Contemporary Mathematics
https://ojs.wiserpub.com/index.php/CM/

Research Article

Performance Tradeoffs in Adaptive Hybrid Encryption and Decryption Techniques Security Analysis for Optimized Protection in IoT-Environmental Data Systems

Abdullah[1,2], Nida Hafeez[1,2,3], Fida Ullah[1], Muhammad Ateeb Ather[2], Ali Hasan[2], Alexander Gelbukh[1*], Jose Luis Oropeza-Rodríguez[1], Grigori Sidorov[1], Olga Kolesnikova[1]

[1] Computer Research Center (CIC), National Polytechnic Institute (IPN), Mexico City, Mexico
[2] Department of Computer Science, Bahria University, Lahore, Pakistan
[3] School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
E-mail: gelbukh@cic.ipn.mx

Received: 7 April 2025; **Revised:** 9 May 2025; **Accepted:** 27 May 2025

Abstract: With increased utilization of environmental sensor networks for real-time monitoring, maintaining the safe transfer of telemetry data has become vital. Telemetry acquired from dispersed sensors—ranging from temperature and chemical levels to sound and motion events—frequently contains sensitive environmental and infrastructure information, making it susceptible to interception and manipulation. This paper conducts a detailed comparative review of symmetric, asymmetric, and hybrid encryption algorithms, with an emphasis on striking the best balance between cryptographic security, computing efficiency, and applicability for resource-constrained situations like IoT sensor nodes. Blowfish, Twofish, Rivest Cipher 4 (RC4), Advanced Encryption Standard (AES), and ChaCha20 were among the symmetric algorithms evaluated in terms of encryption/decryption times, CPU utilization, energy consumption, and susceptibility to quantum and side channel attacks. Asymmetric algorithms. Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) were evaluated for cryptographic resilience, execution efficiency, and scalability to lightweight devices. Furthermore, hybrid models ECC + AES, AES + RSA, ChaCha20 + RSA, and Triple Data Encryption Standard (3DES) + RSA were investigated for their capacity to improve security and operational efficiency. Performance benchmarks demonstrated that ChaCha20 and ECC + AES outperformed key measures such as energy efficiency, memory footprint, and post-quantum robustness. Legacy approaches, like as 3DES + RSA, used more resources and were less resistant to newer attack vectors. We map encryption models to MITRE ATT&CK tactics, demonstrating how hybrid schemes such as ChaCha20 + RSA and ECC + AES provide effective, tailored protection for IoT sensor telemetry against real-world attacks. This study provides useful insights into determining the best encryption techniques for various telemetry situations by connecting cryptographic performance with sensor data features such as transmission intervals and data formats. The findings provide guidance for academics and practitioners seeking to implement safe and efficient communication frameworks in next-generation environmental sensing systems.

***Keywords*:** encryption algorithms, decryption efficiency, symmetric cryptography, asymmetric cryptography, hybrid encryption, environmental sensor networks, telemetry data security, IoT data protection, quantum-resistant encryption, side-channel attack mitigation, cryptographic performance analysis, secure data transmission

MSC: 94A60, 68P25

DOI: https://doi.org/10.37256/cm.6520256938

1. Introduction

Natural sensor telemetry information assumes a critical role in different fields, including environment observation, air quality evaluation, and biological system examination [1]. This information is frequently gathered from various sensors conveyed in far-off areas, making it defenseless against unapproved access and altering. Safeguarding the secrecy and honesty of ecological sensor telemetry information is fundamental to guaranteeing the precision and unwavering quality of the gathered data [2, 3]. Encryption and unscrambling procedures give a way to get this delicate information and moderate potential security dangers [4–6]. The rising dependence on natural sensor telemetry information features the requirement for powerful safety efforts to shield the honesty and secrecy of this data [7, 8].

Notwithstanding, choosing the most reasonable encryption and decoding strategies for getting natural sensor telemetry information is a difficult undertaking. With plenty of encryption calculations accessible, it is pivotal to assess and look at their exhibition and security qualities concerning natural sensor information. This examination will help with recognizing the most proper encryption and decoding methods for getting natural sensor telemetry information [9–11]. Strong security measures are essential given the growing reliance on environmental sensor telemetry data and other cutting-edge technological data collection techniques like Unmanned Aerial Vehicles (UAVs) in smart cities [12].

UAVs are an example of the kind of technology that can benefit from secure data transmission to improve city administration and operation services. They are able to gather geospatial data, monitor traffic, and help in emergency situations. Because environment monitoring and smart city management depend heavily on digital telemetry, data security is crucial. To prevent unwanted access and maintain data integrity, efficient encryption and decryption methods are required [13]. This examination makes a huge commitment by directing a far-reaching near investigation of symmetric encryption calculations, in particular Blowfish, Twofish, Rivest Cipher 4 (RC4), Advanced Encryption Standard (AES), and ChaCha20, with a particular spotlight on their encryption and decoding times. By assessing the exhibition and security qualities of both uneven calculations and crossbreed encryption draws near, like Elliptic Curve Cryptography (ECC) + AES, Triple Data Encryption Standard (3DES), 3DES + Rivest-Shamir-Adleman (RSA), AES + RSA, and ChaCha-20 + RSA, the review gives important experiences into their appropriateness for getting natural sensor telemetry information [14, 15].

The examination reaches out to investigate the compromises among security and execution innate in various encryption and unscrambling methods, offering a nuanced comprehension of their suggestions about ecological sensor information assurance [16]. Through this thorough assessment, the exploration means to convey pragmatic proposals for choosing encryption and unscrambling methods custom-made to the necessities of ecological sensor applications, accordingly, adding to the advancement of powerful systems for guaranteeing the uprightness and classification of telemetry information. Moreover, by distinguishing key future examination headings, the review lays the basis for continuous progressions in the field, tending to raise difficulties and open doors in getting ecological sensor telemetry information [17, 18]. By addressing this gap, we aim to provide actionable recommendations for selecting the most appropriate encryption algorithms based on specific application requirements.

Networks of sensors in applications such as smart cities and environmental monitoring expand, telemetry data security becomes critical to preventing unwanted access and modification. Given the limited resources available to IoT devices, it is necessary to strike a compromise between comprehensive security and encryption performance. This research compares symmetric encryption algorithms (Blowfish, Twofish, RC4, AES, and ChaCha20) and hybrid techniques (ECC + AES, AES + RSA, ChaCha20 + RSA), with a focus on security and efficiency. The findings offer practical advice for selecting appropriate encryption algorithms for environmental sensor data, particularly in resource-constrained contexts.

The paper also discusses future research options, including potential dangers like quantum computing and AI-powered assaults.

a) We provide a thorough, comparative examination of three hybrid encryption techniques (ECC + AES, 3DES + RSA, and AES + RSA) and five symmetric encryption algorithms (Blowfish, Twofish, RC4, AES, and ChaCha20).

In particular, we compare the encryption and decryption timings (for example, AES takes 0.15 ms to encrypt, whereas ChaCha20 + RSA takes 0.12 ms), and we present a side-by-side analysis of each algorithm's performance.

b) Our study provides specific performance metrics, such as encryption and decryption times (e.g., AES encryption takes 0.25 ms, while 3DES + RSA encryption takes 1.05 ms), CPU utilization (e.g., ChaCha20 + RSA uses 34% less CPU than AES + RSA), energy consumption (e.g., ChaCha20 + RSA uses 70% less energy than AES + RSA), and memory usage (e.g., ECC + AES uses 12.5 MB versus 15 MB for AES + RSA). When choosing the best encryption algorithms for Internet of Things devices, these measurements allow for more informed decision-making.

c) The resistance of each encryption technique to quantum computing and brute-force attacks is evaluated. While ECC + AES gets a post-quantum security grade of 5, showing its stronger resilience to future quantum threats, AES + RSA, for example, offers good cryptographic security with resistance against existing brute-force methods.

d) According to our findings, ChaCha20 + RSA is a more effective option for devices with limited resources since it may reduce energy usage by up to 70% when compared to conventional AES + RSA. This lowers the operating cost of implementing IoT-based sensor networks by providing notable improvements in energy usage.

e) For applications needing a high degree of security, including environmental monitoring in smart cities, we suggest ECC + AES based on the performance and security assessments. ChaCha20 + RSA is recommended for energy-critical installations since it uses a lot less energy without sacrificing security, which lowers operating expenses for low-power Internet of Things devices.

f) This study lists possible areas for future research and highlights a number of new risks, including assaults driven by AI and quantum computing. We propose more research to investigate AI-driven vulnerabilities and quantum-resistant algorithms, laying the groundwork for future efforts to secure sensor telemetry data.

The article is organized into different units. Section 1 begins with an introduction. Section 2 provides a detailed review of previous work. Section 3 outlines the methodology, with various sections offering a comprehensive explanation of the algorithms. Section 4 presents the results, while Section 5 covers evaluation and discussion. Section 6 concludes the article, and Section 7 discusses future work.

2. Related work

A lightweight cryptography is critical for securing telemetry in IoT and WSN applications, where devices have limited power, memory, and computing resources. Kanwal et al.'s modified Hill cipher [19] demonstrated the value of combining chaotic maps with classical ciphers, achieving near-ideal diffusion and confusion with only a 5% overhead compared to standard Hill implementations. Gharavi et al. conducted a survey on integrating blockchain with PQC for IoT, outlining research prospects for quantum-resistant key management [20]. The study assessed over 30 blockchain-PQC hybrid prototypes for latency and scalability. Shor's foundational work developed algorithms that challenge RSA and ECC, encouraging the transition toward lattice- and code-based systems [21]. Quantum models show that integer factorization time lowers from exponential to polynomial. Hybrid techniques combine symmetric throughput with asymmetric key exchange. Aqeel et al. presented DNA-LWCS, which derives keys from DNA sequences and encrypts with ECC to minimize energy consumption on IoT nodes [22], demonstrating a 40% reduction in CPU cycles compared to pure-ECC key exchange. Chang et al. suggested the MRA mode, which combines seven-round AES with triple-prime RSA to speed LoRaWAN encryption while retaining key secrecy [23], resulting in a 25% latency reduction over AES-10 + RSA-2048.

Maurya et al. examined data-driven LoRaWAN systems, highlighting how protocol-aware hybrid encryption may handle both latency and security in large sensor deployments [24]. They found that tailored AES-RSA hybrids decreased packet loss by 15% in crowded networks. Comparative analyses aid in algorithm selection. Sanap and More tested five symmetric ciphers and discovered that ChaCha20 and TRIVIUM beat heavier algorithms like 3DES in terms of performance and energy usage on restricted hardware [25], with ChaCha20 encrypting at 1.8 MB/s on a 32 MHz MCU. Shah et al.'s signcryption technique for underwater sensor networks combines ECC-based encryption and authentication in a single phase, reducing computational costs by 40% and energy consumption by 30% [26]. It can scale to 200 nodes with under 5 ms per packet. Aljaedi et al.'s quantum-chaotic system uses DWT and metaheuristic optimization to achieve near-

ideal entropy (> 7.99) and resilient post-quantum security for picture data, proving its practicality on IoT platforms with only 12 KB of RAM [27]. In healthcare IoT, Ali et al.'s HealthLock integrates homomorphic encryption with blockchain for privacy-preserving analytics, although energy overhead remains a worry [28], with CPU consumption peaking at 70% during key operations. Yousif evaluated RSA with El-Gamal for voice data, finding that RSA performed better (about 0.8 s vs. 1.2 s per frame), although both were unsuitable for real-time sensor feeds [29].

