SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V13I9P101

Volume 13 Issue 9, 1-11, September 2026

Original Article

Hardware Evaluation of a Lightweight Secure Communication Protocol for Constrained IoT Devices

Turelle Ange Leslie AROUN FAIMINDI BEREGNON[1], Lawrence NGUGI[2], Irene MUISYO[3]

[1]*Department of Electrical Engineering, Pan African University Institute for Basic Sciences, Technology and Innovation, hosted at Jomo Kenyatta University of Agriculture and Technology, Nairobi, Kenya.*
[2]*Department of Telecommunication and Computer Engineering, Jomo Kenyatta University of Agriculture and Technology, Nairobi, Kenya.*
[3]*Department of Electronic and Computer Engineering, Jomo Kenyatta University of Agriculture and Technology, Nairobi, Kenya.*

[1]*Corresponding Author : faimindileslie@gmail.com*

Received: 24 June 2026 Revised: 20 August 2026 Accepted: 27 August 2026 Published: 29 September 2026

Abstract *- This paper details the design and hardware evaluation of a lightweight communication protocol for constrained Internet of Things (IoT) devices. The design was implemented on one single Nordic nRF52840 Development Kit (ARM Cortex-M4F, 64 MHz) with the help of Zephyr OS, using the micro-ECC library for ECDH key exchange on the secp256r1 curve and TinyCrypt library for key derivation and AES-128-CCM cipher suite. It took 31,405,906 CPU cycles in a handshake operation that comprised generation of key pair, deriving shared secret, key derivation and encryption of 128-byte transcript using AES-CCM, which was performed ten times without variation, taking about 491 ms at 64 MHz frequency. Energy profiling using the Nordic Power Profiler Kit II over a 447.2 ms time window revealed an energy consumption of about 32.8 mJ. The implementation consumes 39 KB Flash and 13 KB RAM, corresponding to 3.8% and 5.1% of the device resource pool, respectively. On an individual stage level, ECDH scalar multiplication over secp256r1 is responsible for 99.8% of the overall cycle count, whereas HKDF-SHA256 and AES-128-CCM cover 0.2% together. The protocol provides key agreement and authenticated encryption but lacks public key authentication functionality. This shows that the full protocol could be implemented on an IoT device with reasonable resource usage, and that hardware evaluation gives a good point of reference for future optimizations.*

Keywords *- ECDH, Elliptic curve cryptography, Energy profiling, Zephyr, Embedded systems.*

1. Introduction

The widespread deployment of IoT devices in smart cities, industrial automation, healthcare, and environmental monitoring has created a pressing need for practical security solutions that fit within the tight resource constraints of embedded hardware [1, 2]. Most IoT nodes operate with limited processing power, small memory, and strict energy budgets, making the direct use of conventional security protocols such as Transport Layer Security (TLS) [3] and Datagram Transport Layer Security (DTLS) [4] impractical due to their high computational overhead, large handshake message sizes, and significant memory requirements.

Several lightweight alternatives have been proposed to address these limitations. TinyDTLS adapts DTLS for constrained nodes [5]. EDHOC reduces key-exchange overhead using compact CBOR/COSE encoding [6]. OSCORE provides application-layer end-to-end security for CoAP-based communication [7]. Despite these advances, all of these frameworks depend on public-key cryptography, with elliptic curve scalar multiplication remaining a significant computational bottleneck on unaccelerated embedded processors [8, 9]. Symmetric operations such as AES-CCM are substantially more efficient once a session key has been established.

There are three common deficiencies in the body of literature reviewed. First, almost all of the existing evaluation efforts characterize cryptographic primitives individually, and, consequently, it is impossible to estimate the total overhead of a complete pipeline consisting of a key-agreement scheme, key derivation function, and authenticated encryption scheme based on these efforts alone. Second, the results of a considerable number of studies rely on simulations or emulations, not on measurements performed on actual hardware. Third, in most cases where the results of hardware-based measurements are available, not all necessary parameters are included, such as

energy measurements without cycles or memory consumption without latency. To determine if a particular security protocol can be implemented on a particular constrained device, it is necessary to know all of the five parameters (CPU cycles, latency, energy, Flash, and RAM), measured for the same firmware image on the same hardware platform using identical conditions.

This paper addresses that gap directly. A lightweight secure communication protocol combining ECDH (secp256r1) for key agreement, HKDF-SHA256 for key derivation, and AES-128-CCM for authenticated encryption was implemented on a Nordic nRF52840 microcontroller running Zephyr RTOS. The complete handshake was profiled on real hardware using the ARM DWT cycle counter and the Nordic Power Profiler Kit II. It should be noted that the implementation provides key agreement and authenticated encryption but does not include public-key authentication; the protocol does not protect against man-in-the-middle attacks unless public keys are pre-distributed or authenticated by other means. This range is clearly defined because it defines the point of comparison in Section 8, where the footprint and the latency reported relate to a key agreement that is not authenticated, which makes it impossible to compare them directly with other values.

The main contributions of this work are:

- A complete hardware implementation of the ECDH + HKDF-SHA256 + AES-128-CCM handshake on the nRF52840 using Zephyr RTOS, micro-ECC, and TinyCrypt.
- Hardware-based measurement of CPU cycles (31,405,906), cycle-based latency (≈491 ms), energy consumption (32.8 mJ over 447.2 ms), Flash usage (39 KB), and RAM usage (13 KB) for the complete handshake.
- Per-cycle stage measurement to determine the costs of each cryptographic operation, finding out that ECDH scalar multiplication is responsible for 99.8% of the handshake process, HKDF-SHA256 for 0.17%, and AES-128-CCM for less than 0.01%, thus discovering which cryptographic operation needs to be optimized.
- An indicative comparison with published results for TinyDTLS, EDHOC, and OSCORE, establishing a reproducible hardware baseline for this class of protocol.

2. Related Work

Existing IoT security research can be grouped into three areas: protocol design and frameworks, lightweight cryptographic implementations, and hardware performance evaluation.

