An Optimized Hybrid Lightweight Encryption Framework Model for Cloud-Based IoT

Majoju Kiran Kumar[1],

Department of Computer science and Engineering,
Talangana University, Nizamabad
kiran4frndship@gmail.com

Prof. Arathi Chitla[2]

Department of Computer science and Engineering,
Talangana University, Nizamabad
arathichitla@gmail.com

Abstract— Cloud-based Internet of Things (IoT) systems must protect data from constrained sensors through fog gateways to third-party cloud storage, yet most hybrid encryption schemes in the literature reuse heavyweight RSA or unauthenticated AES modes and apply one static configuration to every record. This paper proposes OHLEF, an Optimized Hybrid Lightweight Encryption Framework for cloud-based IoT, derived from a review of 19 hybrid and lightweight encryption studies. OHLEF combines Ed25519-signed ephemeral X25519 key exchange, HKDF key separation, authenticated encryption (ChaCha20-Poly1305 or AES-128-GCM) with a sequence-number replay guard, and a session-keyed chaotic dynamic S-box applied only to high-sensitivity data. An optimization module chooses the cipher suite, re-keying interval and S-box use per sensitivity class by minimizing a weighted latency, overhead and key-exposure cost. A Python prototype was benchmarked against RSA-2048 + AES-CBC and per-message ECIES baselines. For 256-byte telemetry, OHLEF needed 9.0 µs and 24 bytes of overhead per message, 44× faster and 92% smaller than the RSA-based hybrid and 21× faster than ECIES. Ciphertext of all-zero and telemetry inputs reached 7.998 bits/byte entropy, passed χ^2 and four NIST SP 800-22 tests, and showed 50.06% key avalanche. Tests also show that chaotic S-boxes are weaker than the AES S-box (nonlinearity 103 versus 112), so OHLEF uses them only as defense in depth.

Keywords: Internet of Things, cloud computing, hybrid encryption, lightweight cryptography, elliptic curve cryptography, ChaCha20-Poly1305, chaotic S-box, optimization

I. INTRODUCTION

The Internet of Things (IoT) and cloud computing have converged into a single "Cloud of Things" (CoT) or CloudIoT paradigm, in which billions of resource-constrained sensors push data to elastic cloud back-ends for storage, analytics and sharing [1], [3]. Healthcare, smart homes and Industry 4.0 cyber-physical systems are the most visible adopters [1], [11], [14]. The same integration widens the attack surface: data is exposed on the wireless link, at the fog or gateway tier, and at rest in third-party cloud storage [4], [12], [19].

Cryptography is the primary control for confidentiality and integrity, but no single primitive fits the CloudIoT setting. Symmetric ciphers such as AES are fast but leave the key-distribution problem open; asymmetric schemes such as RSA and ECC solve key distribution but are too slow for bulk data on microcontrollers [2], [6]. Hybrid encryption, an asymmetric key-establishment step followed by symmetric bulk encryption, has therefore become the dominant design pattern [2], [4], [5], [7], [8].

A. PROBLEM STATEMENT

Existing hybrid schemes leave four practical gaps for cloud-based IoT:

- **Weight.** Many schemes reuse RSA-2048 or layered AES+DES constructions designed for servers, not for Class-1 and Class-2 constrained devices [5], [9], [24].
- **Integrity as an afterthought.** Several pipelines provide confidentiality only (AES-CBC/ECB) and bolt on hashing separately, or omit it [4], [5], [9].
- **Static configuration.** One cipher suite is applied to every record, regardless of its sensitivity or the device that produced it, although data in industrial and healthcare deployments differs sharply in sensitivity [14].

a. **Unamortized key establishment.** The expensive asymmetric step is often repeated per message or per file, which dominates cost for the small, frequent payloads typical of telemetry [7], [8].

B. CONTRIBUTIONS

This paper proposes OHLEF, an **Optimized Hybrid Lightweight Encryption Framework** for cloud-based IoT. Its contributions are:

- A three-tier (device, fog gateway, cloud) architecture that combines authenticated ephemeral elliptic-curve Diffie–Hellman (X25519 with Ed25519 signatures) for key establishment with authenticated encryption with associated data (AEAD) for bulk data.
- A sensitivity-aware data path: low, medium and high sensitivity records receive different protection profiles, the high profile adding a session-keyed chaotic dynamic substitution box (S-box) layer as defense in depth, following [14], [17], [18].
- An optimization module that selects the AEAD primitive and the session re-keying interval per device class and sensitivity level by minimizing a weighted latency–overhead cost under a minimum security constraint.
- A working Python prototype and a reproducible benchmark that compares OHLEF with RSA-2048+AES and ECIES-style per-message hybrid baselines, plus statistical tests of the S-box and ciphertext.

C. PAPER ORGANIZATION

Section 2 reviews the 19 reference works and identifies research gaps. Section 3 presents the OHLEF architecture and threat model. Section 4 details the algorithms and the optimization model. Section 5 analyzes security, Section 6 reports the experimental evaluation, and Section 7 discusses limitations and concludes.

II. Related Work

The 19 reference works converge on one design (ECC for keys, a fast symmetric cipher for data) but differ in weight, integrity handling and adaptivity. They fall into four groups.

A.CloudIoT foundations and surveys

Darwish et al. [1] define the CloudIoT-Health paradigm and identify security and privacy as the leading open problems of integrating IoT with cloud computing. Mohamed et al. [2] review hybrid cryptographic approaches for IoT published from 2013 to 2018 and conclude that AES and ECC are the most popular hybrid pairing because of their speed and security. Mohamed and Ismael [19] show that a fog tier can host security functions close to devices: their GA-optimized neural-network intrusion detection on a Raspberry Pi 4 fog node cut execution time by 16.35% and 37.07% on UNSW-NB15 and ToN_IoT. OHLEF reuses this fog placement for its cryptographic gateway.