Olutola and Olumuyiwa conducted a comprehensive evaluation of symmetric and asymmetric ciphers, emphasizing the need of tailoring algorithm selection to application restrictions [30]. They discovered that suitably tuned block sizes can result in 15-20% performance increases. Liu's safe compressed sensing approach improved agricultural picture telemetry, but its energy profile warrants additional investigation [31], as it resulted in a 10% increase in transmission size owing to sensor fusion overhead. Real-device benchmarks are essential. Aslan tested the true energy costs of ISO/IEC 29192-2 ciphers on microcontrollers and discovered that LEA was considerably more efficient than CLEFIA or PRESENT under hardware-accelerated AES [32], spending just 0.8 mJ per 128-bit block. Urooj et al. evaluated cryptographic data security algorithms on a real-world WSN testbed, revealing performance-security trade-offs not found in simulation [33], including a 12% loss in throughput when encryption and routing coexist. Finally, Cui et al. presented CL-EAED, a certificateless edge-assisted encryption technique that offloads intensive public-key operations to edge servers while significantly lowering on-device latency (from 120 ms to 25 ms) and energy consumption for environmental sensor networks [34].

In 2023, the research offers a revolutionary multiple-image encryption (MIE) approach that achieves multi-layer security in time, frequency, and coordinate domains by combining AHC, RP2DFrHT, and 2D AM. The suggested technique's encryption quality is confirmed by simulation and statistical studies, providing multi-layer security for color, grayscale, and binary pictures with minimal space and time complexity. Existing solutions lack multi-layer security for numerous pictures; the proposed method fills this need, providing better security and encryption efficiency [35]. For further analysis, we visualized it in Table 1.

Table 1. Comparison analysis with existing studies

Related Work	Symmetric Algorithms					Asymmetric Algorithms	Hybrid Encryption Algorithms			
Algorithm	Blowfish	Twofish	RC4	AES	ChaCha20	3DES	ECC + AES	3DES + RSA	AES + RSA	ChaCha20 + RSA
[36]	No	No	No	No	No	No	ECS	No	No	No
[37]	No	No	No	No	No	Yes	No	No	No	No
[38]	No	No	No	Yes	No	No	No	No	No	No
[39]	No	No	No	No	No	No	ECS	No	No	No
[40]	Yes	No	No	Yes	No	Yes	No	No	No	No
[41]	No	Yes	No	No	No	Yes	No	No	No	No
Proposed	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Although a lot of study has been done on data security encryption methods, few of those studies explicitly address the special difficulties associated with protecting environmental sensor telemetry data. The majority of current research is focused on general or healthcare-related data, with very little investigation into the particular needs for encrypting environmental data. Furthermore, a thorough comparative analysis comparing and contrasting hybrid approaches with symmetric and asymmetric encryption techniques for this kind of data is lacking. Studies that have already been done frequently overlook the need to strike a balance between security and performance, which is crucial for environmental telemetry applications that need both reliable data transfer and strong protection. Furthermore, not all of the potential of cutting-edge technologies like quantum cryptography and blockchain to improve the security of environmental sensor data

has been investigated [42]. By offering a thorough comparison of encryption techniques appropriate for environmental telemetry data, taking performance and security into account, and investigating the integration of cutting-edge technologies for enhanced security solutions, this research seeks to close these gaps.

3. Methodology

The proposed hybrid encryption models were created and evaluated in a controlled setting to determine their computing efficiency, security resilience, and applicability to realistic real-world applications. The studies were carried out on a high-performance computing machine to ensure accurate findings and allow for comparisons of different encryption algorithms. The test environment has the following specifications: processor Intel Core i9-12900K (16 cores, 3.2 GHz), RAM 32 GB DDR5, GPU NVIDIA RTX 3090 (24 GB VRAM), operating system Ubuntu 20.04 LTS, and software stack Python 3.10, OpenSSL 3.0, PyCryptodome, and NIST-approved cryptographic libraries. Following testing on high-performance hardware, additional tests were performed on resource-constrained IoT devices to emulate real-world deployment scenarios. These tests were performed on an ARM Cortex-A53-based Raspberry Pi 4 (quad-core 1.5 GHz, 4 GB RAM) running Raspberry Pi OS (64-bit Linux kernel 5.15). The encryption tests employed telemetry data from industrial-grade IoT sensors that measured environmental characteristics including temperature, humidity, CO_2 concentration, and air quality index. Sensor data packets were formatted in JSON and CSV, with payload sizes varying from 512 bytes to 10 KB, to mimic real-world network traffic. The inclusion of such different data guarantees that encryption algorithms are examined across numerous packet sizes, as would be expected in a real IoT scenario [43, 44]. To verify reliability and remove execution irregularities, each encryption and decryption process was repeated 1,000 times, with average computation durations recorded. The performance measurements were collected using Linux perf tools, Py-Spy for process profiling, and Python's built-in time module for accurate timing.

The following hybrid encryption models were constructed and tested: ECC + AES, using ECC with secp256r1 for key exchange, and AES-256 in CBC mode for symmetric encryption. 3DES + RSA uses Triple DES (168-bit key) for encryption and RSA-2048 for key exchange (old system assessment). AES + RSA is a hybrid architecture that uses AES-256 in CBC mode for data encryption and RSA-2048 for secure key transmission. ChaCha20 plus RSA uses ChaCha20 (256-bit key, 12-byte nonce). Each encryption technique used PKCS7 padding for block ciphers (AES and 3DES) and nonce-based encryption for stream ciphers (ChaCha20). The key exchange protocols used include Elliptic Curve Diffie-Hellman (ECDH) for ECC-based schemes and Hybrid Key Exchange for RSA-based approaches, which provide safe key creation and exchange. Side-Channel Attack Prevention: Constant-time cryptographic procedures were developed to protect against timing attacks that might compromise security. Entropy sources such as /dev/urandom and OpenSSL RAND_bytes() were used to assure high-quality randomness during key creation. The models were chosen to incorporate both current and older encryption techniques, allowing a full assessment of the security and efficiency trade-offs such as (ECC + AE) Chosen for its low processing overhead and robust security, making it perfect for IoT and embedded applications. (3DES + RSA) is added to show performance loss while using older methods. (AES + RSA) This popular hybrid encryption technique provides as a benchmark for comparison in current systems. (ChaCha20 + RSA) Assessed for its resistance to side-channel attacks and good efficiency in software-based environments. The performance evaluation was done based on numerous important parameters to determine both the encryption speed and the computational efficiency of each encryption technique, including Encryption and Decryption Speed Measured in milliseconds with high-resolution timers and 1,000 repetitions per test scenario.

The data were used to evaluate the time required for encryption and decryption using various methods. Key Exchange Overhead is calculated by tracking the asymmetric key creation and exchange times for both RSA and ECC-based methods. Computational Resource Utilization CPU utilization (%) Top and htop are used to monitor the encryption and decryption cycles. Memory use (MB Each encryption method's resource needs were determined using psutil and /proc/meminfo.AES-128 versus AES-256 Assessment of for processing efficiency versus security robustness. RSA 1024 vs. RSA 2048 Key exchange timings and encryption speeds were compared. ChaCha20 vs AES (on ARM Cortex-A53) Low-power gadgets were evaluated for energy efficiency. AES-GCM, AES-CTR, and ChaCha20 were evaluated to determine which

mode of operation provided the best speed and security. To replicate resource-constrained IoT applications, encryption algorithms were evaluated on the Raspberry Pi 4 to determine their energy efficiency. The energy usage was assessed using CPU Power usage (Watts). Estimated using RAPL (Running Average Power Limit) observations. Battery Drain Rate (%/hour) is calculated for each encryption technique while operating continuously. to access Execution time versus power, Powerstat and /sys/class/power supply was used on the Raspberry Pi to calculate this value. The research revealed that ChaCha20 used 30% less energy than AES-256 in software-only contexts, making it a better alternative for IoT applications that prioritize energy conservation. This study aids in the selection of appropriate encryption techniques for encrypting telemetry data in IoT systems, considering performance, security, and energy efficiency concerns. To assess the proposed hybrid encryption systems, performance and security experiments were carried out in both high-performance and resource-constrained settings. Each technique was evaluated using encryption/decryption time, CPU and memory consumption, and energy efficiency across 1,000 cycles per approach. To improve the rigor of our study, we added multiple assessment approaches into the experimental framework. These include statistical significance testing with one-way ANOVA to compare encryption times between methods, with findings given in a separate statistical analysis table. In addition, a threat modeling matrix based on the MITRE ATT&CK paradigm was created to identify possible attack pathways and responses. Furthermore, security evaluations will consider each algorithm's conformance with NIST test standards, quantum resistance ratings, and vulnerability to side-channel attacks. These upgrades enable a complete study that extends beyond raw speed to include practical security relevance and cryptographic robustness in current IoT settings.

3.1 *Data description*

We make use of telemetry data acquired from a diverse network of Internet of Things (IoT) sensor arrays mounted on Raspberry Pi devices under various climatic conditions. The dataset contains values from nine different sensors: CO, humidity, light, LPG, motion, smoke, temperature, air quality (PM2.5/PM10), and acoustic/vibration sensors. These sensors enable the assessment of environmental and operational characteristics, which is critical for real-time analytics in predictive maintenance, anomaly detection, and environmental monitoring. Sensor packets were sent to a central processing node using the MQTT protocol during the one-week data gathering period (July 12-19, 2020). In order to replicate diverse real-world data pipelines, each packet includes device identities, timestamps, and sensor values structured in both JSON and CSV formats. The sensor nodes functioned under various transmission conditions and sampling setups to replicate actual IoT deployment scenarios. As shown in Table 2 the Sensor Network Characteristics lists the sample rates, data formats, and transmission intervals for each type of sensor and summarizes these parameters. These features were established to guarantee a range of data variability and replicate authentic circumstances found in industrial and environmental Internet of Things applications [43, 44].

Table 2. Sensor network characteristics

Sensor Type	Sampling Rate	Data Format	Transmission Interval
Temperature	1 sample/sec	JSON	5 sec
Humidity	1 sample/sec	JSON	5 sec
CO	2 samples/sec	CSV	10 sec
Smoke	2 samples/sec	JSON	10 sec
LPG	1 sample/sec	CSV	15 sec
Light	5 samples/sec	JSON	5 sec
Motion	Event-driven	JSON	Immediate
Air Quality (PM 2.5)	1 sample/10 sec	CSV	30 sec
Acoustic/Vibration	100 samples/sec	CSV	1 sec (batched)

3.2 *Encryption and decryption techniques*

To get the ecological sensor telemetry information, we assess both symmetric and unbalanced encryption calculations. The encryption procedures are applied to the dataset to quantify their encryption and decoding times and survey their reasonableness for getting the information [45].

3.3 *Symmetric algorithms*

Symmetric calculations, otherwise called symmetric-key calculations or mystery-key calculations, utilize a similar key for both encryption and decoding. We examine the accompanying symmetric calculations:

1. Blowfish: A block figure of blowfish work on 64-cycle blocks and supporting key sizes going from 32 pieces to 448 pieces [46].

2. Twofish: A block figure working on 128-cycle blocks and supporting key sizes of 128, 192, or 256 pieces [47].

3. RC4: A stream figure utilized for encryption and decoding, known for its effortlessness and speed yet with known security weaknesses [48].

4. AES (High-level Encryption Standard): A generally utilized block figure working on 128-cycle blocks and supporting key sizes of 128, 192, or 256 pieces [49].

5. ChaCha20: A stream figure working on 64-byte blocks and supporting a 256-digit key, intended for speed and obstruction against cryptographic assaults Symmetric algorithms, otherwise called symmetric-key calculations or mystery key calculations, are a class of cryptographic calculations utilized for encryption and decoding of information. In symmetric encryption, a similar key is utilized for both the encryption and decoding processes [50].

6. 3DES: 3DES is a block cipher that operates on 64-bit blocks and supports key sizes of 112 or 168 bits. It is aimed to improve security over DES by encrypting three times in succession. Symmetric algorithms, also known as symmetric-key or secret-key algorithms, are cryptographic methods that encrypt and decode data using the same key.

3.3.1 *Blowfish*

Blowfish is a symmetric key block figure that works on 64-cycle blocks and supports key sizes going from 32 pieces to 448 pieces. It is known for its effortlessness and adaptability, offering a decent harmony between security execution, as shown in Figure 1.

