Comparative Performance and Memory Footprint Analysis of PRESENT-80 and AES-128 on an ESP32-C3 RISC-V IoT Microcontroller

Billy Samuel Setiawan – 18222039
Program Studi Sistem dan Teknologi Informasi,
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
bllysm23@gmail.com, 18222039@std.stei.itb.ac.id

***Abstract*—The growing use of Internet of Things devices motivates the evaluation of cryptographic algorithms under limited processing and memory resources. This paper compares the performance and memory footprint of PRESENT-80 and AES-128 on the ESP32-C3, a 32-bit RISC-V IoT microcontroller with hardware AES support. Three configurations are implemented and benchmarked: hardware-accelerated AES-128 through mbedTLS, pure software AES-128, and pure software PRESENT-80. All implementations are verified against official NIST and PRESENT test vectors prior to measurement. Results show that optimized software PRESENT-80 is functional and suitable for low-data-rate telemetry, but delivers lower throughput per byte than both software and hardware AES-128 on this platform, owing to its smaller 64-bit block size. PRESENT-80 also offers negligible memory savings compared with software AES-128 while providing a weaker 80-bit security level. These findings indicate that PRESENT-80 is technically viable but not a compelling alternative to AES-128 on the ESP32-C3. For practical deployments on similar RISC-V IoT microcontrollers, AES-128 remains the preferred choice due to higher throughput, stronger security, and hardware acceleration support.**

***Index Terms*—PRESENT-80, AES-128, RISC-V, ESP32-C3, IoT, lightweight cryptography, block cipher, benchmark**

I. Introduction

The expansion of Internet of Things (IoT) ecosystems has introduced large numbers of resource-constrained devices into security-sensitive environments, ranging from industrial sensors and smart meters to medical wearables and supply-chain trackers. These devices require cryptographic primitives that can provide adequate security within limited budgets of power, memory, and processing cycles [1].

This tension has historically motivated the development of dedicated lightweight cryptography standards. PRESENT, proposed by Bogdanov et al. in 2007, is among the most widely cited lightweight block ciphers. It was specifically designed for hardware implementation in ultra-constrained environments such as RFID tags, where its bit-oriented permutation layer can be implemented primarily as fixed wiring with negligible logic-gate cost [2]. AES-128, by contrast, is the dominant software and hardware cryptographic standard, designed for efficient implementation on byte-oriented processors and widely supported by dedicated hardware accelerators in modern embedded SoCs [3].

A critical question is: on a modern 32-bit RISC-V IoT microcontroller with hardware AES support, does PRESENT-80 offer any meaningful advantage over AES-128? Prior work has evaluated lightweight block ciphers across constrained software platforms such as AVR and ARM Cortex-M [4], [5], but the 32-bit RISC-V architecture, now increasingly appearing alongside ARM Cortex-M-class devices in cost-sensitive IoT designs, has received comparatively less attention. Furthermore, software PRESENT implementations frequently employ naive bit-by-bit permutation loops that introduce severe architecture-dependent overhead, leading to artificially pessimistic performance figures.

This paper makes several key contributions to the evaluation of lightweight cryptography on modern edge devices. Specifically, it introduces a correctness-verified benchmark suite for the ESP32-C3, validated against official NIST AES and PRESENT test vectors. To ensure an accurate assessment, the study presents a branchless, 32-bit-optimized software implementation of PRESENT-80 that strategically exploits the algebraic structure of its permutation layer. Utilizing this framework, a fair, multi-run performance comparison is conducted among hardware AES-128, software AES-128, and software PRESENT-80, strictly isolating the steady-state encryption phase by excluding key scheduling from the timing loop. Furthermore, the evaluation employs a block-size-aware methodology to explicitly distinguish per-block latency from per-byte throughput. Ultimately, these empirical metrics culminate in an analysis of the practical implications for IoT system designers tasked with selecting appropriate cryptographic algorithms for RISC-V platforms.

II. Background

A. PRESENT-80 Block Cipher

PRESENT is an ultra-lightweight block cipher standardized by ISO/IEC 29192-2:2012 [6]. It operates on a 64-bit block with an

80-bit key and performs 31 rounds. Each round consists of three layers: AddRoundKey, sBoxLayer, and pLayer. AddRoundKey XORs the current 64-bit state with a round subkey derived from the master key. The sBoxLayer applies sixteen parallel 4-bit substitutions using a fixed 4-bit S-box, providing the cipher's nonlinear confusion component. The pLayer applies a 64-bit bitwise permutation that provides diffusion by redistributing bits across nibble boundaries.

