A Reproducible Security–Performance Benchmark Protocol for Lightweight AEAD on Resource-Constrained IoT Nodes

Atul
Allenhouse Institute of Technology
Kanpur, India
Email: atulverma15704@gmail.com

Vivek Shukla
Allenhouse Institute of Technology
Kanpur, India
Email: vivekshukla0552@gmail.com

Mehul Kumar Das
Allenhouse Institute of Technology
Kanpur, India
Email: dasmehulkumar08@gmail.com

Varun Shukla
Allenhouse Institute of Technology
Kanpur, India
Email: varun.shuklaa@gmail.com

Divya Mishra
Allenhouse Institute of Technology
Kanpur, India
Email: divyamishra03125@gmail.com

***Abstract*—Internet-of-Things (IoT) nodes must protect sensed data while operating with limited processing capability, memory, and battery capacity. In August 2025, the National Institute of Standards and Technology (NIST) finalized SP 800-232, which standardizes the Ascon family for constrained devices. This paper presents ASCON-Edge, a reproducible protocol for evaluating the security, performance, memory cost, and directly measured energy cost of standardized *Ascon-AEAD128* on resource-constrained IoT nodes. The protocol compares Ascon-AEAD128 with AES-128-GCM and ChaCha20-Poly1305 using identical message sizes, associated data, timing boundaries, build conditions, and security tests. It defines a deterministic workload, a nonce-uniqueness and replay policy, a hardware-freeze record, raw-data fields, statistical summaries, direct-energy rules, and functional acceptance criteria. It also separates AEAD guarantees from application-layer responsibilities for replay prevention and key handling. A targeted literature synthesis motivates a final-standard, workload-aware comparison protocol. This is a methodology and reproducibility paper: it reports deterministic packet-format overhead, but deliberately does not claim hardware benchmark measurements before a frozen platform and raw data are available.**

***Index Terms*—ASCON, lightweight cryptography, authenticated encryption, Internet of Things, energy measurement, embedded security.**

I. Introduction

The rapid expansion of the Internet of Things (IoT) has changed how physical systems collect, process, and exchange information. Smart homes, industrial control systems, medical monitors, intelligent transportation, smart-agriculture deployments, and environmental sensors all depend on connected electronic devices. These systems produce data whose confidentiality, integrity, authenticity, and availability may be security-critical. NIST notes that lightweight cryptography is intended to protect information generated and transmitted by the billions of devices that make up the IoT and by other small electronics [5].

Unlike servers, laptops, and smartphones, many IoT endpoints operate with restricted processing power, program memory, RAM, storage, and battery capacity. Low-power sensors, RFID tags, wearable devices, medical implants, and embedded controllers can nevertheless be exposed to eavesdropping, message modification, impersonation, and unauthorized access. This mismatch between limited resources and the need for strong protection is the central motivation for lightweight cryptography [5], [1].

Established cryptographic constructions remain essential, but their practical cost can depend on the processor architecture, hardware acceleration, software implementation, message size, and power budget. On highly constrained devices, cryptography may increase execution time, memory consumption, code size, and energy use. Lightweight cryptography seeks an appropriate balance: it aims to retain strong, well-defined security properties while reducing implementation cost for constrained environments [1].

To address this problem, NIST initiated a public Lightweight Cryptography Standardization Process. In February 2019, 57 candidates were submitted for consideration; in March 2021, ten finalists advanced to the final round. The evaluation included public feedback, security analysis, implementation considerations, and benchmarking results [3], [2]. On February 7, 2023, NIST announced its decision to standardize the Ascon family for lightweight cryptography applications [2].

The process culminated on August 13, 2025, when NIST published Special Publication 800-232, *Ascon-Based Lightweight Cryptography Standards for Constrained Devices* [1]. The standard specifies Ascon-AEAD128 for authenticated encryption with associated data, Ascon-Hash256 for hashing, Ascon-XOF128 as an extendable-output function, and Ascon-CXOF128 as a customized extendable-output function. These permutation-based primitives are designed to provide security, efficiency, and flexibility in settings such as IoT devices,

embedded systems, and low-power sensors [1].

Despite Ascon's standardization, its practical behavior still depends on specific hardware and workloads. This paper therefore specifies how standardized Ascon-AEAD128 should be compared with AES-128-GCM and ChaCha20-Poly1305 on a resource-constrained IoT platform. It defines a controlled, reproducible method for measuring execution time, throughput, memory use, code size, directly measured energy, and security-validation behavior without claiming those measurements before data collection.

The contributions of this paper are as follows:

1) A focused, final-standard comparison design for Ascon-AEAD128, AES-128-GCM, and ChaCha20-Poly1305.
2) A repeatable embedded benchmarking protocol for latency, throughput, memory, code size, and directly measured energy per cryptographic operation.
3) A normative packet, nonce-uniqueness, replay, and security-validation design that separates AEAD guarantees from application-layer controls.
4) A reproducibility package specification covering the hardware freeze, raw-data schema, statistical summaries, and reporting rules required before any performance claim is made.

II. Background and Related Work

A. Lightweight Cryptography

Lightweight cryptography focuses on delivering well-defined cryptographic security on systems with limited computational and physical resources. Such constraints can include processor capability, RAM, nonvolatile storage, hardware area, communication bandwidth, and battery energy. They are common in IoT sensors, RFID devices, wearable electronics, embedded controllers, and other low-power systems.

The objective is not merely to make an algorithm smaller. A lightweight design must balance security with implementation efficiency, including execution time, latency, throughput, code size, memory use, hardware area, and energy consumption. NIST initiated its lightweight-cryptography effort for constrained environments in which the performance of then-current NIST standards might not be acceptable; the process sought AEAD schemes and optional hashing functionality for those settings [4]. Consequently, lightweight cryptography is particularly relevant when IoT systems must protect sensitive data without imposing disproportionate computational or energy cost.

B. NIST Lightweight Cryptography Standardization Process

The NIST Lightweight Cryptography (LWC) Standardization Process was a public, multi-round evaluation intended to identify schemes appropriate for constrained devices. Table I summarizes the main milestones. NIST evaluated candidates through public cryptanalysis and feedback together with implementation, benchmarking, flexibility, and security considerations [3], [2].

The ten finalists were Ascon, Elephant, GIFT-COFB, Grain-128AEAD, ISAP, PHOTON-Beetle, Romulus, SPARKLE, TinyJAMBU, and Xoodyak [3]. The process ended with NIST's selection of Ascon in February 2023 and the final publication of SP 800-232 in August 2025 [2], [1].

TABLE I
Key milestones in the NIST LWC process.

Stage	Year	Result
Call for algorithms	2018	Requirements and evaluation criteria published
Initial submissions	2019	57 submissions received; 56 accepted for Round 1
Round 2	2019–2021	32 candidates evaluated
Final round	2021–2023	10 finalists evaluated
Ascon selected	Feb. 2023	Ascon family selected for standardization
Final standard	Aug. 2025	SP 800-232 published

C. Standardized Ascon Family

Ascon is a permutation-based lightweight cryptographic family. Its standardized functions share the Ascon-p permutation while exposing authenticated-encryption, hash, and extendable-output functionality. The following section describes the architecture of the shared core and the Ascon-AEAD128 encryption flow in detail; exact parameters and procedures are defined by SP 800-232 [1].