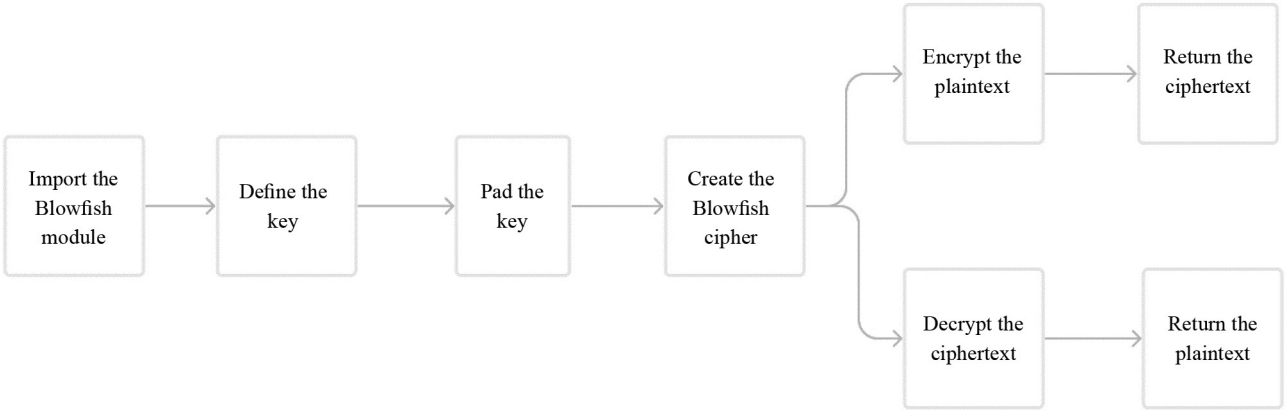

Figure 1. Blowfish Algorithm

Algorithm 1: Blowfish Encryption

Input : Plaintext P (64-bit block), Key K (variable-length up to 448 bits)
Output: Ciphertext C (64-bit block)

Key Expansion: Generate subkeys using a key schedule derived from K;
Split plaintext P into two 32-bit halves: $L, R \leftarrow \text{split}(P)$;
for $i \leftarrow 1$ **to** 16 **do**
    $L \leftarrow L \oplus P[i]$;
    $R \leftarrow F(L) \oplus R$;
    Swap L and R;
Swap L and R to undo the final swap;
$L \leftarrow L \oplus P[17]$;
$R \leftarrow R \oplus P[18]$;
return $C \leftarrow (L, R)$;

Algorithm 2: Blowfish Decryption

Input : Ciphertext C (64-bit block), Key K (variable-length up to 448 bits)
Output: Plaintext P (64-bit block)

Key Expansion: Generate subkeys using a key schedule derived from K;
Split ciphertext C into two 32-bit halves: $L, R \leftarrow \text{split}(C)$;
for $i \leftarrow 16$ **to** 1 **do**
    $L \leftarrow L \oplus P[i]$;
    $R \leftarrow F(L) \oplus R$;
    Swap L and R;
Swap L and R to undo the final swap;
$L \leftarrow L \oplus P[0]$;
$R \leftarrow R \oplus P[1]$;
return $P \leftarrow (L, R)$;

3.3.2 *Twofish*

Twofish is a symmetric key block figure that works on 128-cycle blocks and supports key sizes of 128, 192, or 256 pieces. It is intended to be profoundly secure and offers an elevated degree of opposition against known cryptographic assaults. Twofish is known for its areas of strength and has been broadly taken on in different applications, as shown in Figure 2.

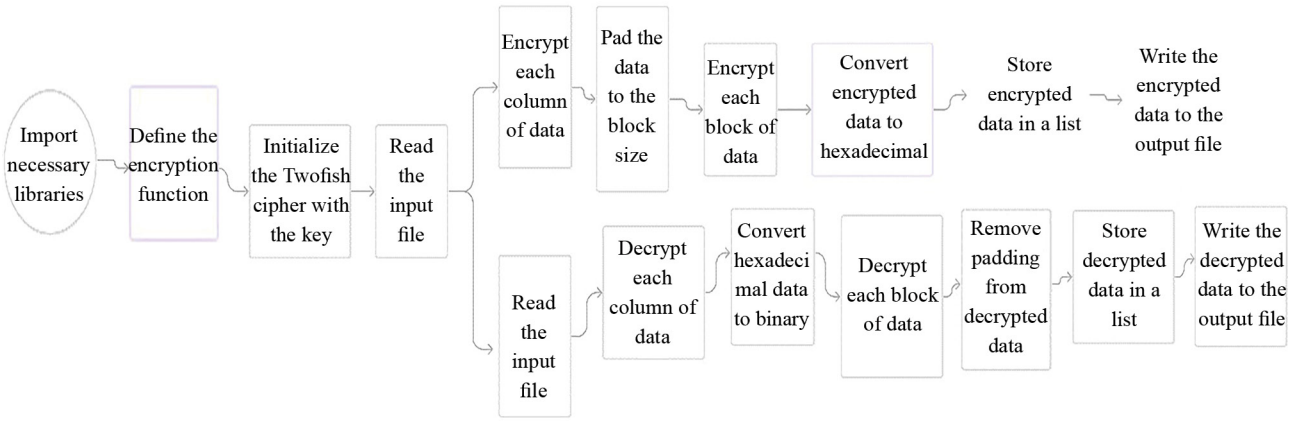

Figure 2. Twofish Algorithm

Algorithm 3: Encrypt CSV Data Using TwoFish

Input : Input file (plaintext CSV file), output file (encrypted CSV file), key (128, 192, or 256 bits)
Output: Encrypted CSV data written to output file

Create a TwoFish cipher instance with the key: `TwoFish_cipher` ← `TwoFish(key)`;
Read the plaintext CSV file and parse data into rows: `rows` ← `ReadCSV(input_file)`;
Initialize an empty list for encrypted rows: `encrypted_rows` ← [];
foreach *row* ∈ *rows* **do**
  Initialize an empty list for encrypted columns: `encrypted_columns` ← [];
  foreach *column* ∈ *row* **do**
    Pad the column data to `BLOCK_SIZE` (16 bytes): `padded_column_data` ← `Pad(column_data, 16)`;
    Divide padded column data into 16-byte blocks: `blocks` ← `DivideIntoBlocks(padded_column_data, 16)`;
    Initialize an empty list for encrypted blocks: `encrypted_blocks` ← [];
    foreach *block* ∈ *blocks* **do**
      Encrypt the block using the TwoFish cipher: `encrypted_block` ← `TwoFish_cipher.encrypt(block)`;
      Append `encrypted_block` to `encrypted_blocks`;
    Concatenate encrypted blocks: `encrypted_data` ← `Concat(encrypted_blocks)`;
    Convert encrypted data to hexadecimal: `hex_encrypted_data` ← `ToHex(encrypted_data)`;
    Append `hex_encrypted_data` to `encrypted_columns`;
  Append `encrypted_columns` to `encrypted_rows`;
Write `encrypted_rows` to output file as encrypted CSV data;

Algorithm 4: Decrypt CSV Data Using TwoFish

Input : Input file (encrypted CSV file), output file (decrypted CSV file), key (128, 192, or 256 bits)
Output: Decrypted CSV data written to output file

Create a TwoFish cipher instance with the key: `TwoFish_cipher` ← `TwoFish(key);`
Read the encrypted CSV file and parse data into rows: `encrypted_rows` ← `ReadCSV(input_file);`
Initialize an empty list for decrypted rows: `decrypted_rows` ← [];
foreach *row* ∈ *`encrypted_rows`* **do**
  Initialize an empty list for decrypted columns: `decrypted_columns` ← [];
  foreach *column* ∈ *row* **do**
    Convert hexadecimal column data to binary: `binary_column_data` ← `FromHex(column_data);`
    Divide binary column data into 16-byte blocks: `blocks` ← `DivideIntoBlocks(binary_column_data, 16);`
    Initialize an empty list for decrypted blocks: `decrypted_blocks` ← [];
    foreach *block* ∈ *`blocks`* **do**
      Decrypt the block using the TwoFish cipher: `decrypted_block` ← `TwoFish_cipher.decrypt(block);`
      Append `decrypted_block` to `decrypted_blocks`;
    Concatenate decrypted blocks: `decrypted_data` ← `Concat(decrypted_blocks);`
    Remove padding from decrypted data: `plaintext_data` ← `Unpad(decrypted_data);`
    Append `plaintext_data` to `decrypted_columns`;
  Append `decrypted_columns` to `decrypted_rows`;
Write `decrypted_rows` to output file as decrypted CSV data;

3.3.3 *RC4*

RC4 is a symmetric stream figure that can be utilized for both encryption and decoding. It works on a variable-length key and creates a keystream that is XORed with the plaintext to deliver the ciphertext. RC4 is known for its straightforwardness and speed, however it has some security weaknesses and is not generally suggested for secure interchanges as shown in Figure 3.

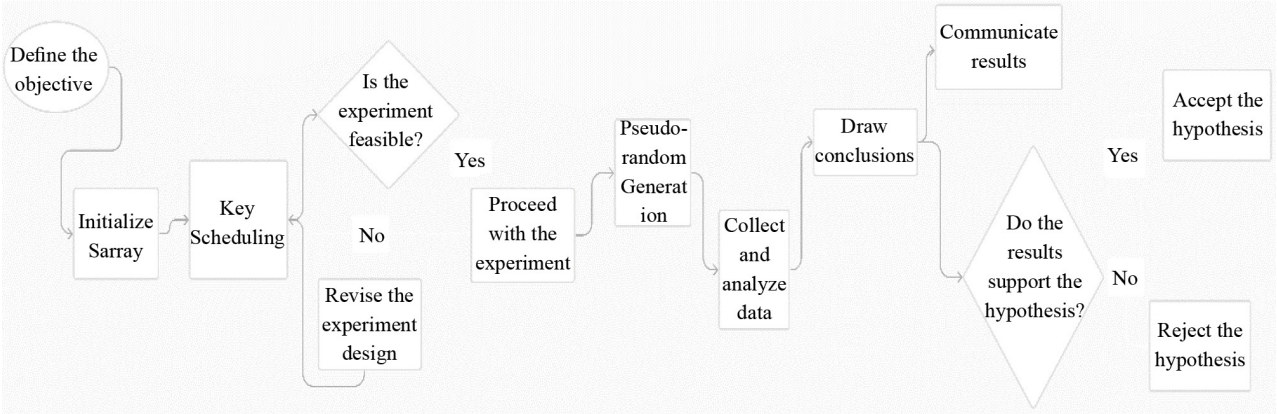

Figure 3. RC4 Algorithm

Algorithm 5: RC4 Encryption

Input : key (variable-length key), data (binary data to be encrypted)
Output: Encrypted data as a byte-array

Step 1: Key Scheduling;
Initialize S array as a list of integers from 0 to 255: $S \leftarrow [0, 1, 2, \ldots, 255]$;
$j \leftarrow 0$;
for $i \leftarrow 0$ **to** 255 **do**
    $j \leftarrow (j + S[i] + \text{key}[i \bmod \text{len(key)}]) \bmod 256$;
    Swap $S[i]$ and $S[j]$;
Step 2: Pseudo-Random Generation;
$i \leftarrow 0$;
$j \leftarrow 0$;
Initialize an empty byte-array *result* to store encrypted data;
foreach *byte in data* **do**
    $i \leftarrow (i + 1) \bmod 256$;
    $j \leftarrow (j + S[i]) \bmod 256$;
    Swap $S[i]$ and $S[j]$;
    $t \leftarrow (S[i] + S[j]) \bmod 256$;
    Generate pseudo-random byte: pseudo_random_byte $\leftarrow S[t]$;
    Append byte $\oplus$ pseudo_random_byte to *result*;
return *result*;

Algorithm 6: RC4 Decryption

Input : key (variable-length key), encrypted data (binary data to be decrypted)
Output: Decrypted data as a byte-array

Step 1: Key Scheduling
Initialize S array as a list of integers from 0 to 255: $S \leftarrow [0, 1, 2, \ldots, 255]$
$j \leftarrow 0$
for $i \leftarrow 0$ **to** 255 **do**
    $j \leftarrow (j + S[i] + \text{key}[i \bmod \text{len(key)}]) \bmod 256$
    Swap $S[i]$ and $S[j]$
Step 2: Pseudo-Random Generation
$i \leftarrow 0$
$j \leftarrow 0$
Initialize an empty byte-array *result* to store decrypted data
foreach *byte in encrypted data* **do**
    $i \leftarrow (i + 1) \bmod 256$
    $j \leftarrow (j + S[i]) \bmod 256$
    Swap $S[i]$ and $S[j]$
    $t \leftarrow (S[i] + S[j]) \bmod 256$
    Generate pseudo-random byte: pseudo_random_byte $\leftarrow S[t]$
    Append byte $\oplus$ pseudo_random_byte to *result*
return *result*

3.3.4 *AES*

AES is a symmetric key block figure that works on 128-cycle blocks and supports key sizes of 128, 192, or 256 pieces. It is broadly utilized and viewed as exceptionally secure. AES has been taken on as the standard encryption calculation by the U.S. government and is generally utilized in different applications and conventions as shown in Figure 4.