B. Hybrid encryption for cloud storage

Alabi et al. [4] encrypt data with AES and then protect the AES key with ECC before cloud storage. Reddy et al. [5] add a second symmetric layer (DES then AES), with the shared key derived through ECC Diffie–Hellman. Shaik and Thummala [6] modify both the symmetric and asymmetric components to cut runtime and report resistance to brute-force, replay and man-in-the-middle attacks; they explicitly name lightweight hybrid schemes for IoT-enabled clouds as future work. Rao and Sujatha [7] generate keys on a lightweight Edwards curve, transform them with identity-based encryption (IBE) and shorten them for AES, reporting 0.000025 s key generation, 0.00349 s encryption and 693.10 kB/s throughput. Ahmad et al. [8] build a key management system (HCA-KMS) that issues ECC keys as QR codes and encrypts data with a hybrid of AES and ECC.

C. Hybrid encryption for healthcare IoT and cloud

Chahin and Mansour [3] integrate a Contiki/Cooja sensor network with a cloud provider using ECC key generation and AES data encryption, and evaluate power, packet delivery ratio and resistance to black-hole attacks. Chinnasamy and Deepalakshmi [9] (HCAC-EHR) encrypt electronic health records with an improved RSA key generation scheme (IKGSR), protect keys with Blowfish and share them through steganography-based access control. Jyotheeswari and Jeyanthi [10] combine symmetric encryption with attribute-based encryption (ABE) for medical IoT data sharing. Popoola et al. [11] pair ECC secp256r1 with AES-128 in EAX mode for smart-home healthcare and report 0.006 s processing, a 25.6% client speed-up and up to 44% lower server energy than RSA-2048. Altherwi et al. [12] (AHO-ABE) use whale optimization for key generation and sheep-flock optimization for monitoring, reporting 5 ms execution and 850 kbps throughput.

D. Lightweight, chaotic and sensitivity-aware schemes

Jammula et al. [13] combine ChaCha and Playfair ciphers with ABE (LWC-ABE) for scalable IoT cloud storage. Garg et al. [14] (DHSE) split industrial data into low, mid and high sensitivity classes and apply AES, IBE and ABE accordingly, averaging 2.07 ms encryption. Zwiad [15] pairs certificate-based ECC with ChaCha20-Poly1305 AEAD on an ARM Cortex-M4 model and reports up to 82% lower execution time, over 60% lower energy, and 34% less RAM than conventional RSA/AES designs. Hussain [16] runs reduced-round AES in parallel with the ASCON authenticated cipher, adds SipHash integrity and derives keys from a 4D chaotic system; all 15 NIST SP 800-22 tests pass. Farahi et al. [17] (SEC-Blowfish) combine ECC key exchange with chaotic maps that generate dynamic S-boxes to resist statistical, linear and differential cryptanalysis. Kadhim and Hussein [18] cascade LEA and PRESENT in counter mode with a 5D chaotic system and HKDF key derivation, reaching ciphertext image entropy of 7.999.

E. Comparative summary

The table 1 we have shown the compared various factors that decides whether the algorithm can be taken as adaptive or optimized.

Table 1 shows Comparative summary of key establishment of various Algorithm's keys.

Table 1 summary of key establishment

Ref	Key establishment	Data cipher	Integrity / AEAD	IoT-lightweight	Adaptive or optimized
[3]	ECC	AES	No	Yes (WSN)	No
[4]	ECC (wraps key)	AES	No	No	No
[5]	ECC Diffie–Hellman	DES + AES	No	No	No
[6]	Modified asymmetric	Modified symmetric	Partial	No	No
[7]	Edwards-curve ECC + IBE	AES	No	Partial	Key-size reduction
[8]	ECC (QR code keys)	AES + ECC	Authenticated encryption	No	No
[9]	Improved RSA (IKGSR)	Blowfish (keys)	No	No	No
[10]	ABE	Symmetric	Yes	Partial	No
[11]	ECC secp256r1	AES-128-EAX	Yes (AEAD)	Yes	Parameter tuning
[12]	ABE	—	Yes	Partial	Whale + sheep-flock optimization
[13]	ABE	ChaCha + Playfair	Partial	Yes	No
[14]	IBE / ABE	AES	Partial	Partial	Sensitivity classes
[15]	ECC certificates	ChaCha20	Poly1305 (AEAD)	Yes	No
[16]	4D chaotic keys	AES-r + ASCON	ASCON + SipHash	Yes	No
[17]	ECC	Blowfish + chaotic S-box	No	Yes	No
[18]	HKDF + 5D chaos	LEA + PRESENT (CTR)	No	Yes	No
OHLEF	**Signed X25519 ECDHE + HKDF**	**ChaCha20 or AES-GCM (+ chaotic S-box for high)**	**Yes (AEAD + replay counter)**	**Yes**	**Cost-based suite and re-key selection per class**

F. Research gaps

1. No reviewed scheme combines forward-secret, mutually authenticated key establishment with AEAD and replay protection in one lightweight pipeline; [15] comes closest but has no per-session re-keying policy or sensitivity tiers.
2. Sensitivity-aware protection [14] and chaotic S-boxes [17], [18] have not been joined with standardized AEAD, so their extra layers sit on top of unauthenticated ciphers.
3. Optimization in the literature tunes key generation [7], [12] or intrusion detection [19], not the cost of the cryptographic pipeline itself (cipher choice and re-keying frequency).

4. Most evaluations measure a single primitive rather than the end-to-end per-message cost, including key establishment, that a device actually pays.

III PROPOSED FRAMEWORK: OHLEF

OHLEF encrypts data once at the device, keeps it encrypted through the fog gateway and stores it encrypted in the cloud, with the protection profile chosen per record sensitivity and device class.