D. Ascon-AEAD128

Ascon-AEAD128 is the AEAD primitive specified in SP 800-232. Given a secret key, nonce, plaintext, and optional associated data, it produces ciphertext and an authentication tag. Confidentiality protects the plaintext from unauthorized disclosure, while tag verification detects modifications to the ciphertext and associated data [1].

Associated data is authenticated but not encrypted. In an IoT protocol, it can include a visible protocol version, device identifier, topic, or packet header. Correct use requires a fresh nonce for every encryption under the same key. Furthermore, AEAD tag verification does not by itself prevent replay of an earlier valid message; a deployed protocol must add a sequence number, timestamp, epoch, or equivalent anti-replay mechanism. For this reason, Ascon-AEAD128 is the most relevant standardized Ascon primitive for the secure IoT communication scenario addressed by this protocol.

E. Ascon-Hash256

Ascon-Hash256 is the fixed-output hash function defined in SP 800-232. It maps an input of arbitrary length to a 256-bit digest [1]. A cryptographic hash can support data-integrity checks, identification of data, digital-signature workflows, and protocol operations. Its standardization extends the Ascon family beyond authenticated encryption while retaining a common lightweight design approach.

F. Ascon-XOF128

Ascon-XOF128 is an extendable-output function (XOF). Unlike a fixed-output hash, an XOF can generate output of an application-selected length. The "128" identifier denotes its

TABLE II
ASCON FUNCTIONS STANDARDIZED IN NIST SP 800-232.

Primitive	Primary function and output
Ascon-AEAD128	Authenticated encryption; ciphertext and authentication tag
Ascon-Hash256	Cryptographic hashing; fixed 256-bit digest
Ascon-XOF128	Extendable-output function; variable-length output
Ascon-CXOF128	Customized XOF; variable-length, domain-separated output

TABLE III
MAJOR DIRECTIONS IN EXISTING ASCON RESEARCH.

Research direction	Typical emphasis
Cryptanalysis and design	Security properties of the permutation and modes
Hardware implementation	Area, latency, throughput, power, and energy on FPGA/ASIC targets
Software optimization	Code size, RAM use, speed, and processor-specific techniques
Implementation security	Side-channel and fault threats and their countermeasures
Application studies	Secure communication in IoT and other embedded deployments

security-strength category rather than a fixed output length [1]. Potential uses include key derivation, generation of protocol-specific cryptographic material, and variable-length digest output.

G. Ascon-CXOF128

Ascon-CXOF128 is a customized XOF. In addition to the input message, it accepts a customization string that provides domain separation: the same input can yield distinct outputs when used in different application contexts. This enables a system to reuse one primitive for, for example, key-derivation and authentication-related processing without conflating their domains [1].

H. Summary of Standardized Ascon Functions

I. Existing Ascon Research

Existing work on Ascon spans several complementary areas. Design and cryptanalysis research studies the security rationale of the permutation and its authenticated-encryption and hashing modes [6], [7]. Hardware research explores area, throughput, power, side-channel protection, and implementation trade-offs in FPGA, ASIC, and emerging-memory designs [9], [10], [11], [13], [14], [15]. Software work studies implementation efficiency on constrained processors, including ARM Cortex-M0, and NIST provides a public microcontroller benchmarking framework for LWC finalists [12], [18]. Recent reviews synthesize implementation strategies, attacks, and countermeasures but do not replace a workload-controlled measurement on a selected platform [16], [17]. The official Ascon project provides reference, optimized, and masked implementations for experimentation [19], [8].

J. Identified Research Gap

The literature establishes Ascon as a serious lightweight-cryptography design, but it also shows that implementation results depend on the platform, version, optimization strategy, and evaluation boundary. This paper therefore does not claim to introduce a new primitive, prove universal superiority, or report a completed benchmark. Its contribution is a controlled, application-oriented protocol for evaluating the *final standardized* Ascon-AEAD128 primitive against AES-128-GCM and ChaCha20-Poly1305 across defined IoT message sizes. The framework fixes timing boundaries, memory reporting, direct-energy rules, security-overhead accounting, nonce policy, and tamper/replay validation before measurement.

K. Literature Review Methodology

The targeted review was conducted on August 22, 2026. It searched NIST CSRC, IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and the IACR ePrint Archive using combinations of *Ascon implementation*, *Ascon performance*, *Ascon ARM Cortex-M*, *Ascon hardware*, *Ascon side channel*, and *Ascon AES-GCM ChaCha20 IoT*. Included sources were NIST primary documents and English-language technical studies published from 2015–2026 that address Ascon implementation, practical security, hardware/software cost, or embedded deployment. Duplicate records, nontechnical summaries, and works outside this scope were excluded. The matrix in Table IV is a synthesis of this targeted corpus, not a claim of an exhaustive search of all publications.

Based on the studies included in this review, the reviewed literature does not provide a controlled comparison of standardized Ascon-AEAD128, AES-128-GCM, and ChaCha20-Poly1305 under the exact platform, implementation boundary, nonce policy, workload, and multi-metric protocol defined here. The gap is consequently specific and testable. Any conclusions from an ASCON-Edge instantiation remain scoped to the selected hardware and implementations.

L. Comparison Baselines

AES-128-GCM is a widely deployed AEAD construction that can benefit substantially from hardware AES instructions. ChaCha20-Poly1305 is a software-oriented AEAD construction often used where AES acceleration is unavailable. These baselines are selected because a useful lightweight-cryptography comparison must report the processor and implementation context rather than treat all general-purpose AEAD schemes as equivalent.

An empirical instantiation of ASCON-Edge is guided by the following research questions:

RQ1: How do encryption and decryption latency, throughput, RAM use, and code size vary across Ascon-AEAD128, AES-128-GCM, and ChaCha20-Poly1305?

TABLE IV
LITERATURE SYNTHESIS AND POSITION OF THE PRESENT STUDY.

Study	Scope or platform	Main emphasis	Limitation relative to ASCON-Edge
Dobraunig *et al.* [6]	Algorithm design	Security and efficiency rationale for Ascon v1.2	Does not evaluate the final SP 800-232 primitive against the two selected baselines on one IoT workload.
Groß *et al.* [10]	Hardware implementations	Area, throughput, and side-channel evaluation	Pre-standard hardware design-space study; no controlled MCU comparison with AES-GCM and ChaCha20-Poly1305.
Kitahara *et al.* [12]	ARM Cortex-M0	Optimized software implementations of LWC candidates	Focuses on LWC-candidate optimization rather than a final-standard, three-AEAD protocol benchmark.
NIST benchmarking [18]	Microcontrollers	Common benchmarking framework for LWC finalists	Provides benchmark infrastructure but not the ASCON-Edge IoT packet, security-validation, and baseline-comparison protocol.
Khan *et al.* [15]	ASIC	Area–throughput trade-offs for Ascon hardware	Hardware-only trade-offs do not characterize software deployment on a selected IoT node.
Mirigaldi *et al.* [16] and Kaur *et al.* [17]	Review studies	Implementation strategies, attacks, and countermeasures	Broad synthesis; no new controlled cross-AEAD measurement on the final standard.
This paper	ESP32-class node; optional Raspberry Pi-class reference	Ascon-AEAD128 versus AES-128-GCM and ChaCha20-Poly1305 across IoT workloads	Defines a reproducible, platform-specific protocol; it does not report a benchmark result set.