2.1. IoT Security Protocols and Frameworks

TLS [3] and DTLS [4] provide well-established security for internet applications, but have high computational and memory requirements that limit their use on constrained IoT nodes. TinyDTLS is a compact DTLS implementation targeting resource-constrained devices; reported implementations on Cortex-M class processors require 60–120 KB of Flash and 20–40 KB of RAM, and introduce handshake latencies in the range of 600–1200 ms [5]. EDHOC [6] reduces handshake message overhead to approximately 100–200 bytes through CBOR/COSE encoding and achieves authenticated key exchange in two or three messages; reported evaluations on constrained IoT hardware show Flash requirements of 45–70 KB, RAM of 10–25 KB, and latency in the range 200–500 ms, as summarized in Table 3. OSCORE [7] complements EDHOC by providing application-layer object security for CoAP messages, offering end-to-end confidentiality and integrity; it has been evaluated on embedded IoT platforms with Flash in the range 50–100 KB and RAM of 15–30 KB, as summarized in Table 3. These protocols focus primarily on protocol-level or component-level evaluation; comprehensive hardware profiling of the complete handshake covering CPU cycles, energy, Flash, and RAM together under a single consistent setup is less commonly reported [10].

2.2. Lightweight Cryptographic Implementation in Embedded Systems

ECC has been widely adopted for IoT key agreement due to its stronger security-to-key-size ratio compared to RSA [11]. TinyECC is a configurable ECC library that demonstrated the feasibility of elliptic curve operations on Wireless Sensor Network platforms, though at comparatively high computational cost due to software-only scalar multiplication [8, 12]. Studies consistently show that ECDH scalar multiplication dominates the execution time and energy cost of ECC-based protocols on constrained processors [8, 9]. In contrast, symmetric operations such as AES in CCM mode are considerably more efficient and are well-suited to authenticated encryption in IoT communications [6]. However, most published works evaluate ECDH, HKDF, or AES-CCM separately rather than as an integrated implementation, making it difficult to assess the combined resource impact.

2.3. Performance and Energy Evaluation of IoT Security Mechanisms

Research into sensor network security has shown that public key operations consume considerable amounts of energy on battery-operated sensors, where scalar multiplication is the most expensive operation [9]. Subsequent studies highlighted that simulation-based evaluations often do not reflect actual hardware performance [5] and called for hardware measurements of CPU cycles, latency, energy, and memory to provide realistic feasibility estimates [5].

A number of recent papers have considered the performance of cryptographic mechanisms and protocol

implementations on embedded IoT devices. Kaganurmath et al. described a lightweight key-sharing method for MQTT communication and presented its computation cost on a constrained device [14]. Shete et al. studied lightweight encryption in GSM-MQTT frameworks in a constrained setup [15].

Naser et al. performed an analysis of the energy consumption of various lightweight authentication schemes for medical IoT systems [16], whereas Radhi et al. [17] and Al-Zubaidi et al. [18] considered identity authentication and physical tamper resistance, respectively. The above papers analyze the properties of particular schemes. Papers that study the whole process of ECDH, key derivation, and authenticated encryption, measured in terms of CPU cycles, latencies, energy consumption, Flash and RAM, from a particular implementation, are few.

2.4. Research Gap

Existing literature on secure communication protocols for constrained IoT systems has extensively explored protocol design and cryptographic building blocks such as ECDH key exchange, HKDF key derivation, and AES-CCM encryption. Several studies have also evaluated hardware performance for specific protocols, including OSCORE, EDHOC, and DTLS-based implementations on embedded platforms.

However, a more detailed gap remains in studies that provide a fully integrated implementation and unified hardware-level profiling of the complete security pipeline under a single consistent experimental setup. In particular, while individual cryptographic primitives are well studied, limited studies report a combined evaluation of ECDH, HKDF-SHA256, and AES-CCM executed sequentially on the same constrained hardware platform while simultaneously capturing CPU usage, execution latency, energy consumption, Flash utilization, and RAM usage under identical conditions.

Furthermore, most existing works tend to focus on either protocol-level performance or isolated cryptographic benchmarking, rather than providing a system-level measurement of the entire secure communication workflow as implemented on a real embedded RTOS environment. This limits the ability to directly understand the cumulative cost of deploying end-to-end security on resource-constrained devices.

To address this gap, this paper presents a complete hardware implementation and profiling of an end-to-end lightweight secure communication process combining ECDH key exchange, HKDF-SHA256 key derivation, and AES-CCM encryption on the nRF52840 platform using Zephyr RTOS. Unlike previous studies, the contribution of this work lies not in proposing new cryptographic algorithms, but in providing a detailed and unified hardware performance analysis of the full security pipeline, including CPU cycles, latency, energy consumption, Flash usage, and RAM utilization measured under a consistent experimental framework.

3. System Architecture

3.1. System Overview

The proposed protocol defines two communicating roles: an Initiator and a Responder. In the evaluated implementation, both roles were executed sequentially on a single Nordic nRF52840 Development Kit, with the public-key exchange performed as a local memory transfer rather than over a wireless link. Figure 1 shows the resulting architecture.

The described communication protocol consists of three key elements in terms of cryptography, which are ECDH (key agreement), HKDF-SHA256 (key derivation function), and AES-CCM (authenticated encryption).

Formal security verification of the protocol is outside the scope of this paper, which is concerned with hardware performance characterization.

3.2. Elliptic Curve Diffie-Hellman (ECDH)

ECDH is applied to derive a shared secret between Initiator and Responder through a non-secure communication channel. Each party generates its own private-public key pair and transmits only the public component. The derivation of a shared secret occurs due to the use of the concept of elliptic curve scalar multiplication for both entities. ECDH was selected because it provides more security with relatively small keys than RSA.

3.3. Key Derivation Function (HKDF-SHA256)

However, the shared secret received above cannot be applied as an encryption key, and the following step involves applying the HKDF-SHA256 protocol to generate a session key. This choice allows improving the quality of the session key and isolating the key derivation stage from the encryption stage.

