Journal of Electronics, Electromedical Engineering, and Medical Informatics
Homepage: jeeemi.org; Vol. 9, No. 1, January 2027, pp: 110-130 e-ISSN: 2656-8632
RESEARCH ARTICLE OPEN ACCESS

Exchange versus Derivation: Comparing ECC Key Exchange and HMAC-SHA256 Key Derivation for IoMT Key Management on the WUSTL-EHMS-2020 Dataset

Sreeja P.[1] and Priyang P. Bhatt[2]

[1] Department of Computer Engineering, Madhuben and Bhanubhai Patel Institute of Technology (MBIT), The Charutar Vidya Mandal (CVM) University, Faculty of Engineering and Technology (FET), New Vallabh Vidyanagar, Gujarat, India
[2] Department of Information Technology, A D Patel Institute of Technology (ADIT), The Charutar Vidya Mandal (CVM) University, Faculty of Engineering and Technology (FET), New Vallabh Vidyanagar, Gujarat, India

Corresponding author: Sreeja P (e-mail: sreejap@mbit.edu.in), **Author(s) Email**: Priyang P. Bhatt (e-mail: priyang.bhatt@cvmu.edu.in)

Abstract

Background: The Internet of Medical Things (IoMT) transmits patient biometrics over untrusted networks. Yet, key management is costly on resource-constrained devices, and most work on the WUSTL-EHMS-2020 dataset detects attacks after they occur rather than protecting data in transit.

Aims: This study aims to compare two ways of supplying a per-record AES-128 key, elliptic-curve Diffie–Hellman (ECDH) key exchange and keyed HMAC-SHA256 derivation from a pre-shared secret, and to test whether an exploratory KaTaPaYadi–LSTM context-binding stage adds any cryptographic benefit.

Methods: The main contributions are a paired comparison of both mechanisms under an identical AES-128-CBC Encrypt-then-MAC stage, three explicitly defined perturbation protocols for the diffusion metrics, and a tested negative result for the context stage. Using all 16,318 records in software simulation, with NIST P-256 for ECC, the study measured key-establishment cost and throughput, applied equivalence testing (TOST) to entropy, avalanche effect, NPCR and UACI, ran nine tampering conditions, and compared four ablation variants.

Results: Without the LSTM stage, HMAC-SHA256 derivation required 5.5 µs per record, 27.6 times less than a fresh P-256 exchange (152.1 µs) and 1.5 times less than a session-amortised exchange (8.4 µs), with throughput of 5,446.5 versus 540.6 KB/s. The two mechanisms were statistically equivalent on every metric, and all modified, substituted, reordered, truncated and forged records were rejected (16,318 of 16,318 in each class); the context stage, in contrast, changed no measured property but cost about 1,900 times the HMAC step.

Conclusion: These findings indicate that HMAC-SHA256 derivation without the context stage is the lower-cost choice where a secret can be pre-provisioned, whereas ECC remains necessary for first-time pairing; hardware validation is required before clinical use.

Keywords IoMT Security; Key Management; Elliptic Curve Cryptography; HMAC-SHA256; Authenticated Encryption; WUSTL-EHMS-2020.

I. Introduction

The Internet of Medical Things (IoMT) links wearable and bedside sensors, such as pulse oximeters, blood-pressure cuffs, ECG patches and respiration monitors, to gateways and clinical servers, enabling continuous remote patient monitoring outside traditional hospital settings. These data cross several untrusted links on their way to a clinician, and an attacker positioned between the gateway and the server can intercept, spoof or alter them. Hady et al. built and released the WUSTL-EHMS-2020 testbed dataset to study this scenario [1]. It contains 16,318 flow records that combine patient biometrics with network traffic: 14,272 normal records and 2,046 man-in-the-middle attacks, of which 1,124 are spoofing and 922 are data alteration. Key supply is the difficult part of protecting such traffic on constrained hardware. AES remains the default cypher in IoMT remote monitoring [2], but a static pre-shared key, once exposed, leaves every message protected under it readable, the weakness that forward secrecy addresses [3]. Two provisioning models address this. In the first, elliptic-curve key exchange negotiates a shared secret between two parties, with RSA-equivalent security at smaller key sizes [4], [5]; its costs across curves and platforms are catalogued in lightweight-cryptography surveys [6]–[9]. In the second, keyed-hash derivation computes each key from a secret that both parties already hold. HMAC-SHA256 [10] is supported by the hardware accelerators found on microcontrollers common in medical wearables [11], [12]. The two models rest on different trust assumptions, so comparing them is a comparison of provisioning models and not merely of algorithms.

Most research on this dataset detects attacks after they occur. Alsolami et al. reported detection accuracy of up to 98.88% with Stacking, Bagging and Boosting ensembles [13]; Alalhareth and Hong proposed a

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

performance-driven meta-learning ensemble [14]; and Wu et al. combined multiple deep-learning models in an ensemble detector [15]. These studies show that attacks can be recognised, but not how the data should be protected in transit. On the cryptographic side, Garg et al. combine elliptic-curve key agreement with one-way hash functions in a lightweight authentication scheme for smart-metering infrastructure [16]. Hash-based key generation has been studied for generic IoT [17], AI-assisted key-generation schemes have been proposed in the same setting [18], and within IoMT, cryptographic work has concentrated largely on medical-image encryption [7], [19], [20].

Three gaps follow from this literature. First, detection-oriented studies on this dataset [13]–[15] do not address key management. Second, ECC and hash benchmarks use generic hardware, smart-grid traffic or medical images, evaluate a single mechanism or compare mechanisms across different platforms, and do not compare exchange and derivation under an identical cypher stage on patient-monitoring records, so reported costs cannot be read directly as costs for such traffic. Third, AI-assisted schemes [18] typically assume that the AI stage contributes to security without testing it against simple baselines, and the image-encryption metrics NPCR and UACI [21], often carried over to such schemes, are reported without stating what is perturbed, which makes results difficult to compare.

To address these gaps, this study compares ECC key exchange and HMAC-SHA256 key derivation as sources of a per-record AES-128 key on all 16,318 records of WUSTL-EHMS-2020, using an identical AES-128-CBC Encrypt-then-MAC stage for both so that differences are attributable to key supply alone. The aim is to determine how the two provisioning models differ in cost and in the statistical quality of the keys and ciphertexts they produce, whether either weakens integrity verification, and whether an exploratory KaTaPaYadi–LSTM context-binding stage adds any measurable benefit. The analysis assumes an on-path attacker who cannot obtain the pre-shared secret used in the HMAC variant; the consequences of its compromise are discussed in Sections II-H and IV-D.

The KaTaPaYadi–LSTM stage is included to test, not to assert, a claim. Binding a key to its record requires only a deterministic function of the session context, and HMAC-SHA256 over the session descriptor already provides this, so the stage is not presumed necessary. It combines KaTaPaYadi encoding, a classical Indian phonetic-to-numeral system documented by Sarma [22], with a fixed-seed, untrained two-layer LSTM [23] that maps the encoded sequence and the record's features to a context-dependent vector placed ahead of key derivation. Because that vector is a deterministic function of known inputs, it adds no entropy, and the key's security rests on the pre-shared secret or the ECDH secret. The hypothesis tested is that the stage improves a measurable property beyond plain context binding, and a four-way ablation (Section III-D) examines it.

The contributions of this study are 1) a paired, per-record comparison of ECC key exchange (NIST P-256, fresh and session-amortised) and HMAC-SHA256 derivation under an identical Encrypt-then-MAC stage, showing the derivation to be 27.6 times faster than a fresh exchange and 1.5 times faster than a session-amortised one, but 69 times slower when the LSTM stage is included; 2) three explicitly defined perturbation protocols for NPCR, UACI and avalanche effect on per-record CBC payloads (NPCR of 99.61% when the IV is refreshed against 61.70% for a plaintext change under a fixed key and IV), with paired equivalence testing showing that the two mechanisms are statistically equivalent under all three; 3) an integrity experiment with nine record-modification conditions in which every modified, substituted, reordered, truncated and forged record was rejected, together with a documented limitation that replay is not prevented by the tag; and 4) a tested negative result for the KaTaPaYadi–LSTM stage, which changed no measured property and cost about 1,900 times the HMAC step, together with deployment guidance on when each mechanism applies.

The remainder of this paper is organised as follows. Section II describes the dataset, the framework and both key-management variants, the threat model, and the evaluation metrics and protocols. Section III reports the results on cost, throughput, diffusion, entropy, integrity verification, the ablation and equivalence testing. Section IV discusses the findings, compares them with related studies, and states the limitations and deployment guidance. Section V concludes.

II. Methods

A. Dataset

This study uses WUSTL-EHMS-2020, a publicly available IoMT security dataset created by Hady et al. at Washington University in St. Louis [1]. It contains 16,318 records: 14,272 normal (87.5%), 1,124 spoofing (6.9%), and 922 data-alteration (5.7%) attack records, each with 45 fields spanning patient biometrics and network flow features.

B. Attack Scenarios Represented in the Dataset

Spoofing attacks (1,124 records) position the attacker between the gateway and the server to passively intercept packets, threatening patient data confidentiality. Data-alteration attacks (922 records) modify packet contents in transit, threatening data integrity in a way that could directly mislead clinical decision-making if the modification goes unnoticed. These two classes define the attacker capabilities assumed in Section II-H: interception and modification of records on the network path. A per-record dynamic key limits the value of any single compromised key to one biometric reading rather than an entire monitoring session. In this framework, confidentiality and integrity are enforced cryptographically, by AES encryption and MAC verification, rather than by detecting attack traffic.

C. Framework Overview

The framework (Fig. 1) has four stages. Stages 1 and 2 feed Variant 2 only, and Stage 4 is identical for both variants. In Stage 1, a per-record session descriptor φ, built from public fields of the record, is converted by KaTaPaYadi encoding into a fixed-length digit sequence.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

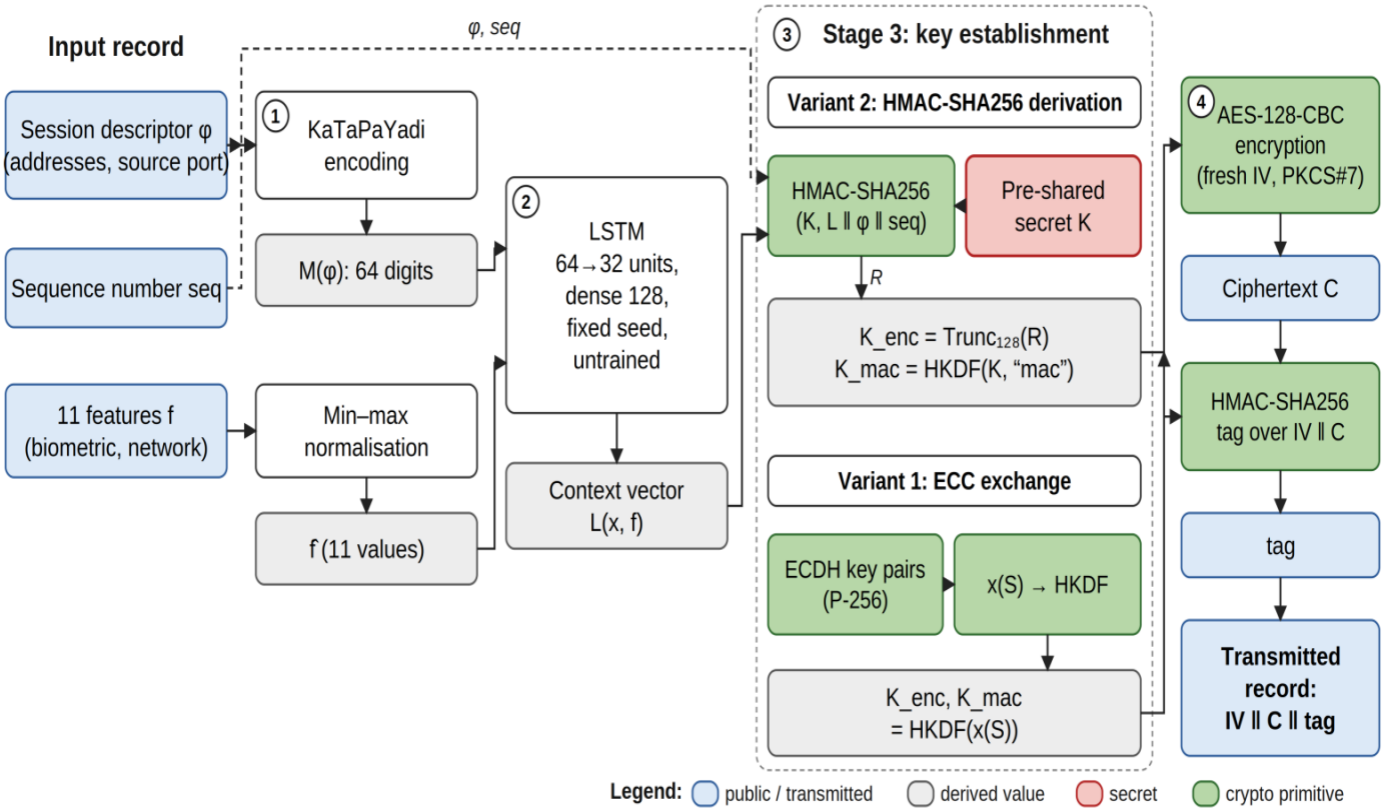

Fig. 1. Overview of the proposed IoMT security framework comparing ECC-based key exchange and HMAC-SHA256-based key derivation. The framework comprises session-context encoding, LSTM-based context-vector generation, cryptographic key establishment, and AES-128-CBC encryption with HMAC-SHA256 authentication. Context encoding and LSTM processing apply to the HMAC-SHA256 variant, while the encryption and authentication stage is shared by both variants. Software simulation only.