RQ2: What is the directly measured incremental energy per successful cryptographic encryption or authenticated-decryption operation at different payload sizes?
RQ3: Do all implementations correctly reject modified ciphertext, tag, nonce, and associated data?
RQ4: Which implementation provides the most suitable trade-off for the tested device and workload, subject to the same security requirements?

III. ASCON ARCHITECTURE

A. High-Level Ascon-AEAD128 Architecture

This section focuses on Ascon-AEAD128 because it is the standardized primitive directly applicable to secure IoT communication. Its encryption function accepts a 128-bit secret key K, a 128-bit nonce N, variable-length associated data A, and variable-length plaintext P. It produces ciphertext C of the same length as P and a 128-bit authentication tag T [1]:

$$\text{Ascon-AEAD128.enc}(K, N, A, P) = (C, T). \quad (1)$$

Decryption uses (K, N, A, C, T) and returns P only when the received tag is valid; otherwise it returns failure. Implementations must not release unauthenticated plaintext. Correct nonce handling is essential: a nonce must not be reused for encryption with the same key [1].

B. The 320-Bit Internal State

The shared Ascon-p core transforms a 320-bit state represented as five 64-bit words:

$$S = (S_0, S_1, S_2, S_3, S_4). \quad (2)$$

The state is updated as it absorbs input, emits output, and processes permutation rounds. In Ascon-AEAD128, the first 128 bits form the rate used for associated-data and plaintext processing; the remainder contributes to the capacity and security of the construction [1].

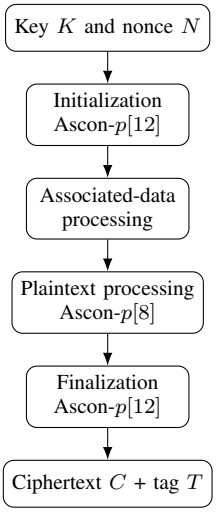

Fig. 1. Conceptual encryption flow of Ascon-AEAD128.

C. Ascon-p Permutation

The Ascon-p permutation is the cryptographic core of the standardized family. Each round applies three transformations in sequence:

$$p = p_L \circ p_S \circ p_C, \quad (3)$$

where p_C is constant addition, p_S is a bitsliced nonlinear substitution layer, and p_L is linear diffusion. The standard defines permutations with different round counts; Ascon-AEAD128 uses Ascon-p[12] during initialization and finalization and Ascon-p[8] between absorbed full blocks [1].

1) Round-Constant Addition: For a round with constant c_r, the constant-addition layer updates the third state word:

$$S_2 \leftarrow S_2 \oplus c_r. \quad (4)$$

TABLE V
ROUND CONSTANTS USED BY ASCON-AEAD128.

Permutation	Constants in application order (hexadecimal)
Ascon-p[8]	`b4, a5, 96, 87, 78, 69, 5a, 4b`
Ascon-p[12]	`f0, e1, d2, c3, b4, a5, 96, 87, 78, 69, 5a, 4b`

The constants differentiate rounds and prevent unwanted structural symmetry. Table V lists the constants used by the two permutations invoked by Ascon-AEAD128.

2) Nonlinear Substitution Layer: The substitution layer applies the same 5-bit S-box in parallel to corresponding bits of $(S_0, S_1, S_2, S_3, S_4)$. Let $(x_0, x_1, x_2, x_3, x_4)$ be a 5-bit input and $(y_0, y_1, y_2, y_3, y_4)$ its output; with $\wedge$ denoting Boolean AND, the standardized S-box is [1]:

$$\begin{aligned}
y_0 &= (x_4 \wedge x_1) \oplus x_3 \oplus (x_2 \wedge x_1) \oplus x_2 \oplus (x_1 \wedge x_0) \oplus x_1 \oplus x_0,\\
y_1 &= x_4 \oplus (x_3 \wedge x_2) \oplus (x_3 \wedge x_1) \oplus x_3 \oplus (x_2 \wedge x_1) \oplus x_2 \oplus x_1 \oplus x_0,\\
y_2 &= (x_4 \wedge x_3) \oplus x_4 \oplus x_2 \oplus x_1 \oplus 1,\\
y_3 &= (x_4 \wedge x_0) \oplus x_4 \oplus (x_3 \wedge x_0) \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0,\\
y_4 &= (x_4 \wedge x_1) \oplus x_4 \oplus x_3 \oplus (x_1 \wedge x_0) \oplus x_1. \qquad (5)
\end{aligned}$$

This nonlinear transformation provides confusion by making the relationship between state inputs and outputs resistant to linear analysis.

3) Linear Diffusion Layer: The linear layer diffuses changes within each 64-bit state word. With ROTR_r denoting a right rotation by r bits, SP 800-232 defines [1]:

$$\begin{aligned}
\Sigma_0(S_0) &= S_0 \oplus \mathrm{ROTR}_{19}(S_0) \oplus \mathrm{ROTR}_{28}(S_0),\\
\Sigma_1(S_1) &= S_1 \oplus \mathrm{ROTR}_{61}(S_1) \oplus \mathrm{ROTR}_{39}(S_1),\\
\Sigma_2(S_2) &= S_2 \oplus \mathrm{ROTR}_{1}(S_2) \oplus \mathrm{ROTR}_{6}(S_2),\\
\Sigma_3(S_3) &= S_3 \oplus \mathrm{ROTR}_{10}(S_3) \oplus \mathrm{ROTR}_{17}(S_3),\\
\Sigma_4(S_4) &= S_4 \oplus \mathrm{ROTR}_{7}(S_4) \oplus \mathrm{ROTR}_{41}(S_4). \qquad (6)
\end{aligned}$$

The diffusion layer supports the avalanche effect: a small input change propagates through the state after successive rounds.

D. Sponge and Duplex-Style Processing

The hash and XOF functions use an absorb–permute–squeeze pattern: input blocks are absorbed into the rate portion of the state and output blocks are subsequently produced from it. Ascon-AEAD128 instead uses duplex-style processing, integrating input absorption, ciphertext generation, and state updates. The common permutation permits these functions to share implementation logic while serving different cryptographic purposes [1].

E. Ascon-AEAD128 Encryption Phases

Ascon-AEAD128 encryption has four phases: initialization, associated-data processing, plaintext processing, and finalization [1]. Table VI summarizes the role of each phase.

Associated data remains visible but is authenticated. Plaintext processing produces ciphertext while updating the state. For the final partial block, the standard applies padding; for a bitstring X shorter than 128 bits, this is $\mathrm{pad}(X, 128) = X \parallel 1 \parallel 0^{127-|X|}$ [1].

TABLE VI
ASCON-AEAD128 ENCRYPTION PHASES.