A System architecture

The framework has three tiers, following the CloudIoT model of [1], [3] and the fog placement of [19]. The figure shows the three architecture of the proposed model.

- **Device tier.** Sensors and wearables (RFC 7228 Class 1–2 devices [24]) hold a provisioned Ed25519 identity key and run the lightweight data path: ephemeral X25519, HKDF, AEAD and, for high-sensitivity data, the chaotic S-box layer.
- **Fog gateway tier.** A gateway (e.g., a Raspberry Pi-class node, as in [19]) terminates device sessions, verifies tags and replay counters, runs the optimization module, and re-wraps data keys for cloud storage. It never needs to decrypt low-sensitivity telemetry if end-to-end mode is configured.
- **Cloud tier.** The cloud stores ciphertext under envelope encryption: each data object is sealed with a data-encryption key (DEK), and the DEK is wrapped with a cloud key-encryption key (KEK) using ECIES. Access control can be layered with ABE as in [10], [12], [13].

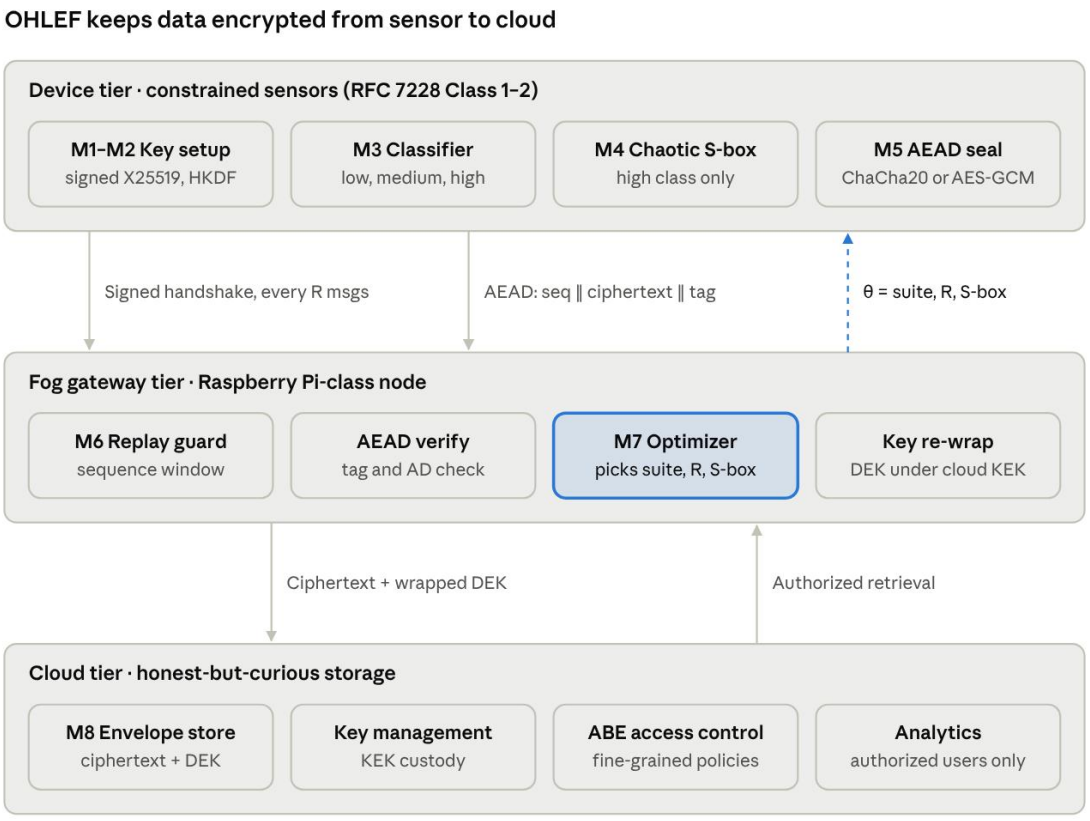


Figure 1 OHLEF architecture

Figure 1. Data is sealed on the device, verified and re-wrapped at the fog gateway, and stored only as ciphertext in the cloud; the optimizer at the gateway pushes the configuration θ back to devices.

The table 2 shows all the modules of the framework and their functions which we have developed a new model based on other researcher's work.

Table 2 Framework Modules

Module	Function	Primitive in prototype	Basis in literature
M1 Authenticated key establishment	Mutual authentication and forward-secret session key	X25519 ECDHE signed with Ed25519	ECC hybrids [3], [4], [8], [11], [15]
M2 Key derivation	Separate keys for AEAD, S-box seed and nonce prefix	HKDF-SHA256 [22]	HKDF in [18]
M3 Sensitivity classifier	Labels each record low, medium or high	Rule table (data type, source)	Three-level split of [14]
M4 Chaotic dynamic S-box	Session-keyed byte substitution with chaining for high data	Logistic map, r ∈ [3.99, 4.0)	Dynamic S-boxes [17], chaos [16], [18]
M5 AEAD encryption	Confidentiality + integrity + associated-data binding	ChaCha20-Poly1305 [20] or AES-128-GCM	AEAD in [11], [15], [16]
M6 Replay guard	Monotonic 64-bit sequence number in nonce and AD	Counter window at gateway	Replay resistance in [6], [11], [15]
M7 Optimizer	Chooses cipher suite, re-key interval and S-box use	Constrained cost minimization	Optimization in [7], [12], [19]
M8 Cloud envelope store	DEK per object, KEK-wrapped	ECIES wrap of DEK	KMS of [8], ECC wrap of [4]

B Threat model

We assume a Dolev–Yao adversary who controls the wireless and Internet links: it can eavesdrop, inject, replay, modify and drop messages, and can attempt man-in-the-middle substitution of key-exchange messages. The cloud provider is honest-but-curious: it stores and serves data correctly but may read anything in plaintext [9], [12]. Devices may be captured; a captured device reveals its own identity key and current session keys but not past session keys. Physical side-channel attacks on the device and denial of service are out of scope; intrusion detection at the fog tier [19] is complementary.