In Stage 2, a fixed-seed two-layer LSTM maps this sequence and the record's 11 normalised features to a 128-dimensional context vector L(x, f). In Stage 3, keying from a pre-shared secret K over L(x, f), φ and the packet sequence number (Variant 2); HKDF then separates this material into independent encryption and MAC keys. In Stage 4, the payload is encrypted with AES-128-CBC and authenticated with an Encrypt-then-MAC construction, and the receiver verifies the tag before decryption, discarding any record that fails.

In Variant 2, Stages 1 and 2 bind each record's session context to the key input in a deterministic, reproducible way. They do not add entropy, and the strength of the derived keys rests on the standardised primitives of Stages 3 and 4 (Section III). The two variants share Stage 4 and differ in how the keying material is supplied, so differences in cost and diffusion are attributable to key supply alone.

D. Formal Notation and Field Classification

This section formalises each processing stage and cites the source of every standard construction. Where an equation is our own formalisation of a described procedure, it is stated as such.

1. KaTaPaYadi encoding

The encoding follows the classical KaTaPaYadi numeral system [22]; Eqs. (1)–(3) [22] are our formalisation of it for session descriptors. The encoder maps a descriptor string to a fixed-length digit sequence:

$$M: \Sigma^* \rightarrow \{0, 1, \ldots, 9\}^{64} \quad (1)$$

$$M(\varphi) = \mathrm{Pad}_{64}\big(\delta(c_1) \parallel \delta(c_2) \parallel \cdots \parallel \delta(c_k)\big) \quad (2)$$

$$\delta(c) = \begin{cases} m(c) & \text{if c is a consonant} \\ c & \text{if c is a decimal digit} \\ \varepsilon & \text{otherwise} \end{cases} \quad (3)$$

where Σ* is the set of all finite character strings, φ = $c_1c_2\ldots c_k$ is the session descriptor after conversion to lowercase, k is its length, m(c) ∈ {0,…,9} is the Kaṭapayādi digit of consonant c (Section II-E), ε is the empty string, ‖ denotes concatenation, and $\mathrm{Pad}_{64}(\cdot)$ zero-pads or truncates its argument to exactly 64 digits. Vowels and separators therefore contribute nothing to M(φ).

2. Feature normalisation

Each of the 11 biometric and network features is min–max normalised to the unit interval, as in Eq. (4) [24]:

$$\hat{f}_i = \frac{f_i - \min(f_i)}{\max(f_i) - \min(f_i)} \quad (4)$$

where f_i is the raw value of feature i (i = 1,…,11), $\min(f_i)$ and $\max(f_i)$ are its minimum and maximum over the full dataset, and $\hat{f}_i \in [0, 1]$ is the normalised value; $\hat{f} = (\hat{f}_1,\ldots,\hat{f}_{11})$.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

3. LSTM context binding

The normalised features and the digit sequence x = M(φ) = (x_1,…,x_{64}) drive a two-layer LSTM. The gate formulation with a forget gate is that of Gers et al. [25], extending the original LSTM of Hochreiter and Schmidhuber [23] (see also [26]). At each time step t = 1,…,64, each layer computes Eqs. (5)–(10) [23], [25], [26]:

$$g_t = \sigma\left(W_g[h_{t-1}, z_t] + b_g\right) \quad (5)$$

$$i_t = \sigma\left(W_i[h_{t-1}, z_t] + b_i\right) \quad (6)$$

$$\tilde{c}_t = \tanh\left(W_c[h_{t-1}, z_t] + b_c\right) \quad (7)$$

$$c_t = g_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (8)$$

$$o_t = \sigma\left(W_o[h_{t-1}, z_t] + b_o\right) \quad (9)$$

$$h_t = o_t \odot \tanh(c_t) \quad (10)$$

where g_t, i_t and o_t are the forget, input and output gates; $\tilde{c}_t$ and c_t are the candidate and cell states; h_t is the hidden state; $[h_{t-1}, z_t]$ is the concatenation of the previous hidden state with the layer input z_t; W_g, W_i, W_c, W_o and b_g, b_i, b_c, b_o are the layer's weight matrices and bias vectors; $\sigma(u) = 1/(1 + e^{-u})$ is the logistic sigmoid; tanh is the hyperbolic tangent; and $\odot$ is the element-wise (Hadamard) product. For layer 1, $z_t = [\hat{f}; x_t] \in \mathbb{R}^{12}$ (the 11 features and the t-th digit) and h_t has 64 units; for layer 2, z_t is the hidden state of layer 1, and h_t has 32 units. Weights are fixed by seed 42 and are not trained.

4. Output layer

A dense sigmoid layer maps the final hidden state of layer 2 to the context vector, as in Eq. (11) [26]:

$$L(x, f) = \sigma\left(W_2 h_T^{(2)} + b_2\right) \quad (11)$$

where $h_T^{(2)} \in \mathbb{R}^{32}$ is the layer-2 hidden state at the last step T = 64, $W_2 \in \mathbb{R}^{128\times32}$ and $b_2 \in \mathbb{R}^{128}$ are the dense weights and bias, and $L(x, f) \in (0,1)^{128}$. L is a deterministic function of its inputs, not a pseudo-random function.

5. HMAC

For a hash function Hash with block size B and key K, the HMAC construction, given by Eq. (12) [27], [28] (security analysis in [10]):

$$\mathrm{HMAC}(K, m) = \mathrm{Hash}\left((K' \oplus \mathrm{opad}) \parallel \mathrm{Hash}\left((K' \oplus \mathrm{ipad}) \parallel m\right)\right) \quad (12)$$

where m is the message, K′ is K zero-padded to B bytes (or Hash(K) zero-padded, if K is longer than B), ipad and opad are the bytes 0x36 and 0x5C repeated B times, and $\oplus$ is bitwise XOR. With Hash = SHA-256 [29], B = 64 bytes and the output has 256 bits.

6. Per-record key

The keying material is derived from the pre-shared secret K, which is provisioned out-of-band at device–gateway pairing and never transmitted, as in Eqs. (13) and (14) [10], [30]:

$$R = \mathrm{HMAC}_{\mathrm{SHA256}}(K, L(x, f) \parallel \varphi \parallel \mathrm{seq}) \quad (13)$$

$$K_{\mathrm{enc}} = \mathrm{Trunc}_{128}(R) \quad (14)$$

Table 1. Classification of the key-derivation inputs and what the attacker is assumed to know about each of them.

Input	Symbol	Class	Origin	Role in Eq. (13)	Known to attacker?
Source address	– (part of φ)	Public	Packet header, transmitted in clear	Component of φ	Yes, observable
Destination address	– (part of φ)	Public	Packet header, transmitted in clear	Component of φ	Yes, observable
Source port	– (part of φ)	Public	Packet header, transmitted in clear	Component of φ	Yes, observable
Session descriptor	φ	Derived (public)	Concatenation of the three header fields above (Section II-E)	Context input	Yes, computable
Sequence number	seq	Public	Packet sequence number	Per-record input	Yes, observable
Context vector	L(x, f)	Derived	LSTM output from M(φ) and $\hat{f}$ (Eq. 11)	Context input	Assumed yes (conservative)
Pre-shared secret	K	Secret	Provisioned out-of-band at pairing; never transmitted	HMAC key	**No**
Keying material	R	Derived (secret)	HMAC output (Eq. 13)	Source of K_{enc} (Eq. 14)	No, without K

where seq is the record's packet sequence number (seq_id in Fig. 1), R is the 256-bit per-record keying material, and $\mathrm{Trunc}_{128}(\cdot)$ returns the leftmost 128 bits. All operands are converted to byte strings before concatenation. Truncating a 256-bit HMAC output to 128 bits is permitted [30] and matches the AES-128 key length [31]. Because K is unknown to the attacker defined in Section II-H, R cannot be recomputed from intercepted traffic, even though L, φ and seq are observable or computable; the security argument rests on the PRF property of HMAC under K [10] (Section II-H). Table 1 classifies every input to Eq. (13) as public, derived or secret.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

E. KaTaPaYadi Linguistic Encoding

For each record, a session descriptor φ is built by concatenating the public header fields (source address, destination address and source port, joined by a separator). These fields are available to both endpoints at runtime. The attack label and the MAC address are deliberately excluded from φ. The label is not available to an endpoint at inference time, and in the WUSTL-EHMS-2020 testbed the MAC-address field separates attack from normal traffic perfectly (one value occurs only in normal records and the other only in attack records), which would make it an unintended proxy for the label. An earlier version of φ included both fields. They were removed, and the category-wise analysis in Section III-B was repeated with the corrected descriptor. After conversion to lowercase, each character of φ is processed from left to right, as formalised in Eqs. (1)–(3). Consonants are mapped to a digit from 0 to 9 following the classical KaTaPaYadi assignment [22], in which several consonants share each digit (for example, ka, ṭa, pa and ya all map to 1); Latin letters are assigned digits as follows: 0 for n; 1 for k, p, q, x and y; 2 for f and r; 3 for b, g and l; 4 for v and w; 5 for m; 6 for c and t; 7 for s; and 8 for d, h, j and z, where letters absent from the classical table (f, q, w, x, z) take the digit of their nearest phonetic equivalent. Vowels and separators are skipped, and digit characters pass through unchanged. The sequence is zero-padded or truncated to exactly 64 digits. In WUSTL-EHMS-2020 the source and destination addresses are constant, so descriptor variability comes entirely from the source port, which is numeric in 16,315 of 16,318 records; only three records carry a service name (dircproxy, fido, tfido), so the consonant mapping is active for 0.02% of records, and the encoder otherwise reduces to digit extraction. Because several consonants share a digit, separators are dropped, and padding uses zeros, distinct descriptors can produce the same sequence; the encoding is lossy. This does not weaken key uniqueness, because φ itself is an input to Eq. (13). The collision rate is quantified in Section III-D.

Table 2. Layer-wise architecture of the LSTM-based session-context network, including layer configurations, input and output dimensions, and parameter counts (fixed, untrained weights).

Layer	Configuration	Input → output	Parameters
Input	Input sequence	64 steps × 12 values (11 features + 1 digit)	–
LSTM 1	64 units, tanh	64 × 12 → 64 × 64	19,712
LSTM 2	32 units, tanh	64 × 64 → 32 (last step)	12,416
Dense	128 units, sigmoid	32 → 128	4,224
Total	Fixed seed 42, untrained	—	36,352 (≈142 KiB at 32-bit)

F. LSTM Contextual Processing

The 64-digit KaTaPaYadi sequence x = M(φ) and the normalised 11-dimensional feature vector f̂ (temperature, SpO_2, pulse rate, systolic and diastolic pressure, heart rate, respiration rate, ST segment, flow duration, total bytes and flow rate) are processed by a two-layer LSTM with 64 and 32 units and tanh activation, followed by a sigmoid dense layer that produces the 128-dimensional context vector L(x, f) (Eqs. (5)–(11); Table 2). At each of the 64 time steps, the first layer receives the concatenation of f̂ and the current digit, so the same feature vector is presented at every step. The architecture is summarised in Table 2.

G. AES Module and Key-Management Variants

Each record's plaintext payload is encrypted with AES-128 [31] in Cipher Block Chaining (CBC) mode [32] with PKCS#7 padding and a fresh 16-byte initialisation vector (IV) drawn from a cryptographically secure random source. An 88-byte payload therefore becomes a 96-byte ciphertext of six blocks. Authentication and integrity are provided by an Encrypt-then-MAC construction [33]: the ciphertext is computed first, and the MAC is then computed over the IV and the ciphertext. Encryption proceeds block by block as in Eq. (15) [31], [32]:

$$C_i = E_{K_{\text{enc}}}(P_i \oplus C_{i-1}),\ C_0 = \text{IV} \qquad (15)$$

where P_i is the i-th 128-bit plaintext block after padding, C_i is the corresponding ciphertext block, E_{Kenc} is AES-128 encryption under the 128-bit encryption key K_{enc}, and IV is the per-record initialisation vector. The authentication tag is given by Eq. (16) [10], [33]:

$$\text{tag} = \text{HMAC} - \text{SHA256}(K_{\text{mac}}, \text{IV} \parallel C) \qquad (16)$$

where K_{mac} is the 128-bit MAC key and $C = C_1 \parallel C_2 \parallel \cdots \parallel C_n$ is the complete ciphertext. The transmitted record is IV ‖ C ‖ tag. The receiver recomputes the tag from the received IV and ciphertext, compares it with the received tag in constant time, and decrypts only if they match; otherwise, the record is discarded before decryption is attempted.

1. Key separation

To avoid using one key in both roles, K_{enc} and K_{mac} come from distinct derivations. Wherever keys are derived from shared secret material, HKDF [34], [35] is used, in two steps. The extract step produces a fixed-length pseudorandom key, Eq. (17) [34], [35]:

$$\text{PRK} = \text{HMAC} - \text{Hash}(\text{salt}, \text{IKM}) \qquad (17)$$

where IKM is the input keying material (K or x(S), defined below), salt is an optional extraction salt (a string of zero bytes if omitted), Hash = SHA-256, and PRK is the resulting pseudorandom key. The expand step derives the output keying material, Eqs. (18) and (19) [34], [35]:

$$\text{OKM} = T(1) \parallel T(2) \parallel \cdots \parallel T(N) \qquad (18)$$

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

$$T(i) = \text{HMAC} - \text{Hash}(\text{PRK}, T(i-1) \parallel \text{info} \parallel i),\ T(0) = \varepsilon \quad (19)$$

where info is a role label ("enc" or "mac") that binds each derived key to its role, i is a one-byte counter, N = $\lceil \ell/256 \rceil$ is the number of blocks needed for ℓ output bits, ε is the empty string, and OKM is truncated to the leftmost ℓ = 128 bits for each key.