The key schedule operates on an 80-bit key register. In each round, the register is rotated left by 61 positions, the S-box is applied to the most significant nibble, and the round counter is XORed into bits 19 through 15 of the key register. This produces 32 distinct 64-bit round keys (one initial key plus 31 round keys) [2].

The pLayer permutation is defined by:

$$P(i) = 16i \bmod 63, \quad \text{for } 0 \leq i < 63; \quad P(63) = 63 \quad (1)$$

This permutation is a critical bottleneck on software platforms. On hardware, the pLayer can be implemented primarily as fixed wiring with negligible gate cost. On a general-purpose processor, however, a naive implementation iterates over all 64 bits individually, extracting each bit from its source position, and inserting it at the destination position. This incurs 64 branch-and-scatter operations per round, or 1,984 such operations per single block encryption. The resulting overhead makes naive software PRESENT significantly slower than its hardware gate-equivalent complexity would suggest.

B. AES-128

The Advanced Encryption Standard (AES), specified in FIPS 197, operates on a 128-bit block with a 128-bit key [3]. AES-128 performs 10 rounds, each comprising four transformations: SubBytes applies a fixed 8-bit S-box to each byte of the state; ShiftRows cyclically shifts the rows of the 4×4 state matrix; MixColumns performs a matrix multiplication in GF(2^8) over each column; and AddRoundKey XORs the state with the round key. The final round omits MixColumns. AES is inherently byte-oriented, making it well aligned to 8-bit and 32-bit architectures. The well-known T-table optimization precomputes the combined SubBytes, ShiftRows, and MixColumns operations into four 256-entry lookup tables, reducing each round to table lookups and XOR operations. Modern SoCs including the ESP32-C3 integrate dedicated AES hardware accelerators, enabling accelerator-backed ECB encryption with substantially lower CPU-side computation than a pure software implementation [7].

C. ESP32-C3 and RISC-V Architecture

The ESP32-C3 is a 32-bit RISC-V single-core SoC designed by Espressif Systems, operating at up to 160 MHz. It features 400 KB of SRAM, supports external Flash via SPI, and includes an integrated hardware AES accelerator that supports 128-bit, 192-bit, and 256-bit key lengths [7]. Its RV32IMC instruction set includes the base integer (I), integer multiplication/division (M), and compressed instruction (C) extensions, supporting efficient 32-bit integer arithmetic and compact code density. These characteristics make it suitable for word-aligned cryptographic operations. The ESP32-C3 also integrates Wi-Fi and Bluetooth LE connectivity, positioning it as a representative platform for the emerging class of RISC-V-based IoT edge nodes that require both wireless communication and on-device security.

III. Methodology

A. Implementation Details

Three configurations were implemented and benchmarked.

Hardware AES-128 (HW-AES): This configuration uses the mbedTLS `mbedtls_aes_crypt_ecb()` function. The AES context is initialized and the key schedule (`mbedtls_aes_setkey_enc`) is performed once outside the timing loop. Only the ECB encryption call is timed. ECB is used only as a raw block-cipher benchmarking mode and is not recommended for application-level encryption.

Software AES-128 (SW-AES): This configuration is a pure C implementation following FIPS 197. The AES state matrix is loaded and stored in column-major order as specified by the standard: `state[row][col] = input[row + 4*col]`. Key expansion is performed once prior to the timing loop. MixColumns is implemented using the `xtime` GF(2^8) multiplication helper without lookup tables.

Software PRESENT-80 (SW-PRESENT): This configuration is a pure C implementation following the PRESENT specification [2]. The key schedule generates all 32 round keys once prior to the timing loop. The 80-bit key register is stored in big-endian byte order. The 61-bit left rotation is implemented as:

```
new[j] = (old[(j+8)%10] >> 3)
       | (old[(j+7)%10] << 5);
```

The critical optimization is the branchless bit-gather permutation layer described in Section III-B.

B. Permutation Layer Optimization

The mathematical definition of the PRESENT permutation layer is not arbitrary; it is an algebraic mechanism strictly designed to achieve optimal cryptographic diffusion using minimal hardware logic [2]. For a block cipher to be secure against linear and differential cryptanalysis, it must achieve rapid diffusion, meaning the four output bits of any single S-box must be distributed to four completely different S-boxes in the subsequent round.