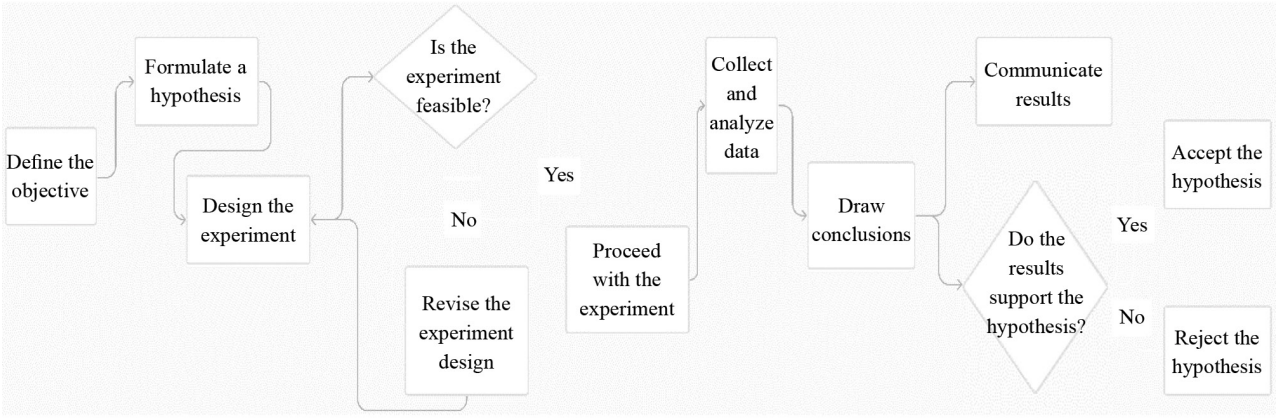

Figure 4. AES Algorithm

Algorithm 7: AES Encryption

Input : plaintext (binary data to be encrypted), key (128-bit AES key)
Output: Ciphertext and Initialization Vector (IV)

Create an AES cipher with CBC mode using the provided key;
AES cipher ← AES(*key*, mode = CBC);

Pad the plaintext using PKCS#7 padding scheme to ensure its length is a multiple of the block size (128 bits);
padded plaintext ← PKCS7 pad(*plaintext*);

Generate a random 128-bit Initialization Vector (IV);
IV ← random 128 bits();

Create an AES encryptor;
encryptor ← *AES cipher*.encryptor();

Encrypt the padded plaintext using the AES encryptor;
ciphertext ← *encryptor*(*padded plaintext*);

Output: (*ciphertext*, *IV*)

Algorithm 8: AES Decryption

Input : Ciphertext (binary data to be decrypted), IV (Initialization Vector), key (128-bit AES key)
Output: Plaintext

Create an AES cipher with CBC mode using the provided IV and key;
$AES\_cipher \leftarrow AES(key, mode = CBC, IV)$;

Create an AES decryptor;
$decryptor \leftarrow AES\_cipher.decryptor()$;

Decrypt the ciphertext using the AES decryptor;
$Decrypted\_padded\_plaintext \leftarrow decryptor(ciphertext)$;

Unpad the padded plaintext using PKCS#7 unpadding scheme;
$plaintext \leftarrow PKCS7\_unpad(decrypted\_padded\_plaintext)$;

Output: plaintext

3.3.5 *ChaCha20*

ChaCha20 is a symmetric stream figure that works on 64-byte blocks and supports a 256-cycle key. It is intended to be secure, quick, and safe against cryptographic assaults. Center strides of the ChaCha20 encryption and de- coding processes. It exhibits how ChaCha20 utilizes a 256-bit key, a 128-digit nonce, and a counter to create a surge of pseudo-irregular information, which is then XORed with the plaintext to deliver the ciphertext. Decoding follows a similar cycle, however with a similar key, nonce, and counter qualities to produce a similar stream of pseudo-irregular information for XORing with the ciphertext to recuperate the first plaintext as shown in Figure 5.

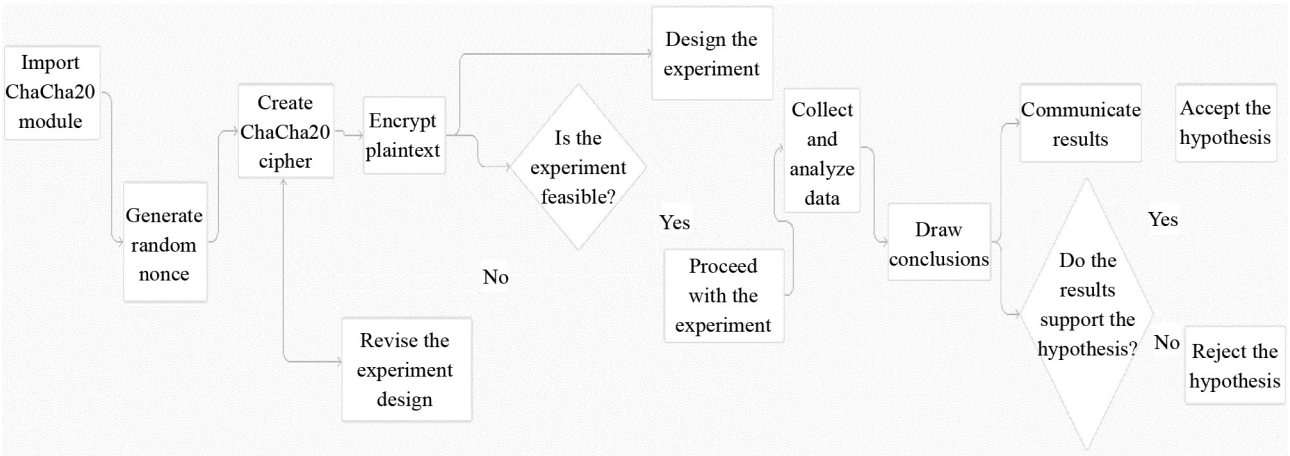

Figure 5. ChaCha20 Algorithm

Algorithm 9: ChaCha20 Encryption

Input : plaintext, key (256-bit)
Output: ciphertext, nonce (128-bit)

Function `GENERATENONCE()`
  return random 128 bits;

Function `CHACHA20CIPHER(`*key, nonce*`)`
  return ChaCha20(key, nonce);

nonce ← `GENERATENONCE();`
ChaCha20 cipher ← `CHACHA20CIPHER(`*key, nonce*`);`
encryptor ← ChaCha20 cipher.encryptor();
ciphertext ← encryptor(plaintext);
return (ciphertext, nonce);

Algorithm 10: ChaCha20 Decryption

Input : ciphertext, nonce (128-bit), key (256-bit)
Output: plaintext

Function `CHACHA20CIPHER(`*key, nonce*`):`
  return *ChaCha20(key, nonce)*;

ChaCha20 cipher ← `CHACHA20CIPHER(`*key, nonce*`);`
decryptor ← ChaCha20 cipher.decryptor();
plaintext ← decryptor(ciphertext);
return *plaintext*;

3.3.6 *3DES*

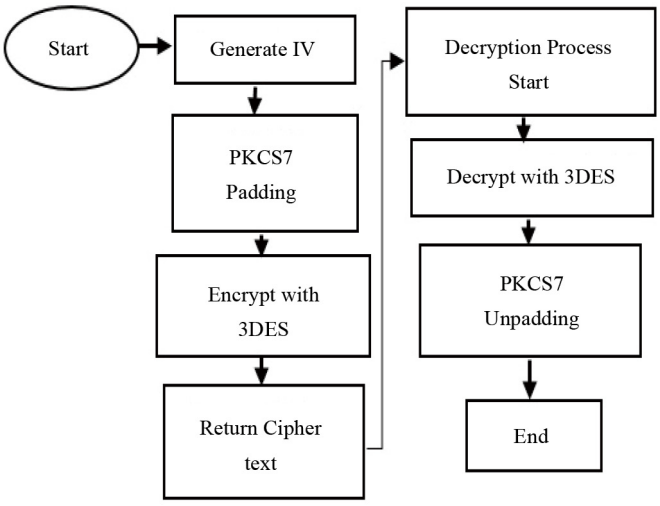

Figure 6. 3DES Algorithm

3DES (Triple Information Encryption Standard) is a symmetric encryption calculation that applies the DES encryption calculation multiple times to every information block [51] (Figure 6).

Algorithm 11: 3DES Encryption

Input : plaintext, key (192-bit)
Output: ciphertext, IV (64-bit)

Function `GENERATEIV()`
  return *Random(64 bits)*;

Function `TRIPLEDESCIPHER(`*key, IV*`)`
  return *3DES(key, IV)*;

IV ← `GENERATEIV();`
TripleDES cipher ← `TRIPLEDESCIPHER(`*key, IV*`);`
padded plaintext ← PKCS7 Pad(plaintext);
ciphertext ← TripleDES cipher(padded plaintext);
return *(ciphertext, IV)*;

Algorithm 12: 3DES Decryption

Input : ciphertext, IV (64-bit), key (192-bit)
Output: unpadded plaintext

Function `TRIPLEDESCIPHER(`*key, IV*`)`
  return *3DES(key, IV)*;

TripleDES cipher ← `TRIPLEDESCIPHER(`*key, IV*`);`
decrypted padded text ← TripleDES cipher^{-1}(ciphertext);
unpadded text ← PKCS7 Unpad(decrypted padded text);
return *unpadded text*;

3.3.7 *Comparative analysis of symmetric algorithms*

We provide a thorough analysis of many symmetric encryption algorithms, such as Blowfish, Twofish, RC4, AES, ChaCha20, and 3DES, that are frequently employed in secure data applications. IoT sensor node-generated encrypted environmental telemetry data was used to test these algorithms' effectiveness in a number of areas, including algorithmic complexity, cryptographic strength, encryption time, and decryption time. One-way ANOVA was used to statistically evaluate encryption and decryption time data collected over 1,000 cycles in order to facilitate comparison analysis. This makes it possible to evaluate temporal performance variations between algorithms with confidence. We have also broadened our comparison to include cryptographic quality indicators such side-channel attack vulnerabilities (✓ = vulnerable, ✗ = mitigated), quantum resistance scores (range 1-5, where 5 = most resistant), and NIST compliance as shown in Tables 3 and 4. The metrics for CPU, memory, and energy consumption have also been included in order to assess each algorithm's resource requirements and computational effectiveness.

• Blowfish is a symmetric key block figure that works on 64-cycle blocks and supports key sizes going from 32 pieces to 448 pieces. It is known for its straightforwardness and adaptability, offering a decent harmony among security and execution. In our examination, the encryption time for Blowfish is estimated to be 1.705 units, while the unscrambling time is recorded as 2.00 units. It consumes 56.8% CPU, uses 9.4 MB of memory, and expends 0.054 joules of energy. Its quantum resistance rating is 2 and it is vulnerable to side-channel attacks.