Phase	Standardized operation
Initialization	Form the initial state from the algorithm parameters, K, and N; apply Ascon-p[12] and incorporate K.
Associated data	Pad and absorb 128-bit blocks into the rate, applying Ascon-p[8] after each block; then set the domain-separation bit.
Plaintext	Absorb plaintext blocks into the rate, emit ciphertext from the updated rate, and apply Ascon-p[8] after full blocks.
Finalization	Incorporate K, apply Ascon-p[12], and derive T from the final state and K.

TABLE VII
KEY ARCHITECTURAL CHARACTERISTICS OF ASCON-AEAD128.

Component	Role
320-bit state	Five 64-bit words transformed by Ascon-p
Round constants	Differentiate individual permutation rounds
S-box	Provides bitsliced nonlinearity and confusion
Linear layer	Spreads changes through rotations and XOR operations
128-bit rate	Processes associated data and plaintext blocks
AEAD finalization	Generates a tag for ciphertext and associated-data verification

F. Finalization and Authentication

After plaintext processing, finalization incorporates the key, applies Ascon-p[12], and derives a tag. In the untruncated standardized form, the tag is 128 bits. The decryption procedure recomputes the required authentication information and returns plaintext only if the received tag validates; otherwise it returns failure [1]. This supplies confidentiality and integrity/authenticity, while anti-replay protection still requires a protocol-level sequence number, timestamp, or epoch.

G. Architectural Characteristics

IV. PROPOSED FRAMEWORK: ASCON-EDGE

A. Overview of the Proposed Framework

ASCON-Edge is a reproducible benchmarking framework for evaluating NIST-standardized Ascon-AEAD128 in resource-constrained IoT communication. It models a sensor node that creates a telemetry message, protects that message with an AEAD algorithm, and sends the resulting packet to an edge gateway. The framework compares Ascon-AEAD128 with AES-128-GCM and ChaCha20-Poly1305 under identical workload, compiler, and measurement conditions.

The primary experiment measures only the cryptographic operation. Packet serialization, radio transmission, network congestion, and server processing are excluded from the timing interval. This separation prevents network variation from being incorrectly attributed to the AEAD algorithm. A separately reported end-to-end experiment may be added later if an MQTT, CoAP, HTTP, or TCP deployment is implemented.

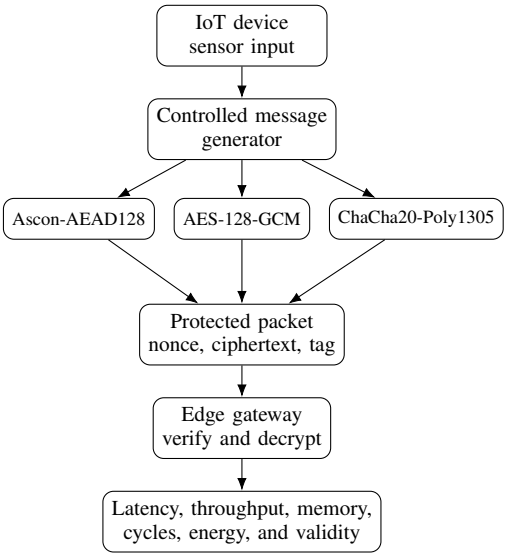

Fig. 2. ASCON-Edge benchmarking framework.

TABLE VIII
THREAT MODEL FOR ASCON-EDGE.

Threat	Attacker action	Expected protection
Eavesdropping	Reads protected packets	Payload confidentiality
Modification	Alters ciphertext or associated data	Tag-verification failure
Injection	Sends a forged packet	Tag-verification failure
Replay	Resends an earlier valid packet	Authenticated epoch and sequence rule
Key compromise	Obtains a device secret	Outside the primary model

TABLE IX
STANDARDIZED PARAMETER SETS EVALUATED BY ASCON-EDGE.

Algorithm	Key	Nonce	Tag	Specification
Ascon-AEAD128	128 b	128 b	128 b	NIST SP 800-232
AES-128-GCM	128 b	96 b	128 b	NIST GCM profile
ChaCha20-Poly1305	256 b	96 b	128 b	RFC 8439 AEAD

B. System Architecture

1) IoT Device Layer: The IoT device represents a constrained sensor node. It constructs a deterministic byte fixture of the requested length, invokes one selected AEAD implementation, and emits a protected packet. Payload semantics are not an independent performance factor: cryptographic timing is evaluated as a function of the exact input bytes and payload length, not whether those bytes represent binary, sensor, or telemetry data.

2) Communication Layer: For algorithm a, the conceptual transmitted packet is

$$H \parallel N_a \parallel C_a \parallel T_a, \tag{7}$$

where H is the visible authenticated header, N_a is the algorithm-specific nonce, C_a is ciphertext, and T_a is the authentication tag. The header is supplied as associated data so that it is authenticated but remains readable by network devices. The benchmark records cryptographic latency separately from any optional network latency.

3) Edge Gateway Layer: The edge gateway receives the packet, constructs the same associated data, verifies the tag, and decrypts only if verification succeeds. A valid packet is accepted and logged; an invalid packet is rejected without releasing plaintext. This enforces the principal AEAD deployment rule that data must not be treated as authentic until tag verification succeeds.

C. Threat Model and Security Assumptions

The attacker controls the communication channel but does not know the secret key. The attacker may observe packets, alter ciphertext or associated data, inject forged packets, and replay previously valid packets. Table VIII maps these capabilities to the expected defense.

AEAD does not by itself supply complete application-level replay protection. The normative nonce and replay policy below prevents reuse of an algorithm-specific nonce with the same key and defines gateway acceptance. Physical compromise, power/electromagnetic side channels, fault injection, weak random-number generation, denial of service, and key extraction are outside the initial evaluation scope.

D. Cryptographic Algorithms

Ascon-AEAD128 is the primary subject because SP 800-232 standardizes it for authenticated encryption in constrained devices [1]. AES-128-GCM is a general-purpose AEAD baseline specified by NIST, and ChaCha20-Poly1305 is an IETF AEAD construction designed for protocol use [20], [21]. The comparison does not assert that the three schemes have identical internals; it evaluates their practical costs under a common workload.

The algorithms are evaluated with their respective standardized parameter sets. Ascon-AEAD128 and AES-128-GCM use 128-bit keys, whereas ChaCha20-Poly1305 uses its required 256-bit key [1], [20], [21]. The comparison controls payload size, associated data, fixture, trial count, compiler settings, and measurement boundary rather than artificially enforcing a common key length.

E. Nonce Construction and Replay Policy

The transmitted header is fixed as $H = \mathit{version}_8 \parallel \mathit{algorithm_id}_8 \parallel \mathit{key_id}_{32} \parallel \mathit{device_id}_{32} \parallel \mathit{epoch}_{16} \parallel \mathit{sequence}_{48}$, a total of 18 bytes. All multi-byte fields use unsigned big-endian encoding. H is supplied as associated data and is transmitted in cleartext. The nonce is algorithm-indexed so that its length and construction are explicit.

For each key, *device_id* is provisioned uniquely. The sender stores *epoch* and *sequence* in persistent state. On boot it increments and commits the epoch before transmitting, then begins with sequence zero; a failed persistent-state update

TABLE X
NORMATIVE NONCE CONSTRUCTION AND UNIQUENESS POLICY.