The modulo arithmetic from Equation 1 guarantees this exact dispersion. To prove this mathematically, let i be any bit index

where $0 \leq i \leq 62$. We can express i as a combination of its S-box assignment:

$$i = 4k + j \tag{2}$$

where k represents the S-box index ($0 \leq k \leq 15$) and j represents the specific bit position within that S-box ($0 \leq j \leq 3$). Substituting this definition into the permutation formula yields:

$$P(i) = 16(4k + j) \bmod 63 = (64k + 16j) \bmod 63 \tag{3}$$

Because $64 \equiv 1 \pmod{63}$, the $64k$ term simplifies to k, reducing the equation to:

$$P(i) = (k + 16j) \bmod 63 \tag{4}$$

This simplified equation dictates the exact output routing for the four bits exiting a single S-box: bit $j = 0$ routes to position k, $j = 1$ to $k+16$, $j = 2$ to $k+32$, and $j = 3$ to $k+48$. Because the next round's S-boxes ingest bits in consecutive blocks of four, bits spaced exactly 16 positions apart are mathematically guaranteed to land in different S-boxes.

Furthermore, a valid cryptographic permutation must be a bijection—a perfect one-to-one mapping where no bits collide. If the modulo formula were blindly applied to the final bit ($i = 63$), the result would be $16 \times 63 \bmod 63 = 0$. This would route bit 63 to position 0, which is already occupied by bit 0 (since $16 \times 0 \bmod 63 = 0$). To prevent this collision and maintain the bijection, $i = 63$ is treated as an explicit mathematical exception and is mapped to itself [2].

From a data-structure perspective, this modulo operation is geometrically identical to structuring the 64-bit state as a 4×16 bit-matrix and transposing it. Exploiting this structure, the permutation can be computed without a lookup table or modulo arithmetic using a shift-and-mask bit-gather technique. The key primitive is an `extract_row` function that collects every fourth bit from a 32-bit word:

```
static inline uint32_t
extract_row(uint32_t w, int row) {
  uint32_t t = (w >> row) & 0x11111111UL;
  t = (t | (t >> 3)) & 0x03030303UL;
  t = (t | (t >> 6)) & 0x000F000FUL;
  t = (t | (t >> 12)) & 0x000000FFUL;
  return t;
}
```

The 64-bit state is split into two native `uint32_t` words: `lo` for bits 0–31 and `hi` for bits 32–63. Four `extract_row` calls per word gather the four transposed rows, which are then packed into the output state:

```
uint32_t r0 = extract_row(lo, 0)
         | (extract_row(hi, 0) << 8);
uint32_t r1 = extract_row(lo, 1)
         | (extract_row(hi, 1) << 8);
uint32_t r2 = extract_row(lo, 2)
         | (extract_row(hi, 2) << 8);
uint32_t r3 = extract_row(lo, 3)
         | (extract_row(hi, 3) << 8);
uint32_t new_lo = r0 | (r1 << 16);
uint32_t new_hi = r2 | (r3 << 16);
```

This optimization removes the 64-iteration branch-and-scatter loop used in a naive pLayer implementation. The optimized method uses only 32-bit shifts, masks, and logical operations, all of which execute in a single cycle on the ESP32-C3's RISC-V core. The total operation count per pLayer invocation is approximately 40 ALU instructions, compared to 64 conditional branches and 128 shift-and-OR operations in a naive implementation. Over 31 rounds, this translates to a substantial reduction in total instruction count per block encryption.

C. Correctness Verification

All implementations were verified against official test vectors before timing measurements were taken. If verification fails, the benchmark aborts immediately.

- SW-AES and HW-AES: NIST FIPS-197 Appendix B test vector. Key: `2B7E151628AED2A6ABF7158809CF4F3C`. Plaintext: `3243F6A8885A308D313198A2E0370734`. Expected ciphertext: `3925841D02DC09FBDC118597196A0B32`.
- SW-PRESENT: Official PRESENT test vector. Key: `00000000000000000000`. Plaintext: `0000000000000000`. Expected ciphertext: `5579C1387B228445`.

Both HW-AES and SW-AES produced identical ciphertext, `66E94BD4EF8A2C3B884CFA59CA342B2E`, for an all-zero key and plaintext. This further cross-validates the software AES implementation against the hardware accelerator.

This two-stage verification process, which consists of testing against published test vectors followed by a cross-comparison between independent implementations, ensures high confidence that the benchmark timings reflect correctly functioning algorithms rather than faster but incorrect code paths.

