	980.88
	size	0.00	88.00	2276	411	0.09	1255.83
QARMAv2 (r=9)	speed	0.00	90.00	5274	403	0.09	1259.38
	size	0.00	112.00	2276	412	**0.07**	**1602.22**

root cause is their 4-bit S-box operations and matrix multiplications, which require many instructions on a byte/word-aligned architecture. Both of these ciphers should not be deployed in software on MSP430.

5.2 Stream Ciphers

Table 3 reports stream cipher results; Figure 2 visualises the key metrics.

Ascon. Ascon's 64-bit word operations are emulated on the 16-bit MSP430, increasing execution time compared to 32/64-bit platforms. Nevertheless, Ascon delivers acceptable results across all metrics (40.57 nAh/kB, 1902 B FRAM), and is particularly good in charge consumption. NIST standardisation (Turan et al., 2025) provides the strongest available security guarantee. The AEAD mode adds roughly 30 nAh/kB overhead, but provides authenticated encryption a significant advantage for IoT applications where message integrity must also be verified.

ChaCha20. Despite its excellent security properties and wide adoption in higher-performance systems, ChaCha20 performs poorly on the MSP430 platform: 0.64–0.94 kB/s throughput and a charge consumption of 119–203 nAh/kB, 3–5 times worse than Ascon. The reason is that ChaCha20 operates internally on 32-bit words in a 4×4 matrix, requiring extensive software emulation on the 16-bit MSP430.

Hummingbird-2. Designed specifically for 16-bit CPUs, Hummingbird-2 achieves the best throughput among stream ciphers (3.29–3.40 kB/s) with a modest FRAM footprint. However, its current draw (563–569 μA) is the highest of all stream ciphers. Having received limited cryptanalytic attention (Fan and Gong, 2012; Chai and Gong, 2012), this cipher inspires less confidence than standardized alternatives.

5.3 Results Analysis and Discussion

Our findings reveal that native 16-bit word-size alignment is the primary determinant of performance on the MSP430. We identify a clear efficiency hierarchy based on word size:

- **Optimal:** Ciphers optimized for 16-bit operations

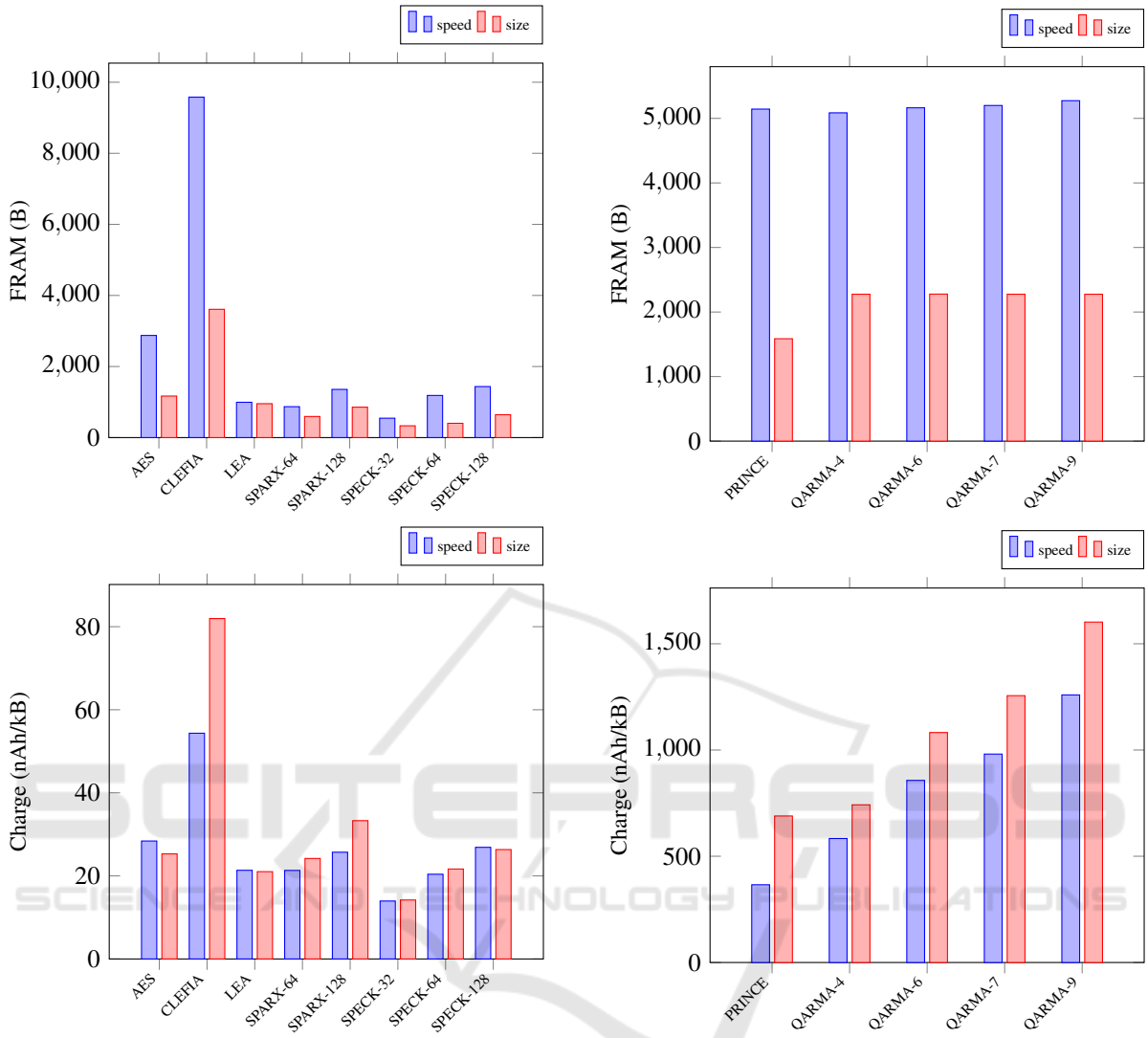

Figure 1: Software and hardware oriented block ciphers: FRAM footprint and charge consumption.

(SPECK-32/64, SPARX-64/128, Hummingbird-2) exhibit the highest throughput.

- **Moderate:** 32-bit primitives (LEA, SPECK-64/128) incur increased overhead, as the absence of a hardware barrel shifter significantly inflates the cost of rotation operations.
- **Sub-optimal:** 64-bit designs (Ascon, ChaCha20) face a severe performance penalty due to the necessity of software-level word emulation.
- **Inefficient:** Ciphers relying on nibble-level operations (PRINCE, QARMAv2) demonstrate the lowest efficiency on this architecture.

Our analysis indicates that word-size alignment exerts a more significant influence on performance than other design paradigms, such as S-box vs. ARX structures or total round count. Crucially, this effect is architecture-specific, implying that results derived from 8-bit or 32-bit platforms cannot be extrapolated to the MSP430 without substantial correction.

In the context of LWE, the FELICS (*Fair Evaluation of Lightweight Cryptographic Systems*) framework (Dinu et al., 2015) is a reference for benchmarking software implementations of lightweight cryptographic primitives. While a direct numerical comparison of the present work with FELICS is precluded by differences in clock frequencies, operating voltages and measurement methodologies, the qualitative trends remain instructive. For instance, (Sevin and Mohammed, 2021) identifies SPECK and LEA as highly efficient also on 8-bit AVR platforms, which aligns with our findings. Similarly, CLEFIA and AES maintain consistent relative rankings across both architectures. However, significant performance dis-

Table 3: Stream cipher benchmark results on MSP430FR6989 at 4 MHz. See Table 2 for the legend.

Algorithm		**Init.** (ms)	**Exec.** (ms)	**FRAM** (B)	**Current** (μA)	**Rate** (kB/s)	**Charge** (nAh/kB)
Ascon	speed	8.60	11.40	1902	410	2.81	**40.57**
	size	9.20	12.30	1764	418	2.60	44.63
Ascon+AEAD	speed	19.90	19.90	2090	410	1.61	70.82
	size	21.40	21.40	1952	419	1.50	77.84
ChaCha20	speed	0.05	34.20	**2556**	**401**	0.94	119.05
	size	0.05	50.00	1564	468	**0.64**	**203.13**
Hummingbird-2	speed	2.44	9.40	2172	**569**	**3.40**	46.43
	size	2.48	9.72	**1464**	564	3.29	47.59
HB-2+AEAD	speed	2.44	16.40	2396	565	1.95	80.43
	size	2.48	17.20	1666	563	1.86	84.06

Table 4: Algorithm recommendations by use case (notice that security levels differ).