2. Variant 2 (HMAC-SHA256 derivation)

The per-record encryption key is $K_{enc} = \text{Trunc}_{128}(R)$, with R from Eq. (13), so that each record is encrypted under a distinct key derived from the pre-shared secret K, the context vector L(x, f), the session descriptor φ and the sequence number. The MAC key K_{mac} is obtained once per device–gateway pair by HKDF with IKM = K and info = "mac", so the two keys never coincide.

3. Variant 1 (ECC key exchange)

Variant 1 simulates a device–gateway key exchange by elliptic-curve Diffie–Hellman (ECDH) [36] on the NIST P-256 curve (secp256r1) [37], whose security strength matches that of AES-128 [38]. The legacy SECP192R1 curve used in earlier work falls below the 112-bit minimum strength and is no longer an approved curve, and it is therefore retained in Section III-A only for continuity. The device (A) and the gateway (B) each generate a key pair over the curve's base point G of order n, as in Eq. (20) [36], [37]:

$$d_A, d_B \in_R [1, n-1];\ \ Q_A = d_A \cdot G,\ \ Q_B = d_B \cdot G \quad (20)$$

where d_A and d_B are the private scalars, kept secret at each endpoint, and Q_A and Q_B are the public points exchanged over the network path, and therefore available to the attacker defined in Section II-H. Each party computes the shared point by scalar multiplication, Eq. (21) [36]:

$$S = d_A \cdot Q_B = d_B \cdot Q_A = (d_A d_B) \cdot G \quad (21)$$

By the commutativity of scalar multiplication, both parties obtain the same point S, and only Q_A and Q_B cross the network. The x-coordinate x(S) serves as IKM in Eqs. (17)–(19): K_{enc} is derived with info = "enc" and K_{mac} with info = "mac". The public points are assumed to be authenticated (for example by provisioned certificates); the cost of that authentication is not included in the measurements. A session-amortized configuration, in which one handshake per session establishes x(S) and per-record keys are derived from it by HKDF, is also evaluated (Section III-A). Algorithm 1 summarises the complete per-record procedure for both variants.

H. Threat Model

1. Adversary

The framework is evaluated against the threat model implicit in the WUSTL-EHMS-2020 testbed: an attacker on the network path between the gateway and the visualisation server who knows every algorithm and parameter and observes every transmitted field (Table 1). The attacker may passively intercept records (spoofing), and may actively modify, replay, substitute, reorder or truncate them (data alteration). Conservatively, the attacker is also assumed able to compute the context vector L(x, f), so no security claim rests on L being secret. The attacker has no physical access to sensor, gateway or server hardware, and has no knowledge of the pre-shared secret K or of the private scalars d_A and d_B. Fig. 2 summarises the transmission path, the attacker's capabilities and the receiver's verification of each record.

2. Assumptions and provisioning

For Variant 2, K is a uniformly random secret of at least 128 bits, unique to each device–gateway pair, provisioned out-of-band at pairing, and held in protected memory, for example, a secure element on the device and a hardware security module or encrypted key store at the gateway. Rotation is by re-pairing, and revocation by deleting K at the gateway and re-provisioning the device. These are deployment assumptions, not mechanisms evaluated in this study. For Variant 1, the public points Q_A and Q_B are assumed authenticated, for example by provisioned

Algorithm 1: Per-record cryptographic key establishment and authenticated encryption using ECC-based key exchange or HMAC-SHA256-based key derivation, including context processing, encryption-key generation, AES-128-CBC encryption, and HMAC-SHA256 authentication-tag computation (Fig. 1; Eqs. (1)–(21)).

Input: session descriptor φ, features f, pre-shared secret K, sequence number seq, plaintext P, variant ∈ {ECC, HMAC}
Output: transmitted record IV ‖ C ‖ tag; receiver accept/reject decision
Function PROTECT(φ, f, K, seq, P, variant):
  $\hat{f}$ ← min–max normalisation of f; φ ← source address ‖ destination address ‖ source port
  if variant = HMAC **then**
    x ← KaTaPaYadi(φ); L ← LSTM(x, $\hat{f}$)
    R ← HMAC-SHA256(K, L ‖ φ ‖ seq); K_{enc} ← $\text{Trunc}_{128}(R)$
    K_{mac} ← HKDF(K, "mac")
  else
    generate (d_A, Q_A) and (d_B, Q_B); S ← $d_A \cdot Q_B$
    K_{enc} ← HKDF(x(S), "enc"); K_{mac} ← HKDF(x(S), "mac")
  end if
  IV ← 16 random bytes; C ← AES-128-CBC(K_{enc}, IV, PKCS7(P))
  tag ← HMAC-SHA256(K_{mac}, IV ‖ C); transmit IV ‖ C ‖ tag
  Receiver: recompute keys as above; tag′ ← HMAC-SHA256(K_{mac}, IV ‖ C)
  if tag′ ≠ tag (constant-time comparison) **then**
    discard the record
  else
    decrypt and unpad
  end if
Return transmitted record and accept/reject decision

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

Table 3. Threat coverage of the two key-management variants and the evidence supporting each statement.

Threat	Variant 2 (HMAC-SHA256)	Variant 1 (ECC)	Evidence
Passive interception (confidentiality)	Protected while K is secret	Protected if ECDH holds and Q is authenticated	Analytical (Section II-H)
Modification of IV, ciphertext or tag	Rejected by tag	Rejected by tag	Section III-C
Substitution of public points	Not applicable (no public exchange)	Not prevented unless Q is authenticated	Analytical
Replay, reordering, whole-record dropping	Not prevented; needs sequence check	Not prevented; needs sequence check	Section III-C (limitation)
Compromise of long-term secret	K exposes all records of that pair (no forward secrecy)	Ephemeral keys protect past records	Analytical
Device capture, gateway compromise (side channels and DoS out of scope)	Exposes K of the pair (device) or of every pair held (gateway)	Exposes long-term and live session keys; earlier ephemeral sessions protected	Analytical

certificates; the cost of that authentication is not included in the measurements.

3. Security argument

The confidentiality claims for Variant 2 rest on the pseudo-random-function (PRF) security of HMAC-SHA256 under the secret K [10]: for distinct inputs, the keys K_{enc} = $Trunc_{128}(R)$ are computationally indistinguishable from independent random keys to an attacker without K, even when consecutive records have correlated inputs, since PRF security is defined against adaptive, correlated queries. Records are encrypted under AES-128-CBC with a fresh random IV [32]. For integrity, the Encrypt-then-MAC construction [33] with an HMAC-SHA256 tag means that any modification of the IV or ciphertext is rejected before decryption, except with negligible probability: an attacker without K_{mac} succeeds with probability at most about 2^{-256} per forgery attempt, plus its advantage against HMAC, and cannot do better than exhaustive search over the 128-bit K_{mac}. This establishes cryptographic integrity verification: a modified record is verifiably rejected. It is not an intrusion-detection capability, and the framework does not classify or locate attacks. The claims rest on standard, independently analysed assumptions, and no formal proof is offered for the combined pipeline beyond them.

4. Consequences of compromise

If K is compromised, an attacker who observes the public fields can recompute R, and hence K_{enc}, for every past and future record of that pair, and can forge tags using K_{mac}. Variant 2 therefore provides no forward secrecy and loses both confidentiality and integrity for the affected pair; compromise is confined to that pair, since K is unique per pair. Variant 1 with ephemeral key pairs keeps past records protected if a long-term authentication key is later exposed (forward secrecy). In the session-amortized configuration, the exposure of a session secret reveals all records derived from it. Conversely, ECDH without authenticated public points is vulnerable to an active attacker who substitutes Q_A or Q_B, which is exactly the spoofing scenario of the dataset; Variant 1 is therefore only as strong as its public-key authentication, and Variant 2 only as strong as the secrecy of K.

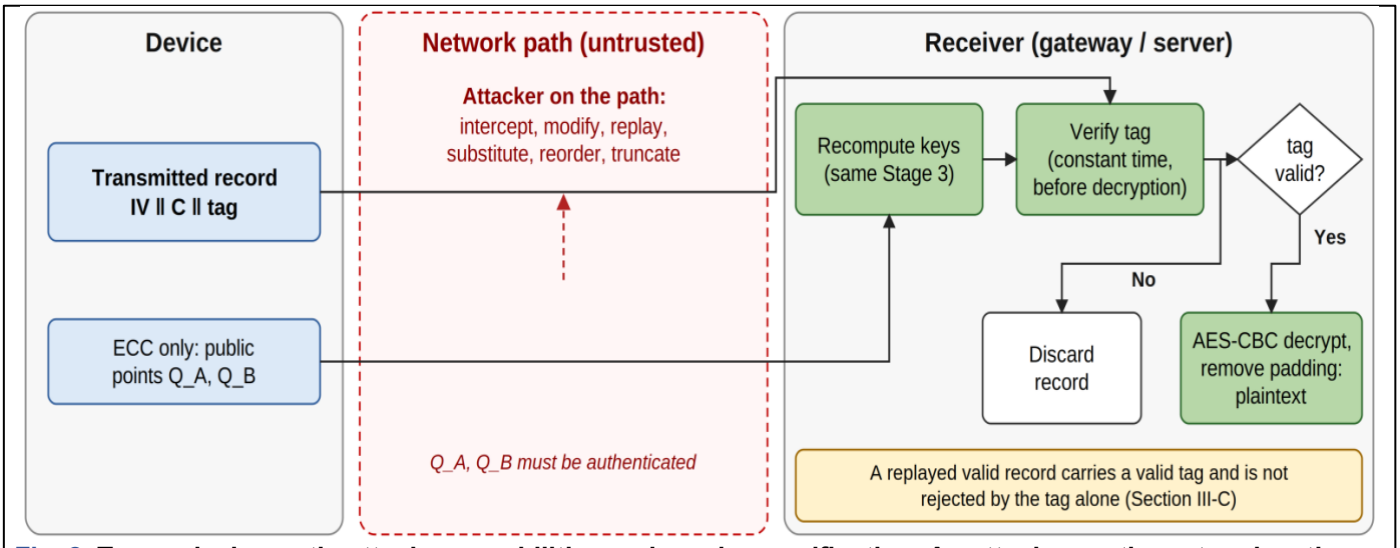

Fig. 2. Transmission path, attacker capabilities and receiver verification. An attacker on the network path can intercept, modify, replay, substitute, reorder or truncate records; the receiver recomputes the keys, verifies the tag in constant time before decryption, and discards any record that fails. A replayed valid record is not rejected by the tag alone.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

5. Replay, reordering and truncation

The tag covers IV ‖ C only. A replayed, reordered or dropped whole record carries a valid tag, so these are not prevented by key derivation alone and require a receiver-side check on monotonic sequence numbers, which this study does not implement. Modification of any part of a record, including truncation within a record, changes IV ‖ C and is rejected.

6. Out of scope

Side-channel attacks and denial of service are not considered. Physical capture of a device and compromise of a gateway lie outside the adversary model, but their consequences follow from the analysis above and were analysed, not tested. Capturing a device exposes its K, and so every record of that device–gateway pair under Variant 2, whereas under Variant 1 it exposes the long-term authentication key and leaves earlier ephemeral sessions protected. Compromising a gateway exposes every K that it stores, so a gateway holding one K per device exposes the whole fleet under Variant 2, whereas under Variant 1 it exposes only the session secrets it holds at that time. Table 3 summarises coverage.

I. Security Analysis and Evaluation Metrics

1. Security analysis

The effective key space of both variants is bounded by the 2^{128} keys of AES-128 [31]. The P-256 exchange of Variant 1 offers a matching strength of about 128 bits [38]. The KaTaPaYadi and LSTM stages diversify the input to HMAC-SHA256 but do not extend this bound. The security of HMAC rests on the secrecy of K and not on the secrecy of its message [10], so the network-flow features in f (duration, total bytes, flow rate), which are observable to the attacker in Section II-H and partly correlated with traffic category, do not weaken the derivation in Eq. (13). The explicit limitations are as follows. The LSTM is not a cryptographically secure random bit generator in the sense of NIST SP 800-90A [39]. No formal proof is offered for the combined pipeline beyond the PRF assumption on HMAC-SHA256 and the standard properties of AES-CBC and Encrypt-then-MAC. The statistical metrics defined below are indicators of diffusion and randomness, not proofs of security, and all results are from software simulation.

2. Entropy

Randomness of byte sequences is measured by Shannon entropy (Eq. (22) [40]):

Table 4. Perturbation protocols used to compute the diffusion metrics and what each of them measures.

Protocol	Perturbation	Key	IV	What it measures
P1 Plaintext sensitivity	One random plaintext bit flipped	Unchanged	Unchanged	Propagation of a plaintext change through CBC
P2 IV re-randomisation	Same bit flip	Unchanged	Fresh random IV	Ciphertext independence under a new IV
P3 Key sensitivity	One random bit of K_{enc} flipped	Flipped	Unchanged	Dependence of the ciphertext on the key

$$H(X) = -\sum_{j=0}^{255} p_j \log_2 p_j \quad (22)$$

where X is the analysed byte sequence, p_j is the relative frequency of byte value j in X, and terms with $p_j = 0$ are omitted. Entropy is computed (a) over the pooled bytes of all derived encryption keys (261,088 bytes, maximum 8 bits/byte), and (b) per record over the 96-byte ciphertext, whose maximum is $\log_2 96 \approx 6.585$ bits/byte. Because the plug-in estimator is biased downward in finite samples [41], ideal random data does not reach the maximum: it gives about 7.9993 bits/byte for the pooled keys and about 6.24 bits/byte per ciphertext. Reported values are compared with these expected values, not with the maxima.