D. Benchmark Procedure

Benchmarks were conducted on an ESP32-C3 Super Mini development board operating at 160 MHz using the Arduino framework. The timing procedure for each algorithm was:

1) Perform key schedule or context setup once outside the timing loop.
2) Perform 100 warmup encryptions to reduce first-call and cache effects.
3) Execute 10 independent benchmark runs, each encrypting 1000 consecutive blocks using `micros()` for timing.
4) Compute average, minimum, and maximum timing across the 10 runs.

The reported timing excludes key schedule and context initialization. Therefore, the results represent steady-state encryption throughput under a fixed key rather than one-shot encryption latency. This methodology is appropriate for evaluating algorithms in scenarios where a session key is established once and then used to encrypt many blocks, which is the common

case in IoT sensor streaming and telemetry applications.

Memory footprint was measured from the Arduino IDE compiler output with each algorithm compiled in isolation. Each build includes only one cipher implementation alongside the benchmark harness, ensuring that the reported Flash and RAM values reflect the deployment-level cost of each algorithm without interference from unused cipher code.

IV. Results

A. Execution Speed

TABLE I
Execution Speed Benchmark Results (ESP32-C3, 160 MHz)

Algorithm	Block Size	Avg. 1000-Blk Time (μs)	Avg./Blk (μs)	Tput (kB/s)
HW AES-128	16 B	29,336.5	29.34	545.4
SW AES-128	16 B	46,999.1	47.00	340.4
SW PRESENT-80	8 B	88,499.5	88.50	90.4

The coefficient of variation for all algorithms is below 0.1%, indicating stable and repeatable timing under the benchmark setup.

Hardware AES-128 is the fastest configuration at 29.34 μs per 16-byte block. It benefits from the ESP32-C3's dedicated AES accelerator, which offloads the AES round computation from the main RISC-V core. Software AES-128 achieves 47.00 μs per 16-byte block, only 1.60$\times$ slower per block than hardware AES. Software PRESENT-80 achieves 88.50 μs per 8-byte block.

Because AES and PRESENT use different block sizes, two comparisons are necessary. On a per-invocation basis, software PRESENT-80 is 1.88$\times$ slower than software AES-128. However, normalized by processed bytes, software PRESENT-80 is 3.77$\times$ slower than software AES-128. Similarly, software PRESENT-80 is 3.02$\times$ slower than hardware AES per invocation, but 6.03$\times$ slower when normalized by processed bytes.

TABLE II
Relative Speed Comparison

Comparison	Per Cipher Invocation	Per Byte / Equal Payload
SW AES vs HW AES	1.60$\times$ slower	1.60$\times$ slower
SW PRESENT vs HW AES	3.02$\times$ slower	6.03$\times$ slower
SW PRESENT vs SW AES	1.88$\times$ slower	3.77$\times$ slower

The impact of the permutation optimization is substantial. A naive bit-by-bit pLayer implementation yielded approximately 841.82 μs per PRESENT block. With the optimized branchless bit-gather pLayer, the latency was reduced to 88.50 μs per block, a 9.5$\times$ speedup. Without this optimization, PRESENT would appear 28.7$\times$ slower than hardware AES and 17.9$\times$ slower than software AES on a per-block basis.

For IoT telemetry payloads of 8 to 64 bytes transmitted intermittently, the measured software PRESENT throughput of 90.4 kB/s is technically sufficient. However, AES-128 provides much higher throughput, especially when hardware acceleration is available.

B. Memory Footprint

TABLE III
Memory Footprint (Per Algorithm, Compiled in Isolation)

Algorithm	Flash (B)	RAM (B)	Flash Δ vs. SW-AES	RAM Δ vs. SW-AES
HW AES-128	278,386	13,500	+1,342	+160
SW AES-128	277,044	13,340	baseline	baseline
SW PRESENT-80	276,468	13,340	-576	0

The reported Flash and RAM figures include the Arduino ESP-IDF framework baseline. Therefore, the absolute memory values are dominated by framework overhead rather than cipher code alone. The measured Flash difference between software PRESENT-80 and software AES-128 is only 576 bytes, and the RAM difference is zero bytes. Against hardware AES, software PRESENT-80 saves 1,918 bytes of Flash and 160 bytes of RAM.