Use case	**Recommended algorithm**	**Notes**
Best overall balance	**LEA-128**	ISO standard, low energy, compact
Highest throughput	**SPECK-32/64**	NSA-provenance caveat
Authenticated encryption	**Ascon-AEAD 128**	NIST standard
Widely adopted and implemented	**AES-128**	better suited when hardware coprocessor available
Provably secure ARX	**SPARX-64/128**	
Hardware deployment	**PRINCE / QARMAv2**	not software based

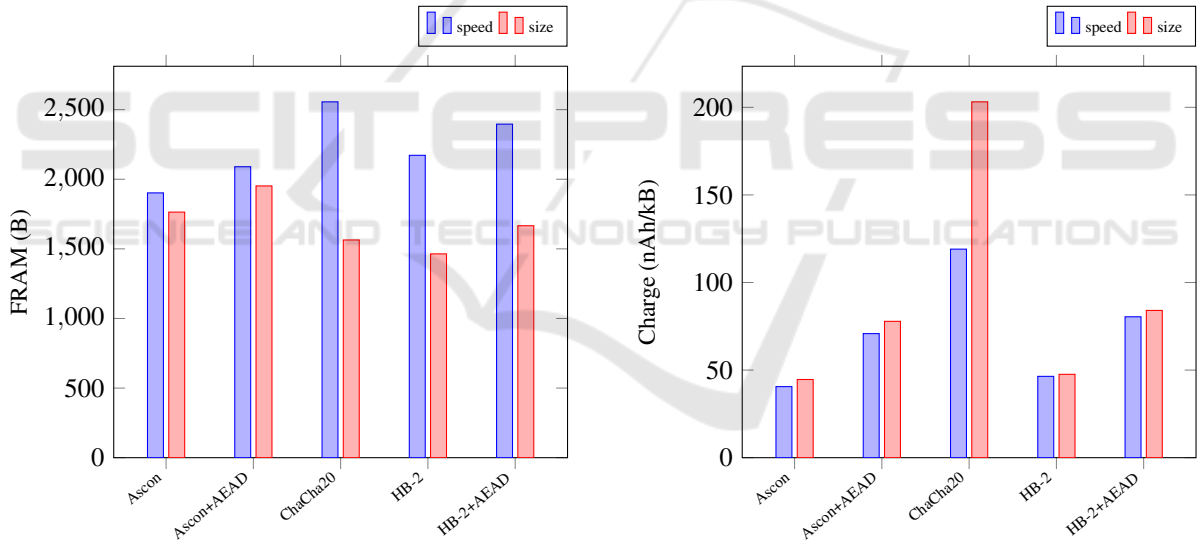

Figure 2: Stream ciphers: FRAM footprint and charge consumption.

crepancies emerge for specific primitives. PRINCE incurs only a 2× overhead relative to AES on the AVR—compared to 15× on the MSP430—due to the AVR's superior support for bit-level rotations. Conversely, 16-bit ARX designs (e.g., SPECK-32/64, SPARX-64) favor the MSP430, which executes 16-bit Addition/XOR operations in a single cycle, whereas the AVR requires multiple 8-bit instructions.

These results underscore that lightweight cryptographic selection must be meticulously tailored to the target ISA, as no single algorithm is universally optimal across all constrained environments.

6 CONCLUSIONS

We provided a comprehensive hardware-level benchmarking of ten lightweight cryptographic algorithms on the MSP430FR6989 16-bit microcontroller. By evaluating software-only implementations, we established a standardized baseline to assess the inherent computational efficiency of each design paradigm.