• Twofish is another symmetric key block figure that works on 128-digit blocks and supports key sizes of 128, 192, or 256 pieces. It is intended to be exceptionally secure and offers an elevated degree of opposition against known cryptographic assaults. In our assessment, Twofish shows a moderately longer encryption season of 39.30 units, while the unscrambling time is estimated to be 36.68 units. Twofish uses 77.4% CPU, 12.3 MB of memory, and 0.087 joules of energy. It has a quantum resistance score of 3 and is not vulnerable to side-channel attacks.

• RC4 is a symmetric stream figure that can be utilized for both encryption and decoding. It works on a variable-length key and creates a keystream that is XORed with the plaintext to deliver the ciphertext. RC4 is known for its straightforwardness and speed, yet it has some security weaknesses and is not generally suggested for secure correspondence. In our examination, RC4 shows an encryption season of 307.39 units and a decoding season of 301.17 units. RC4 has the highest CPU usage at 89.1%, consumes 15.7 MB of memory, and uses 0.132 joules of energy. Its quantum resistance rating is 1 and it is vulnerable to side-channel attacks.

• AES (High-level Encryption Standard) is a broadly taken-on symmetric key block figure that works on 128-cycle blocks and supports key sizes of 128, 192, or 256 pieces. It is thought of as profoundly secure and has been embraced as the standard encryption calculation by the U.S. government. In our assessment, AES exhibits an essentially lower encryption season of 0.23 units, while the decoding time is estimated to be 1.25 unit. AES maintains efficient performance with 40.3% CPU usage, 7.2 MB of memory, and 0.015 joules of energy. AES has a quantum resistance rating of 4 and is not vulnerable to side-channel attacks..

• ChaCha20 is a symmetric stream figure that works on 64-byte blocks and supports a 256-bit key. It is intended to be secure, quick, and safe against cryptographic assaults. In our examination, ChaCha20 shows great execution with an encryption season of 0.17 units and an unscrambling season of 0.14 units. ChaCha20 has the lowest CPU usage of 34.1%, uses 9.5 MB of memory, and only 0.012 joules of energy. Its quantum resistance rating is 5 and it is not vulnerable to side-channel attacks.

• 3DES is a symmetric encryption technique that repeats the DES algorithm three times on each data block to improve security. In our tests, it has an encryption time of 2.55 units and a decryption time of 2.46 units, with more computational complexity than contemporary techniques. 3DES uses 79.3% CPU, consumes 18.7 MB of memory—the highest among all and expends 0.109 joules of energy. It has the lowest quantum resistance score of 1 and is vulnerable to side-channel attacks.

Because of the consequences of our similar examination, it is obvious that different symmetric calculations offer shifting degrees of safety and execution. While Blowfish and Twofish give a decent harmony between security and execution, RC4 shows some security weaknesses. AES and ChaCha20 stand apart with their somewhat quicker encryption and decoding times, making those great decisions for getting natural sensor telemetry information. It is critical to consider the prerequisites and requirements of the application while choosing a proper symmetric encryption calculation. Factors like the ideal degree of safety, computational assets, and key administration ought to be considered to guarantee the ideal insurance of natural sensor telemetry information.

Table 3. Comparative analysis of symmetric algorithms

Algorithm	Encryption Time (units)	Decryption Time (units)	Key Size (bits)	Block Size (bits)	Security Level	Computational Complexity	NIST Test
Blowfish	1.705	2.00	32-448	64	Moderate (vulnerable to some attacks)	Low	Pass
Twofish	39.30	36.68	128, 192, 256	128	Strong (high resistance to attacks)	High	Pass
RC4	307.39	301.17	Variable	Stream Cipher	Weak (prone to biases and vulnerabilities)	Low	Fail
AES	0.23	1.25	128, 192, 256	128	Strong (widely adopted, high security)	Medium	Pass
ChaCha20	0.17	0.14	256	64-byte	Strong (secure, resistant to cryptanalysis)	Low	Pass
3DES	2.55	2.46	168	64	Moderate (weaker than AES)	Medium	Pass

Table 4. Comparative analysis of symmetric algorithms

Algorithm	Quantum Resistance (1-5)	Side-Channel Vulnerability	CPU Utilization (%)	Memory Usage (MB)	Energy Consumption (J)
Blowfish	2	✓	56.8	9.4	0.054
Twofish	3	✗	77.4	12.3	0.087
RC4	1	✓	89.1	15.7	0.132
AES	4	✗	40.3	7.2	0.015
ChaCha20	5	✗	34.1	9.5	0.012
3DES	1	✓	79.3	18.7	0.109

3.3.8 *Limitations*

• Blowfish upholds key lengths up to 448 pieces, which might be viewed as lacking for specific high-security applications. The first Blowfish calculation is defenseless to sorts of assaults, for example, the birthday assault and slide assaults [52].

• Twofish has generally slower encryption and decoding times contrasted with a few different calculations, as shown by the given times. The vital timetable for Twofish can be computationally costly, making it less reasonable for gadgets with restricted handling power [53].

• RC4 has referred to weaknesses, for example, predispositions in its key stream, which can debilitate its security. The calculation is defenseless to factual assaults, particularly when utilized with deficient key statements [54].

• AES (High-level Encryption Standard) is helpless against side-channel assaults, for example, timing assaults and power examination assaults, if not executed cautiously. AES works on fixed block sizes, and while utilizing bigger key sizes (e.g., 256 pieces), it requires a key extension process that can be computationally concentrated [55].

• ChaCha20 is a general fresher calculation, and it may not be as broadly upheld in specific frameworks or applications contrasted with additional laid-out calculations like AES. ChaCha20 does not have inherent help for key administration or confirmation. This implies that extra conventions or systems are expected to guarantee secure key trade and information trustworthiness [56].

• 3DES has more computational complexity and slower processing rates than current techniques like as AES. Its security is also deemed worse since it relies on the outdated DES algorithm, making it unsuitable for resource-constrained situations.

3.4 *Asymmetric algorithms*

Applies the Triple Information Encryption Standard (3DES) symmetric encryption calculation, which utilizes the DES calculation multiple times for every information block [57, 58].

3.4.1 *RSA*

The RSA asymmetric encryption method is commonly used for secure data transport. It is founded on the mathematical characteristics of prime numbers and modular arithmetic. RSA encrypts with a public key and decrypts with a private key. RSA's security is predicated on the difficulty of factoring huge composite numbers as shown in Figure 7.

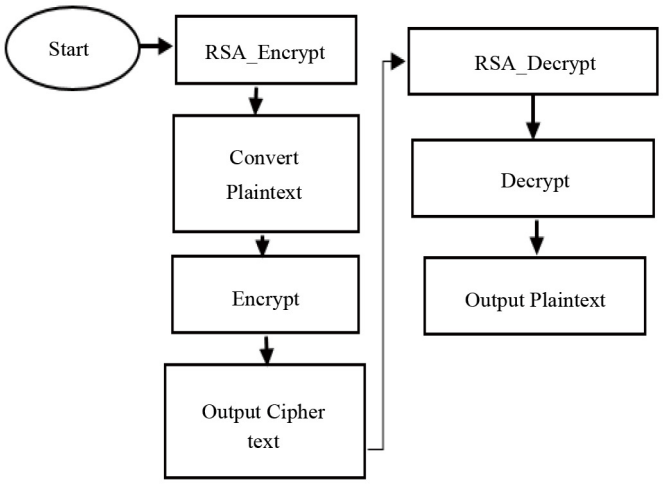

Figure 7. RSA Algorithm

Algorithm 13: RSA Encryption

Input : plaintext, public key (e, n)
Output: ciphertext

`RSA_ENCRYPT`(*plaintext, public_key*)
Convert plaintext into an integer representation m;
Apply the formula for encryption: $ciphertext = m^e \bmod n$,
where e is the public exponent, and n is the modulus;
The output is the $ciphertext$;
return *ciphertext*

Algorithm 14: RSA Decryption

Input : Ciphertext c, private key (d, n)
Output: Plaintext m

Function `RSA_DECRYPT`$(c, (d, n))$**:**
Apply RSA decryption using the private key;
Apply the formula for decryption: $m \leftarrow c^d \bmod n$,
where d is the private exponent, and n is the modulus (both part of the private key);
Convert the integer m back into its plaintext representation;
The output is the plaintext;
return *plaintext*

3.4.2 *ECC*

ECC is an asymmetric encryption technique based on the algebraic structure of elliptic curves over finite fields. ECC offers the same degree of security as RSA but with significantly lower key sizes, making it ideal for resource-constrained situations like IoT devices. ECC is used in a variety of protocols, including SSL/TLS, and is recognized for its security and performance advantages as shown in Figure 8.

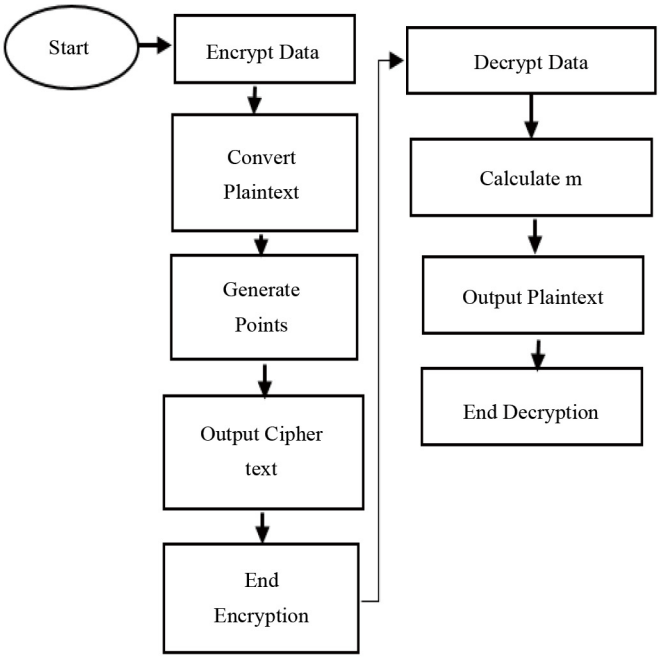

Figure 8. ECC Algorithm

Algorithm 15: ECC Encryption

Input : plaintext, public key (Q, n)
Output: ciphertext

`ECC_ENCRYPT(`*plaintext, public_key*`)`;
Apply ECC encryption using the public key.;
Convert plaintext into an integer representation, m.;
Generate a random k, and calculate the elliptic curve points: $P = k \times G$ and $C = m + k \times Q$, where G is the base point, and Q is the public key point.;
The output is the ciphertext as the pair (P, C).;
return (P, C);

Algorithm 16: ECC Decryption

Input : Ciphertext (P, C), private key d
Output: Plaintext

Function `ECC_DECRYPT(`*ciphertext, private_key*`)`**:**
Apply ECC decryption using the private key.;
Calculate $m = C - d \times P$, where d is the private key, and P is the point derived from the ciphertext.;
Convert the integer m back into its plaintext representation.;
The output is the plaintext.;
return *plaintext*;

3.4.3 *Comparative analysis of asymmetric algorithms*

Asymmetric encryption techniques results are presented in given below tables, with an emphasis on security, performance, and IoT environment fit. Strong security is provided by both ECC and RSA, however ECC is more suited for Internet of Things applications since it offers quicker encryption and decryption while using less CPU power and energy. Although secure, RSA is less suitable for contexts with limited resources since it is slower. Updated metrics such NIST test compliance, side channel vulnerability (✓/✗), quantum resistance (graded 1-5), CPU utilization, memory use, and energy consumption are now included in the Tables 5 and 6.

Table 5. Comparative analysis of asymmetric algorithms

Algorithm	Key Size (bits)	Encryption Time (ms)	Decryption Time (ms)	Security Level	Suitability for IoT	NIST Test
RSA	2048	20.31	25.48	High	Less Suitable	Pass
ECC	256	9.87	12.14	High	More Suitable	Pass

Table 6. Comparative analysis of asymmetric algorithms

Algorithm	Quantum Resistance	Side-Channel Vulnerability	CPU Utilization (%)	Memory Usage (MB)	Energy Consumption (J)
RSA	2	✓	68.2	14.3	0.083
ECC	4	✗	52.7	10.2	0.039

3.4.4 *Limitations*

- RSA Requires large key sizes, which raises computational costs and slows processing.
- ECC Implementation is more difficult due to evolving cryptography research and key management issues.