Algorithm	Nonce N_a (big-endian fields)	Uniqueness scope
Ascon-AEAD128	$kid_{32} \parallel did_{32} \parallel epoch_{16} \parallel seq_{48}$	Per key/device
AES-128-GCM	$did_{32} \parallel epoch_{16} \parallel seq_{48}$	Per key/device
ChaCha20-Poly1305	$did_{32} \parallel epoch_{16} \parallel seq_{48}$	Per key/device

disables transmission. The sender increments the sequence after each encryption. At $2^{48}-1$, it advances and commits the epoch before resetting the sequence; at $2^{16}-1$, it provisions a new key and key identifier before restarting the epoch. The gateway retains the greatest accepted $(\textit{epoch}, \textit{sequence})$ pair for each $(\textit{key_id}, \textit{device_id})$ and, in this baseline protocol, accepts only strictly increasing pairs. Out-of-order delivery and replay windows are deliberately outside the initial scope.

Nonce construction, persistent-state updates, packet serialization, and logging are outside the timed cryptographic interval. The raw-data metadata records the nonce-policy identifier and fixture identifier but never publicizes secret keys or nonce-generation state.

F. Framework Contributions and Workflow

ASCON-Edge contributes a standardized-Ascon benchmark protocol, a controlled comparison design against AES-128-GCM and ChaCha20-Poly1305, a message-size and resource-cost analysis plan, and functional acceptance criteria for tampering and misuse handling. It does not claim a new cryptographic primitive, a measured implementation advantage, or universal superiority for any scheme.

The workflow is: select validated implementations; freeze the hardware and software configuration; generate controlled IoT workloads; execute encryption, decryption, resource, and security tests; analyze the recorded distributions; and interpret the security–performance trade-offs within the stated platform scope. This workflow supports evaluation of Ascon-AEAD128 for constrained IoT communication without converting unmeasured quantities into results.

V. EXPERIMENTAL SETUP

A. Experimental Environment

ASCON-Edge is intended for a constrained IoT node and, optionally, a more capable edge reference platform. An ESP32-class board is the primary embedded target because it represents the constrained-device setting addressed by the protocol; a Raspberry Pi-class platform may be added as a separately reported reference environment. The two platforms are never pooled: each represents a distinct configuration and must be analyzed separately.

This paper intentionally does not nominate a specific board revision. Any empirical instantiation must complete the hardware-freeze checklist in Table XI before data collection. Until then, ASCON-Edge remains a benchmark protocol rather than a source of benchmark findings.

TABLE XI
HARDWARE-FREEZE CHECKLIST FOR AN ASCON-EDGE EMPIRICAL INSTANTIATION.

Item	Required record	Why it is required
Board and revision	Manufacturer, board, revision, and processor identifier	Reproducible target
CPU configuration	Architecture, core count, active clock, cache, and frequency-lock policy	Timing comparability
Memory configuration	Usable flash, RAM, task-stack allocation, and linker map	Resource accounting
Crypto features	Enabled AES, SHA, DMA, or other cryptographic acceleration	Fair baseline interpretation
Software build	Framework/OS, compiler and version, flags, library commits, and linker options	Repeatable binaries
Power path	Supply voltage, current/voltage instrument, sampling rate, trigger, and idle baseline	Valid energy measurement

B. Hardware Configuration

The hardware record is required because AES hardware acceleration, processor frequency, cache behavior, RAM, and compiler target can materially affect results. Comparisons with and without a hardware accelerator are valid only when they are reported as distinct configurations.

C. Operating System and Execution Environment

The reference embedded integration uses native C/C++ with ESP-IDF and FreeRTOS. An optional Raspberry Pi-class reference integration uses native C/C++ on Linux with a documented GCC or Clang version. Python is reserved for parsing raw files, statistical analysis, automation, and graph generation; it is excluded from the timed cryptographic path.

For every platform, the following must be recorded: firmware framework or operating system version, compiler and linker version, optimization flags, board clock configuration, cryptographic library name and commit/version, and any enabled hardware cryptographic feature. All implementations must first pass their published known-answer tests. The official Ascon C repository supplies reference and optimized implementations conforming to SP 800-232 [19].

D. Algorithms and Parameter Handling

The setup evaluates the parameter sets in Table IX. A fresh N_a is assigned to every encryption under a key according to Table X; the test harness rejects a requested reuse before calling the AEAD API. The nonce is an input to the API and is not generated inside the timed interval.

Associated data is the fixed 18-byte header H defined in the nonce policy. Its exact serialized bytes, the algorithm profile, and the implementation API are included in the build manifest because they affect execution time and packet overhead.

TABLE XII
CONTROLLED MESSAGE-SIZE WORKLOAD.

Fixture identifier	Payload bytes
`ascon-edge-fixture-v1-16`	16
`ascon-edge-fixture-v1-32`	32
`ascon-edge-fixture-v1-64`	64
`ascon-edge-fixture-v1-128`	128
`ascon-edge-fixture-v1-256`	256
`ascon-edge-fixture-v1-512`	512
`ascon-edge-fixture-v1-1024`	1024

E. Test Messages

The benchmark treats only exact byte length as a workload factor. It does not use binary, sensor, and telemetry semantics as separate performance classes because AEAD processing depends on the bytes and their length, not on their application interpretation. For a requested length L, the public fixture `ascon-edge-fixture-v1-`L contains bytes $F_L[i] = (\texttt{0xA5} + 29i + 17L) \bmod 256$ for $0 \le i < L$. Fixtures are initialized before measurement and their identifier is recorded with every result.

F. Experimental Procedure

Each condition is defined by platform, algorithm/profile, fixture identifier, payload size, and operation. The procedure is:

1) Build every algorithm using the same platform-specific compiler and optimization policy, then verify known-answer and encrypt–decrypt round-trip tests.
2) Execute 100 warm-up operations per condition without recording them, to exclude startup and allocation effects.
3) Execute 1,000 measurement runs per condition. For each run, use a fresh valid nonce, time only the cryptographic API, and avoid serial logging inside the timed interval.
4) Benchmark successful encryption and successful authenticated decryption separately. Record authentication-failure tests separately from successful decryption.
5) Export one raw record per run together with build metadata and the measurement-boundary definition.

If an operation is long enough to make 1,000 runs impractical, a lower count may be used only when it is justified and reported. The timer source, its resolution, the clock-stabilization policy, and any outlier rule must be recorded.

G. Performance and Resource Metrics

Encryption and authenticated-decryption latency are defined as

$$T_{\mathrm{enc}} = t_{\mathrm{end,enc}} - t_{\mathrm{start,enc}}, \qquad T_{\mathrm{dec}} = t_{\mathrm{end,dec}} - t_{\mathrm{start,dec}}. \tag{8}$$

The timing interval begins immediately before the selected AEAD API and ends immediately after it returns. Fixture construction, nonce construction, persistent-state updates, packet serialization, output allocation, logging, and network I/O are excluded; ciphertext processing, tag verification, and plaintext recovery are included in T_{dec}. For a payload of L bytes processed in t seconds, throughput is

$$\mathit{Throughput} = \frac{8L}{t}. \tag{9}$$

Where a cycle counter is available, the protocol reports cycles per byte,

$$\mathit{Cycles/byte} = \frac{\text{CPU cycles}}{L}, \tag{10}$$

instead of an ambiguous CPU-utilization percentage.