Our main findings are summarized as follows:

- LEA-128 is the optimal choice for the MSP430, delivering 25% better energy efficiency and a 66% smaller code footprint (speed-optimized) than AES. Its status as an ISO/IEC standard further reinforces its suitability for industrial IoT.
- SPECK-32/64 achieves the highest raw performance, with a throughput of 11.5 kB/s and a charge consumption of 14 nAh/kB. While highly efficient, its adoption may be restricted in certain regulated sectors due to its provenance.
- Among authenticated and stream ciphers, Ascon-128 stands out as the most balanced option, particularly following its selection as the NIST LWC standard. Conversely, ChaCha20 proved unsuitable for this 16-bit architecture due to the heavy overhead of 64-bit word emulation.
- Hardware-oriented ciphers like PRINCE and QARMAv2 exhibit poor software performance on this RISC architecture, with energy consumption reaching 1600 nAh/kB.

Ultimately, we have demonstrated that native word-size alignment is the primary driver of performance. Ciphers designed for 16-bit operations leverage the MSP430 architecture most effectively, while those relying on 64-bit or nibble-level operations incur severe penalties. Furthermore, our results indicate that size-optimized compilation often provides the most advantageous trade-off, slashing FRAM usage with negligible throughput loss and, for certain ciphers like AES, even reducing current draw.

In Table 4 we summarise our final recommendations, according to different use cases.

Future Work. While this study focused on the MSP430FR series, the insights could be applicable to the wider 16-bit RISC landscape. Future research will expand this evaluation to include additional primitives such as Simon, GIFT, and Elephant, as well as an analysis of decryption-side performance. Also, we plan to explicitly compare FRAM and flash-based MSP430 models to quantify the effects the non-volatile memory architecture has on charge consumption. Finally, we intend to investigate the scalability of these results across 8-bit (AVR) and 32-bit (ARM Cortex-M0) platforms to establish a definitive cross-architectural performance baseline.

ACKNOWLEDGEMENTS

The authors thank Maddalena SpA for providing measurement instrumentation and industrial context, and Luca Campa for valuable advice throughout this work.

REFERENCES

Avanzi, R., Banik, S., Dunkelman, O., Eichlseder, M., Ghosh, S., Nageler, M., and Regazzoni, F. (2023). The QARMAv2 family of tweakable block ciphers. IACR ePrint 2023/929.

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., and Wingers, L. (2013). The SIMON and SPECK families of lightweight block ciphers. IACR ePrint 2013/404.

Biryukov, A. and Nikolic, I. (2019). Security analysis of the block cipher CLEFIA. *Final Report for CRYPTREC*.

Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E. B., Knezevic, M., Knudsen, L. R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S. S., and Yalçın, T. (2012). Prince – a low-latency block cipher for pervasive computing applications. In *Advances in Cryptology – ASIACRYPT 2012*, pages 208–225. Springer.

Buchanan, W. J., Li, S., and Asif, R. (2017). Lightweight cryptography methods. *Journal of Cyber Security Technology*, 1(3-4):187–201.

Buhrow, B., Riemer, P., Shea, M., Gilbert, B., and Daniel, E. (2014). Block cipher speed and energy efficiency records on the MSP430: System design trade-offs for 16-bit embedded applications. In *International Conference on Cryptology and Information Security in Latin America*, pages 104–123. Springer.

Cazorla, M., Gourgeon, S., Marquet, K., and Minier, M. (2015). Survey and benchmark of lightweight block ciphers for MSP430 16-bit microcontroller. *Security and Communication Networks*, 8(18):3564–3579.

Chai, Q. and Gong, G. (2012). A cryptanalysis of HummingBird-2: The differential sequence analysis. IACR ePrint 2012/233.

Dhanda, S. S., Singh, B., and Jindal, P. (2020). Lightweight cryptography: A solution to secure IoT. *Wireless Personal Communications*, 112:1947–1980.

Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., and Biryukov, A. (2016). Design strategies for ARX with provable bounds: Sparx and LAX. In *Advances in Cryptology – ASIACRYPT 2016*, pages 484–513. Springer.

Dinu, D.-D., Biryukov, A., Großschädl, J., Khovratovich, D., Corre, Y. L., and Perrin, L. (2015). Felics – fair evaluation of lightweight cryptographic systems. In *Proceedings of the NIST Workshop on Lightweight Cryptography 2015*.