IV METHODOLOGY

OHLEF runs an authenticated handshake once per re-key interval R, then seals each message with one AEAD call (plus one S-box pass for high-sensitivity data), so the asymmetric cost is amortized over R messages.

A. Authenticated ephemeral key establishment (M1)

Device D and gateway G each hold a long-term Ed25519 key pair (sk_D, pk_D) and (sk_G, pk_G), provisioned at enrollment; each knows the other's public key or a certificate for it. For every session:

i. D draws an ephemeral X25519 key a and sends A = a·P together with σ_D = Sign(sk_D, A ‖ ID_D).
ii. G draws b, sends B = b·P and σ_G = Sign(sk_G, B ‖ ID_G).
iii. Each side verifies the peer signature and aborts on failure; this defeats key substitution by a man in the middle.
iv. Both compute the shared secret Z = a·B = b·A and erase the ephemeral private key, giving forward secrecy.

B. Key derivation (M2)

The transcript T = A ‖ B ‖ suite ‖ class binds the keys to the negotiated parameters, so a downgrade of the cipher suite or sensitivity class changes every derived key:

$$(K_{enc}, K_{sbox}, N_{pre}) = SHA-256(salt = H(T), IKM = Z, info = "OHELF\ v1" || suit)$$

C. Chaotic dynamic S-box (M4)

For high-sensitivity sessions the S-box is regenerated from K_sbox, so each session has a different, secret substitution. The logistic map is iterated with an initial value x_0 ∈ (0.01, 0.99) and control parameter r ∈ [3.99, 4.0), both taken from SHA-256(K_sbox):

$$x_{n+1} = rx_n(1 - x_n)$$

After 1,000 transient iterations, 256 samples are collected (keeping every third iterate to reduce serial correlation). The S-box is the rank order of those samples, S = argsort(x_1, …,

x_256), which is a permutation of 0…255 and therefore always bijective. Bytes are substituted with chaining so that a change in one byte propagates to all later bytes:

$$c_i = S[p_i \oplus c_{i-1}],\ p_i = S^{-1}[c_i] \oplus c_{i-1},\ c_0 = K_{sbox}[0]$$

D. AEAD sealing and replay guard (M5, M6)

Each message carries a 64-bit sequence number s that the sender increments and the receiver requires to be strictly greater than the last accepted value:

$$N = N_{pre} \,||s,\ AD = ID_D \,||\, s \,||\, class,\ M_{out} = s \,||AEAD.Enc(K_{Enc}, N, \tilde{p}, AD)$$

Here $\tilde{p}$ is the plaintext after the optional S-box pass. Because N_{pre} is implicit, the wire overhead is 24 bytes per message: 8 bytes of sequence number and a 16-byte tag. Nonces never repeat within a session, which is the core safety condition of both AEAD modes.

```
Algorithm 1  OHLEF-Seal(session, ID_D, p)
  s  <- session.seq ; session.seq <- s + 1
  N  <- session.N_pre || BE64(s)
  AD <- ID_D || BE64(s) || session.class
  if session.class = high then p <- ChaoticSub(p, session.S,
session.c0)
  return BE64(s) || AEAD.Enc(session.K_enc, N, p, AD)

Algorithm 2  OHLEF-Open(session, ID_D, m)
  s  <- BE64^-1(m[0:8])
  if s <= session.last then reject (replay)
  N  <- session.N_pre || m[0:8] ; AD <- ID_D || m[0:8] ||
session.class
  p  <- AEAD.Dec(session.K_enc, N, m[8:], AD)      // reject
on tag failure
  session.last <- s
  if session.class = high then p <- InvChaoticSub(p, session.S,
session.c0)
  return p
```

E. Optimization model (M7)

The optimizer chooses a configuration θ = (suite, R, sbox) for each sensitivity class k. The device-side cost per message of a 256-byte payload is modeled from calibrated primitive timings: handshake H, AEAD time E_suite, S-box pass S and S-box generation G:

$$C(\theta) = \frac{H}{R} + E_{suit} + 1_{sbox}\left(S + \frac{G}{R}\right),\ B(\theta) = 24 + \frac{HB}{R}$$

HB = 192 bytes is the handshake traffic (two 32-byte ephemeral keys and two 64-byte signatures). Long re-key intervals lower cost but enlarge the window of messages exposed if a session key leaks, so exposure is penalized. The objective and constraints are:

$$\min_{\theta} \quad J_k(\theta) = w_1 \frac{C(\theta)}{C_{ref}} + w_2 \frac{B(\theta)}{B_{ref}} + w_3 \frac{R}{R_k^{max}} \quad s.t\ R \le R_k^{max}, sbox = 1\ if\ k = high$$

We use w = (0.5, 0.2, 0.3), C_{ref} = 0.5 ms, B_{ref} = 64 bytes, and R^{max} = 1000, 200 and 50 for the low, medium and high classes. The search space is {ChaCha20-Poly1305, AES-128-GCM} × {10, 25, 50, 100, 200, 500, 1000} × {off, on}, only 28 points per class, so exhaustive enumeration finds the exact optimum in microseconds. Because H, E, S and G are measured on the target at provisioning, the optimizer self-calibrates: on hardware with AES instructions it tends to choose AES-GCM, while on microcontrollers without them ChaCha20-Poly1305 is expected to win, as reported in [15]. Metaheuristics such as whale optimization [12] become useful only if the space grows, for example with per-link energy budgets.

V SECURITY ANALYSIS

OHLEF's confidentiality and integrity rest on standardized primitives (X25519, Ed25519, HKDF, and IND-CCA-secure AEAD), not on the chaotic layer; the S-box adds defense in depth for high-sensitivity data without any claimed security reduction.