3.4. Authenticated Encryption using AES-CCM

AES-CCM uses the approach of authenticated encryption that is done by integrating counter mode encryption along with CBC MAC integrity check using a single key [13].

This technique suits well for the IoT environment because confidentiality and integrity are achieved through a single cipher primitive, not two different processes. In this regard, ECDH is used in order to provide key agreement, HKDF-SHA256 is used for deriving the session key, while AES-CCM makes sure that the message remains secret and its integrity.

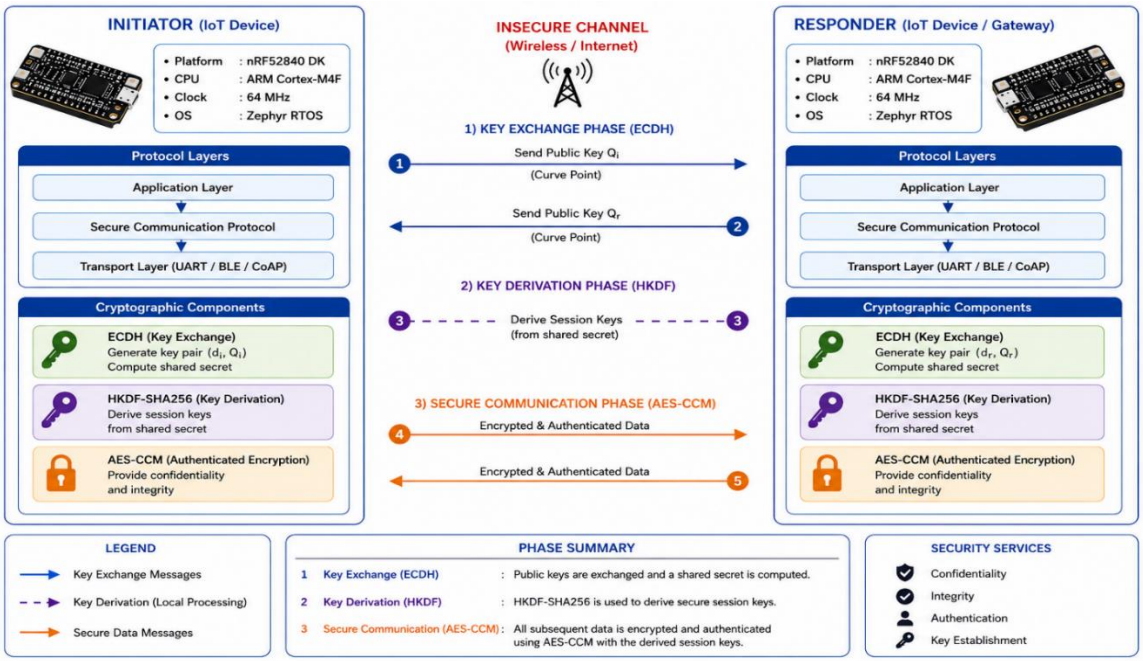

Fig. 1 Architecture of the proposed lightweight secure communication protocol

4. Methodology and Protocol Operation

4.1. Handshake Process

This secure communication connection is created using a lightweight handshake protocol. The aim of this protocol is to ensure that both the Initiator and Responder reach a consensus on the session key despite the communication link being insecure. The handshake protocol consists of five steps, namely: creation of key pairs, public key exchange, calculation of shared secrets, key derivation, and secure communication. The entire energy analysis of CPU cycle consumption involves all five steps: creation of key pairs, public key exchange (local memory transmission), calculation of shared secrets, key derivation, and AES-CCM encryption of the 128-byte transcript.

Step 1: Key Pair Generation

Each entity generates an elliptic curve key pair on the secp256r1 (P-256) curve, consisting of a private key and a corresponding public key. The private key remains stored locally, while the public key is used for exchange with the peer device.

Step 2: Public Key Exchange

The Initiator and the Responder exchange their public keys through the communication channel. Since only public keys are exchanged, the private keys are never transmitted.

Step 3: Shared Secret Computation

After receiving the peer public key, each entity computes the shared secret using ECDH. For the Initiator, the shared secret can be expressed as:

$$S = d_A \times Q_B \quad (1)$$

Where d_A is the Initiator's private key, and Q_B is the Responder's public key. The Responder performs the corresponding computation using its private key and the Initiator's public key. Both entities independently derive the same shared secret.

Step 4: Key Derivation using HKDF

The shared secret is run through HKDF-SHA256 in order to extract two separate 16-byte keys, Ksession and Kmac, using the labels ‘session’ and ‘mac’, respectively. Separation into two separate keys with the help of different labels guarantees the separation of domains and prevents deriving one key from another:

$$K_{session} = HKDF - SHA256\,(S, salt = 0, info = session) \quad (2)$$

$$K_{mac} = HKDF - SHA256\,(S, salt = 0, info = mac) \quad (3)$$

No additional salt was used; context labels 'session' and 'mac' were used to separate the two derived keys.

Step 5: Secure Communication using AES-CCM

The AES-128-CCM algorithm is used on the 128-byte transcript constructed by joining together the public keys of the Initiator and Responder, which are each 64 bytes long. The algorithm works based on a single 16-byte key, which is the Kmac in the current implementation.

In addition, the 13-byte nonce and 8-byte tag are utilized. Through encryption of the transcript, both public keys are attached to the generated key material, and thus any modification to the keys during exchange will be noted by the tag verification.

4.2. Handshake Sequence

Figure 2 presents the handshake procedure that takes place between the Initiator and the Responder. In this process, the Initiator starts by sending its public key to the Responder, following which the Responder sends back its own public key to the Initiator. Both parties independently calculate the shared secret based on the ECDH algorithm and derive the session key using the HKDF-SHA256 protocol. After successfully deriving the session key, communication proceeds under protection from AES-CCM.