TABLE XIII
EXPERIMENTAL DESIGN MATRIX.

Factor	Values	Measurements
Platform	ESP32; optional Raspberry Pi	Platform-specific performance
Algorithm	Three AEAD schemes	Encrypt, decrypt, reject invalid input
Payload	16–1024 B	Latency, throughput, cycles
Fixture	One deterministic byte fixture per length	Exact input reproducibility
Build	One image per algorithm plus baseline	Flash/RAM deltas, stack, heap
Energy	Instrumented runs only	Incremental cryptographic energy

Each algorithm is built into a separate, otherwise identical firmware image alongside a no-AEAD baseline. The report gives both whole-image flash/static-RAM values and baseline-subtracted deltas. Static RAM comes from the linker map. The documented platform mechanism measures peak stack for the benchmark task; for ESP-IDF/FreeRTOS, it is the configured task stack minus `uxTaskGetStackHighWaterMark` after the fixed workload. Timed code performs no dynamic allocation, and any observed heap metric is reported separately. Compiler/linker flags, including section garbage collection when enabled, are part of the manifest. Table XIII summarizes the condition matrix and expected measurements.

H. Energy and Security Overhead

Energy is reported only when voltage and current are measured with suitable instrumentation. The reported quantity is baseline-subtracted incremental cryptographic energy,

$$E_{\mathrm{crypto}} = \int_{t_0}^{t_1} \left(P_{\mathrm{active}}(t) - P_{\mathrm{idle}}(t)\right) dt. \tag{11}$$

Here instantaneous active power is $P_{\mathrm{active}}(t) = V(t)I(t)$, and P_{idle} is measured over the same trigger window with the same clock configuration and benchmark harness but without an AEAD call. The report must state the power instrument, supply voltage, sampling rate, trigger point, averaging rule, and idle-baseline procedure. Radio transmission, packet serialization, and network activity are excluded; an end-to-end communication-energy study, if added, is a separate experiment.

TABLE XIV
EXPERIMENTAL CONTROLS.

Parameter	Controlled configuration
Message sizes	16, 32, 64, 128, 256, 512, and 1024 bytes
Warm-up and measurement runs	100 warm-up; 1,000 recorded runs
Associated data	Fixed byte sequence within each experiment
Compiler and optimization	Same platform-specific policy for all algorithms
Timing boundary	Cryptographic API only; no logging or network I/O
Nonce handling	Fresh valid nonce for every encryption under a key
Power conditions	Fixed; active and idle windows reported when energy is measured

If no direct energy instrument is available, energy is recorded as unavailable rather than estimated from latency.

Security overhead is reported separately as authentication-tag bytes, nonce bytes transmitted by the packet format, associated-data/header bytes, and ciphertext expansion. This is important for short IoT packets, where metadata can be a large proportion of the transmitted data.

I. Experimental Controls

Table XIV lists the controls that ensure the algorithms are compared fairly. The only intentionally varied factors are the algorithm/profile, payload size, and platform.

J. Result Recording and Statistical Analysis

Raw results are stored as CSV records with the following fields:

```
platform, algorithm, parameter_set_id, fixture_id,
message_size, operation, run_number,
execution_time_ns, cpu_cycles, flash_image_bytes,
flash_delta_bytes, static_ram_bytes,
static_ram_delta_bytes, peak_stack_bytes,
heap_peak_bytes, energy_crypto_uj, nonce_policy_id,
and build_id.
```

The analysis converts execution time to microseconds only when producing human-readable summaries. No secret key, nonce value, or nonce-generation state is written to the public CSV. Unavailable metrics are recorded as missing values, never as zero.

For each condition, the analysis reports mean, median, standard deviation, minimum, maximum, and a 95% confidence interval. The manuscript must state the confidence-interval method and any excluded observations. A single timing value is not treated as a result.

K. Security Validation Setup

The functional validation in Table XV is run for every algorithm and message size. A valid encrypt–decrypt round trip must recover the original plaintext. Each negative test must return authentication failure without disclosing plaintext.

TABLE XV
FUNCTIONAL SECURITY-VALIDATION TESTS.

Test	Expected result
Valid encrypt–decrypt round trip	Recovered plaintext equals original plaintext
One-bit ciphertext modification	Authentication failure and plaintext rejection
One-bit tag modification	Authentication failure and plaintext rejection
One-bit nonce modification	Authentication failure and plaintext rejection
Associated-data modification	Authentication failure and plaintext rejection
Nonce-reuse attempt	Sender-side prevention; policy violation before encryption
Replay of valid packet	Gateway sequence/epoch rule rejects replay

L. Experimental Setup Summary

In an empirical instantiation, the primary embedded study uses one frozen ESP32-class configuration, native C/C++ implementations, seven payload sizes, 100 warm-up runs, 1,000 recorded runs, and a fixed associated-data policy. An optional Raspberry Pi-class experiment uses the same logical workload and reporting protocol but is analyzed as a separate platform. A result is valid only after the exact hardware, software, library revisions, and measurement instrumentation in Table XI have been recorded.

VI. PROTOCOL OUTPUTS AND REPORTING RULES

A. Status of Evidence

ASCON-Edge is a methodology and reproducibility contribution. This manuscript therefore reports no encryption or decryption latency, throughput, cycle, memory, stack, heap, energy, or functional-test outcome from a physical device. Such quantities depend on the frozen configuration in Table XI; reporting them before a board, implementation revision, build, and instrument are fixed would be misleading.

The protocol provides two immediately usable outputs. First, it specifies deterministic packet-format overhead. Second, it defines the evidence that an empirical study must release before making a comparative claim. These outputs support review of the study design while preserving the distinction between a planned measurement and an observed result.

B. Deterministic Communication Overhead

For the packet format defined in Section V, ciphertext has the same length as plaintext and every algorithm uses a 16-byte authentication tag. The fixed 18-byte header is authenticated associated data. Ascon-AEAD128 carries a 16-byte nonce, while the AES-GCM and ChaCha20-Poly1305 profiles carry 12-byte nonces. Thus, the total packet length is determined without a hardware experiment, as shown in Table XVI. The 4-byte nonce difference matters most for short messages; it is a packet-format property, not a performance ranking.

Figure 3 extends the deterministic packet-format calculation to all seven workload sizes by expressing the fixed packet-format cost as a percentage of plaintext size. The AES-128-GCM and ChaCha20-Poly1305 profiles coincide because

TABLE XVI
DETERMINISTIC PACKET-FORMAT OVERHEAD UNDER THE ASCON-EDGE PROTOCOL.