3. Diffusion metrics

For two ciphertexts C and C′ of N bytes (N = 96), the bit-level avalanche effect [42], the number of pixels change rate (NPCR) and the unified average changing intensity (UACI) [21] are given by Eq. (23) [42] and Eqs. (24) and (25) [21]:

$$A = 100 \cdot \frac{\mathrm{HD}(C, C')}{8N} \quad (23)$$

$$\mathrm{NPCR} = \frac{100}{N}\sum_{u=1}^{N}[C_u \neq C'_u] \quad (24)$$

$$\mathrm{UACI} = \frac{100}{N}\sum_{u=1}^{N}\frac{|C_u - C'_u|}{255} \quad (25)$$

where HD(C, C′) is the Hamming distance in bits, C_u and C'_u are the u-th bytes read as integers 0–255, and [·] equals 1 when its condition holds and 0 otherwise. For independent uniform random bytes, the expected values are A = 50%, NPCR = 255/256 ≈ 99.61% and UACI ≈ 33.46%. These metrics originate in image-encryption analysis [21], and they are used here only as diffusion indicators. Because their value depends on what is perturbed, three protocols are defined (Table 4), and every reported value names its protocol:

For each record, the same perturbation (bit position, IV) is applied to both key-management variants, so the two variants are compared pairwise.

4. Equivalence testing

ECC and HMAC-SHA256 are compared by the paired Two One-Sided Tests (TOST) procedure [43]. For n records, let $\bar{d}$ and s_d be the mean and standard deviation of the paired differences $d_i = X_{ECC,i} - X_{HMAC,i}$ of a metric, and let δ be the equivalence margin. The null hypothesis of non-equivalence is rejected when both one-sided tests reject at α = 0.05, as in Eqs. (26) and (27) [43]:

$$t_L = \frac{\bar{d} + \delta}{\frac{s_d}{\sqrt{n}}} > t_{1-\alpha,n-1} \quad (26)$$

$$t_U = \frac{\bar{d} - \delta}{\frac{s_d}{\sqrt{n}}} < -t_{1-\alpha,n-1} \quad (27)$$

where $t_{1-\alpha, n-1}$ is the critical value of Student's t distribution with n − 1 degrees of freedom. This is equivalent to the

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

90% confidence interval of $\bar{d}$ lying inside (−δ, δ). The margins are ±2 percentage points for avalanche and UACI, ±1 percentage point for NPCR and ±0.05 bits/byte for per-record entropy. They correspond to about 0.7–1.6 times the per-record standard deviation expected for ideal random 96-byte ciphertexts (1.80 pp for A, 2.42 pp for UACI, 0.64 pp for NPCR, 0.072 bits/byte for entropy, from a Monte-Carlo simulation of 20,000 ciphertext pairs), so that differences smaller than the natural fluctuation of a perfect cypher are treated as negligible. We report $\bar{d}$, the 90% confidence interval, the p-value of the larger of the two one-sided tests (given as <0.001 when it is smaller, with the test statistics in Section III-E), and the paired effect size $d_z = \bar{d} / s_d$.

5. Randomness tests

The pooled derived keys (261,088 bytes, 2,088,704 bits) were also assessed with a subset of the NIST SP 800-22 test suite [44]: frequency (monobit), block frequency, runs, longest run of ones in a block, binary matrix rank, discrete Fourier transform, cumulative sums (forward and reverse), approximate entropy and serial (two statistics), eleven statistics in all, with the results reported in Section III-F. The remaining tests of the suite (template matching, Maurer's universal test, linear complexity and the random-excursion tests) were not run, because most of them are specified for sequences of 10^6 bits or more and give reliable pass proportions only over many such sequences, which 2.09 Mbit of keys cannot supply. The tests were implemented in Python and checked before use. The implementation reproduced the P-values of the worked examples in SP 800-22 for the frequency (0.109599), block frequency (0.706438), runs (0.500798), cumulative sums (forward 0.219194, reverse 0.114866), approximate entropy (0.261961) and serial (0.808792 and 0.670320) tests. On 20 sequences of 100,000 bits from a software pseudo-random generator, every statistic met the minimum pass proportion of 0.9233 (at least 19 of 20 sequences at α = 0.01), and the P-values were uniformly distributed. A deliberately biased stream with 50.5% ones was rejected by six of the eleven statistics (frequency, runs, longest run, both cumulative sums and the first serial statistic), which shows that the implementation detects non-randomness. These tests indicate statistical randomness and do not prove security.

Table 5. Simulation configuration used for all results reported in this study.

Parameter	Value
Records processed	16,318 (full dataset)
Cipher and mode	AES-128-CBC with PKCS#7; 88-byte payload, 96-byte ciphertext
Authentication	Encrypt-then-MAC, HMAC-SHA256 tag over IV ‖ C
Variant 2 (HMAC)	$K_{enc} = Trunc_{128}$(HMAC-SHA256(K, L ‖ φ ‖ seq)); K_{mac} by HKDF-SHA256
Variant 1 (ECC)	ECDH on P-256 (primary), SECP192R1 (legacy); keys by HKDF-SHA256
Session-amortized ECC	One handshake per 100-record session
LSTM	64 and 32 units, 128-dimensional output, 36,352 parameters, seed 42, untrained
Diffusion and TOST sample	All 16,318 records, paired across variants (protocols P1–P3, Table 4)
Random sources	OS secure random (ECC keys); NumPy RandomState(42) (IVs, LSTM weights, bit positions)
Software	Python 3; NumPy; SciPy; cryptography
Hardware	Timing run: cloud virtual machine, Intel Xeon 2.20 GHz, 2 vCPUs, Linux 6.6 (x86-64), Python 3.13
Timing	200 warm-up runs, then 3,000 timed runs (500 for the LSTM, 1,500 for ECDH); mean ± 95% CI

J. Simulation Environment

The framework was implemented in Python using NumPy (LSTM), the cryptography library (AES, ECDH and HKDF) and SciPy (statistics), and executed on all 16,318 records of the dataset [1]. In the evaluation, initialisation vectors and sampled perturbation positions were drawn from NumPy's RandomState with seed 42, so that all non-timing results are reproducible; ECC key pairs used the operating system's secure random source, and a deployment must draw every IV from a cryptographically secure generator. The LSTM weights were fixed by the same seed. ECC key exchange used the NIST P-256 curve as the primary configuration, with SECP192R1 retained as a legacy comparison, and HKDF used SHA-256 as the underlying hash. Key-establishment times were measured per record with a wall-clock timer after a warm-up period and repeated runs, and are reported as mean ± 95% confidence interval. Because timing was measured in an interpreted software environment on a single shared virtual machine, absolute values do not represent embedded IoMT hardware; only the relative comparison between mechanisms is meaningful, and validation on microcontroller targets is left to future work. All code, configuration files, random seeds and evaluation scripts are available from the corresponding author upon reasonable request (Code Availability). Table 5 summarises the configuration used throughout Section III.

III. Results

Results are reported for all 16,318 records of WUSTL-EHMS-2020 in software simulation. Unless stated otherwise, values are means; the evaluation protocols (P1–P3), equivalence margins and metric definitions are those of Section II-I.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

A. Key-Establishment Cost, Throughput, and Diffusion Metrics

The computational cost of key establishment was measured for both variants under four configurations: per-record elliptic-curve exchange on the NIST P-256 curve (the primary ECC configuration), the same exchange on the legacy SECP192R1 curve (retained for continuity with earlier work), and HMAC-SHA256 derivation without and with the LSTM context stage. The LSTM stage is reported separately because the context vector L(x, f) must be computed for every record before the key can be derived in Variant 2. A session-amortised baseline, in which one ECDH handshake serves a 100-record session and per-record keys are derived by HKDF, was also measured. Throughput, a standard performance criterion in lightweight-cryptography evaluation [6], is computed by Eq. (28) [6] as

Table 6. Comparison of per-record key-establishment time, end-to-end time and encryption throughput for the ECC P-256, legacy ECC P-192, HMAC-SHA256 and HMAC-SHA256 + LSTM configurations, reported as mean values (key-establishment times with 95% confidence intervals) under the software simulation environment of Table 5.

Metric	ECC P-256	ECC P-192 (legacy)	HMAC (no LSTM)	HMAC + LSTM
Key establishment, per record (µs)	152.1 ± 1.4	3189.3 ± 108.7	5.5 ± 0.1	10,562.0 ± 829.9
Session-amortised (µs)	8.4	38.8	5.5	10,562.0
End-to-end per record (µs)	162.8	3199.9	16.2	10,572.6
Throughput (KB/s)	540.6	27.5	5446.5	8.3

$$\Theta = \frac{S}{t_{\text{key}} + t_{\text{enc}} + t_{\text{tag}} [+ t_{\text{ctx}}]} \quad (28)$$

where Θ is throughput, S = 88 bytes is the payload size, tkey is the key-derivation or key-exchange time, t_{enc} and t_{tag} are the AES-128-CBC encryption and HMAC-SHA256 tag times, and t_{ctx} is the LSTM context-vector time, included only in the "HMAC + LSTM" configuration, for which the key-establishment entry in Table 6 is the sum of the key-derivation and context-vector times. Throughput is expressed in KB/s with 1 KB = 10^3 bytes. All timings are mean ± 95% confidence interval from repeated wall-clock measurements after a warm-up period (Section II-J) and are summarised in Table 6 and Fig. 3. Without the LSTM stage, HMAC-SHA256 derivation took 5.5 µs per record, which is 27.6 times faster than per-record P-256 exchange (152.1 µs) and 1.5 times faster than the session-amortised baseline (8.4 µs). Throughput for the 88-byte payload was 5446.5 KB/s for HMAC derivation against 540.6 KB/s for P-256 exchange, about 10.1 times higher. Session amortisation reduces the per-record cost of P-256 exchange by a factor of 18, so the advantage of keyed derivation over a realistic ECC deployment is 1.5 times, not the much larger factor obtained against a fresh handshake for every record. When the LSTM stage is included, the picture reverses. Variant 2 required 10,562.0 µs per record, 69 times slower than per-record P-256 exchange, and throughput fell to 8.3 KB/s. The LSTM context vector alone (10,556.5 µs)

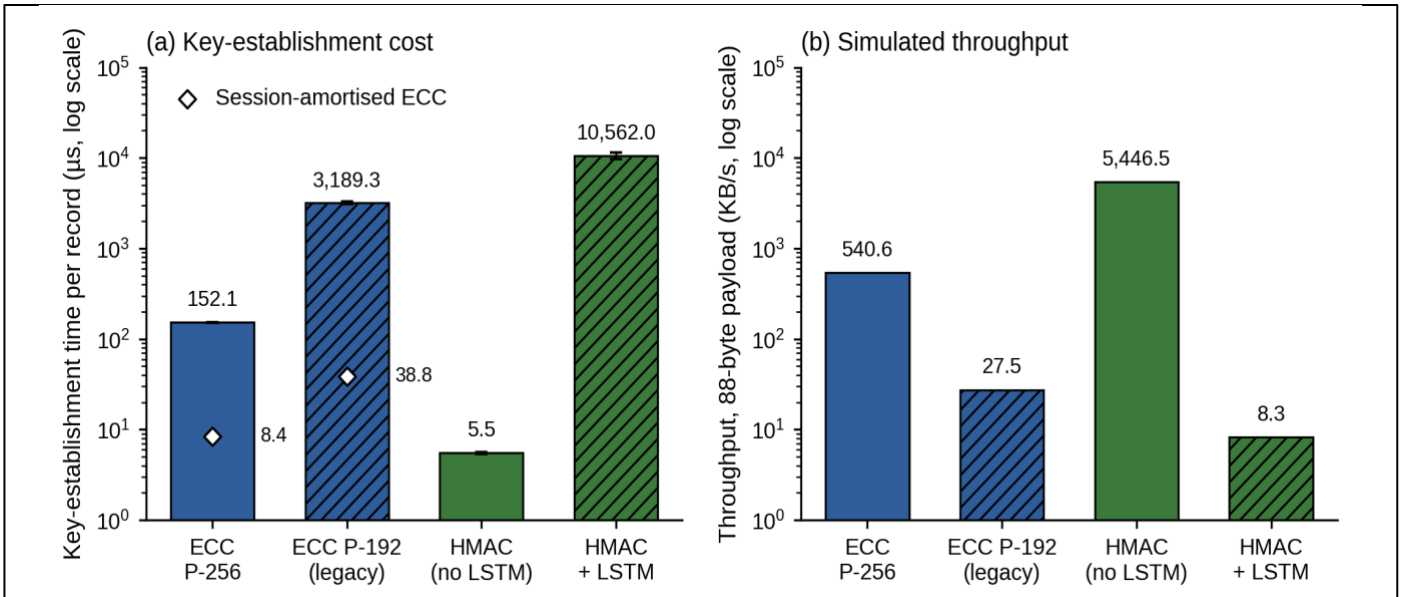

Fig. 3. Per-record cost and throughput of key establishment for an 88-byte payload (log scale). (a) Key-establishment time, mean ± 95% CI; diamonds mark session-amortised ECC. (b) Simulated throughput. HMAC + LSTM includes the context-vector computation.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

cost about 1,900 times the HMAC step. Keyed derivation is therefore cheaper than elliptic-curve exchange only when the LSTM stage is omitted, and Section III-D shows that this stage adds no measurable cryptographic property. The legacy SECP192R1 curve took 3189.3 µs per record in this environment. This cost depends strongly on how the underlying cryptographic library implements the curve, whereas the optimised P-256 implementation is stable; ratios against the legacy curve are therefore reported for completeness only and are not used for the principal conclusions. Whether these relative costs hold on constrained IoMT microcontrollers requires the hardware validation listed in Section IV-F. The ECC figure is the cost of a complete two-party exchange measured in one process (both key pairs, the shared-secret computation and HKDF), whereas the HMAC figure is a single derivation. The cost borne by a constrained device alone is therefore lower than 152.1 µs, and the 27.6-fold ratio overstates the device-side saving. A device-side measurement on target hardware is part of the validation in Section IV-F. Diffusion metrics for the encrypted records are reported under the three perturbation protocols of Table 4 (Table 7 and Fig. 4). In protocol P1, a single plaintext bit is flipped with the key and IV unchanged; in P2, the same flip is re-encrypted under a fresh IV; and in P3, a single bit of the encryption key is flipped.