4.3. Design Considerations

Design of the proposed protocol is based on the constraints that apply to the embedded IoT devices. Public key calculations are limited to handshake only in order to minimize repetitive overhead while transferring the data. Energy efficiency is addressed by confining public-key operations to the handshake and using AES-128-CCM for all subsequent session traffic; the cost of that post-handshake traffic was not measured in this work and is noted as a limitation in Section 9.

Use of memory is minimized through minimizing the complexity of the protocol. In terms of security, ECDH is used for key agreement, HKDF-SHA256 is used for the derivation of the session key, and AES-CCM ensures confidentiality and integrity.

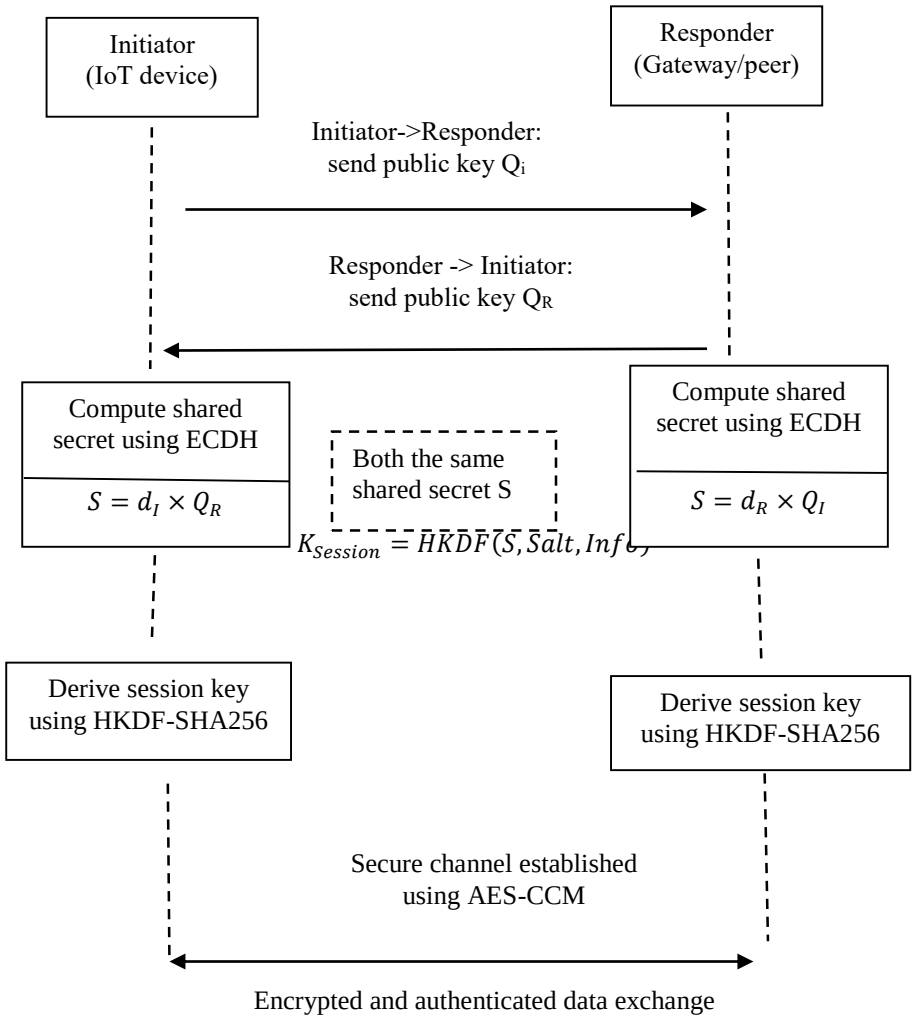

Fig. 2 Handshake sequence of the proposed secure communication protocol

5. Implementation

Development Kit, with an ARM Cortex-M4F microcontroller clocked at 64 MHz, 1 MB flash memory, and 256 KB RAM. The software stack was Zephyr RTOS and nRF Connect SDK, offering hardware abstractions, timer functions, and memory management. The Initiator and Responder functionalities were implemented in the same firmware application and sequentially run on the same board.

No wireless communication was involved; the public-key exchange was completed as a local memory exchange between the two roles in the same execution context. Fresh

ECDH key pairs were generated for each handshake execution.

The protocol was decomposed into four major modules: (1) ECDH key pair generation and public key exchange using secp256r1 (P-256); (2) ECDH shared secret calculation; (3) HKDF-SHA256 session key derivation; and (4) AES-128-CCM authenticated encryption of a 128-byte transcript. Micro-ECC was utilized for the ECDH over secp256r1, and TinyCrypt was used for the HKDF-SHA256 and AES-128-CCM without modifications to the library source. AES-128-CCM was set up with a 13-byte nonce and an 8-byte authentication tag used for the encryption of a 128-byte transcript. To enhance execution predictability, static memory allocation, reduction in use of intermediate buffers, and elimination of redundant actions were utilized. Instrumentation was conducted accurately using the ARM DWT cycle counter. A single counter pair was used to cover the whole handshake, starting from the key pair generation up to AES-CCM encryption of the transcript to provide an aggregate number of cycles spent on the whole handshake. Stage-wise costs were calculated using dedicated firmware builds for each operation, so that the cost of ECDH, HKDF-SHA256, and AES-128-CCM operations described in Section 7 are actual measurements and not estimations based on known primitive properties.

For the per-stage build measurements, the AES-128-CCM performance was measured using a 64-byte payload and the HKDF-SHA256 using only one key expansion, but for the integrated handshake performance, the encryption involves 128 bytes and the expansion of keys twice. These two per-stage figures are therefore lower bounds; as both operations together account for less than 0.2% of the total cost, this does not affect the identification of ECDH scalar multiplication as the dominant contributor. The PPK2 energy measurement window was aligned to the handshake boundaries.