A.Security strength

X25519 and Ed25519 provide about 128-bit classical security [21], matching AES-128 and ChaCha20. Equivalent RSA strength would need a 3,072-bit modulus, which is why RSA-based hybrids [9] are heavier than ECC-based ones [11], [15]. None of these primitives is quantum-resistant; Section 7 discusses migration.

B. Resistance to common attacks

Table 3 shows resistance to various types of common attacks that can be achieved through the model.

Table 3 Resistance to common attacks

Attack	OHLEF mechanism	Verified in prototype
Eavesdropping	AEAD encryption under a fresh session key	Ciphertext entropy 7.998 bits/byte (Section 6.4)
Man-in-the-middle key substitution	Ed25519 signature over each ephemeral key and identity	Forged ephemeral key rejected

Attack	OHLEF mechanism	Verified in prototype
		(InvalidSignature)
Message tampering or forgery	128-bit Poly1305/GHASH tag over ciphertext and AD	1-bit flip rejected (InvalidTag)
Replay	Strictly increasing sequence number bound into nonce and AD	Re-sent message rejected
Cross-device or class splicing	Device ID and class bound into AD	Tag fails on mismatched AD
Cipher suite downgrade	Suite and class bound into the HKDF transcript	Different suites derive different keys
Key compromise (past traffic)	Ephemeral ECDH; keys erased after each re-key interval	By construction (forward secrecy)
Key compromise (exposure window)	Re-key interval capped by R^max per class	Optimizer constraint (Section 4.5)
Statistical and pattern analysis	AEAD output, plus chained chaotic S-box for high data	NIST-style tests pass; adjacent-byte correlation below 0.01
Honest-but-curious cloud	Data stays encrypted; DEKs wrapped by KEK	By construction (envelope encryption)

C. Role and limits of the chaotic layer

The chaotic S-box is keyed by K_{sbox}, which is independent of K_{enc} through HKDF, so it cannot weaken the AEAD: an attacker who could break the combined scheme could also break the AEAD alone. Its value is that each high-sensitivity session uses a fresh, secret substitution, so even a future weakness in one AEAD would face an additional unknown permutation, in the spirit of [16], [17]. Its weaknesses are also measurable: a random permutation from a logistic map is cryptographically weaker than the algebraically designed AES S-box (Section 6.4), and floating-point chaos can behave differently across platforms, so device and gateway must use an identical implementation. We therefore treat the layer as optional hardening, not as a security foundation.

VI. PERFORMANCE EVALUATION

For 256-byte telemetry, OHLEF costs 9.0 µs and 24 bytes of overhead per message, 44× faster and 92% smaller than an RSA-2048 + AES-CBC hybrid and 21× faster than per-message ECIES, while its ciphertext passes every statistical test we ran.

A. Experimental setup

The prototype is written in Python 3 using pyca/cryptography 50.0.1 (OpenSSL backend) and NumPy 2.5.3, and runs on a 2-vCPU Intel Xeon at 2.1 GHz with AES instructions. Every timing is the median of 20 to 2,000 repetitions, after one warm-up call. Both endpoints run in one process, so network latency is excluded. This is a server-class platform: absolute times will be one to three orders of magnitude larger on microcontrollers, but the relative ranking of schemes is the quantity of interest. The S-box layer is pure Python and is therefore disproportionately slow next to the C-backed AEADs.

Five configurations are compared on 1,000 messages of 256 bytes:

1. **Baseline A**: RSA-2048-OAEP wraps a fresh AES-128 key per message, data in AES-128-CBC (the pattern of [4], [9]).
2. **Baseline B**: ECIES-style per-message ephemeral ECDH on P-256, HKDF, then AES-128-GCM (the pattern of [7], [8], [11]).
3. **OHLEF-low (ChaCha)**: signed X25519 handshake every 100 messages, ChaCha20-Poly1305.
4. **OHLEF-low (GCM)**: as above with AES-128-GCM.
5. **OHLEF-high**: chaotic S-box + ChaCha20-Poly1305, re-key every 100 messages.

B. Key establishment cost

The table 4 shows key establishment cost of selected algorithms

A complete, mutually authenticated OHLEF handshake for both parties costs about the same as one RSA-2048 private-key decryption, and RSA key generation is about 105× more expensive than the whole handshake. This matches the motivation for ECC-based hybrids in [2], [7], [11].

C. ***Bulk encryption cost***

In the table 5 shows bulk encryption of selective algorithms. The AEAD ciphers are both the fastest and the only single-pass options that also authenticate. Encrypt-then-MAC with

CBC costs 7.6× more than AES-GCM for 256 B and adds four times the overhead. ECB and CBC run slower than GCM here mainly because padding is done in Python; this does not change the conclusion that the unauthenticated modes used in several hybrids [4], [5] buy no speed.

D. End-to-end per-message cost

The table 6 shows end-to end -per message cost in which it is observed that OHLEF's gain comes from amortizing the asymmetric step: Baselines A and B pay a public-key operation on every message, OHLEF once per 100. Overhead falls from 288 to 24 bytes (−92%) because no wrapped key or explicit nonce travels with each message. Even the high profile, with its pure-Python S-box, is 5.0× faster than Baseline A and 2.4× faster than Baseline B.