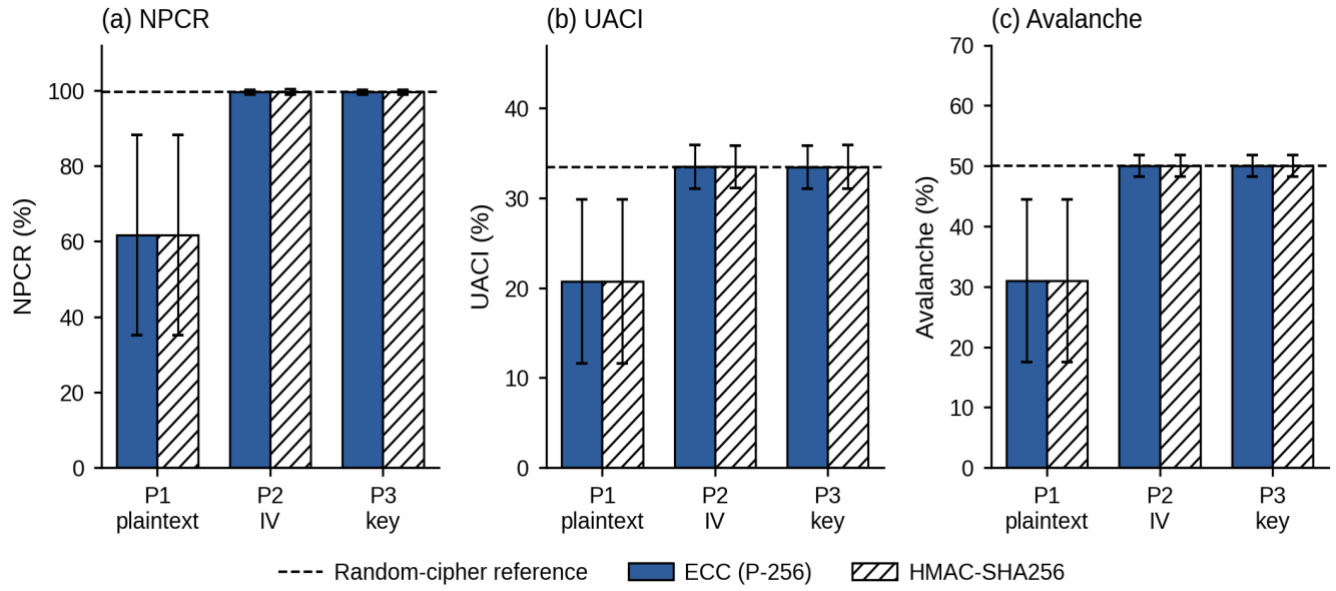

Fig. 4. Diffusion metrics under the three perturbation protocols of Table 4 (mean ± SD, 16,318 records). (a) NPCR. (b) UACI. (c) Avalanche. Dashed lines show the expected values for random ciphertext; ECC and HMAC-SHA256 are equivalent under every protocol.

Under protocols P2 and P3 both variants match the random reference on every metric: for ECC and HMAC-SHA256 respectively, NPCR was 99.60% and 99.61% (P2) and 99.61% and 99.61% (P3), against 99.61% expected, and avalanche was 50.02% and 50.00% (P2) and 49.99% and 50.01% (P3), against 50%. Under protocol P1, the values are lower (NPCR 61.70% and 61.69%, UACI 20.73% and 20.74%, avalanche 30.97% and 30.98%) and vary widely between records (NPCR SD of 26.6 pp). This is a property of CBC mode, not a weakness of either key-management variant: a flipped plaintext bit changes its own ciphertext block and every

Table 7. Diffusion metrics under the three perturbation protocols for both key-management variants (mean ± SD).

Protocol	Metric	ECC	HMAC-SHA256	Random reference
P1 plaintext	NPCR (%)	61.70 ± 26.60	61.69 ± 26.60	99.61
	UACI (%)	20.73 ± 9.12	20.74 ± 9.12	33.46
	Avalanche (%)	30.97 ± 13.42	30.98 ± 13.44	50.00
P2 IV	NPCR (%)	99.60 ± 0.64	99.61 ± 0.64	99.61
	UACI (%)	33.47 ± 2.41	33.47 ± 2.38	33.46
	Avalanche (%)	50.02 ± 1.82	50.00 ± 1.80	50.00
P3 key	NPCR (%)	99.61 ± 0.63	99.61 ± 0.63	99.61
	UACI (%)	33.42 ± 2.41	33.44 ± 2.42	33.46
	Avalanche (%)	49.99 ± 1.80	50.01 ± 1.79	50.00

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

later block, but not earlier ones, so NPCR depends on the position of the flipped bit. For a flip in blocks 1 to 6, the mean NPCR was 99.6, 83.0, 66.4, 49.8, 33.2 and 16.6%, respectively, matching the values expected from the chaining structure. Because the underlying block cypher is identical in both variants, the near-identical values for ECC and HMAC-SHA256 under every protocol are an expected consequence of the shared cypher stage and not an independent finding about either mechanism; their statistical equivalence is tested formally in Section III-E.

B. Entropy by Traffic Category

Entropy was evaluated separately for the three traffic categories to test whether key randomness depends on the type of record. Pooled over all 261,088 key bytes, entropy was 7.9993 bits/byte for both ECC and HMAC-SHA256, equal to the 7.9993 expected for ideal random keys at this sample size (Section II-I), and well above the 7.5 bits/byte threshold of NIST SP 800-90B [45]. This pooled value is a descriptive statistic of the key-byte distribution and is not a formal NIST SP 800-90B entropy-source validation, which additionally requires min-entropy estimation under the restart and conditioning tests of that standard. Per-record ciphertext entropy over the 96-byte ciphertext was 6.2413 ± 0.0718 bits/byte for HMAC-SHA256 and 6.2407 ± 0.0717 for ECC, against 6.24 expected for 96 random bytes.

By category, pooled key entropy for normal records (n = 14,272) was 7.9992 (HMAC-SHA256) and 7.9993 (ECC); for spoofing records (n = 1,124), 7.9892 for both variants; and for data-alteration records (n = 922), 7.9862 (HMAC-SHA256) and 7.9886 (ECC). The two attack categories sit lower than the normal category, but this follows from sample size. The plug-in entropy estimator is biased downward in finite samples [41], and the expected value for ideal random keys of the same size is 7.9992 (n = 14,272 keys), 7.9897 (n = 1,124) and 7.9875 (n = 922), with standard deviations of 0.0001, 0.0009 and 0.0011 bits/byte, obtained by Monte-Carlo simulation. Every observed value lies within 1.2 standard deviations of its expected value (for example, 7.9862 against 7.9875 ± 0.0011 for HMAC-SHA256 in data-alteration records), so there is no evidence that key entropy depends on traffic category, and the difference between ECC and HMAC-SHA256 within a category is within sampling variation. These observations hold because each key is derived independently of the traffic label. The label was excluded from the descriptor (Section II-E), and the keys depend only on the pre-shared secret and the per-record inputs.

C. Integrity Verification (Encrypt-then-MAC)

To evaluate whether the Encrypt-then-MAC construction (Section II-G) protects against the data-alteration attacks of the dataset, a controlled experiment was run on all 16,318 Variant 2 records. Each record was subjected, in turn, to nine conditions simulating an attacker with network-path access but no knowledge of K_{mac}: no modification (to measure false rejections), a single flipped ciphertext bit, eight flipped ciphertext bits, a flipped IV bit, a flipped tag bit, truncation of the last ciphertext block, substitution of another record's ciphertext, reordering of two ciphertext blocks, and a wholly forged record of random bytes. Two replay conditions were also tested: a duplicate of a valid record, and a valid record presented at a different sequence position. Rejection rates are reported with exact (Clopper–Pearson) 95% confidence intervals (Table 8).

Table 8. Integrity verification results for nine record-modification conditions and two replay conditions (n = 16,318 per condition).

Condition	Rejected	Rate (95% CI)
None (false-rejection check)	0	0.00% [0.00, 0.02]
Ciphertext, 1 bit	16,318	100% [99.98, 100]
Ciphertext, 8 bits	16,318	100% [99.98, 100]
IV, 1 bit	16,318	100% [99.98, 100]
Tag, 1 bit	16,318	100% [99.98, 100]
Truncation (last block)	16,318	100% [99.98, 100]
Substitution (other record's ciphertext)	16,318	100% [99.98, 100]
Block reordering	16,318	100% [99.98, 100]
Forged record (random)	16,318	100% [99.98, 100]
Replay: duplicate of a valid record	0	Accepted in 16,318 of 16,318
Replay: valid record at another position	0	Tag accepted in 16,318 of 16,318

All modified, substituted, reordered, truncated and forged records were rejected before decryption, and no unmodified record was rejected (0 of 16,318; 95% CI 0.00–0.02%). The 95% lower bound on the rejection rate is 99.98% for every attack class. This is consistent with the 2^{-256} forgery bound of HMAC-SHA256, and it shows cryptographic integrity verification for modification, but the framework does not classify or locate attacks. Replay is not prevented by the tag. A duplicate of a valid record carries a valid tag and was accepted in every case. A valid record presented at a different sequence position also passes tag verification; decrypted under that position's key, it yielded valid padding in only 60 of 16,318 cases (0.37%), which is incidental and not a defence. Preventing replay requires a receiver-side check of monotonic sequence numbers, which this study does not implement (Section II-H). No empirical experiment was run for passive interception; confidentiality against it rests on AES-128-CBC with a secret key and a fresh IV, and remains an analytical, not experimental, claim.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

D. Ablation and Key Independence

To isolate the contribution of each stage, four variants of Variant 2 were evaluated on all 16,318 records: (A) HMAC-SHA256 over the raw descriptor and sequence number, HMAC(K, φ ‖ seq); (B) with KaTaPaYadi encoding added, HMAC(K, M(φ) ‖ φ ‖ seq); (C) with the LSTM context vector but bypassing KaTaPaYadi, in which the LSTM receives zero digits; and (D) the full pipeline of Eq. (13). For each variant, pooled key entropy, the percentage of key bits that change when one digit of φ is altered, and the ciphertext avalanche under the same descriptor perturbation are reported (Table 9).

The four variants were indistinguishable: key entropy differed by 0.0001 bits/byte (all equal to the 7.9993 expected for ideal random keys), the key-change percentage by 0.04 percentage points, and the ciphertext avalanche by 0.039 percentage points. Cryptographic diffusion therefore derives entirely from HMAC-SHA256, consistent with Section II-I, and neither stage adds a measurable property. This finding comes at a cost: the LSTM stage takes about 10,556 µs per record, roughly 1,900 times the HMAC step (Section III-A). The stage is therefore reported as a negative result.

Table 9. Ablation of the KaTaPaYadi and LSTM stages on key entropy, key change and ciphertext avalanche (n = 16,318).

Variant	Key entropy (bits/byte)	Key change under φ perturbation (%)	Ciphertext avalanche (%, mean ± SD)
A: HMAC-SHA256 alone	7.9993	49.98	50.01 ± 1.81
B: KaTaPaYadi + HMAC-SHA256	7.9994	50.02	50.00 ± 1.82
C: LSTM, no KaTaPaYadi	7.9993	50.00	50.01 ± 1.81
D: Full pipeline	7.9993	50.01	49.97 ± 1.81

Keys were independent in both variants. All 16,318 keys were distinct; the Hamming distance between the keys of consecutive records was 49.98 ± 4.44% (HMAC-SHA256) and 50.00 ± 4.40% (ECC), against 50% for independent random keys; and, for HMAC-SHA256, changing one digit of φ changed 50.01 ± 4.48% of the 128 key bits, and changing the sequence number by one changed 49.97%.

The behaviour of the encoder on this dataset explains why it has no effect. The source and destination addresses are constant, so descriptors differ only through the source port (16,314 distinct descriptors among 16,318 records). Only three records carry a non-numeric port (dircproxy, fido, tfido), so the consonant mapping acts on 0.02% of records, and the encoder otherwise reduces to digit extraction. The 16,314 distinct descriptors produced 16,313 distinct sequences, a single collision, which does not affect keys because φ itself is an input to Eq. (13). M(φ) was unchanged under case and separator variants for 100% of descriptors, identical to a one-line normaliser that lowercases and strips separators.

E. Statistical Equivalence Testing

To test formally whether the two key-management variants are equivalent in cryptographic diffusion, the paired Two One-Sided Tests (TOST) procedure of Section II-I (Eqs. (26)–(27)) was applied to per-record measurements over all 16,318 records, with margins of ±1 percentage point for NPCR, ±2 for UACI and avalanche effect, and ±0.05 bits/byte for entropy. For each metric and protocol, Table 10 reports the mean paired difference

Table 10. Paired Two One-Sided Tests (TOST) evaluating statistical equivalence between ECC and HMAC-SHA256 cryptographic diffusion and ciphertext-entropy metrics across perturbation protocols, including mean differences, 90% confidence intervals, equivalence margins, p-values and standardized effect sizes (n = 16,318 paired records).