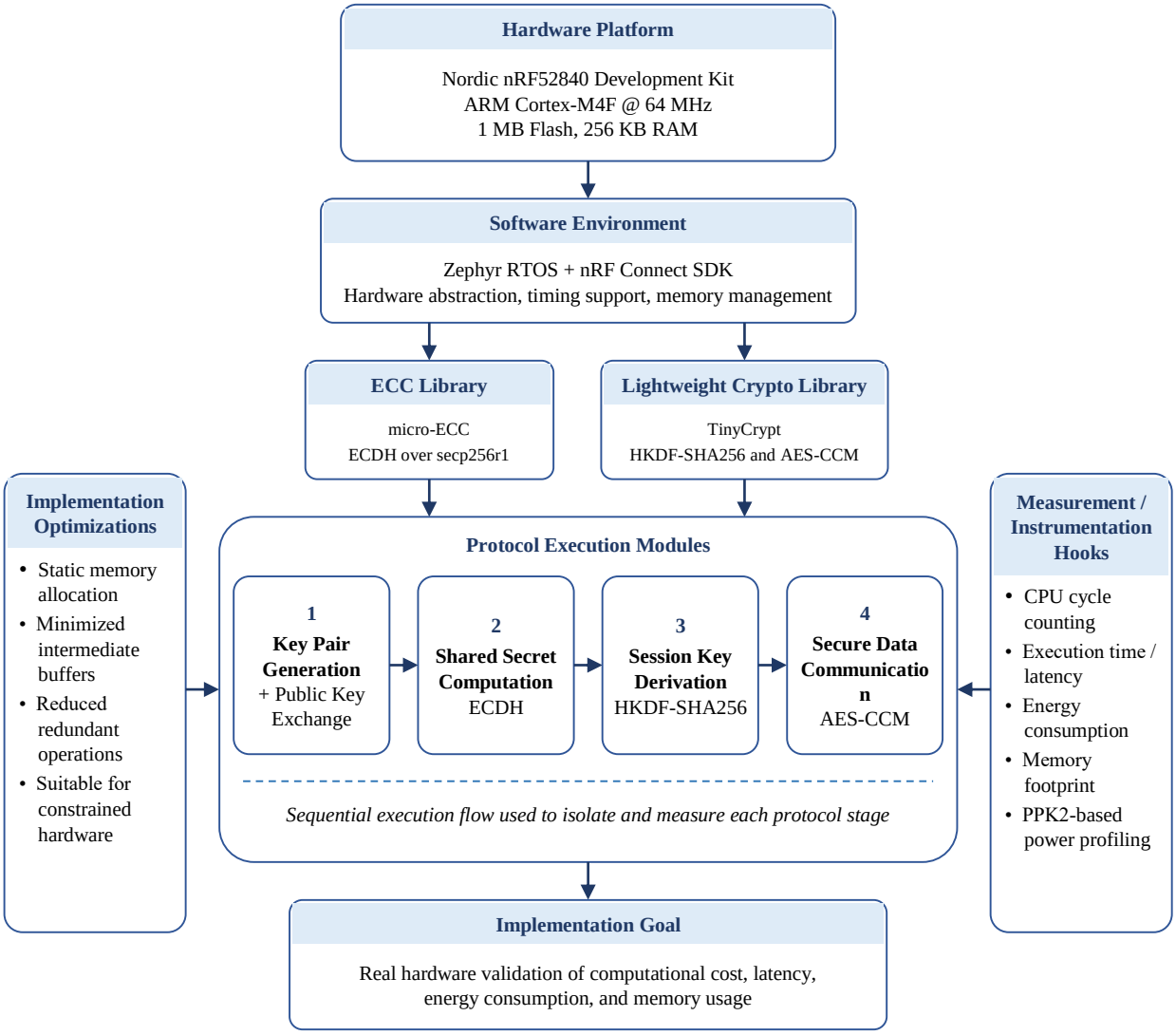

Fig. 3 Implementation framework of the proposed protocol on the nRF52840 platform

The 39 KB flash footprint and 13 KB RAM footprint of the protocol refer to the complete firmware, which includes the Zephyr RTOS kernel, micro-ECC, TinyCrypt libraries and the protocol itself. The firmware was built using the Zephyr RTOS default build configuration and size optimization flag (-Os). All components depicted in Figure 3 on the embedded platform, cryptographic libraries, protocol components, and measurement hooks were implemented in practice and not conceptualized only.

6. Experimental Setup

The experiment was performed with a single Nordic nRF52840 Development Kit by measuring through hardware the runtime performance of the designed protocol through hardware. The Initiator and the Responder operations were performed successively on the same device without any wireless link between them. The test was performed for CPU cycles, cycle-time of execution, energy consumption, and memory usage. The experimental setup is shown in Table 1 below.

Table 1. Experimental setup and evaluation configuration

Parameter	Description
Embedded Platform	Nordic nRF52840 Development Kit
Processor	ARM Cortex-M4F
Clock frequency	64 MHz
Flash memory	1 MB
RAM	256 KB
Operating system	Zephyr RTOS
SDK	nRF Connect SDK
Key Exchange	ECDH using secp256r1
Key Derivation	HKDF-SHA256
Authenticated encryption	AES-128-CCM (13-byte nonce, 8-byte tag)
ECC library	micro-ECC (uECC secp256r1)
Symmetric crypto library	TinyCrypt (HMAC-SHA256, AES-128-CCM)
CPU cycle measurement	ARM DWT cycle counter
Energy measurement	Nordic Power Profiler Kit II (PPK2)
Memory measurement	Zephyr build output
Evaluation metrics	CPU cycles, latency, energy consumption, Flash, RAM
Encrypted payload	128-byte transcript (concatenated public keys)

6.1. Measurement of CPU Cycles

The measurement of CPU cycle consumption was done through the use of DWT (Data Watchpoint and Trace) cycle counter on the ARM Cortex-M4F processor. The cycle counter was set at the beginning of key pair generation and halted at the end of AES-128-CCM encryption of the payload to be transmitted, involving the five handshakes mentioned above. These include the following steps: key pair generation, public key exchange (transfer between local memories), computing of shared secret, key derivation using HKDF-SHA256 and AES-128-CCM encryption of the 128-byte transcript, which includes the initiator and responder's public keys (64 bytes each), with a 128-bit key, 13-byte nonce, and 8-byte tag.