These differences are negligible in the context of the ESP32-C3's available memory (400 KB SRAM, up to 16 MB external Flash). For comparison, the 576-byte Flash saving of PRESENT-80 over software AES-128 represents only 0.21% of the total firmware Flash usage. PRESENT's well-known hardware-area advantage, which requires only approximately 1,570 gate equivalents compared to roughly 3,400 for AES-128 in ASIC implementations [2], does not translate into a meaningful software memory advantage on this platform, because both algorithms are dominated by the shared framework overhead of the ESP-IDF runtime.

V. Discussion

A. Architecture Mismatch and Software Cost

PRESENT was designed as a hardware-first cipher. Its defining feature, the bit-permutation pLayer, can be implemented primarily as fixed wiring in hardware, but it has non-trivial cost in software. Even after branchless optimization, 31 pLayer operations remain a major contributor to SW-PRESENT execution time.

AES, by contrast, maps naturally onto byte-oriented and word-oriented processors. On a 32-bit RISC-V core, each AES column operation processes four bytes simultaneously within a single 32-bit register, and the 256-byte S-box table fits comfortably within the ESP32-C3's cache. The ShiftRows operation reduces to simple byte-level register manipulation, and MixColumns can be computed with a small number of XOR and shift operations per column using the `xtime` helper.

This architectural fit explains why SW-AES remains faster than SW-PRESENT despite AES having a larger block size, more complex round transformations, and a larger S-box.

Additionally, AES benefits from extensive optimization research spanning over two decades since its standardization in 2001, with well-documented techniques for both software and hardware implementations. PRESENT, while designed to be compact in hardware, has received comparatively less attention for software optimization on 32-bit platforms.

B. Security Considerations

Performance alone is not sufficient for algorithm selection. PRESENT-80 provides an 80-bit key, while AES-128 provides a 128-bit key. The difference of 48 bits in key length corresponds to a factor of approximately 2^{48} (roughly 2.8×10^{14}) in brute-force search space, which is substantial even considering that exhaustive key search is not the most likely attack vector for either cipher.

Although 80-bit security may still be acceptable for some low-value, short-lifetime embedded deployments where the protected data has a limited validity window, it provides a much smaller long-term security margin than AES-128. NIST's lightweight cryptography standardization process explicitly reflects the industry's shift away from 80-bit keys by selecting the Ascon family, which defaults to 128-bit security and provides authenticated encryption with associated data (AEAD), as the new standard for constrained environments [9]. The selection of Ascon in 2023 signals that even resource-constrained IoT devices are expected to provide at least 128-bit security going forward.

Furthermore, PRESENT's 64-bit block size introduces a practical limitation: the birthday-bound collision probability becomes non-negligible after approximately 2^{32} blocks (32 GB) of data encrypted under a single key. AES-128's 128-bit block size raises this threshold to 2^{64} blocks, providing a much larger margin before key rotation is required. While 32 GB far exceeds typical IoT telemetry volumes, it remains a relevant consideration for long-lived deployments or devices that aggregate data from multiple sensors.

C. Practical Recommendations

Based on the experimental results, the following recommendations are made for ESP32-C3 and similar 32-bit RISC-V IoT platforms:

1) Use hardware AES-128 whenever available. It provides the highest throughput and strongest security among the tested options.
2) Use software AES-128 as the preferred fallback when hardware acceleration is unavailable. It provides higher per-byte throughput than software PRESENT-80 while maintaining 128-bit security.
3) Use software PRESENT-80 only when there is a specific compatibility or legacy requirement. It is technically viable for low-data-rate telemetry, but it is slower per byte, has weaker security, and provides negligible memory savings on ESP32-C3.
4) Avoid interpreting per-block latency without considering block size. Since PRESENT uses 64-bit blocks and AES uses 128-bit blocks, per-byte throughput is the more relevant metric for equal-payload comparisons.

D. Limitations

This study has several limitations. First, the benchmark measures raw ECB block encryption only. As specified in NIST SP 800-38A [10], ECB mode lacks semantic security and is highly vulnerable to pattern leakage; it is utilized in this methodology strictly to isolate and measure the bare-metal throughput of the underlying cryptographic primitives. Real-world IoT deployments must utilize appropriate authenticated encryption modes.

Second, the reported timing excludes key schedule and context initialization. This is appropriate for measuring steady-state throughput under a fixed key but does not represent one-shot encryption latency for a single short message.

Third, direct energy profiling via a shunt resistor or dedicated high-frequency power monitor was outside the scope of this study. Consequently, this paper relies on execution time as a proxy for active CPU duty cycles. While execution time strongly correlates with energy drain in pure software implementations, the hardware AES accelerator possesses a distinct power profile that requires direct electrical measurement to calculate absolute energy efficiency.