Metric (protocol)	Margin (δ)	$\bar{d}$	90% CI	p	d_z	Equivalent?
NPCR (P1)	±1 pp	+0.012	[+0.003, +0.021]	<0.001	+0.017	Yes
UACI (P1)	±2 pp	−0.008	[−0.043, +0.027]	<0.001	−0.003	Yes
Avalanche (P1)	±2 pp	−0.010	[−0.036, +0.016]	<0.001	−0.005	Yes
NPCR (P2)	±1 pp	−0.002	[−0.014, +0.010]	<0.001	−0.002	Yes
UACI (P2)	±2 pp	−0.003	[−0.047, +0.041]	<0.001	−0.001	Yes
Avalanche (P2)	±2 pp	+0.022	[−0.011, +0.055]	<0.001	+0.009	Yes
NPCR (P3)	±1 pp	−0.007	[−0.019, +0.004]	<0.001	−0.008	Yes
UACI (P3)	±2 pp	−0.016	[−0.060, +0.028]	<0.001	−0.005	Yes
Avalanche (P3)	±2 pp	−0.021	[−0.053, +0.011]	<0.001	−0.008	Yes
Ciphertext entropy	±0.05 bits/byte	−0.0007	[−0.0020, +0.0006]	<0.001	−0.007	Yes

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

(ECC − HMAC-SHA256), its 90% confidence interval, the p-value of the TOST, and the paired effect size.

The two mechanisms were equivalent on every metric under every protocol: the largest mean difference was 0.022 percentage points (avalanche, P2), against margins of 1 to 2 points, every 90% confidence interval lay well inside its margin, and all effect sizes were below 0.02. The smaller of the two one-sided test statistics exceeded 62 in absolute value in every row (computed from the tabulated means and intervals), so every p-value is far below 0.001, and the entries in Table 10 are conservative bounds. The 90% intervals do not depend on the margin, so they also show that equivalence would be concluded at much tighter margins: every interval lies within ±0.021 percentage points for NPCR, ±0.060 for UACI, ±0.055 for avalanche and ±0.0020 bits/byte for entropy, so margins of ±0.05, ±0.1, ±0.1 and ±0.0025, respectively (twenty times tighter), give the same result. For per-record ciphertext entropy, the means were 6.2407 (ECC) and 6.2413 (HMAC-SHA256) bits/byte. Entropy over a 96-byte ciphertext is bounded above by $\log_2 96 \approx 6.585$ bits/byte, and the expected value for 96 random bytes is about 6.24 because of the finite-sample bias of the estimator [41]; the observed means match this expectation, so they do not indicate weaker keys. Equivalence was expected, since both variants feed the same AES-128-CBC stage; Table 10 shows that the source of the key does not affect diffusion, not that either mechanism improves it. Statistical equivalence of these diffusion indicators is not equivalence of security: the two mechanisms differ in forward secrecy, provisioning assumptions and the consequences of compromise (Section II-H), none of which these metrics measure.

F. Randomness and Key-Collision Tests

The derived keys of both variants were tested with the eleven statistics of the SP 800-22 subset described in Section II-I. The HMAC-derived stream is the 2,088,704-bit stream of the evaluation keys. For the ECC variant, the stream came from 16,318 ECDH P-256 exchanges with fixed-seed private scalars, so that the test is reproducible; the exchanges of Section III-A used operating-system randomness. Each stream was cut into 20 sequences of 100,000 bits and every statistic was applied to every sequence (Table 11). Every statistic met both acceptance criteria for both variants: at least 19 of 20 sequences passed at $\alpha = 0.01$ (the acceptance region for 20 sequences is a pass proportion of at least 0.9233), and the 20 P-values were uniformly distributed (uniformity $P \geq 0.0001$). The lowest pass count was 19 of 20, observed for one statistic of the HMAC-derived keys (longest run) and four statistics of the ECDH-derived keys (runs, longest run, discrete Fourier transform, and the first serial statistic), and the lowest uniformity P-value was 0.0010 (discrete Fourier transform, ECDH-derived keys). With 20 sequences, at least one failure occurs by chance for a given statistic with probability 0.18, so a few 19-of-20 outcomes among eleven statistics are expected; the same implementation rejected a stream with 50.5% ones (Section II-I). Key collisions were checked at a larger scale. Two million HMAC-derived keys, formed by varying the sequence number over a fixed context vector and descriptor, contained no duplicates (the birthday bound for 128-bit keys at this number of keys is about 5.9×10^{-27}), and 20,000 ECDH P-256 exchanges gave 20,000 distinct keys; together with the 16,318 distinct keys of the evaluation (Section III-D), no collision occurred. These counts are far below the number of keys at which collisions among 128-bit keys become likely, so they show the absence of structural defects and do not bound the collision probability of a deployed fleet. The tests indicate statistical randomness of the keys and do not prove security, which rests on the PRF assumption for HMAC-SHA256 (Section II-H).

IV. Discussion

A. Interpretation of Principal Findings

The central result is a cost asymmetry that depends on whether the LSTM stage is included. Without it, HMAC-SHA256 derivation established a key in 5.5 µs per record, against 152.1 µs for a fresh per-record P-256 exchange (27.6 times faster) and 8.4 µs for a session-amortised P-256 baseline (1.5 times faster). Throughput for the 88-byte payload followed the same order: 5,446.5 KB/s for HMAC derivation against 540.6 KB/s for P-256, about 10 times higher (Section III-A). With the LSTM context vector included, the order reverses: Variant 2 required 10,562.0 µs per record, 69 times slower than a per-record P-256 exchange, because the LSTM alone takes about 1,900 times the HMAC step. Keyed derivation is therefore the cheaper mechanism only when the context stage is omitted, and the ablation (Section III-D) shows that this omission costs nothing measurable: entropy, avalanche effect and key independence were unchanged across the four variants. The configuration that this study supports is HMAC-SHA256 derivation over the session descriptor and sequence number, without the KaTaPaYadi–LSTM stage.

Key quality was not a differentiator. Entropy, avalanche effect, UACI and NPCR were statistically equivalent between ECC and HMAC-SHA256 under all three perturbation protocols (Section III-E), with mean differences of at most 0.022 percentage points. Because both variants feed the same AES-128-CBC stage, this is the expected outcome, and it shows that the choice of mechanism affects computational cost and provisioning requirements, not diffusion. The Encrypt-then-MAC construction rejected every modified, substituted, reordered, truncated and forged record (16,318 of 16,318 in each class) with no false rejection, so per-record keying does not come at the expense of integrity verification; replay, however, is not prevented by the tag alone (Section III-C). The cost comparison is not a comparison of equal security properties. HMAC-SHA256 derivation requires a pre-shared secret provisioned in advance and, if that secret is compromised, exposes every record of the affected device–gateway pair with no forward secrecy (Section II-H). Ephemeral ECC exchange provides forward secrecy and needs no pre-shared secret, but is secure against the active attacker of this dataset only if the public points are authenticated, a cost that is not included in the ECC timings. The ECC figures are therefore a lower bound for a deployable system, which

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

Table 11. NIST SP 800-22 results for the derived keys of both variants: sequences passed out of 20 at α = 0.01 and the uniformity P-value of the 20 P-values.

Statistic	HMAC: passed	HMAC: uniformity P	ECDH: passed	ECDH: uniformity P
Frequency (monobit)	20/20	0.6371	20/20	0.5341
Block frequency (M = 128)	20/20	0.9114	20/20	0.8343
Runs	20/20	0.1223	19/20	0.2133
Longest run of ones (M = 128)	19/20	0.6371	19/20	0.4373
Binary matrix rank (32 × 32)	20/20	0.7399	20/20	0.9643
Discrete Fourier transform	20/20	0.0127	19/20	0.0010
Cumulative sums, forward	20/20	0.6371	20/20	0.2133
Cumulative sums, reverse	20/20	0.0909	20/20	0.8343
Approximate entropy (m = 8)	20/20	0.2757	20/20	0.8343
Serial (m = 8), first	20/20	0.4373	19/20	0.2757
Serial (m = 8), second	20/20	0.8343	20/20	0.2757

strengthens the case for HMAC derivation where a pre-shared secret is acceptable, and does not remove the need for ECC where it is not. These findings are consistent with the broader lightweight-cryptography literature [6]–[9] and extend the detection-focused work on WUSTL-EHMS-2020 [1], [13]–[15] by addressing how data should be protected in transit rather than how attacks are recognised. The framework does not detect attacks: Section III-C measures cryptographic integrity verification, and no classification of attack traffic was attempted.

1. Tested and untested statements

The following were tested in this study: the per-record cost ratios, the equivalence of diffusion metrics, the rejection of nine record-modification classes, the null effect of the KaTaPaYadi and LSTM stages, and the inertness of the encoder on this dataset. The following were not tested and rest on analysis or on cited standards: forward secrecy and compromise consequences (analytical, Section II-H), the PRF security of HMAC-SHA256 [10], behaviour on constrained hardware, replay resistance, and the cost of authenticating ECC public keys.

B. Quantitative Comparison with Related Studies

Table 12 places this work beside five related studies and quotes each study's own reported figures where they could be retrieved. The comparison is order-of-magnitude and not a controlled benchmark, because the studies differ in six respects that matter for the figures: the data and payload (network records, benchmark inputs or images), the hardware platform, the cryptographic curve or primitive, the computational environment, the scope of the timed operation, and the evaluation metrics. These differences are stated in each row and discussed below. Figures for [4], [5], [7] and [20] are the studies' own reported results. For [19] only the abstract was retrievable and it quotes no values, so none is attributed. Where a source reports no value for a metric, the cell is marked "–".

1. Key-establishment time

The retrieved figures differ widely: 1.37–41.76 ms for key-pair generation across six software libraries on a Raspberry Pi 4B [4], a spread of about 30 times, and 991 cycles for a hardware ECC core [5], roughly 5–10 µs at an assumed 100–200 MHz clock (the frequency is not stated, so this is illustrative). Against these, this work's fresh P-256 exchange (152.1 µs) is 9.0 times below the fastest library in [4] and 275 times below the slowest, and 15–30 times above the illustrative range for [5]. Its session-amortised cost (8.4 µs) is about 160 times below the fastest library in [4], and its HMAC-SHA256 derivation (5.5 µs) is about 250 times below it and within the illustrative range for [5]. When the LSTM context vector is included, Variant 2 (10,562.0 µs) is 7.7 times above the fastest library in [4] and 4.0 times below the slowest. These ratios must be read with the scope of each timed operation in mind: this work's ECC figure covers both key pairs, the shared-secret computation and HKDF, while the figures of [4] cover key-pair generation alone, so the two are not like-for-like. The 3.001 ms of [20] measures image processing and is not a key-establishment cost.

2. Entropy and diffusion

Pooled key entropy in this work (7.9993 bits/byte for both variants) lies within the range of [7] (7.9967–7.9999) and exceeds the value of [20] (7.997633) by 0.0017 bits/byte. Under the key and IV perturbation protocols (P2, P3), NPCR was 99.60–99.61% and UACI 33.42–33.47%, within 0.04 percentage points of the 33.46% expected for random data [21], while [7] reports UACI of 30.31–44.76% and NPCR of 99.60–99.84%. Under the plaintext-sensitivity protocol (P1), NPCR was 61.70% and UACI 20.73%, lower than the image-cypher figures because CBC propagates a plaintext change only forward (Section III-A); image-cypher NPCR and UACI thresholds therefore do not transfer to block-cypher modes. [20] reports neither metric.

3. What can and cannot be concluded

No quoted figure is a matched, same-device timing of HMAC-SHA256 derivation against ECC exchange, so the ratios of this work (27.6 times against fresh P-256 exchange and 1.5 times against the session-amortised baseline) are single-platform measurements. No software ECC step in Table 12 is faster than 1.37 ms on embedded-class hardware, and the ordering hash

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

derivation, then amortised ECC, then fresh ECC, is not contradicted by any compared study. Absolute times, however, depend on platform, curve, library and operation scope, and cannot be transferred between studies.

C. Why the Results Differ Across Studies

Table 12 shows three differences: the size of the ECC cost, the gap in pooled entropy between this work and the image-encryption studies, and the spread of NPCR and UACI. For each, this subsection separates what this study's own measurements establish from interpretations that were not tested.

1. Established by this study's measurements

ECC cost depends on the curve implementation. On the same machine, library and timing procedure, a per-record exchange took 152.1 µs on P-256 and 3,189.3 µs on SECP192R1, a 21-fold difference caused by the curve and its implementation alone. The same legacy curve took 1,209.4 µs in an earlier run of this study, a 2.6-fold difference between two environments. Cost therefore depends strongly on implementation, which is consistent with [4], where six libraries on one Raspberry Pi 4B differ about 30-fold (41.76 ms against 1.37 ms), and it is why the primary comparison here uses the optimised P-256 curve. The ratio between mechanisms depends on how often the ECC step is paid. HMAC-SHA256 derivation was 27.6 times faster than a fresh per-record P-256 exchange but only 1.5 times faster than a session-amortised exchange in which one handshake serves 100 records (Section III-A). The reported advantage of keyed derivation therefore depends on the ECC baseline chosen, and both baselines are reported for that reason. Entropy shortfalls follow from sample size. The plug-in Shannon estimator is biased downward in finite samples [41]; for J = 256 byte values and n sampled bytes, the expected shortfall below the maximum is approximately, by Eq. (29) [41],

$$\Delta H \approx \frac{J-1}{2n \ln 2} \quad (29)$$

where ΔH is the expected downward bias in bits/byte, J = 256 is the number of possible byte values, n is the number of bytes in the sample, and ln is the natural logarithm. For this work's n = 261,088 key bytes, ΔH ≈ 0.0007 bits/byte, matching the observed 7.9993 bits/byte for both variants, and the category-level values lie within 1.2 standard deviations of the values expected for random keys of the same size (Section III-B).

NPCR and UACI depend on what is perturbed and on the cypher mode. Under the key and IV protocols (P2, P3) NPCR was 99.60–99.61% and UACI 33.42–33.47%, matching the random reference of 99.61% and 33.46% [21]; under the plaintext-sensitivity protocol (P1) NPCR was 61.70% and UACI 20.73%, because CBC chaining propagates a plaintext change only to later blocks (Section III-A). A single reported NPCR or UACI value is therefore not interpretable without the protocol that produced it.