6.2. Latency Measurement

Latency was determined directly from the cycle count for the DWT, as a ratio between the total number of cycles measured and the processor frequency of 64 MHz. Thus, a deterministic calculation of the execution time of the cryptographic handshake was obtained for the specified implementation and platform.

6.3. Energy Consumption

Energy consumption was measured through the use of the Nordic Power Profiler Kit II (PPK2). The PPK2 measures the current used by the embedded system in real time. Generally, the energy consumption can be defined as follows:

$$E = V \times I \times t \quad (4)$$

where E is the energy consumption, V is the voltage, I is the current, and t is the duration of the measurement.

Since the measured charge is reported by the PPK2, too, the energy consumption can be computed as follows:

$$E = V \times Q \quad (5)$$

where Q is the measured charge, this charge-based method is used to calculate the energy consumed during the chosen window of time. Such an approach gives a real estimate of the energy costs for running the protocol.

The supply voltage for the energy computation is taken as 3.013 V, which was set up in the configuration of the PPK2 source meter. The measurement window of the PPK2 is 447.2 ms, which was manually chosen with respect to covering the main handshake procedures. The difference of about 44 ms between the PPK2 window and the cycle-based one (491 ms) is caused by manual window selection in the PPK2.

6.4. Memory Footprint

Both flash and RAM sizes were determined using the build output from Zephyr after building the whole firmware. These memory footprints include the Zephyr RTOS, the micro-ECC library and TinyCrypt library, as well as the protocol implementation code. No Zephyr modules were

disabled beyond the defaults, so the reported footprint corresponds to the standard Zephyr build configuration with size optimisation (-Os) enabled.

6.5. Measurement Procedure

The firmware performed the handshake in a loop with a three-second pause in between each execution so that each execution can be seen as a separate peak of current in the PPK2 trace. The DWT timer counted 31,405,906 cycles for each of the ten executions in a row without any variance. The stage-specific builds exhibited identical behaviour with constant cycle counts for each execution. This consistency is due to the deterministic nature of the control flow of the cryptographic primitives running at a constant clock frequency of the processor.

7. Results and Discussions

7.1. Experimental Result

The performance of the developed lightweight secure communication protocol is assessed through the hardware implementation as described in Section 5. The test consists of CPU cycle consumption, cycle-based latency, energy consumption, and memory footprint. The full handshake performance includes all five stages: generation of the ECDH key pairs, calculation of the shared secret, deriving K_session and K_mac from the shared secret using the HKDF-SHA256 function, and encrypting the 128-byte transcript with AES-128-CCM. The firmware executes the handshake in a loop, returning 31,405,906 cycles across ten consecutive executions with no observed variation.

Table 2. Experimental performance results of the proposed protocol

Metric	Value
Handshake CPU Cycles	31,405,906 cycles
Cycle-based time	≈ 491 ms
PPK II selected window	447.2 ms
Energy consumption	32.8 mJ
Flash memory	39 KB
RAM usage	13 KB
ECDH cycles (component)	≈ 31,350,529 cycles (≈ 99.8% of total)
HKDF-SHA256 cycles (component)	52,269 cycles (≈ 0.17% of total)
AES-128-CCM cycles (component)	1,108 cycles (< 0.01% of total)
Number of runs	10 times

The cycle count per stage was derived using separate firmware builds, each timing one cryptographic step. The total number of cycles, 31,403,906, represents 99.994% of the total number of cycles of 31,405,906 for the overall handshake timing, with a difference of 0.006%. The remaining 2,000 cycles belong to the steps that are part of the integrated build only, including the second HKDF expansion step, transcript construction, and a bigger AES-CCM message size.

7.2. Analysis and Computational Costs

According to component-level measurements, the ECDH operation consists of ≈ 31,350,529 cycles (≈ 99.8% of the total amount), HKDF-SHA256 consumes 52,269 cycles (≈ 0.17%), and AES-128-CCM requires 1,108 cycles (less than 0.01%). This confirms that scalar multiplication on the elliptic curve, namely, key pair generation and shared secret calculation on the secp256r1 curve, is the most costly computation in terms of resources. Symmetric computations do not require much computational effort. This distribution is simply a result of the operation counts required: two key generations and two shared-secret computations need four secp256r1 scalar multiplications, where each one contains 256 point doublings and point additions on a 256-bit prime field using software, while HKDF-SHA256 has a few invocations of the compression function and AES-128-CCM works on just 128 bytes through a ten-round block cipher.

7.3. Latency Analysis

The execution time based on the number of cycles was approximately 491 ms, considering 31,405,906 cycles obtained at 64 MHz. The PPK2 measurement window of 447.2 ms is slightly shorter by 44 ms and is explained by the fact that the boundaries of the PPK2 measurement window were manually defined and were not strictly coincident with the DWT measurement window start/stop points. The handshake latency of approximately 491 ms is suitable for IoT devices where the secure session is established periodically, such as sensor nodes sending their measurements periodically.

7.4. Energy Consumption Analysis

Energy consumption of approximately 32.8 mJ was determined based on the formula E = V x Q, where V = 3.013 V and Q = 10.89 mC as provided by PPK2 in a 447.2 ms measurement window. PPK2 was used in the source meter mode with 3,013 mV supply voltage. Idle current during 3 seconds of sleep between handshake executions was excluded from the reported measurement. Based on the CPU cycle distribution, the largest part of energy consumption is related to the process of ECDH scalar multiplication. HKDF-SHA256 and AES-128-CCM provide insignificant energy overhead.

7.5. Footprint Analysis

This implementation uses up to 39 KB of Flash and 13 KB of RAM, which constitute about 3.8% and 5.1% of total Flash and RAM resources available on the nRF52840 MCU, respectively. This is the entire size of the firmware build with the Zephyr RTOS kernel, micro-ECC and TinyCrypt libraries, and the protocol logic itself. As one can see from Table 3, this footprint is less than reported values for

TinyDTLS-based implementations (60–120 KB Flash, 20–40 KB RAM) [5], as well as lower than those for EDHOC-based implementations (45–70 KB Flash, 10–25 KB RAM) [6].