Fourth, memory footprint is measured through Arduino compiler output, which includes substantial framework overhead. The absolute Flash and RAM values should therefore be interpreted as deployment-level footprints rather than pure algorithmic code sizes.

VI. Conclusion

This paper presented a correctness-verified comparative benchmark of PRESENT-80 and AES-128 on the ESP32-C3, a 32-bit RISC-V IoT microcontroller. A branchless bit-gather optimization derived from the algebraic structure of the PRESENT pLayer was implemented, reducing PRESENT-80 encryption time from approximately 841.82 μs per block to 88.50 μs per block. This 9.5$\times$ speedup demonstrates the importance of architecture-aware algorithm optimization in embedded cryptographic benchmarking.

The main findings are as follows:

- Hardware AES-128 is the fastest option at 29.34 μs per 16-byte block (545.4 kB/s).
- Software AES-128 achieves 47.00 μs per 16-byte block (340.4 kB/s).

- Optimized software PRESENT-80 achieves 88.50 μs per 8-byte block (90.4 kB/s).
- While PRESENT is only 1.88× slower than software AES on a per-invocation basis, it is 3.77× slower when normalized by processed bytes due to its smaller 64-bit block size.
- Memory savings are negligible: software PRESENT-80 saves only 576 bytes of Flash and zero bytes of RAM compared with software AES-128.

The conclusion for ESP32-C3 is that PRESENT-80 is technically viable for low-data-rate IoT telemetry, but it is not a compelling alternative to AES-128. AES-128 offers higher throughput per byte, stronger 128-bit security, hardware acceleration support, and virtually identical deployment-level memory usage. Therefore, AES-128 is recommended for practical cryptographic deployments on ESP32-C3 and similar 32-bit RISC-V IoT platforms.

Future work could extend this analysis to include the NIST-standardized Ascon lightweight AEAD family, which provides authenticated encryption with 128-bit security and may offer competitive performance on RISC-V platforms. Additionally, direct power measurement using a precision shunt resistor or dedicated power monitor would enable a more complete energy-efficiency comparison beyond the execution-time proxy used in this study.

Repository Link

The complete source code and benchmark scripts for this project can be accessed at:
https://github.com/billysams21/present_aes_paper

Video Link

A video demonstration of the ESP32-C3 benchmark execution is available at:
https://www.youtube.com/watch?v=RsLaoK2TfPM

Acknowledgment

First and foremost, the author thanks God Almighty for His blessings and grace that enabled the completion of this paper. The author also expresses deep gratitude to Dr. Ir. Rinaldi Munir, M.T., as the course lecturer, for his guidance and valuable knowledge. Furthermore, the author would like to thank the open-source RISC-V community and cryptographic library developers who provide documentation and implementation references publicly.

Declaration

I hereby declare that this paper is my own original work, not an adaptation or translation of another's paper, and does not contain plagiarism.

Bandung, June 19, 2026

Billy Samuel Setiawan – 18222039

References

[1] K. McKay, L. Bassham, M. Sönmez Turan, and N. Mouha, "Report on Lightweight Cryptography," NIST Internal Report 8114, National Institute of Standards and Technology, 2017.

[2] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," in *Cryptographic Hardware and Embedded Systems — CHES 2007*, Lecture Notes in Computer Science, vol. 4727, pp. 450–466, Springer, 2007.

[3] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," FIPS Publication 197, Nov. 2001.

[4] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A Survey of Lightweight-Cryptography Implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522–533, Nov.–Dec. 2007.

[5] D. Dinu, Y. Le Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, "Triathlon of Lightweight Block Ciphers for the Internet of Things," *Journal of Cryptographic Engineering*, vol. 9, pp. 283–302, 2019.

[6] ISO/IEC 29192-2:2012, *Information technology — Security techniques — Lightweight cryptography — Part 2: Block ciphers*, International Organization for Standardization, 2012.

[7] Espressif Systems, "ESP32-C3 Series Datasheet," Espressif Systems. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf

[8] National Institute of Standards and Technology, "Lightweight Cryptography Project," Computer Security Resource Center. [Online]. Available: https://csrc.nist.gov/projects/lightweight-cryptography

[9] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Lightweight Authenticated Encryption and Hashing," *Journal of Cryptology*, vol. 34, no. 3, article 33, 2021.

[10] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques," NIST Special Publication 800-38A, National Institute of Standards and Technology, 2001.