Table 4 Key establishment cost

Operation	Median time (ms)
RSA-2048 key generation	48.466
OHLEF full handshake, both sides (2 × keygen, sign, verify, ECDH)	0.460
RSA-2048 OAEP decrypt	0.458
RSA-2048 sign	0.324
Ed25519 verify	0.105
ECDH P-256 (key generation + exchange)	0.098
X25519 (key generation + exchange)	0.093
ECDSA P-256 verify	0.085
RSA-2048 OAEP encrypt	0.040
Ed25519 sign	0.032
RSA-2048 verify	0.029
ECDSA P-256 sign	0.025

Table 5 Bulk encryption cost

Cipher	256 B time (µs)	4 KB throughput (MB/s)	1 MB throughput (MB/s)	Overhead (bytes)	Integrity
AES-128-GCM	0.95	3,134	10,852	16 tag + 12 nonce	Yes
ChaCha20-Poly1305	2.10	1,232	2,520	16 tag + 12 nonce	Yes
AES-128-ECB	4.07	840	861	0–16 padding	No
AES-128-CBC + HMAC-SHA256	7.24	328	303	64 (IV, padding, MAC)	Yes
Chaotic S-box layer (Python)	36.5	6.1	—	0	No

Table 6 End-to-end -per message cost

Scheme	Time per message (µs)	Overhead per message (bytes)	Speed-up of OHLEF-low (ChaCha)
Baseline A: RSA-2048 + AES-128-CBC	398.0	288	44.1×
Baseline B: ECIES P-256 + AES-128-GCM	191.3	61	21.2×
OHLEF-high (S-box + ChaCha20-Poly1305)	79.0	24	—

Scheme	Time per message (µs)	Overhead per message (bytes)	Speed-up of OHLEF-low (ChaCha)
OHLEF-low (ChaCha20-Poly1305)	9.0	24	1.0×
OHLEF-low (AES-128-GCM)	9.0	24	—

E. Effect of the re-key interval

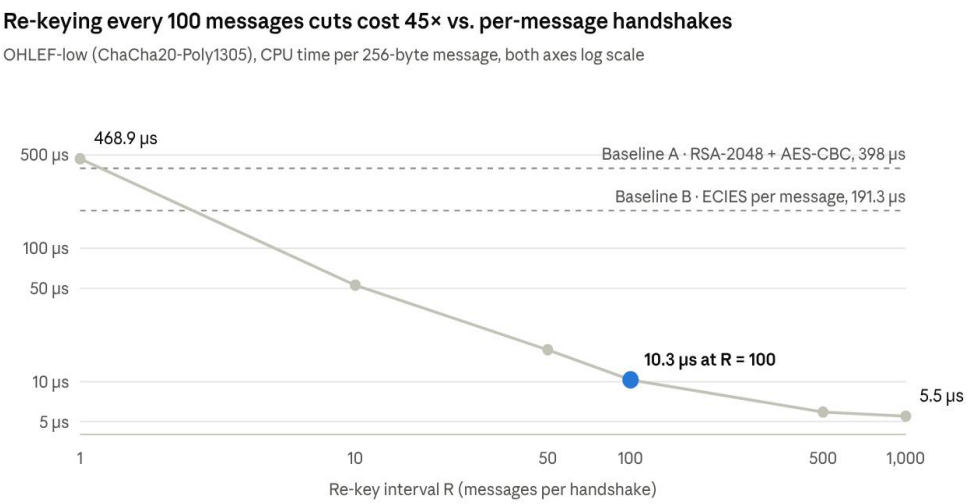


Figure 2

Per-message cost falls steeply as the handshake is shared over more messages, from 468.9 µs when re-keying every message to 5.5 µs at R = 1,000, with most of the benefit reached by R = 100.

Figure 2. Past R ≈ 100 the curve flattens because the AEAD call, not the handshake, dominates; the cap R^max then exists to limit key exposure rather than cost. At R = 1 OHLEF is slower than both baselines, because every handshake is mutually authenticated with two signatures; it overtakes them within the first few messages of a session. The 10.3 µs at R = 100 here versus 9.0 µs in Section 6.4 reflects run-to-run variance between separate benchmark runs.

F. Optimizer output

In the Table 8 shows that exposure term pushes R well below its cap, trading a few microseconds for a 5–20× shorter key-exposure window. On this AES-capable platform AES-GCM wins narrowly; the margin is under 1%, so a calibration on a microcontroller without AES hardware would flip the choice to ChaCha20-Poly1305

G.Statistical security tests:

The statistical security tests are shown in the table 7.
The chaotic S-boxes satisfy the avalanche criterion and are always bijective, and a 1-bit change of the seed changes **S-box quality.** The chaotic S-box was generated from 30 random seeds and compared with the AES S-box.

99.6% of S-box entries. They are, however, clearly weaker than the AES S-box on nonlinearity and differential uniformity, typical of random permutations. This supports Section V-C: the layer adds per-session secrecy, not cryptographic strength, and the AEAD carries the security

Table 7 statistical security test

Metric	AES S-box	Chaotic S-box, mean (min–max over 30 seeds)	Ideal
Bijective	Yes	Yes (30/30)	Yes

Metric	AES S-box	Chaotic S-box, mean (min–max over 30 seeds)	Ideal
Minimum nonlinearity	112	92.9 (88–96)	High (max 120)
Average nonlinearity	112	103.5 (103.1–103.8)	High
Strict avalanche criterion	0.505	0.502 (0.493–0.513)	0.5
Differential uniformity	4	11.3 (10–12)	Low (min 2)
Maximum linear bias	0.063	0.137 (0.125–0.156)	Low
Fixed points	0	0.8 (0–3)	0

Ciphertext randomness. The table 9 shows the ciphertext randomness in which we encrypted 400 messages of 256 bytes (102,400 bytes) of two highly redundant plaintexts: all zeros, and JSON vital-sign telemetry.Every ciphertext reaches entropy within 0.002 of the 8-bit maximum, every χ^2 value is below the 5% critical value of 293.2, and all 16 NIST SP 800-22 p-values exceed the 0.01 threshold [23]. Changing one bit of the shared secret flips 50.06% (±1.06%) of ciphertext bits over 200 trials, and a 1-bit change in the first plaintext byte changes 100% of the S-box layer output bytes because of chaining.