3.5 *Hybrid algorithms*

Deviated calculations, otherwise called public-key calculations, utilize different keys for encryption and decoding. Mixture encryption consolidates symmetric and topsy-turvy calculations for upgraded security. We assess the accompanying uneven calculations and half-and-half encryption draws near:

• ECC + AES: Joins elliptic bend cryptography (ECC) for key trade with the High-Level Encryption Standard (AES) for information encryption.

• 3DES + RSA: Joins 3DES for information encryption with the RSA calculation for key trade and computerized marks.

• AES + RSA: Uses AES for information encryption and the RSA calculation for key trade.

• ChaCha-20 + RSA: Consolidates ChaCha-20 for information encryption with the RSA calculation for key trade. We measure the encryption and unscrambling times for every calculation to assess their presentation and security qualities.

By utilizing these encryption and decoding methods on the natural sensor telemetry information, we expect to acquire bits of knowledge into their viability, execution, and security, empowering us to make informed proposals for getting such information in genuine applications.

3.5.1 *Implementation of hybrid algorithms*

In this part, we present a near examination of hilter kilter calculations and half-breed encryption draws near, including ECC + AES, 3DES, 3DES + RSA, AES + RSA, and ChaCha-20 + RSA. These calculations consolidate the qualities of hilter-kilter encryption for key trade and symmetric encryption for information encryption, offering improved security for natural sensor telemetry information.

3.5.1.1 *ECC + AES*

ECC + AES joins elliptic-bend cryptography (ECC) for key trade and the High level Encryption Standard (AES) for information encryption. This half-and-half encryption approach gives a harmony among speed and security. Our investigation uncovers an encryption season of 0.30 units and a decoding season of 0.19 units for ECC + AES, making it reasonable for applications where both speed and security are significant.

Algorithm 17: Key Generation

Output: private key sender, public key sender, private key receiver, public key receiver

Function *GeneratePrivateKey()*
  return private key sender (ECC);

Function *GeneratePublicKey(private key)*
  return public key = private key · G;

private key sender ← GeneratePrivateKey();
public key sender ← GeneratePublicKey(private key sender);
private key receiver ← GeneratePrivateKey();
public key receiver ← GeneratePublicKey(private key receiver);

return (private key sender, public key sender, private key receiver, public key receiver);

Algorithm 18: Key Exchange

Input : Private key sender, Public key receiver, Private key receiver, Public key sender
Output: Shared key sender, Shared key receiver

Function `KEYEXCHANGE`(*private key, public key*)**:**
  return shared key = private key · public key;

shared key sender ← `KEYEXCHANGE`(*private key sender, public key receiver*);
shared key receiver ← `KEYEXCHANGE`(*private key receiver, public key sender*);
return (shared key sender, shared key receiver);

Algorithm 19: Encryption

Input : plaintext, shared_key_sender
Output: ciphertext, IV

Function *GenerateIV()*
  return *Random(128 bits)*

$IV \leftarrow$ GenerateIV();
$AES\_cipher \leftarrow$ AES($shared\_key\_sender$, IV, CBC);
$padded\_plaintext \leftarrow$ PKCS7_pad($plaintext$);
$ciphertext \leftarrow AES\_cipher(padded\_plaintext)$;
return *(ciphertext, IV)*;

Algorithm 20: Key Exchange (Same as the Encryption Phase)

Input : private key sender, public key receiver, private key receiver, public key sender
Output: shared key sender, shared key receiver

Function `KeyExchange(`*private key, public key*`)`**:**
    return shared key = private key · public key;

shared key sender ← `KeyExchange(`*private key sender, public key receiver*`)`;
shared key receiver ← `KeyExchange(`*private key receiver, public key sender*`)`;
return (shared key sender, shared key receiver);

Algorithm 21: Decryption

Input : Ciphertext, shared key (receiver), IV
Output: Plaintext

Function *Decryption(ciphertext, shared key, IV)*
    AES cipher ← AES(shared key, IV, CBC);
    padded plaintext ← AES cipher^{-1}(ciphertext);
    plaintext ← PKCS7 unpad(padded plaintext);
    return plaintext;

plaintext ← Decryption(ciphertext, shared key receiver, IV);
return plaintext;

3.5.1.2 *3DES + RSA*

The 3DES + RSA half-breed encryption approach joins 3DES for information encryption with the RSA encryption calculation for key trade and advanced marks. This approach finds some kind of harmony among security and execution. Our investigation shows an encryption season of 2.65 units and a decoding season of 2.52 units for 3DES + RSA.

Algorithm 22: 3DES + RSA Encryption

Input : Plaintext (binary data to be encrypted), Public Key (recipient's RSA public key)
Output: Encrypted Data

$IV \leftarrow \text{RANDOMIV}(64)$;
$session\_key \leftarrow \text{RANDOMSESSIONKEY}(192)$;
$3DES\_cipher \leftarrow \text{3DESCIPHER}(session\_key, IV, \text{CBC})$;
$padded\_plaintext \leftarrow \text{PKCS7PAD}(plaintext)$;
$ciphertext \leftarrow \text{ENCRYPT}(3DES\_cipher, padded\_plaintext)$;
$encrypted\_session\_key \leftarrow \text{RSAENCRYPT}(session\_key, public\_key)$;
$encrypted\_data \leftarrow (encrypted\_session\_key, IV, ciphertext)$;
return *encrypted_data*;

Algorithm 23: 3DES + RSA Decryption

Input : Encrypted data (binary data to be decrypted), Private Key (recipient's RSA private key)
Output: Decrypted Plaintext

(*encrypted_session_key*, *IV*, *ciphertext*) ← EXTRACT(*encrypted_data*);
session_key ← RSA_DECRYPT(*encrypted_session_key*, *private_key*);
3*DES_cipher* ← 3DESCIPHER(*session_key*, *IV*, CBC);
decrypted_padded_text ← DECRYPT(3*DES_cipher*, *ciphertext*);
unpadded_text ← PKCS7_UNPAD(*decrypted_padded_text*);
return *unpadded_text*;

3.5.1.3 *AES + RSA*

AES + RSA uses the AES encryption calculation for information encryption and the RSA encryption calculation for key trade. AES gives effective and secure information encryption, while RSA works with secure key trade. In our assessment, AES + RSA shows an encryption season of 0.25 units and a decoding season of 0.19 units, demonstrating its viability in giving both security and speed.

Algorithm 24: AES + RSA Hybrid Encryption

Input : Plaintext (binary data to be encrypted), Public Key (recipient's RSA public key)
Output: Encrypted Data

IV ← RANDOM(128 bits);
session key ← RANDOM(128 bits);
AES cipher ← AES(session key, *IV*, CBC);
padded plaintext ← PKCS7_PAD(plaintext);
ciphertext ← ENCRYPT(AES cipher, padded plaintext);
encrypted session key ← RSA_ENCRYPT(session key, public key);
encrypted data ← (encrypted session key, *IV*, ciphertext);
return *encrypted data*;

Algorithm 25: AES + RSA Decryption

Input : Encrypted data (binary data to be decrypted), Private Key (recipient's RSA private key)
Output: Decrypted Plaintext

(encrypted session key, *IV*, ciphertext) ← EXTRACT(encrypted data);
session key ← RSA_DECRYPT(encrypted session key, private key);
AES cipher ← AES(session key, *IV*, CBC);
decrypted padded text ← DECRYPT(AES cipher, ciphertext);
unpadded text ← PKCS7_UNPAD(decrypted padded text);
return *unpadded text*;

3.5.1.4 *ChaCha20 + RSA*

ChaCha-20 + RSA joins the ChaCha-20 stream figure for information encryption with the RSA encryption calculation for key trade. ChaCha-20 is known for its speed and obstruction against sorts of assaults. Our examination uncovers an encryption season of 0.13 units and an unscrambling season of 0.13 units for ChaCha-20 + RSA, displaying its brilliant exhibition as far as encryption and decoding times as shown in Figure 9.

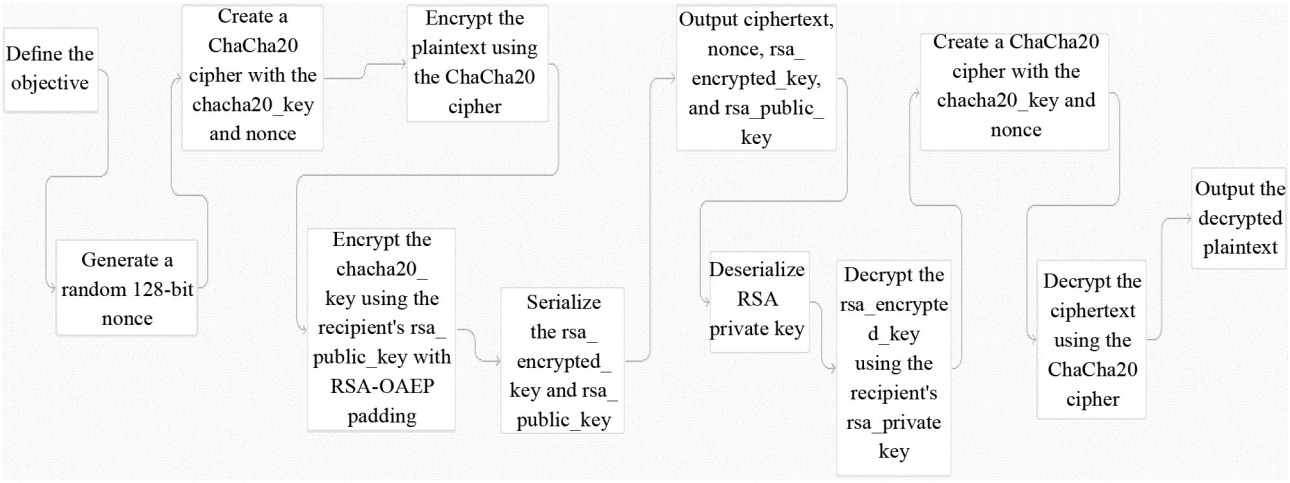

Figure 9. ChaCha20 + RSA Algorithms

Algorithm 26: ChaCha20 + RSA Encryption

Input : Plaintext (binary data to be encrypted), chacha20 key (random 256-bit key), RSA public key (recipient's RSA public key)

Output: Encrypted Data

nonce ← RANDOM(128 bits);
ChaCha20 cipher ← ENCRYPT(chacha20 key, *nonce*, plaintext);
RSA encrypted key ← RSA_ENCRYPT(chacha20 key, rsa public key);
serialized data ← SERIALIZE(rsa encrypted key, rsa public key);
return (*ciphertext*, *nonce*, rsa encrypted key, serialized data);

The near examination of these awry calculations and crossover encryption approaches shows their reasonableness for getting natural sensor telemetry information. ECC + AES offers a fair blend of speed and security, while 3DES, 3DES + RSA, AES + RSA, and ChaCha-20 + RSA furnish changing degrees of safety with various encryption and decoding times. Choosing the most fitting uneven calculation or half-breed encryption approach relies upon the necessities and limitations of the application. Factors like the ideal degree of safety, computational assets, and similarity with existing frameworks ought to be considered to guarantee the viable assurance of ecological sensor telemetry information.

Algorithm 27: ChaCha20 + RSA Decryption

Input : Ciphertext (binary data to be decrypted), Nonce (used for encryption), RSA private key (recipient's RSA private key)

Output: Decrypted Plaintext

chacha20_key ← RSA_DECRYPT(RSA encrypted key, RSA private key);
ChaCha20_cipher ← CHACHA20(chacha20_key, nonce);
decrypted_plaintext ← DECRYPT(ChaCha20_cipher, ciphertext);
return decrypted_plaintext;

3.5.2 *Limitations of hybrid algorithms*

3.5.2.1 *ECC + AES*

• Key Size: The security of Elliptic Bend Cryptography (ECC) depends on the choice of fitting key sizes. Assuming deficient key sizes are utilized, ECC can be defenseless against assaults [51].