Algorithm	Plaintext	Ciphertext	Tag	Nonce	Header	Total packet
Ascon-AEAD128	16 B	16 B	16 B	16 B	18 B	66 B
Ascon-AEAD128	64 B	64 B	16 B	16 B	18 B	114 B
Ascon-AEAD128	256 B	256 B	16 B	16 B	18 B	306 B
AES-128-GCM	16 B	16 B	16 B	12 B	18 B	62 B
AES-128-GCM	64 B	64 B	16 B	12 B	18 B	110 B
AES-128-GCM	256 B	256 B	16 B	12 B	18 B	302 B
ChaCha20-Poly1305	16 B	16 B	16 B	12 B	18 B	62 B
ChaCha20-Poly1305	64 B	64 B	16 B	12 B	18 B	110 B
ChaCha20-Poly1305	256 B	256 B	16 B	12 B	18 B	302 B

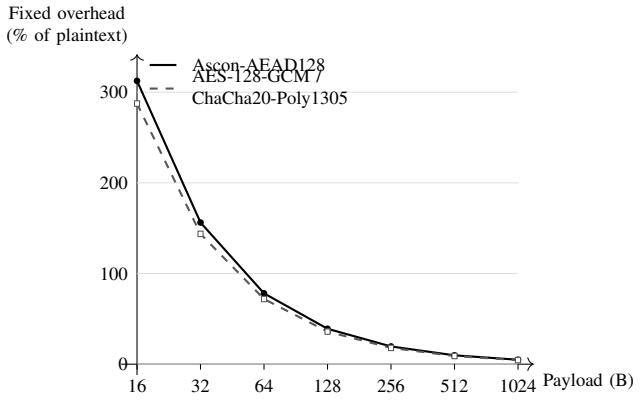

Fig. 3. Deterministic fixed packet-format overhead across the ASCON-Edge workload. The values follow $100(50/L)\%$ for Ascon-AEAD128 and $100(46/L)\%$ for the two baseline profiles; the graph depicts packet-format cost, not measured performance.

they use the same 12-byte nonce, 16-byte tag, and 18-byte header under this protocol. The graph is included to visualize communication-format cost only; it is not a latency, throughput, energy, or security ranking.

C. Evidence Required for Empirical Claims

A complete empirical release comprises the frozen experiment manifest, one raw record per successful run, a security-validation ledger, per-condition summaries, and the analysis code used to generate tables and figures. Table XVII makes the minimum reporting threshold explicit. An unavailable metric remains unavailable; in particular, timing or cycles cannot be presented as an energy measurement. A failed functional test must be reported rather than excluded from the evaluation.

The functional checks in Table XV are acceptance criteria, not claimed test results. A successful future instantiation must show that every implementation accepts valid input, rejects the specified modifications without releasing plaintext, prevents sender-side nonce reuse, and rejects gateway replays under the stated sequence rule.

VII. DISCUSSION

A. Interpretation Rules and Message-Size Effects

An empirical instantiation should interpret the measured latency, throughput, memory, and cycle distributions rather than rely on a single timing value. It should examine small-message behavior, scaling from 16 to 1024 bytes, and the difference between encryption and authenticated decryption. Every conclusion remains platform-specific: an advantage observed for one algorithm on one board does not establish universal superiority.

TABLE XVII
MINIMUM EVIDENCE REQUIRED BEFORE REPORTING ASCON-EDGE BENCHMARK FINDINGS.

Evidence item	Minimum content
Frozen manifest	Board and revision, clock, supply, toolchain, flags, library commits, and acceleration state
Raw measurements	One successful record per run with workload, operation, time, cycles, resource fields, and energy where directly measured
Statistical summary	Run count, mean, median, standard deviation, minimum, maximum, 95% confidence interval, and stated exclusion rule
Security ledger	Known-answer, round-trip, tampering, nonce-reuse, and replay-test outcomes for every relevant configuration
Energy evidence	Instrument, sampling rate, trigger, active and idle windows, baseline method, and retained raw trace or equivalent record
Reproducible analysis	Versioned scripts that regenerate every reported aggregate, table, and figure from the released data

B. Hardware Acceleration and IoT Deployment Implications

AES acceleration, processor architecture, cache behavior, compiler optimization, and implementation choice can materially affect the observed ranking. For a tiny sensor node, memory demand, cycles, and directly measured energy may be more important than peak throughput. For a gateway or a device with AES acceleration, throughput may become more important. ASCON-Edge therefore requires deployment recommendations to be derived from the recorded configuration rather than inferred from an algorithm name alone.

C. Security–Performance Trade-off

The trade-off analysis asks which algorithm offers the most suitable balance for the *tested* platform and workload, rather than which is universally fastest. It considers latency, throughput, memory, cycles, directly measured energy where available, communication overhead, and hardware dependence.

If a single composite score is required, its normalization and weights must be fixed before interpretation. Let $x_{a,m}$ be the measured value for algorithm a and metric m. The normalized metric score is

$$s_{a,m} = \begin{cases} \dfrac{\min\limits_{a'} x_{a',m}}{x_{a,m}}, & \text{for lower-is-better metrics,} \\ \dfrac{x_{a,m}}{\max\limits_{a'} x_{a',m}}, & \text{for higher-is-better metrics.} \end{cases} \tag{12}$$

The application-specific composite is then

$$S_a = \sum_m w_m s_{a,m}, \qquad \sum_m w_m = 1. \tag{13}$$

For the battery-powered-node profile, the pre-registered weights are latency 0.25, throughput 0.20, memory 0.20, cycles 0.20, and direct energy 0.15. A score is not calculated if any required metric is unavailable; in particular, unavailable direct energy is not silently replaced by latency or omitted after observing the data. The weights may be changed only for a separately named deployment profile defined before analysis.

D. Threats to Validity and Comparison with Previous Research

Threats to validity include limited hardware coverage, library and compiler choices, timer resolution, thermal or frequency variation, measurement-boundary mistakes, and power-instrument bandwidth. An empirical report should compare its frozen configuration and workload with the literature-review matrix rather than make broad cross-platform claims. It should include latency, throughput, cycles-per-byte, memory, and direct-energy figures only when their underlying raw data and reporting requirements in Table XVII have been met.

VIII. Conclusion

This manuscript presents ASCON-Edge, a reproducible benchmark protocol for NIST-standardized Ascon-AEAD128 in resource-constrained IoT communication. It defines a controlled comparison with AES-128-GCM and ChaCha20-Poly1305, including a deterministic workload, timing boundaries, memory accounting, direct-energy requirements, nonce and replay controls, functional acceptance criteria, a hardware-freeze checklist, and a raw-data and statistical-reporting schema.

The paper makes no unmeasured benchmark claim. Its directly derived packet-format analysis shows that Ascon-AEAD128 adds four nonce bytes relative to the selected 96-bit-nonce baselines, while all three profiles add a 16-byte tag. Any performance or energy conclusion must be supplied by a future frozen-platform instantiation that releases the evidence specified by ASCON-Edge. The contribution is thus a transparent, reviewable method for measuring security–performance trade-offs without treating planned values as experimental results.

A. Limitations

Any empirical findings produced with ASCON-Edge will be limited to the selected hardware, implementations, compiler configuration, and enabled hardware accelerators; they cannot be generalized automatically to all IoT architectures. The protocol focuses on software-level performance and functional AEAD behavior, not physical side-channel attacks, fault injection, formal cryptanalysis, or dedicated hardware implementations. Direct energy measurement may also be limited by the availability and bandwidth of the power-monitoring setup.