2. Interpretations not tested here

The following explanations are consistent with the evidence but were not examined experimentally. The fastest key-pair generation in [4] (1.37 ms) is a single operation on a Raspberry Pi 4B, while this work's 152.1 µs covers two key pairs, the shared-secret computation and HKDF on a cloud virtual machine, so curve, library, runtime, host and operation scope are confounded and cannot be separated using the published figures. The 991-cycle hardware core of [5] would take about 5–10 µs at an assumed 100–200 MHz clock, which is plausibly explained by dedicated hardware, but its frequency is not stated, and this was not measured.

ECDH is dominated by a scalar multiplication whose cost grows with the group size, whereas HMAC-SHA256 costs a small, fixed number of hash-compression calls. This is a general property of the primitives that this study's

Table 12. Comparative summary of related key-management and encryption approaches and of this work.

Study	Data and payload	Platform and primitive	Key-establishment time	Entropy (bits/byte)	NPCR / UACI
Cekis et al. [4]	Benchmark; no dataset	Raspberry Pi 4B, six software ECC libraries	Key-pair generation 1.37 ms (RELIC) to 41.76 ms (libecc)	–	–
Dhanda et al. [5]	Benchmark; no dataset	FPGA/ASIC core, 163-bit ECC	991 clock cycles (minimum latency); µs not derivable	–	–
Kinani et al. [7]	Medical images	Software; ECC + chaos cipher	Not reported	7.9967–7.9999	NPCR 99.60–99.84%; UACI 30.31–44.76%
Hasan et al. [19]	Medical image	Lightweight symmetric cipher, no key exchange	Not applicable	Not retrievable	Not retrievable
Senthilkumar et al. [20]	Medical image	Software; ECDH + KDF hybrid	Image encryption 1.998 ms (decryption 1.001 ms)	7.997633	Not reported
This work	WUSTL-EHMS-2020, 88-byte records (n = 16,318)	Cloud VM, Intel Xeon 2.2 GHz, Python (software); P-256 ECDH + HKDF; HMAC-SHA256	ECC 152.1 µs (fresh), 8.4 µs (session-amortised); HMAC 5.5 µs (10,562.0 µs with LSTM)	7.9993 (ECC and HMAC; pooled, n = 261,088 bytes)	P2, P3: NPCR 99.61%, UACI 33.42–33.47%; P1: 61.70% / 20.73%

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

single-platform measurements are consistent with but do not isolate. The 0.0017 bits/byte by which this work's 7.9993 exceeds the 7.997633 of [20] is of the order of the estimator bias for an image of a few tens of thousands of pixels: $\Delta H \approx 0.0028$ bits/byte for a 256 × 256 image ($n = 65{,}536$), against the observed shortfall of 0.0024. The image size in [20] is not stated, so this is an order-of-magnitude illustration and not an explanation that has been verified. The wide UACI spread of 30.31–44.76% reported in [7] is plausibly due to structured image pixels and a bespoke ECC-plus-chaos cypher, in which any departure from 33.46% marks a departure from ideal diffusion. This is not tested here, and the near-ideal values in this work follow from the shared block cypher and are not evidence that the KaTaPaYadi or LSTM stages improve diffusion (Section III-D). The ordering of cost, hash derivation below session-amortised ECC below fresh ECC, is consistent with the retrievable sources, although none of them timed hash derivation or session-amortised exchange, so it rests on this study's single-platform measurements, whereas absolute times vary by more than two orders of magnitude with implementation, platform and operation scope. A same-board benchmark with a common curve and library (Section IV-F) is needed to separate these factors.

D. Practical Deployment Guidance

The guidance below follows from the measurements and analysis of this study and holds only under the conditions stated; none of it has been validated on constrained hardware.

1. Choosing a mechanism

If a device and its gateway can be provisioned in advance with a unique secret K, HMAC-SHA256 derivation of per-record keys is the lower-cost choice. It was 27.6 times faster than a fresh per-record P-256 exchange and 1.5 times faster than a session-amortised exchange, with no measurable loss of diffusion (Sections III-A and III-E). It should be deployed over the session descriptor and sequence number without the KaTaPaYadi–LSTM stage, which added about 1,900 times the HMAC cost and no measured benefit (Section III-D). If a device is paired with a gateway for the first time, no secret has been provisioned, or forward secrecy is required, ECC exchange remains necessary despite its higher cost: ephemeral ECDH on P-256, with the public points authenticated, for example, by provisioned certificates, because unauthenticated ECDH does not resist the active attacker in this dataset (Section II-H). The legacy 192-bit curve should not be used. A mixed deployment, in which ECC establishes the initial pairing and HMAC derivation protects the ongoing session, follows naturally but was not evaluated.

2. Conditions the HMAC result depends on

The cost advantage holds only if the following are in place. (i) K is a uniformly random secret of at least 128 bits, unique to each device–gateway pair, and held in protected storage. (ii) A rotation and revocation procedure exists, because compromise of K exposes every past and future record of that pair and Variant 2 offers no forward secrecy (Section II-H). (iii) The receiver enforces monotonically increasing sequence numbers, because the tag does not prevent replay (Section III-C). (iv) The tag is verified in constant time and before decryption. (v) A fresh random IV is drawn for every record.

3. Trade-offs in amortisation

Amortising an ECC handshake over a session reduces the per-record cost (8.4 µs for 100-record sessions, against 152.1 µs for a handshake per record), but exposure of the session secret then reveals every record derived from it, so shorter sessions trade cost for a smaller exposure window. The cost of authenticating the public points is excluded from all ECC figures, so the ECC costs here are lower bounds.

4. Limits of these recommendations

All timings come from a software simulation on a cloud virtual machine. On microcontrollers with hardware hash or elliptic-curve accelerators, the ratios between mechanisms may differ considerably, and the decision should be re-evaluated on the target hardware before deployment (Section IV-F).

E. Limitations

Every result is a software-simulation figure measured in interpreted Python on a cloud virtual machine (Intel Xeon, 2.2 GHz, 2 vCPUs), so no deployment or clinical-use claim should be drawn from them. Timing distributions were skewed on this shared host (for the LSTM stage, the standard deviation, 9,445.7 µs, approached the mean, 10,556.5 µs), and ECC times depend strongly on the curve implementation (Section IV-C). The ECC timing covers both parties' key pairs, the shared-secret computation and HKDF in one process, and excludes network latency and public-key authentication, so it is a lower bound for a deployed exchange. On microcontrollers with hardware hash or elliptic-curve accelerators, the ratios between mechanisms may differ considerably.

1. Resource estimate of the LSTM stage

An analytical estimate follows from the architecture in Table 2. With the Keras parameter convention (LSTM: $4 \times (\text{units} \times \text{input_dim} + \text{units}^2 + \text{units})$; dense: $\text{input_dim} \times \text{units} + \text{units}$), layer 1 (64 units, input dimension 12: 11 features and one digit) has 19,712 parameters, layer 2 (32 units, input dimension 64) has 12,416, and the dense output layer (128 units, input dimension 32) has 4,224, a total of 36,352 parameters. Stored as 32-bit floats, this is about 142 KiB of weights (about 71 KiB at 16-bit precision); the weights alone fit in the SRAM of representative IoMT-class microcontrollers such as the ESP32 (520 KB) or the nRF52840 (256 KB). Inference requires about 2.04 million multiply–accumulate operations per record (64-time steps through both layers plus the dense layer), which is the analytical counterpart of its measured cost. Measured with Python's tracemalloc in the NumPy implementation, one single-record inference allocated a further 231.3 KiB of activations and temporaries at peak (output vector 512 bytes), so weights and working memory together take about 373 KiB. This fits the ESP32 but exceeds the 256 KB of the nRF52840, and a microcontroller implementation would need reduced-precision weights and in-place activation buffers. These are software-simulation figures: inference-

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

framework overhead, input and output buffers and cryptographic working state were not measured.

2. Scope of the security analysis

Key independence rests on the PRF assumption for HMAC-SHA256 [10] and on the standard properties of AES-CBC and Encrypt-then-MAC; no formal proof is given for the combined pipeline (Section II-H). Variant 2 offers no forward secrecy and loses confidentiality and integrity for a device–gateway pair if K is compromised; the tag does not prevent replay (Section III-C); and the ECC variant assumes authenticated public points, whose cost and security were not evaluated. The experiments cover nine record-modification classes and two replay conditions. Passive interception and active substitution of ECDH public points were analysed but not tested experimentally, and gateway compromise, side-channel attacks and denial of service are outside the model.

3. Statistical evidence

Entropy, avalanche effect, NPCR and UACI are indicators that originate in image-encryption analysis; they do not prove key-management security, and the NIST SP 800-22 suite [44] was applied only in part (Section III-F). The TOST margins were justified relative to the variability expected for ideal random ciphertext and not derived from a domain-specific security threshold; the conclusions hold regardless, because the largest observed mean difference (0.022 percentage points) is far inside the margins of 1 to 2 points, and a sensitivity check with margins twenty times tighter gave the same result (Section III-E). All results are from a single seeded run: the confidence intervals describe sampling variation across the 16,318 records, not variation across independent runs, and ECC keys were not seeded.

4. Dataset and encoder

The study uses a single dataset, collected from a controlled testbed with a patient simulator instead of real patients, so generalisation to other IoMT devices, sensor modalities, traffic patterns and clinical conditions is untested. In this dataset, the source and destination addresses are constant, and the descriptor varies only through the source port, with only three of 16,318 records carrying a non-numeric port; the finding that the KaTaPaYadi stage has no effect therefore applies to numeric descriptors and may differ for datasets with alphanumeric identifiers. Letters absent from the classical Kaṭapayādi table were mapped to their nearest phonetic equivalent, which affects at most those three records. MAC addresses were excluded from the descriptor because they separate attack from normal traffic perfectly in this testbed (Section II-E), so the evaluation does not cover descriptors that include hardware identifiers.

F. Future Work

Future work should prioritise the following, in order of how directly each addresses a limitation of this study.

1. Hardware validation

Deployment on medical-device-representative microcontrollers (for example ESP32- or nRF52840-class devices, ideally with hardware hash and elliptic-curve accelerators), measuring directly the quantities that this study could only estimate: (a) RAM footprint, including activation buffers, framework overhead and AES/HMAC working state; (b) flash usage for the cryptographic library and firmware (and for model weights if the LSTM stage is retained); (c) energy per key-establishment operation and per record, from current-trace measurements; (d) end-to-end per-record latency; (e) sustained throughput for the 88-byte payload; and (f) the incremental cost of ECC exchange against HMAC-SHA256 derivation and of Encrypt-then-MAC against unauthenticated encryption. This should include a same-board benchmark with a common curve and cryptographic library, which would separate the platform, implementation and curve effects discussed in Section IV-C.

2. Closing the security gaps identified in Section II-H

Replay protection through receiver-side sequence-number enforcement, or by deriving K_{mac} per record so that the tag binds the sequence position, with its cost measured; a forward-secure variant of keyed derivation (for example, key ratcheting) to remove the loss of forward secrecy on compromise of K; and an evaluation of authenticated ECC public keys (certificates or signatures), so that the cost of the ECC variant is measured for a deployable system and not for the key exchange alone. A hybrid deployment, in which ECC establishes the initial pairing and HMAC derivation protects the ongoing session, should be evaluated together with these.

3. Broader statistical and formal evaluation

Application of the remaining NIST SP 800-22 tests [44] and of SP 800-90B entropy estimation [45] to derived keys, repetition across independent seeds to report run-to-run variability, and a formal security argument for the combined construction. A comparison with an authenticated-encryption mode such as AES-GCM or ChaCha20-Poly1305 would test whether CBC with a separate HMAC tag is the right baseline.

4. Other datasets and identifier types

Evaluation on additional IoMT datasets and device classes, including datasets with alphanumeric device identifiers, would show whether the null effect of the KaTaPaYadi stage seen here, where descriptors were numeric, persists when identifiers vary in case, separators and format, and would test the stage against simple normalisation under those conditions.

V. Conclusion

This study compared elliptic-curve key exchange and keyed HMAC-SHA256 derivation as sources of per-record AES-128 keys for IoMT traffic, using all 16,318 records of WUSTL-EHMS-2020 in software simulation, and tested whether a KaTaPaYadi–LSTM context-binding stage adds any cryptographic benefit. HMAC-SHA256 derivation took 5.5 µs per record, 27.6 times less than a fresh P-256 exchange (152.1 µs) and 1.5 times less than a session-amortised one (8.4 µs). The two mechanisms were statistically equivalent in entropy, avalanche effect, NPCR and UACI, and every modified, substituted, reordered, truncated or forged record was rejected (16,318 of 16,318 in each class). The context stage changed no measured property yet cost about 1,900 times the HMAC step. Where a secret can be provisioned

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

in advance, HMAC-SHA256 derivation without the context stage is therefore the lower-cost choice, whereas ECC remains necessary for first-time pairing or forward secrecy. The principal limitation is that all results come from single-platform software simulation, and keyed derivation provides neither forward secrecy nor replay protection by itself. Future work should validate both mechanisms on microcontroller hardware, add replay protection and authenticated public keys, and extend the evaluation to other datasets.

Acknowledgment

The authors thank Washington University in St. Louis for making the WUSTL-EHMS-2020 dataset publicly available for IoMT security research.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Author Contributions

Sreeja P: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Visualization, Writing – original draft, Writing – review & editing. Priyang P. Bhatt: Supervision, Methodology, Formal analysis (review), Writing – review & editing. All authors have read and approved the final version of the manuscript and agree to be accountable for their respective contributions.