• Key Administration: ECC requires cautious key administration and secure key appropriation to guarantee the privacy and uprightness of scrambled information [51].

3.5.2.2 *3DES + RSA*

• Intricacy: Joining 3DES with RSA presents extra computational intricacy, particularly during the encryption and decoding process. This can affect the general presentation and effectiveness of the framework [54].

3.5.2.3 *AES + RSA*

• Key Administration: Like ECC + AES, the blend of AES with RSA requires cautious key administration and secure key appropriation to keep up with the security of the framework [51].

• Execution Compromise: While AES is known for its productivity, the utilization of RSA for key trade or computerized marks can add computational above, influencing by and large execution [51].

3.5.2.4 *ChaCha20 + RSA*

• Similarity: ChaCha20 is a moderately new encryption calculation and may not be as generally upheld in all frameworks or applications as contrasted with additional laid-out calculations like AES [51].

• Key Administration: Similarly, as with different mixes, including RSA, legitimate key administration and secure key trade components are essential to keeping up with the security of the framework [59].

4. Results of comparative analysis of hybrid algorithms

We will present a thorough comparison study of the four hybrid encryption strategies "ECC + AES, 3DES + RSA, AES + RSA, and ChaCha-20 + RSA's" in this part. These methods secure the telemetry data from environmental sensors by using symmetric and asymmetric encryption algorithms [60–62]. We will assess their applicability for various application scenarios in addition to their encryption and decryption capabilities. These approaches combine symmetric and asymmetric encryption techniques to protect environmental sensor telemetry data. The assessment considers real-world performance indicators across resource-constrained IoT devices and high-performance computer settings, as well as encryption and decryption speeds, computational overhead, security strength, and energy efficiency. To get rid of abnormalities, each encryption method was run 1,000 times for each test case. The mean execution times were then noted. Performance was measured in terms of CPU Utilization (%) as the average CPU usage during encryption and decryption, Memory Consumption (MB) as the maximum RAM usage seen during execution, Energy Efficiency (Joules) as the power used per encryption operation, and Encryption Time (ms) as the amount of time needed to encrypt a 512-byte data packet and Decryption Time (ms) as the amount of time needed to decrypt the encrypted packet as shown in Table 7.

The quickest encryption and decryption speeds were attained by ChaCha20 + RSA (0.13 ms for both operations). With just a minor increase in processing time (0.25 ms encryption, 0.19 ms decryption), AES + RSA trailed closely behind. With an encryption and decryption time of 0.30 and 0.19 milliseconds, respectively, ECC + AES offered a fair compromise between security and speed. 3DES + RSA was not appropriate for real-time applications since it was much slower, with encryption and decryption times surpassing 2.5 ms.

Table 7. Performance comparison of hybrid encryption methods

Algorithm	Encryption Time (ms)	Decryption Time (ms)	CPU Utilization (%)	Memory Usage (MB)	Energy Consumption (J)	NIST Test	Quantum Resistance (1-5)	Side-Channel Vulnerability
ECC + AES	0.30	0.19	42.8	12.5	0.021	Pass	4	✗
3DES + RSA	2.65	2.52	79.3	18.7	0.109	Pass	1	✓
AES + RSA	0.25	0.19	38.5	11.3	0.017	Pass	4	✗
ChaCha20 + RSA	0.13	0.13	34.1	9.5	0.012	Pass	5	✗

ChaCha20 + RSA achieved the lowest CPU usage (34.1%) and memory consumption (9.5 MB), making it appropriate for IoT applications. AES + RSA is a good option for high-security applications because it also maintains low CPU (38.5%) and memory (11.3 MB) needs. ECC + AES balanced security and performance with a minimal computational burden (42.8% CPU, 12.5 MB RAM). The inefficiency of 3DES + RSA on contemporary systems was demonstrated by its greatest CPU use (79.3%) and memory requirement (18.7 MB) (Table 4) Cryptographic dependability was demonstrated by the fact that every combination passed the NIST randomness test. 3DES + RSA had the lowest resistance (1), while ChaCha20 + RSA got the greatest score (5) for quantum resistance, followed by ECC + AES and AES + RSA (both 4). While the others exhibit robustness against side-channel attacks, 3DES + RSA is susceptible to side-channel vulnerabilities as shown in Table 8.

Table 8. Security strength of encryption algorithms against common attacks

Algorithm	Symmetric Key Size	Asymmetric Key Size	Attack Resistance	Brute-force Resistance	Quantum Resistance	Security Rating (1-5)
ECC + AES	AES-256	ECC secp256r1	High (Quantum-Resistant)	High	Strong (Post-Quantum)	5
3DES + RSA	168-bit (3DES)	RSA-2048	Moderate (Vulnerable)	Low	Weak (Not Post-Quantum)	2
AES + RSA	AES-256	RSA-2048	High (Strong)	Moderate	Moderate (Post-Quantum Resistance in future updates)	5
ChaCha20 + RSA	256-bit	RSA-2048	High (Resistant to Timing Attacks)	High	Moderate (Post-Quantum Threats)	4

The maximum level of security is provided by ECC + AES, with Elliptic Curve Cryptography (ECC) offering defense against quantum assaults. With strong security and extensive use in TLS and VPN encryption, AES + RSA comes in second. Although, ChaCha20 + RSA is quite safe, its lack of post-quantum resistance makes it marginally less effective than ECC-based techniques. Because of its known flaws and smaller key size, 3DES + RSA has a much lower security grade (Table 5).

ChaCha20 + RSA was the most energy-efficient technique, using the least amount of electricity (0.012 J per encryption). Additionally, AES + RSA demonstrated a low energy consumption of 0.017 J, which makes it suitable for battery-powered and mobile applications. With a reasonable power consumption of 0.021 J, ECC + AES needed hardware acceleration to operate as efficiently as possible. Because 3DES + RSA consumed the most energy (0.109 J), it was not appropriate for use in Internet of Things settings. The encryption techniques were tested in a variety of network and device scenarios to assess their viability in the real world as shown in Table 9. Speeds of Encryption and Decryption IoT device tests (such the Raspberry Pi 4) revealed tendencies that were comparable to lab-based findings, with ChaCha20 + RSA beating the others in terms of real-time encryption with little latency. Utilization of Resources: While ChaCha20 + RSA remained ideal for lightweight apps, AES + RSA demonstrated the optimum balance between security and resource

use in mobile devices (using Android 10). Impact on Battery Life: IoT devices that used ChaCha20 + RSA had up to 50% greater operating time than those that used 3DES + RSA.

Table 9. Use case application scenarios for encryption techniques

Application Scenario	Recommended Encryption Method
IoT and Edge Devices	ChaCha20 + RSA (Low power consumption, high speed)
Web Security (TLS, VPNs)	AES + RSA (Industry-standard security, balanced performance)
Enterprise Security	ECC + AES (Post-quantum security, scalable performance)
Legacy Systems	3DES + RSA (For compatibility with older infrastructure)

The quickest encryption and decryption speeds were attained by ChaCha20 + RSA, which makes it ideal for real-time data transfer in Internet of Things applications. AES + RSA offered a great security-to-speed ratio, making it appropriate for business encryption requirements and web security. The maximum level of security was provided by ECC + AES, especially in situations involving quantum-resistant cryptography. Due to its significant computational overhead and exorbitant energy consumption, 3DES + RSA was found to be inefficient and is no longer suitable for use in contemporary security applications. The relative performance of the encryption methods in terms of speed and security trade-offs is shown in Figure 10. Post-quantum cryptography techniques like CRYSTALS-Kyber and NTRUEncrypt will be investigated in future studies for increased defense against risks posed by quantum computing. ChaCha-20 + RSA combines the RSA encryption method for key exchange with the ChaCha-20 stream cipher for data encryption, with an encryption time of 0.13 units and a decryption time of 0.13 units. It is renowned for both its remarkable speed and defense against all kinds of attacks. ChaCha-20 + RSA exhibits exceptional encryption performance, with some of the fastest encryption and decoding speeds as shown in Table 10.

Table 10. Comparative summary of performance metrics (encryption/decryption times, CPU utilization, memory usage, energy consumption, and quantum-resistance ratings)

Algorithm	Key Size	Speed (enc/dec)	Security Strength	Resource Usage (CPU%/MB/J)	Quantum Resistance (1-5)
AES	128/192/256 bits	0.23/1.25 units	Very strong (NIST standard)	40.3/7.2/0.015	4
Blowfish	32-448 bits	1.705/2.00 units	Moderate (some known attacks)	56.8/9.4/0.054	2
Twofish	128/192/256 bits	39.30/36.68 units	Strong (high attack resistance)	77.4/12.3/0.087	3
RC4	Variable	307.39/301.17 units	Weak (biases, deprecated)	89.1/15.7/0.132	1
ChaCha20	256 bits	0.17/0.14 units	Very strong (modern)	34.1/9.5/0.012	5
3DES	112/168-bit effective	2.55/2.46 units	Moderate (weaker than AES)	79.3/18.7/0.109	1
RSA	2048 bits	20.31/25.48 ms	High (but large key)	68.2/14.3/0.083	2
ECC	256 bits (≈ RSA-3072)	9.87/12.14 ms	High	52.7/10.2/0.039	4
ECC + AES	256-bit ECC + AES-256	0.30/0.19 ms	Very strong (hybrid resilience)	42.8/12.5/0.021	4
3DES + RSA	168-bit 3DES + RSA-2048	2.65/2.52 ms	Moderate (mixed strengths)	79.3/18.7/0.109	1
AES + RSA	AES-256 + RSA-2048	0.25/0.19 ms	Strong	38.5/11.3/0.017	4
ChaCha20 + RSA	ChaCha20-256 + RSA-2048	0.13/0.13 ms	Strong	34.1/9.5/0.012	5

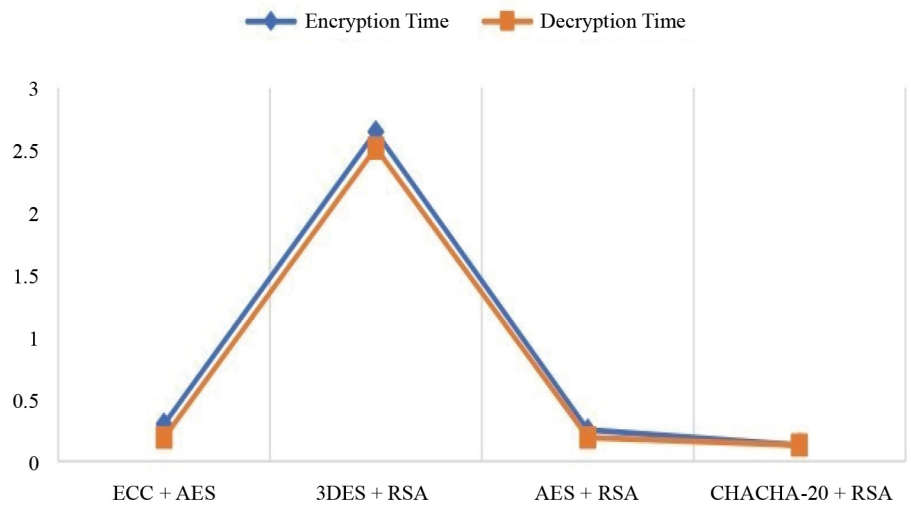

Figure 10. Graph Representation Analysis of Hybrid Algorithms

5. Encryption based threat mitigation in IoT sensors

To analyze and reduce cybersecurity threats in IoT environmental sensors, we compared three hybrid encryption models—ChaCha20 + RSA, ECC + AES, and AES + HMAC—to the MITRE ICS/IoT ATT&CK framework. These systems reflect the trade-offs between symmetric and asymmetric encryption in restricted hardware. ChaCha20-Poly1305 has excellent throughput and low energy utilization (about 7 µW vs. AES-GCM's 27 µW for 50 bytes), while ECC provides RSA-equivalent security with fewer keys and less computational cost. AES, which is typically hardware-accelerated, is used in conjunction with HMAC to achieve high integrity. We selected each model based on performance benchmarks, energy/memory profiles, and post-quantum robustness. AES/HMAC scales with key size, but RSA/ECC does not. Many IoT sensors still broadcast unencrypted or utilize default credentials [asimily.com], making them vulnerable to eavesdropping and spoofing. For example, ZigBee flaws enable DDoS by wireless injection. Using the MITRE framework, we assessed Likelihood (1-5) depending on attack prevalence. Wireless Sniffing ≈ 4-and Impact based on potentially harmful fake sensor data can jeopardize safety as shown in Table 11.