7.6. Discussion

As follows from experimental measurements, the full ECDH + HKDF-SHA256 + AES-128-CCM handshake is able to take place on the nRF52840 chip in terms of 39 KB of Flash footprint (3.8%), 13 KB of RAM (5.1%), the latency of 491 ms, and energy consumption of 32.8 mJ. ECDH key exchange operation constitutes up to 99.8% of the total computational and energy costs, while AES-128-CCM-protected data exchange after handshake establishment imposes negligible energy costs.

Several limitations need to be mentioned. Firstly, tests have been conducted using only one platform; different hardware will yield different results, especially when it comes to hardware without built-in floating-point processing. Secondly, no wireless channel was used for profiling of the handshake; actual wireless transmission would incur channel latency and cost of retransmissions, which were not taken into account here. Finally, the protocol lacks a public key authentication procedure; a fully-fledged implementation will need to use some external means of authentication of public keys.

8. Comparison with Existing Work

8.1. Comparative Analysis

In Table 3, there is an indicative comparison between the proposed approach and selected existing lightweight IoT security protocols. These figures are ranges taken from previous literature studies for TinyDTLS [5], EDHOC [6], OSCORE [7], and ECC-based protocols [8, 9, 11]. The above figures were not achieved on the same hardware infrastructure, cryptographic setup, and experimental conditions. It is notable that the proposed protocol supports key agreement and authenticated encryption without supporting public-key authentication; this means that the scope of this protocol is different from that of EDHOC, where authenticated key exchange is supported.

Table 3. Indicative comparison with existing lightweight IoT security protocols (N/R = Not Reported)

Protocol	Security Mechanism	Platform	Flash / RAM (KB)	Energy (mJ)	Latency (ms)
TinyDTLS [5]	DTLS secure comm.	Cortex-M3/M4	60–120 / 20–40	60–120	600–1200
EDHOC [6]	Lightweight AKE	Embedded IoT	45–70 / 10–25	30–70	200–500
OSCORE [7]	App-layer security	Embedded IoT	50–100 / 15–30	N/R	N/R
ECC [8, 9, 11]	ECC key agreement	Sensor/IoT	N/R / N/R	N/R	ECC-dom.
This work	ECDH+HKDF+AES-128-CCM	nRF52840	39 / 13	32.8	≈491

This comparison is indicative since the presented results have been obtained on different platforms, protocol settings, cryptographic choices, and measurement techniques. Some works among existing studies do not provide the full set of implementation characteristics, emphasizing the necessity of hardware profiling.

8.2. Discussion of the Comparison

The footprint measurement of 39 KB Flash and 13 KB RAM is smaller than the reported TinyDTLS range (60-120 KB Flash, 20-40 KB RAM) [5] and is on the border of reported EDHOC realisations (45-70 KB Flash, 10-25 KB RAM) [6]. The reason for that difference is in the scope of the protocol rather than in the efficiency of the implementation.

TinyDTLS includes support for record layer state machine, fragmentation and retransmissions, cipher suite negotiation, and X.509 certificate processing. EDHOC protocol includes CBOR and COSE encodings along with credential management for key agreement. The protocol studied here includes none of that functionality: it just implements unauthenticated two-message key agreement with static parameters; thus, the firmware includes only ECC arithmetic, HKDF key derivation and AES-CCM encryption/decryption modes.

Energy can be assessed the same way. The measured 32.8 mJ falls in line with EDHOC's energy consumption (30-70 mJ), and is below TinyDTLS' energy consumption (60-120 mJ) [5, 6], and the per-stage analysis indicates the source of this value: 99.8% of energy spent during the handshake is due to a single operation, secp256r1 scalar multiplication performed in software. Protocols that also authenticate either certificate or signature do additional scalar multiplications whose energy consumption is roughly proportional to that number.

The latency of 491 ms is measured on the upper bound of the reported range for EDHOC (200-500 ms). It is expected, as no hardware acceleration was used: nRF52840 contains a CryptoCell-310 accelerator for cryptography, but this implementation performs all ECC arithmetic in software via micro-ECC library, while lower bound for EDHOC latency values is usually reported with accelerated implementation.

It is not a statement of superiority that constitutes the main value of this comparison. Rather, it is the fact that all five measures listed have been derived from a single image of firmware on a single platform under exactly the same circumstances, and the per-stage contribution of each of the operations was measured directly. The comparative measures

have been extracted from different sources, which makes them indicative rather than definitive.

8.3. Key Insights

Based on the comparison, the following key insights can be derived:

1. The proposed design requires 39 KB of Flash (3.8% of nRF52840 Flash), a reduction of 35–68% relative to reported TinyDTLS designs (60–120 KB) [5].
2. The RAM requirement of 13 KB (5.1% of nRF52840 RAM) falls within or below the range of existing lightweight protocol designs [5].
3. An energy consumption of 32.8 mJ in 447.2 ms falls within the range of EDHOC designs (30-70 mJ) [6] and is also lower than TinyDTLS (60-120 mJ) [5].
4. ECDH scalar multiplication represents 99.8% of the total CPU cycles of 31,405,906, whereas the combination of HKDF-SHA256 and AES-128-CCM makes up 0.2% (0.17% and less than 0.01% respectively). Any optimization of anything but scalar multiplication will not result in a meaningful reduction of the handshake cost.
5. The absence of public-key authentication accounts for part of the footprint advantage over EDHOC and TinyDTLS, and any future addition of certificate or signature handling would increase both memory and cycle cost.

9. Conclusion

In this paper, the implementation of a lightweight secure communication protocol using the cryptographic primitives such as ECDH secp256r1, HKDF-SHA256, and AES-128-CCM was presented. It is implemented on one nRF52840 Development Kit using Zephyr RTOS. Implementation of both sides was done sequentially on one board in a non-wireless manner. This study is not about inventing new cryptographic primitives but is about hardware profiling of the protocol used to achieve secure communication. Security verification was not within the scope of the work because it focuses on hardware characterization and does not contain public-key authentication.