Table 8 Optimizer output

Class	R^max	Chosen suite	Chosen R	S-box	Cost (µs/msg)	Overhead (bytes/msg, incl. handshake share)	Runner-up
Low	1,000	AES-128-GCM	50	Off	10.1	27.8	ChaCha20-Poly1305, R = 50 (J +1.0%)
Medium	200	AES-128-GCM	25	Off	19.3	31.7	ChaCha20-Poly1305, R = 25 (J +0.7%)
High	50	AES-128-GCM	10	On	98.0	43.2	ChaCha20-Poly1305, R = 10 (J +0.4%)

H. Position against reported results

Published figures are not directly comparable, because platforms, payloads and timing scopes differ. With that caveat, OHLEF's 0.46 ms full handshake and 0.95–2.1 µs per-message AEAD sit well inside the 2.07 ms average encryption of DHSE [14], the 3.49 ms encryption of HECC [7] and the 5 ms execution of AHO-ABE [12], and its 24-byte overhead and AEAD design are in line with the ECC + ChaCha20-Poly1305 framework of [15]. Its ciphertext entropy matches the 7.999 reported by chaotic schemes [16], [18] without relying on chaos for security.

I. Position against reported results

Published figures are not directly comparable, because platforms, payloads and timing scopes differ. With that caveat, OHLEF's 0.46 ms full handshake and 0.95–2.1 µs per-message AEAD sit well inside the 2.07 ms average encryption of DHSE [14], the 3.49 ms encryption of HECC [7] and the 5 ms execution of AHO-ABE [12], and its 24-byte overhead and AEAD design are in line with the ECC + ChaCha20-Poly1305 framework of [15]. Its ciphertext entropy matches the 7.999 reported by chaotic schemes [16], [18] without relying on chaos for security.

Table 9

Plaintext → profile	Entropy (bits/byte)	χ^2 (255 df)	Adjacent-byte correlation	NIST p-values: frequency / block / runs / cusum
Zeros (plaintext)	0.000	2.6×10^7	—	—
Zeros → OHLEF-low	7.9981	277.3	−0.0003	0.903 / 0.920 / 0.368 / 0.117
Zeros → OHLEF-high	7.9983	235.5	0.0090	0.923 / 0.119 / 0.335 / 0.701

Plaintext → profile	Entropy (bits/byte)	χ^2 (255 df)	Adjacent-byte correlation	NIST p-values: frequency / block / runs / cusum
Telemetry (plaintext)	0.979	2.0×10^7	0.4228	—
Telemetry → OHLEF-low	7.9981	262.9	−0.0025	0.134 / 0.351 / 0.739 / 0.038
Telemetry → OHLEF-high	7.9983	248.5	0.0018	0.482 / 0.540 / 0.759 / 0.619

VII. DISCUSSION AND LIMITATIONS

The results show that, for cloud-based IoT, the biggest saving comes not from a faster cipher but from paying for public-key cryptography less often, under an explicit bound on key exposure The chaotic S-boxes satisfy the avalanche criterion and are always bijective, and a 1-bit change of the seed changes 99.6% of S-box entries. They are, however, clearly weaker than the AES S-box on nonlinearity and differential uniformity, typical of random permutations.

- **Amortization is the dominant lever.** Moving from per-message to per-100-message key establishment cuts cost by about 45× (Figure 2); swapping AES-GCM for ChaCha20 changes it by under 1%.
- **Authenticated encryption costs nothing extra.** AEAD modes were faster than the unauthenticated or encrypt-then-MAC constructions still common in cloud hybrids [4], [5], [9], so integrity need not be traded for speed.
- **Chaos is hardening, not a foundation.** The chaotic S-boxes behave like random permutations (nonlinearity about 103 versus 112 for AES), consistent with the cautions in [18]; the layer is justified only for high-sensitivity data where an extra secret per session is worth roughly 9× more CPU in our Python build.

LIMITATIONS.

First, measurements were taken on a server CPU, not on an ARM Cortex-M or ESP32 device, and energy was not measured; a hardware study is needed to confirm the ChaCha20 choice and to quantify battery impact. Second, the optimizer weights and R^max caps are policy choices, not derived from a formal risk model. Third, the S-box layer is in pure Python and would need a constant-time C implementation for deployment; table look-ups indexed by secret data are a cache-timing risk. Fourth, the cloud envelope store and ABE access control were designed but not benchmarked. Fifth, X25519 and Ed25519 are not quantum-resistant.

VIII. CONCLUSION AND FUTURE WORK

This paper presented OHLEF, an optimized hybrid lightweight encryption framework for cloud-based IoT built from a review of 19 hybrid and lightweight encryption studies. OHLEF joins signed ephemeral X25519 key establishment, HKDF key separation, AEAD with replay protection, sensitivity-tiered chaotic S-box hardening and a cost-based optimizer for cipher choice and re-keying. In a working prototype it secured 256-byte telemetry for 9.0 µs and 24 bytes per message, 44× faster and 92% smaller than an RSA-2048 + AES-CBC hybrid, while passing entropy, χ^2, correlation and NIST SP 800-22 randomness tests.

Future work will (1) port the device path to ARM Cortex-M4 and ESP32 and measure energy; (2) add the NIST lightweight standard Ascon-AEAD128 [25] as a third suite for the optimizer; (3) replace X25519 with a hybrid X25519 + ML-KEM key exchange for quantum resistance, as urged by [6], [11]; (4) benchmark ABE-based cloud access control [10], [13]; and (5) feed fog-tier intrusion alerts [19] into the optimizer so that re-keying tightens automatically under attack.

REFERENCES

[1] A. Darwish, A. E. Hassanien, M. Elhoseny, A. K. Sangaiah, and K. Muhammad, "The impact of the hybrid platform of internet of things and cloud computing on healthcare systems: opportunities, challenges, and open problems," *Journal of Ambient Intelligence and Humanized Computing*, 2017, doi: 10.1007/s12652-017-0659-1.