Engels, D., Saarinen, M.-J. O., Schweitzer, P., and Smith, E. M. (2012). The Hummingbird-2 lightweight authenticated encryption algorithm. In *RFID Security and Privacy*, pages 19–31. Springer.

Fan, X. and Gong, G. (2012). On the security of Hummingbird-2 against side channel cube attacks. In *Research in Cryptology*, pages 18–29.

Feistel, H. (1973). Cryptography and computer privacy. *Scientific American*, 228(5):15–23.

Hong, D., Lee, J.-K., Kim, D.-C., Kwon, D., Ryu, K. H., and Lee, D.-G. (2014). LEA: a 128-bit block cipher for fast encryption on common processors. In *Information Security Applications*, pages 3–27. Springer.

ISO (2019). Information security — lightweight cryptography — part 2: Block ciphers. Technical Report IEC 29192-2, Geneva.

Katz, J. and Lindell, Y. (2015). *Introduction to Modern Cryptography*. CRC Press, 2 edition.

National Institute of Standards and Technology (US) (2023). Advanced encryption standard (AES). Technical report, Washington, D.C.

Nir, Y. and Langley, A. (2018). ChaCha20 and Poly1305 for IETF Protocols. RFC 8439.

Rana, M., Mamun, Q., and Islam, R. (2022). Lightweight cryptography in IoT networks: A survey. *Future Generation Computer Systems*, 129:77–89.

Sevin, A. and Mohammed, A. A. O. (2021). A survey on software implementation of lightweight block ciphers for iot devices. *Journal of Ambient Intelligence and Humanized Computing*, 14(3):1801–1815.

Shirai, T., Shibutani, K., Akishita, T., Moriai, S., and Iwata, T. (2007). The 128-bit blockcipher CLEFIA (extended abstract). In *Fast Software Encryption*, pages 181–195. Springer.

Texas Instruments (1998). *MSP430 Family Instruction Set Summary*.

Texas Instruments (2018). *MSP430FR698x, MSP430FR598x Mixed-Signal Microcontrollers Datasheet*. Revisione D.

Texas Instruments (2020). *MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User's Guide*. Revisione P.

Texas Instruments (2021). *MSP430 Optimizing C/C++ Compiler — User's Guide*.

Turan, M. S., McKay, K. A., Chang, D., Kang, J., and Kelsey, J. (2025). Ascon-based lightweight cryptography standards for constrained devices. NIST Special Publication SP 800-232.

White, E. (2024). *Making Embedded Systems*. O'Reilly Media, 2 edition.

APPENDIX

Ciphers Pseudocodes

Algorithm 1: AES pseudocode.

procedure AES(in, Nr, w)
  $state \leftarrow in$
  $state \leftarrow$ AddRoundKey($state, w[0 \cdots 3]$)
  for $round = 1$ **to** $Nr - 1$ **do**
    $state \leftarrow$ SubBytes($state$)
    $state \leftarrow$ ShiftRows($state$)
    $state \leftarrow$ MixColumns($state$)
    $state \leftarrow$ AddRoundKey($state, w[4 * round \cdots 4 * round + 3]$)
  end for
  $state \leftarrow$ SubBytes($state$)
  $state \leftarrow$ ShiftRows($state$)
  $state \leftarrow$ AddRoundKey($state, w[4 * Nr \cdots 4 * Nr + 3]$)
  return $state$
end procedure

Algorithm 2: CLEFIA pseudocode.

procedure CLEFIA$_r$(WK, RK, P)
  $T \leftarrow P_0 || (P_1 \oplus WK_0) || P_2 || (P_3 \oplus WK_1)$
  $T \leftarrow GFN_{4,r}(RK, T)$
  return $T_0 || (T_1 \oplus WK_2) || T_2 || (T_3 \oplus WK_3)$
end procedure

Algorithm 3: LEA pseudocode.