The handshake process involved a total of 31,405,906 CPU cycles in ten successive attempts without any variations, equivalent to about 491 ms using 64 MHz. The energy consumption was determined to be about 32.8 mJ using an energy profiling window of 447.2 ms. The calculation involved the supply voltage of 3.013 V and a charge of 10.89 mC. The size taken by the firmware in the Flash memory and RAM memory of nRF52840 is 39 KB (3.8%) and 13 KB (5.1%), respectively. The analysis of the protocol per stage showed that 99.8% of the CPU cycles were used for ECDH scalar multiplication and 0.2% for HKDF-SHA256 and AES-128-CCM processes.

There are four limitations associated with this research. First, the measurements were conducted using one specific embedded platform without any wireless connectivity; the protocol performance may differ in other hardware environments or wireless setups. Second, the protocol does not feature any authentication process using public keys, which is necessary for the deployment of the protocol to prevent man-in-the-middle attacks. Third, only the handshake was timed; the costs of the AES-128-CCM-encrypted traffic were not calculated. Finally, per-stage timing was conducted using special firmware images where the workload of AES-128-CCM and HKDF-SHA256 was slightly smaller compared to the workload in the integrated handshake; therefore, the component figures are lower bounds than expected, but this is not important to identify scalar multiplication as the most costly part.

Future works should involve the use of hardware acceleration for the ECDH, considering that the nRF52840 features a cryptographic accelerator called CryptoCell-310 that was not used in the current implementation and which directly works with the cost percentage determined in this study (99.8%). Furthermore, public key authentication should be included in the protocol, as well as IEEE 802.15.4 wireless evaluation.

Author Contribution Statement

TALAFB conceived and designed the research, implemented the protocol, and performed the hardware measurements. LN and IM supervised the work and reviewed the manuscript. All authors read and approved the manuscript.

Conflict of Interest

The authors declare that they have no conflicts of interest.

Acknowledgment

The authors would like to express their sincere gratitude to the African Union (AU) for their invaluable support and commitment to advancing scientific research and technological innovation within the continent. This work was supported in part by the African Union Commission, which provided the necessary resources and platform to conduct this research.

References

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito, "The Internet of Things: A Survey," *Computer Networks*, vol. 54, no. 15, pp. 2787-2805, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[2] Ala Al-Fuqaha et al., "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[3] Gabriele Restuccia, Hannes Tschofenig, and Emmanuel Baccelli, "Low-Power IoT Communication Security: On the Performance of DTLS and TLS 1.3," *2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks (PEMWN)*, Berlin, Germany, pp. 1-6, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[4] Angelo Capossele et al., "Security as a CoAP Resource: An Optimized DTLS Implementation for the IoT," *2015 IEEE International Conference on Communications (ICC)*, London, UK, pp. 549-554, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[5] Shahid Raza et al., "Lithe: Lightweight Secure CoAP for the Internet of Things," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3711-3720, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[6] Rikard Höglund et al., "Secure Communication for the IoT: EDHOC and (Group) OSCORE Protocols," *IEEE Access*, vol. 12, pp. 49865-49877, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[7] Martin Gunnarsson et al., "Evaluating the Performance of the OSCORE Security Protocol in Constrained IoT Environments," *Internet of Things*, vol. 13, pp. 1-16, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[8] An Liu, and Peng Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, St. Louis, MO, USA, pp. 245-256, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[9] A.S. Wander et al., "Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks," *Third IEEE International Conference on Pervasive Computing and Communications*, Kauai, HI, USA, pp. 324-328, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[10] A. H. Al-Omari, "Lightweight Dynamic Crypto Algorithm for Next Internet Generation," *Engineering, Technology & Applied Science Research*, vol. 9, no. 3, pp. 4203-4208, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[11] U. Iftikhar et al., "Evaluating the Performance Parameters of Cryptographic Algorithms for IoT-based Devices," *Engineering, Technology & Applied Science Research*, vol. 11, no. 6, pp. 7867-7874, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[12] D.J. Malan, M. Welsh, and M.D. Smith, "A Public-Key Infrastructure for Key Distribution in TinyOS based on Elliptic Curve Cryptography," *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, Santa Clara, CA, USA, pp. 71-80, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[13] Simon Heron, "Advanced Encryption Standard (AES)," *Network Security*, vol. 2009, no. 12, pp. 8-12, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[14] Sharadadevi Kaganurmath, Nagaraj G. Cholli, and M. R. Anala, "DLKS-MQTT: A Lightweight Key Sharing Protocol for Secure IoT Communications," *Engineering, Technology & Applied Science Research*, vol. 15, no. 2, pp. 21532-21538, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[15] Rupali P. Shete, Anupkumar M. Bongale, and Deepak Dharrao, "Lightweight Cryptographic and Scalable IoT Systems for Encryption across GSM-MQTT Architectures in Resource-Constrained Aquaculture Environment," *Engineering, Technology & Applied Science Research*, vol. 15, no. 4, pp. 25133-25139, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[16] Hayder Yasir Naser et al., "A Comparison of Lightweight Cryptographic Protocols for Energy-Efficient and Sustainable IoMT Authentication," *Engineering, Technology & Applied Science Research*, vol. 15, no. 4, pp. 25746-25756, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[17] Batool Mohammed Radhi et al., "A Lightweight Identity Authentication Protocol for Nano-Scale IoT Devices," *Engineering, Technology & Applied Science Research*, vol. 15, no. 5, pp. 27938-27946, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[18] Waleed Khalid Al-Zubaidi et al., "Resilient IoT Authentication Against Physical Tampering and Side Channel Attacks," *Engineering, Technology & Applied Science Research*, vol. 15, no. 6, pp. 28787-28795, 2025. [CrossRef] [Google Scholar] [Publisher Link]