[2] N. N. Mohamed, Y. Mohd Yussoff, M. A. Saleh, and H. Hashim, "Hybrid cryptographic approach for Internet of Things applications: A review," *Journal of Information and Communication Technology*, vol. 19, no. 3, pp. 279–319, 2020, doi: 10.32890/jict2020.19.3.1.

[3] N. Chahin and A. Mansour, "Improving the IoT and cloud computing integration using hybrid encryption," *Design, Construction, Maintenance*, vol. 3, 2023, doi: 10.37394/232022.2023.3.1.

[4] O. Alabi, A. F.-B. Thompson, B. K. Alese, and A. J. Gabriel, "Cloud application security using hybrid encryption," *Communications on Applied Electronics*, vol. 7, no. 33, May 2020.

[5] K. K. Reddy, A. R. Chadha, P. S. Nikhil, and S. Sountharrajan, "Hybrid cryptography techniques for data security in cloud computing," in *Proc. 2024 Int. Conf. Computing, Power and Communication Technologies (IC2PCT)*, IEEE, 2024, doi: 10.1109/IC2PCT60090.2024.10486794.

[6] N. J. Shaik and P. K. Thummala, "A hybrid encryption solution to improve cloud computing security using modified symmetric and asymmetric cryptography algorithms," *International Journal of Applied Mathematics*, vol. 38, no. 8s, 2025.

[7] B. R. Rao and B. Sujatha, "A hybrid elliptic curve cryptography (HECC) technique for fast encryption of data for public cloud security," *Measurement: Sensors*, vol. 29, Art. no. 100870, 2023.

[8] S. Ahmad, S. Mehfuz, and J. Beg, "Hybrid cryptographic approach to enhance the mode of key management system in cloud environment," *The Journal of Supercomputing*, vol. 79, pp. 7377–7413, 2023, doi: 10.1007/s11227-022-04964-9.

[9] P. Chinnasamy and P. Deepalakshmi, "HCAC-EHR: hybrid cryptographic access control for secure EHR retrieval in healthcare cloud," *Journal of Ambient Intelligence and Humanized Computing*, 2021, doi: 10.1007/s12652-021-02942-2.

[10] P. Jyotheeswari and N. Jeyanthi, "Hybrid encryption model for managing the data security in medical internet of things," *International Journal of Internet Protocol Technology*, vol. 13, no. 1, pp. 25–31, 2020.

[11] O. Popoola, M. A. Rodrigues, J. Marchang, A. Shenfield, A. Ikpehai, and J. Popoola, "An optimized hybrid encryption framework for smart home healthcare: Ensuring data confidentiality and security," *Internet of Things*, vol. 27, Art. no. 101314, 2024, doi: 10.1016/j.iot.2024.101314.

[12] A. Altherwi et al., "A hybrid optimization approach for securing cloud-based e-health systems," *Multimedia Tools and Applications*, vol. 84, pp. 16525–16560, 2025, doi: 10.1007/s11042-024-19688-6.

[13] M. Jammula, V. M. Vakamulla, and S. K. Kondoju, "Hybrid lightweight cryptography with attribute-based encryption standard for secure and scalable IoT system," *Connection Science*, vol. 34, no. 1, pp. 2431–2447, 2022, doi: 10.1080/09540091.2022.2124957.

[14] D. Garg, S. Rani, N. Herencsar, S. Verma, M. Wozniak, and M. F. Ijaz, "Hybrid technique for cyber-physical security in cloud-based smart industries," *Sensors*, vol. 22, Art. no. 4630, 2022, doi: 10.3390/s22124630.

[15] A. H. Zwiad, "A lightweight hybrid framework for secure communication in low-resource IoT devices," *Journal of Al-Qadisiyah for Computer Science and Mathematics*, vol. 17, no. 4, pp. 309–318, 2025, doi: 10.29304/jqcsm.2025.17.42564.

[16] W. A. Hussain, "A lightweight hybrid encryption algorithm for Internet of Things security using ASCON and AES with chaotic system," *Alkadhim Journal for Computer Science*, vol. 4, no. 2, 2026, doi: 10.61710/pghstr62.

[17] R. Farahi, A. Ghaffari, N. Derakhshanfard, and M. Osmani, "A lightweight hybrid encryption scheme for secure IoT communications based on elliptic curve cryptography and a chaotic dynamic substitution box," *Discover Artificial Intelligence*, 2026 (in press), doi: 10.1007/s44163-026-01949-y.

[18] E. N. Kadhim and K. A. Hussein, "A new hybrid lightweight LEA-PRESENT (HLP) algorithm to encrypt color images," *International Journal of Safety and Security Engineering*, vol. 16, no. 5, pp. 1009–1017, 2026, doi: 10.18280/ijsse.160507.

[19] D. Mohamed and O. Ismael, "Enhancement of an IoT hybrid intrusion detection system based on fog-to-cloud computing," *Journal of Cloud Computing*, vol. 12, Art. no. 41, 2023, doi: 10.1186/s13677-023-00420-y.

[20] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," RFC 8439, IETF, 2018.

[21] A. Langley, M. Hamburg, and S. Turner, "Elliptic curves for security," RFC 7748, IETF, 2016; S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (EdDSA)," RFC 8032, IETF, 2017.

[22] H. Krawczyk and P. Eronen, "HMAC-based extract-and-expand key derivation function (HKDF)," RFC 5869, IETF, 2010.

[23] A. Rukhin et al., "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST SP 800-22 Rev. 1a, 2010.

[24] C. Bormann, M. Ersue, and A. Keranen, "Terminology for constrained-node networks," RFC 7228, IETF, 2014.

[25] M. S. Turan et al., "Ascon-based lightweight cryptography standards for constrained devices," NIST SP 800-232, 2025