procedure LEA(P,RK)
  $X_0 \leftarrow P$
  for $i = 0$ **to** $r - 1$ **do**
    $X_{i+1}[0] \leftarrow LROT_9((X_i[0] \oplus RK_i[0]) \boxplus (X_i[1] \oplus RK_i[1]))$
    $X_{i+1}[1] \leftarrow RROT_5((X_i[1] \oplus RK_i[2]) \boxplus (X_i[2] \oplus RK_i[3]))$
    $X_{i+1}[2] \leftarrow RROT_3((X_i[2] \oplus RK_i[4]) \boxplus (X_i[3] \oplus RK_i[5]))$
    $X_{i+1}[3] \leftarrow X_i[0]$
  end for
  $C \leftarrow X_r$
  return C
end procedure

Algorithm 4: SPARX pseudocode.

procedure SPARX(p,k)
  $y \leftarrow p$
  for $s = 0$ **to** $n_s - 1$ **do**
    for $i = 0$ **to** $w - 1$ **do**
      for $r = 0$ **to** $r_a - 1$ **do**
        $y_i \leftarrow y_i \oplus k_r$
        $y_i \leftarrow A(y_i)$
      end for
      $k \leftarrow K_n(k)$
    end for
    $y \leftarrow \lambda_w(y)$
  end for
  $y \leftarrow y \oplus k$
  $c \leftarrow y$
  return c
end procedure

Algorithm 5: PRINCE pseudocode.

procedure PRINCE(m,k)
  $state \leftarrow P$
  $state \leftarrow state \oplus k_0$
  $state \leftarrow state \oplus k_1$
  $state \leftarrow state \oplus RC_0$
  for $r = 1$ **to** 5 **do**
    $state \leftarrow S(state)$
    $state \leftarrow M \times state$
    $state \leftarrow state \oplus RC_r$
    $state \leftarrow state \oplus k_1$
  end for
  $state \leftarrow S(state)$
  $state \leftarrow M' \times state$
  $state \leftarrow S^{-1}(state)$
  for $r = 6$ **to** 10 **do**
    $state \leftarrow state \oplus k_1$
    $state \leftarrow state \oplus RC_r$
    $state \leftarrow M^{-1} \times state$
    $state \leftarrow S^{-1}(state)$
  end for
  $state \leftarrow state \oplus RC_{11}$
  $state \leftarrow state \oplus k_1$
  $state \leftarrow state \oplus k_0'$
  $c \leftarrow state$
  return c
end procedure

Algorithm 6: SPECK pseudocode.

procedure SPECK(p,K)
  for $i = 0$ **to** $T - 2$ **do**
    $\ell_{i+m-1} \leftarrow (k_i \boxplus S^{-\alpha}(\ell_i)) \oplus i$
    $k_{i+1} \leftarrow S^{\beta}(k_i) \oplus \ell_{i+m-1}$
  end for
  $x \leftarrow p_1, y \leftarrow p_0$
  for $i = 0$ **to** $T - 1$ **do**
    $x \leftarrow (S^{-\alpha}(x) \boxplus y) \oplus k_i$
    $y \leftarrow S^{\beta}(y) \oplus x$
  end for
  $c \leftarrow x||y$
  return c
end procedure

Algorithm 7: QARMAv2 pseudocode.

procedure QARMAv2$_r$(K,W,T,P)
  $t_0 \leftarrow T_0, t_1 \leftarrow T_1$
  $k_0 \leftarrow K_0, k_1 \leftarrow K_1$
  $\mathcal{S} \leftarrow P \oplus k_0$
  $\mathcal{S} \leftarrow S(\mathcal{S})$
  for $i = 1$ **to** r **do**
    $\mathcal{S} \leftarrow \mathcal{S} \oplus k_{(i \bmod 2)} \oplus t_{(i \bmod 2)} \oplus c_i$
    $\mathcal{S} \leftarrow (S \circ M \circ \tau)(\mathcal{S})$
    if $i \equiv 1 \mod 2$ **then**
      $t_1 \leftarrow \varphi(t_1)$
    else
      $t_0 \leftarrow \varphi^{-1}(t_0)$
    end if
  end for
  $k_0 \leftarrow o(k_0) + \alpha, k_1 \leftarrow o^{-1}(k_1) + \beta$
  $\mathcal{S} \leftarrow \tau(\mathcal{S})$
  $\mathcal{S} \leftarrow M(\mathcal{S} \oplus W_{(r+1 \bmod 2)}) \oplus W_{(r \bmod 2)}$
  $\mathcal{S} \leftarrow \tau^{-1}(\mathcal{S})$
  for $i = r$ **down to** 1 **do**
    $\mathcal{S} \leftarrow (\tau^{-1} \circ M^{-1} \circ S^{-1})(\mathcal{S})$
    $\mathcal{S} \leftarrow \mathcal{S} \oplus k_{(i+1 \bmod 2)} \oplus t_{(i+1 \bmod 2)} \oplus c_i$
    if $(i > 1) \wedge (i \equiv 0 \mod 2)$ **then**
      $t_1 \leftarrow \varphi(t_1)$
    else
      $t_0 \leftarrow \varphi^{-1}(t_0)$
    end if
  end for
  $\mathcal{S} \leftarrow S^{-1}(\mathcal{S})$
  $C \leftarrow \mathcal{S} \oplus k_1$
  return C
end procedure