IX. Future Work

Future work can extend the study to additional microcontrollers, FPGA or ASIC implementations, dedicated Ascon acceleration, protected or masked implementations for side-channel resistance, and end-to-end MQTT or CoAP deployments. Other directions include secure key provisioning and rotation, post-quantum key establishment combined with Ascon-AEAD128 for bulk data protection, and an adaptive edge policy that selects a measured AEAD configuration according to device memory, battery, payload size, workload, and available hardware acceleration. A hardware side-channel study should use appropriate acquisition equipment and statistical methodology rather than infer physical-attack resistance from software timing results alone.

Appendix A
Experiment Manifest

Before data collection, the experiment manifest must record the exact board model and revision, CPU clock configuration, supply voltage, operating environment, compiler and optimization flags, cryptographic-library commits, enabled acceleration features, timer source, power instrument, and associated-data encoding. It accompanies the raw CSV data and makes each reported configuration reproducible.

Appendix B
Raw Result Format and Security Tests

Each raw record contains the platform, algorithm, parameter-set identifier, fixture identifier, message size, operation, run number, execution time, cycle count, whole-image and baseline-subtracted memory fields, direct-energy field where measured, nonce-policy identifier, and build identifier. Public raw data must never contain a secret key, nonce value, or nonce-generation state. The test suite includes known-answer tests, encrypt–decrypt round trips, modified-ciphertext, modified-tag, modified-nonce, and modified-associated-data tests; sender-side nonce-reuse prevention; and gateway replay rejection. Nonexperimental test fixtures may validate the analysis workflow, but they are not experimental evidence and must never appear as benchmark measurements.

References

[1] M. S. Turan, K. McKay, J. Kang, J. Kelsey, and D. Chang, *Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and Extendable Output Functions*, NIST SP 800-232, National Institute of Standards and Technology, Gaithersburg, MD, USA, Aug. 2025, doi: 10.6028/NIST.SP.800-232.

[2] M. S. Turan, K. McKay, D. Chang, L. Bassham, J. Kang, N. Waller, J. Kelsey, and D. Hong, *Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process*, NIST IR 8454, National Institute of Standards and Technology, Gaithersburg, MD, USA, Jun. 2023, doi: 10.6028/NIST.IR.8454.

[3] M. S. Turan, K. McKay, D. Chang, C. Calik, L. Bassham, J. Kang, and J. Kelsey, *Status Report on the Second Round of the NIST Lightweight Cryptography Standardization Process*, NIST IR 8369, National Institute of Standards and Technology, Gaithersburg, MD, USA, Jul. 2021, doi: 10.6028/NIST.IR.8369.

[4] National Institute of Standards and Technology, "Lightweight Cryptography," Computer Security Resource Center. [Online]. Available: https://csrc.nist.gov/Projects/Lightweight-Cryptography. Accessed: Aug. 22, 2026.

[5] National Institute of Standards and Technology, "NIST finalizes 'lightweight cryptography' standard to protect small devices," Aug. 13, 2025. [Online]. Available: https://www.nist.gov/news-events/news/2025/08/nist-finalizes-lightweight-cryptography-standard-protect-small-devices. Accessed: Aug. 22, 2026.

[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, Art. no. 33, 2021, doi: 10.1007/s00145-021-09398-9.

[7] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2—Analysis of security and efficiency," presented at the NIST Lightweight Cryptography Workshop, 2019.

[8] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon status update (finalist)," NIST Lightweight Cryptography Standardization Process, Sep. 2022.

[9] H. Groß, E. Wenger, C. Dobraunig, and C. Ehrenhöfer, "Suit up!–Made-to-measure hardware implementations of Ascon," in *Proc. Euromicro Conf. Digital System Design*, 2015, pp. 645–652.

[10] H. Groß, E. Wenger, C. Dobraunig, and C. Ehrenhöfer, "Ascon hardware implementations and side-channel evaluation," *Microprocessors and Microsystems*, vol. 52, pp. 470–479, 2017, doi: 10.1016/j.micpro.2016.10.006.

[11] W. Diehl, F. Farahmand, A. Abdulgadir, J. P. Kaps, and K. Gaj, "Face-off between the CAESAR lightweight finalists: ACORN versus Ascon," in *Proc. Int. Conf. Field-Programmable Technology*, 2018, pp. 330–333.

[12] T. Kitahara, R. Hira, Y. Hara-Azumi, D. Miyahara, Y. Li, and K. Sakiyama, "Optimized software implementations of Ascon, Grain-128AEAD, and TinyJambu on ARM Cortex-M0," in *Proc. 10th Int. Symp. Computing and Networking Workshops*, 2022, pp. 316–322, doi: 10.1109/CANDARW57323.2022.00030.

[13] N. Roussel, O. Potin, G. Di Pendina, J.-M. Dutertre, and J.-B. Rigaud, "CMOS/STT-MRAM based Ascon LWC: A power efficient hardware implementation," in *Proc. IEEE Int. Conf. Electronics, Circuits and Systems*, 2022, pp. 1–4.

[14] S. Khan, W. K. Lee, and S. O. Hwang, "Scalable and efficient hardware architectures for authenticated encryption in IoT applications," *IEEE Internet of Things Journal*, vol. 8, no. 14, pp. 11260–11275, 2021.

[15] S. Khan *et al.*, "Securing the IoT ecosystem: ASIC-based hardware realization of Ascon lightweight cipher," *International Journal of Information Security*, vol. 23, pp. 3653–3664, 2024, doi: 10.1007/s10207-024-00904-1.

[16] M. Mirigaldi, V. Piscopo, M. Martina, and G. Masera, "The quest for efficient ASCON implementations: A comprehensive review of implementation strategies and challenges," *Chips*, vol. 4, no. 2, Art. no. 15, 2025, doi: 10.3390/chips4020015.

[17] J. Kaur, A. Cintas Canto, M. M. Kermani, and R. Azarderakhsh, "A survey on the implementations, attacks, and countermeasures of the NIST lightweight cryptography standard: ASCON," *ACM Computing Surveys*, vol. 58, no. 1, Art. no. 6, pp. 1–16, 2026, doi: 10.1145/3744640.

[18] National Institute of Standards and Technology, "Benchmarking of NIST LWC finalists on microcontrollers," GitHub repository. [Online]. Available: https://github.com/usnistgov/Lightweight-Cryptography-Benchmarking. Accessed: Aug. 22, 2026.

[19] Ascon Team, "Ascon C: Reference, optimized, masked C and assembly implementations of NIST SP 800-232," GitHub repository. [Online]. Available: https://github.com/ascon/ascon-c. Accessed: Aug. 22, 2026.

[20] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, NIST SP 800-38D, National Institute of Standards and Technology, Gaithersburg, MD, USA, Nov. 2007, doi: 10.6028/NIST.SP.800-38D.

[21] Y. Nir and A. Langley, *ChaCha20 and Poly1305 for IETF Protocols*, RFC 8439, Internet Engineering Task Force, Jun. 2018, doi: 10.17487/RFC8439.