Data Availability

The WUSTL-EHMS-2020 dataset [1] is publicly available for research use at https://www.cse.wustl.edu/~jain/ehms/index.html. Supporting materials are available as described under Code Availability.

Code Availability

The source code (ECC key exchange, HMAC-SHA256 key derivation, KaTaPaYadi encoding, LSTM context processing and evaluation) is not publicly available because it is part of ongoing doctoral research. It may be obtained from the corresponding author (sreejap@mbit.edu.in) upon reasonable request, with the configuration files, random seeds, dependencies and instructions needed to reproduce the results.

Ethical Approval

This study used the publicly available, de-identified WUSTL-EHMS-2020 dataset [1]; no new data were collected from human participants. As a secondary analysis, it was exempt from institutional ethical approval.

Consent To Participate

Not applicable. No new data were collected from human participants, so participant consent was not required.

Consent for Publication

Not applicable. The manuscript contains no identifiable personal information, and the dataset contains no identifiable participant information.

Competing Interests

The authors declare that they have no known financial or non-financial competing interests that could have influenced the research reported in this manuscript.

Declaration of Generative Artificial Intelligence and AI-Assisted Technologies in the Writing Process

During the preparation of this manuscript, the authors used Claude Sonnet 5.5 (Anthropic) to assist with developing and debugging the simulation and analysis code, drafting and revising text, language editing, and checking the consistency of numerical values, equations, tables and references. All AI-assisted content was critically reviewed, verified, and revised by the authors. The experimental design, results and conclusions are the authors' own, and the authors take full responsibility for the accuracy, originality, scientific validity, and integrity of the manuscript. The AI tool was not listed as an author.

Research Integrity and Originality

The authors confirm that the manuscript is their original work, has not been published previously, and is not under consideration for publication elsewhere. All sources, datasets, algorithms, and previously published methods have been appropriately acknowledged and cited.

References

[1] A. A. Hady, A. Ghubaish, T. Salman, D. Unal, and R. Jain, "Intrusion Detection System for Healthcare Systems Using Medical and Network Data: A Comparison Study," *IEEE Access*, vol. 8, pp. 106576–106584, 2020, doi: 10.1109/ACCESS.2020.3000421.

[2] Z. K. Mohammed, M. A. Mohammed, K. H. Abdulkareem, D. A. Zebari, A. Lakhan, H. A. Marhoon, J. Nedoma, and R. Martinek, "A metaverse framework for IoT-based remote patient monitoring and virtual consultations using AES-256 encryption," *Appl. Soft Comput.*, vol. 158, art. 111588, 2024, doi: 10.1016/j.asoc.2024.111588.

[3] W. Diffie, P. C. van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Des. Codes Cryptogr.*, vol. 2, no. 2, pp. 107–125, 1992, doi: 10.1007/BF00124891.

[4] İ. K. Çekiş, A. Toros, N. Apaydın, and İ. Ozcelik, "Performance comparison of ECC libraries for IoT devices," *Eskişehir Technical University Journal of Science and Technology A – Applied Sciences and Engineering*, vol. 25, no. 2, pp. 278–288, 2024, doi: 10.18038/estubtda.1427488.

[5] S. S. Dhanda, S. Kumari, V. Kumar, and S. K. Gupta, "A low-latency 163-bit ECC processor for IoT applications," *J. Electr. Eng.*, vol. 76, no. 3, pp. 241–255, 2025, doi: 10.2478/jee-2025-0025.

[6] N. G. Zinabu, Y. W. Marye, K. K. Tune, and S. A. Demilew, "Comprehensive Analysis of Lightweight Cryptographic Algorithms for Battery-Limited Internet of Things Devices," *Int. J. Distrib. Sens. Netw.*, vol. 2025, art. no. 9639728, 2025, doi: 10.1155/dsn/9639728.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

[7] K. Kinani, F. Amounas, S. Bendaoud, M. Azrour, and M. Badiy, "New Image Crypto-Compression Scheme Based on ECC and Chaos Theory for High-Speed and Reliable Transmission of Medical Images in the IOMT," *Cybern. Inf. Technol.*, vol. 24, no. 4, pp. 108–125, 2024, doi: 10.2478/cait-2024-0038.

[8] D. Canavese, L. Mannella, L. Regano, and C. Basile, "Security at the Edge for Resource-Limited IoT Devices," *Sensors*, vol. 24, no. 2, art. 590, 2024, doi: 10.3390/s24020590.

[9] B. Guşită, A. A. Anton, C. S. Stângaciu, D. Stănescu, L. I. Găină, and M. V. Micea, "Securing IoT Edge: A Survey on Lightweight Cryptography, Anonymous Routing, and Communication Protocol Enhancements," *Int. J. Inf. Secur.*, vol. 24, art. 149, 2025, doi: 10.1007/s10207-025-01071-7.

[10] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in Advances in Cryptology — CRYPTO '96, LNCS vol. 1109, 1996, pp. 1–15, doi: 10.1007/3-540-68697-5_1.

[11] N. F. Mufidah and H. H. Nuha, "Performance and Security Analysis of Lightweight Hash Functions in IoT," *Jurnal Informatika: Jurnal Pengembangan IT*, vol. 9, no. 3, pp. 264–270, 2024, doi: 10.30591/jpit.v9i3.7633.

[12] I. Baird, I. Wadhaj, B. Ghaleb, C. Thomson, and G. Russell, "Evaluating the Energy Costs of SHA-256 and SHA-3 (KangarooTwelve) in Resource-Constrained IoT Devices," *IoT*, vol. 6, no. 3, art. 40, 2025, doi: 10.3390/iot6030040.

[13] T. Alsolami, B. Alsharif, and M. Ilyas, "Enhancing Cybersecurity in Healthcare: Evaluating Ensemble Learning Models for Intrusion Detection in the Internet of Medical Things," *Sensors*, vol. 24, no. 18, art. 5937, 2024, doi: 10.3390/s24185937.

[14] M. Alalhareth and S.-C. Hong, "Enhancing the Internet of Medical Things (IoMT) Security with Meta-Learning: A Performance-Driven Approach for Ensemble Intrusion Detection Systems," *Sensors*, vol. 24, no. 11, art. 3519, 2024, doi: 10.3390/s24113519.

[15] W. Wu, H. Fouzi, S. Sidi-Mohammed, et al., "Improving cyber-attack detection in Internet of Medical Things using ensemble deep learning methods," *Cluster Comput.*, vol. 28, art. 891, 2025, doi: 10.1007/s10586-025-05660-y.

[16] S. Garg, K. Kaur, G. Kaddoum, J. J. P. C. Rodrigues, and M. Guizani, "Secure and Lightweight Authentication Scheme for Smart Metering Infrastructure in Smart Grid," *IEEE Trans. Ind. Informat.*, vol. 16, no. 5, pp. 3548–3557, May 2020, doi: 10.1109/TII.2019.2944880.

[17] D. Guo, K. Cao, J. Xiong, D. Ma, and H. Zhao, "A lightweight key generation scheme for the internet of things," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 12137–12149, 2021, doi: 10.1109/JIOT.2021.3060438.

[18] A. Sarkar, "A symmetric neural cryptographic key generation scheme for IoT security," *Applied Intelligence*, vol. 53, no. 8, pp. 9344–9367, 2023, doi: 10.1007/s10489-022-03904-7.

[19] M. K. Hasan, S. Islam, R. Sulaiman, S. Khan, et al., "Lightweight Encryption Technique to Enhance Medical Image Security on Internet of Medical Things Applications," *IEEE Access*, vol. 9, pp. 47731–47742, 2021, doi: 10.1109/ACCESS.2021.3061710.

[20] M. Senthilkumar, K. Suthendran, and V. Ravi, "Enhancing Medical Image Security through a Novel Framework: Crypto-Aware Elliptic Curve Diffie-Hellman with Key Derivation Function (CAECDH-KDF)," *Open Bioinform. J.*, vol. 17, art. e18750362303634, 2024, doi: 10.2174/0118750362303634240624112446.

[21] Y. Alghamdi, A. Munir, and J. Ahmad, "A Lightweight Image Encryption Algorithm Based on Chaotic Map and Random Substitution," *Entropy*, vol. 24, no. 10, art. no. 1344, 2022, doi: 10.3390/e24101344.

[22] S. R. Sarma, "The Kaṭapayādi system of numerical notation and its spread outside Kerala," *Revue d'histoire des mathématiques*, vol. 18, no. 1, pp. 37–66, 2012, doi: 10.24033/rhm.167.

[23] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[24] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques* , 3rd ed. Waltham, MA, USA: Morgan Kaufmann, 2012, doi: 10.1016/C2009-0-61819-5.

[25] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000, doi: 10.1162/089976600300015015.

[26] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017, doi: 10.1109/TNNLS.2016.2582924.

[27] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, Internet Engineering Task Force, Feb. 1997, doi: 10.17487/RFC2104.

[28] National Institute of Standards and Technology, "The Keyed-Hash Message Authentication Code (HMAC)," FIPS PUB 198-1, Gaithersburg, MD, USA, Jul. 2008, doi: 10.6028/NIST.FIPS.198-1.

[29] National Institute of Standards and Technology, "Secure Hash Standard (SHS)," FIPS PUB 180-4, Gaithersburg, MD, USA, Aug. 2015, doi: 10.6028/NIST.FIPS.180-4.

[30] Q. Dang, "Recommendation for Applications Using Approved Hash Algorithms," NIST Special Publication 800-107 Rev. 1, Gaithersburg, MD, USA, Aug. 2012, doi: 10.6028/NIST.SP.800-107r1.

[31] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," FIPS 197, 2001, doi: 10.6028/NIST.FIPS.197.

[32] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

Operation: Methods and Techniques," NIST SP 800-38A, 2001, doi: 10.6028/NIST.SP.800-38A.

[33] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," in Advances in Cryptology — ASIACRYPT 2000, LNCS vol. 1976, pp. 531–545, 2000, doi: 10.1007/3-540-44448-3_41.

[34] H. Krawczyk, "Cryptographic Extraction and Key Derivation: The HKDF Scheme," in Advances in Cryptology — CRYPTO 2010, LNCS vol. 6223, pp. 631–648, 2010, doi: 10.1007/978-3-642-14623-7_34.

[35] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, Internet Engineering Task Force, May 2010, doi: 10.17487/RFC5869.

[36] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography," NIST Special Publication 800-56A Rev. 3, Gaithersburg, MD, USA, Apr. 2018, doi: 10.6028/NIST.SP.800-56Ar3.

[37] L. Chen, D. Moody, K. Randall, A. Regenscheid, and A. Robinson, "Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters," NIST Special Publication 800-186, Gaithersburg, MD, USA, Feb. 2023, doi: 10.6028/NIST.SP.800-186.

[38] E. Barker, "Recommendation for Key Management: Part 1 – General," NIST Special Publication 800-57 Part 1 Rev. 5, Gaithersburg, MD, USA, May 2020, doi: 10.6028/NIST.SP.800-57pt1r5.

[39] National Institute of Standards and Technology, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," NIST SP 800-90A, 2012, doi: 10.6028/NIST.SP.800-90A.

[40] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, no. 3–4, pp. 379–423, 623–656, 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x.

[41] L. Paninski, "Estimation of Entropy and Mutual Information," *Neural Comput.*, vol. 15, no. 6, pp. 1191–1253, 2003, doi: 10.1162/089976603321780272.

[42] A. F. Webster and S. E. Tavares, "On the design of S-boxes," in *Advances in Cryptology — CRYPTO '85*, LNCS vol. 218, 1986, pp. 523–534, doi: 10.1007/3-540-39799-X_41.

[43] D. J. Schuirmann, "A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability," *Journal of Pharmacokinetics and Biopharmaceutics*, vol. 15, no. 6, pp. 657–680, 1987, doi: 10.1007/BF01068419.

[44] National Institute of Standards and Technology, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," NIST SP 800-22 Rev. 1a, 2010, doi: 10.6028/NIST.SP.800-22r1a.

[45] National Institute of Standards and Technology, "Recommendation for the Entropy Sources Used for Random Bit Generation," NIST SP 800-90B, 2018, doi: 10.6028/NIST.SP.800-90B.

Author Biography



Sreeja P. received the B.Tech. degree from Kannur University, Kerala, India, and the M.Tech. degree in Computer Science and Engineering from Glocal University, Uttar Pradesh, India. She is currently pursuing the Ph.D. degree at CVM University, Gujarat, India, and is an Assistant Professor in the Department of Computer Science and Engineering at Madhuben & Bhanubhai Patel Institute of Technology (MBIT), Charutar Vidyamandal (CVM) University, Vallabh Vidyanagar, Gujarat, India. Her research interests include lightweight cryptography, IoT and IoMT security, key management, and machine learning applications in security. She is the first and corresponding author of this work and carried out the conceptualisation, implementation, experiments, and drafting of the manuscript. She can be contacted at sreejap@mbit.edu.in



Dr. Priyang P. Bhatt received his Ph.D. in Computer Engineering. He is an Associate Professor and Head of the Department of Information Technology at ADIT, CVM University, New Vallabh Vidyanagar, Gujarat, India. His research interests include network security, cryptographic protocols, IoT system design, and machine learning applications. He has extensive experience supervising doctoral research and has published widely. Dr. Bhatt is a member of professional bodies including IEEE. He supervises the first author's doctoral research at CVM University, and in this work he supervised the study, guided the methodology of the comparison between ECC key exchange and HMAC-SHA256 key derivation, and reviewed the analysis and the manuscript. He can be contacted at priyang.bhatt@cvmu.edu.in.

Manuscript received 8 August 2026; Accepted 14 October 2026; Available online 26 November 2026
Digital Object Identifier (**DOI**): https://doi.org/10.35882/jeeemi.v9i1.1996
Copyright © 2026 by the authors. This work is an open-access article and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).