Algorithm 8: Ascon-AEAD128 pseudocode.

procedure ASCON(K,N,A,P)
  $IV \leftarrow$ 0x00001000808c0001
  $\mathcal{S} \leftarrow IV||K||N$
  $\mathcal{S} \leftarrow Ascon\text{-}p[12](\mathcal{S})$
  $\mathcal{S} \leftarrow \mathcal{S} \oplus K$
  if $|A| > 0$ **then**
    $A_m \leftarrow \text{PAD}(\tilde{A}_m)$
    for $i = 0$ **to** m **do**
      $\mathcal{S} \leftarrow Ascon\text{-}p[8]((\mathcal{S}_{[0:127]} \oplus A_i)||\mathcal{S}_{[128:319]})$
    end for
  end if
  $\mathcal{S} \leftarrow \mathcal{S} \oplus (0^{319}||1)$
  $\ell \leftarrow |\tilde{P}_n|$
  for $i = 0$ **to** $n-1$ **do**
    $\mathcal{S}_{[0:127]} \leftarrow \mathcal{S}_{[0:127]} \oplus P_i$
    $C_i \leftarrow \mathcal{S}_{[0:127]}$
    $\mathcal{S} \leftarrow Ascon\text{-}p[8](\mathcal{S})$
  end for
  $\mathcal{S}_{[0:127]} \leftarrow \mathcal{S}_{[0:127]} \oplus \text{PAD}(\tilde{P}_n)$
  $\tilde{C}_n \leftarrow S_{[0:\ell-1]}$
  $\mathcal{S} \leftarrow Ascon\text{-}p[12](\mathcal{S} \oplus (0^{128||}K||0^{64}))$
  $T \leftarrow \mathcal{S}_{[192:319]} \oplus K$
  return C,T
end procedure

Algorithm 9: ChaCha20 pseudocode (first part).

procedure QUARTERROUND(a,b,c,d)
  $a \leftarrow a \boxplus b$
  $d \leftarrow d \oplus a$
  $d \leftarrow d \lll 16$
  $c \leftarrow c \boxplus d$
  $b \leftarrow b \oplus c$
  $b \leftarrow b \lll 12$
  $a \leftarrow a \boxplus b$
  $d \leftarrow d \oplus a$
  $d \leftarrow d \lll 8$
  $c \leftarrow c \boxplus d$
  $b \leftarrow b \oplus c$
  $b \leftarrow b \lll 12$
end procedure
procedure ROUND(M)
  QUARTERROUND(M_0,M_4,M_8,M_{12})
  QUARTERROUND(M_1,M_5,M_9,M_{13})
  QUARTERROUND(M_2,M_6,M_{10},M_{14})
  QUARTERROUND(M_3,M_7,M_{11},M_{15})
  QUARTERROUND(M_0,M_5,M_{10},M_{15})
  QUARTERROUND(M_1,M_6,M_{11},M_{12})
  QUARTERROUND(M_2,M_7,M_8,M_{13})
  QUARTERROUND(M_3,M_4,M_9,M_{14})
end procedure

Algorithm 10: ChaCha20 pseudocode (second part).

procedure BLOCK(k,n,b_c)
  $M \leftarrow c||k||b_c||n$
  $M_{init} \leftarrow M$
  for $i = 1$ **to** 10 **do** ROUND(M)
  return $M \boxplus M_{init}$
end procedure
procedure CHACHA20(p,k,n,b_c)
  for $i = 0$ **to** m **do**
    $ks \leftarrow \text{BLOCK}(k,n,b_c+i)$
    $c_i \leftarrow p_i \oplus ks$
  end for
  return c
end procedure

Algorithm 11: Hummingbird-2 pseudocode.

procedure HUMMINGBIRD-2(P,K,IV)
  $R^{(0)} \leftarrow IV_1||IV_2||IV_3||IV_4||IV_1||IV_2||IV_3||IV_4$
  for $i = 0$ **to** 3 **do**
    $t_1 \leftarrow \text{WD16}(R_1^{(i)} \boxplus i,K_1,K_2,K_3,K_4)$
    $t_2 \leftarrow \text{WD16}(R_2^{(i)} \boxplus t_1,K_5,K_6,K_7,K_8)$
    $t_3 \leftarrow \text{WD16}(R_3^{(i)} \boxplus t_2,K_1,K_2,K_3,K_4)$
    $t_4 \leftarrow \text{WD16}(R_4^{(i)} \boxplus t_3,K_5,K_6,K_7,K_8)$
    $R_1^{(i+1)} \leftarrow (R_1^{(i)} \boxplus t_4) \lll 3$
    $R_2^{(i+1)} \leftarrow (R_2^{(i)} \boxplus t_1) \ggg 1$
    $R_3^{(i+1)} \leftarrow (R_3^{(i)} \boxplus t_2) \lll 8$
    $R_4^{(i+1)} \leftarrow (R_4^{(i)} \boxplus t_3) \lll 1$
    $R_5^{(i+1)},R_6^{(i+1)} \leftarrow R_5^{(i)} \oplus R_1^{(i+1)},R_6^{(i)} \oplus R_2^{(i+1)}$
    $R_7^{(i+1)},R_8^{(i+1)} \leftarrow R_7^{(i)} \oplus R_3^{(i+1)},R_8^{(i)} \oplus R_8^{(i+1)}$
  end for
  for $i = 0$ **to** n **do**
    $t_1 \leftarrow \text{WD16}(R_1^{(i)} \boxplus P_i,K_1,K_2,K_3,K_4)$
    $t_2 \leftarrow \text{WD16}(R_2^{(i)} \boxplus t_1,K_5 \oplus R_5^{(i)},K_6 \oplus R_6^{(i)},K_7 \oplus R_7^{(i)},K_8 \oplus R_8^{(i)})$
    $t_3 \leftarrow \text{WD16}(R_3^{(i)} \boxplus t_2,K_1 \oplus R_5^{(i)},K_2 \oplus R_6^{(i)},K_3 \oplus R_7^{(i)},K_4 \oplus R_8^{(i)})$
    $C_i \leftarrow \text{WD16}(R_4^{(i)} \boxplus t_3,K_5,K_6,K_7,K_8) \boxplus R_1^{(i)}$
    $R_1^{(i+1)},R_2^{(i+1)} \leftarrow R_1^{(i)} \boxplus t_3,R_2^{(i)} \boxplus t_1$
    $R_3^{(i+1)},R_4^{(i+1)} \leftarrow R_3^{(i)} \boxplus t_2,R_4^{(i)} \boxplus R_1^{(i)} \boxplus t_3 \boxplus t_1$
    $R_5^{(i+1)} \leftarrow R_5^{(i)} \oplus (R_1^{(i)} \boxplus t_3)$
    $R_6^{(i+1)} \leftarrow R_6^{(i)} \oplus (R_2^{(i)} \boxplus t_1)$
    $R_7^{(i+1)} \leftarrow R_7^{(i)} \oplus (R_3^{(i)} \boxplus t_2)$
    $R_8^{(i+1)} \leftarrow R_8^{(i)} \oplus (R_4^{(i)} \boxplus R_1^{(i)} \boxplus t_3 \boxplus t_1)$
  end for
  return C
end procedure