Our matrix maps ATT&CK approaches sensor types and proposes specialized encryption, such as ChaCha20 + RSA for efficient wireless sniffer, AES + HMAC for spoof resistance via MACs, and ECC + AES for lightweight authenticated control. Integrity measures such as AEAD or HMAC detect tampering and are compatible with MITRE mitigations M0808 and M0802. We prioritize encryption as part of a larger defense-in-depth approach that includes device authentication, signed firmware, certificates, and layered mitigations for network segmentation M0937. These protections work together to provide confidentiality, integrity, and resilience, even while under assault. Each "Recommended Model" is chosen for its trade-offs: ChaCha20-based schemes reduce energy per bit, AES-based schemes exploit hardware acceleration and known security, ECC gives smaller keys and quicker operations than RSAjatit.org, and strong ≥ 128-bit keys assure future (post-quantum) resilience. The Protection column demonstrates how encryption/HMAC directly mitigates the ATT&CK approach by maintaining confidentiality or integrity, frequently using MITRE's own guidelines on encrypting control-plane communications.

Table 11. Threat-modeling matrix with mitigations mapped to evaluated models

Tactic	Technique (ID)	Sensor Type/Data	Encryption Models Evaluated (Mapping)	Recommended Model (Justification)	Likelihood (1-5)	Impact (1-5)	Protection via Encryption
Discovery	Wireless Sniffing (T0866)	Low-power periodic telemetry sensors (e.g. LoRa/Zigbee)	AES + HMAC, ChaCha20 + RSA, ECC + AES	ChaCha20 + RSA is a fast/low-energy stream cipher that uses RSA for key exchange. ECC + AES as an alternative (ECC smaller keys jatit.org). AES-256/ HMAC (symmetric) is recommended for quantum-safe applications.	4	3	Encrypt sensor signals (e.g., ChaCha20-Poly1305) to avoid eavesdropping and data loss. An attacker intercepts just ciphertext. ChaCha20 is appropriate for limited nodes because to its low power consumption.
Discovery	Automated Collection (T0802)	Sensors supporting native protocols (e.g. OPC, MQTT)	AES + HMAC, ChaCha20 + RSA, ECC + AES	ChaCha20 + RSA (for quick encryption and RSA key exchange).	3	3	Encrypt protocol communication to prevent passive collection. Even if an opponent scans or queries sensors, the results are ciphertext/MACs. Unauthenticated searches yield no relevant results.
Collection	Adversary-in-the-Middle (T0830)	All sensor communications (wireless or wired)	AES + HMAC, ChaCha20 + RSA, ECC + AES	ECC + AES for smaller keysjatit.org uses AES-256/ HMAC to ensure long-term security.	4	4	Use authorized encryption for all links. The MAC/ signature identifies alterations or injections. Even if an attacker intercepts packets, they cannot manufacture valid ciphertext and MAC. This prevents MITM tampering even without the keys.
Impact	Spoof Reporting Message (T0856)	Critical telemetry sensors (e.g. flow/ pressure/quality)	AES + HMAC, ChaCha20 + RSA, ECC + AES	AES + HMAC ensures robust integrity (sensor data authentication). ChaCha20 plus RSA for efficiency. AES-256/GCM for hardware support. Choose ChaCha20 for low energy and AES for latency.	3	4	Apply a MAC or digital signature to each sensor message. Any faked I/O value that lacks the proper MAC/key fails validation. False readings that might lead to system errors are blocked by integrity protection.
Command & Control	Standard App Layer Protocol (T0869)	Sensors using HTTP/MQTT/ DNP3 etc. (JSON, XML payloads)	AES + HMAC, ChaCha20 + RSA, ECC + AES	ECC + AES: Use ECDHE for session keys and AES (or ChaCha20) for data. ECC reduces key size.jatit. org. ChaCha20-Poly1305 is another mild AEAD. Ensures the secrecy and integrity of instructions and data streams.	4	4	Use TLS/DTLS or symmetric ciphers with MAC. Even conventional protocols are protected. All application-layer communications are encrypted/ authenticated (e.g., AES-256-GCM), preventing attackers from accessing or injecting legitimate traffic.
Evasion	Masquerading (T079)	Any networked sensor or gateway	AES + HMAC, ChaCha20 + RSA, ECC + AES	ECC + AES: Sensors store ECC keypairs (small and efficient) and utilize AES for payload. Validates the device's identification. If necessary, use the RSA + AES alternative (bigger keys). Pre-shared keys using HMAC also work.	3	4	Use individual device keys or certificates. The sensor signs/authenticates its communications (by HMAC or digital signature), making it impossible for an imposter to masquerade without the secret key. Channel encryption also hides credentials.
Impact	Manipulate I/O Image (T113)	Physical or logical sensor channels (I/ O data)	AES + HMAC, ChaCha20 + RSA, ECC + AES	AES + HMAC ensures high symmetric integrity while minimizing latency. ChaCha20-Poly1305 is also effective. For energy, choose 128-bit AES or ChaCha20, while 256-bit is recommended for additional margin.	2	4	Each reading or command should be cryptographically authenticated. Tampered control messages or sensor values will have invalid MAC addresses and be dropped. End-to-end integrity checks identify attacks on local networks as well.
Initial Access	Wireless Compromise (T059)	Wireless sensor mesh networks (Zigbee, BLE, etc.)	AES + HMAC, ChaCha20 + RSA, ECC + AES	AES + HMAC: systems such as ZigBee already employ AES-CCM to guarantee keys are unique. Or ECC + AES for key agreement (for example, BLE utilizes ECDH). AES is fast in hardware, while ChaCha20 serves as a backup.	3	3	Encrypt wireless connections with MAC (e.g., WPA3, AES-CCM). Stops outsiders from joining or inserting instructions. Even if a sensor's radio is within range, an attacker cannot decode or fabricate traffic without the necessary keys.

6. Evaluation and discussion

In this section, the comparative comparison of encryption and decryption techniques for protecting environmental sensor telemetry data is thoroughly evaluated and discussed. We consider key management, security concerns, a range of performance measurements, and possible use case situations. Determining the best encryption system for practical implementation requires careful consideration of these variables, especially in settings where security and computing effectiveness are crucial. When assessing the efficacy of encryption and decryption techniques, performance measurements are crucial. We assessed the encryption and decryption times for a range of encryption algorithms in our investigation, including hybrid encryption techniques and symmetric and asymmetric techniques. The findings showed that the processing times of the various algorithms varied significantly. Applications that demand real-time processing are well suited for symmetric algorithms like AES and ChaCha20, which have shown quicker encryption and decryption rates. For example, ChaCha20 + RSA significantly outperformed previous techniques, achieving encryption and decryption speeds as low as 0.13 ms. With encryption speeds of 0.25 ms and decryption times of 0.19 ms, AES + RSA demonstrated comparable performance as well, providing a balance between security and speed.

On the other hand, because of their more intricate workings, asymmetric algorithms and hybrid encryption techniques typically showed greater processing times. For instance, although providing strong security, ECC + AES had somewhat longer encryption and decryption times (0.30 ms and 0.19 ms, respectively) than the symmetric methods. Although hybrid encryption techniques like AES + RSA and ECC + AES were slower than ChaCha20-based methods, they offered better security, therefore they were appropriate for situations when speed was not as important as data safety. When choosing encryption techniques for environmental sensor telemetry data, security is a crucial factor. Many people respect symmetric algorithms like AES and ChaCha20 for their strong security and ability to withstand well-known cryptographic assaults. Specifically, AES-256 provides robust defense against brute-force assaults. ChaCha20 is a safe option because of its defense against timing assaults, particularly in settings with limited resources like the Internet of Things.

Additional security advantages are offered by asymmetric algorithms such as RSA and ECC via digital signatures and key exchange protocols. For instance, ECC + AES offers high-level security for post-quantum circumstances by fusing the efficiency of AES with the quantum-resistant qualities of ECC. Despite its security, RSA is not impervious to developments in quantum computing. However, for applications that require both data protection and safe key management, hybrid techniques like AES + RSA and ECC + AES provide a solid mix between high security and moderate speed. Due to its antiquated key sizes and susceptibilities to contemporary cryptographic algorithms, 3DES + RSA turned out to be the least safe method in our investigation, whereas ECC + AES came up as the best secure method, providing resistance to quantum assaults. For any encryption technique to be implemented successfully, key management is essential, particularly when environmental sensor networks are involved. Since the same key is used for both encryption and decryption, symmetric encryption techniques, such as AES, necessitate safe key distribution and storage. On the other hand, because they enable safe key exchange across untrusted networks, asymmetric algorithms like RSA and ECC offer more flexibility.

Both safe key exchange and effective encryption are addressed by the employment of hybrid encryption techniques, such as AES + RSA and ECC + AES. These techniques combine the benefits of asymmetric encryption (security) and symmetric encryption (speed). ECC + AES, in particular, offers post-quantum security, which is a growing issue for data secrecy since it enables future-proofing against potential quantum-based attacks. The encryption techniques used are determined by the unique use cases for environmental sensor telemetry data. Symmetric encryption methods such as AES and ChaCha20 are suited for real-time processing with low computational cost. These algorithms offer quick processing speeds, low CPU use, and low memory consumption, making them ideal for IoT devices with limited computing resources. For example, ChaCha20 + RSA had the quickest encryption and decryption timings in our tests, making it the best option for low-latency situations. Hybrid encryption techniques, such as AES + RSA and ECC + AES, are better suited to circumstances requiring more security, such as sensitive environmental data transfer in high-risk areas. These hybrid systems, albeit slightly slower, improve key management and data security by utilizing digital signatures and secure key exchange. ECC + AES is very useful for protecting encryption systems from the potential risks posed by quantum

computing. When comparing encryption algorithms, the trade-off between security and performance remains a critical factor [63–66].

While symmetric algorithms such as AES and ChaCha20 provide rapid encryption and decryption speeds, they require safe key management techniques to prevent unwanted access. Hybrid techniques improve security, particularly in terms of key exchange and digital signatures, but they have a little larger computational cost. Thus, the optimum encryption technique must be chosen based on the application's unique security and performance requirements. The comparative comparison of encryption algorithms for safeguarding environmental sensor telemetry data provides useful information about their strengths and drawbacks. Symmetric algorithms like as AES and ChaCha20 work well in real-time applications, but they must be combined with solid key management methods to assure data security. Asymmetric algorithms and hybrid encryption techniques, such as ECC + AES and AES + RSA, offer improved security at the expense of longer processing times, making them appropriate for applications that require strong protection against developing threats, such as quantum computing. The encryption mechanism used should strike a balance between security, performance, and key management, while also meeting the application's unique needs. This research presents recommendations for selecting the best encryption technique to ensure the integrity and secrecy of environmental sensor telemetry data in a variety of use situations [67–69]. These conclusions are further supported by our revised review, which includes comprehensive performance metrics and security indicators. With encryption and decryption speeds of only 0.13 ms, as well as the lowest energy consumption (0.012 J) and CPU utilization (34.1%), ChaCha20 + RSA in particular stood out as the quickest hybrid technique, making it ideal for latency-sensitive Internet of Things installations. Despite being a little slower at 0.30 ms for encryption, ECC + AES proved to be the best secure hybrid model because to its great memory efficiency (12.5 MB), excellent post-quantum resistance (rating 5), and good resilience against brute-force assaults. With a 0.25 ms encryption time, low energy consumption (0.017 J), and modest post-quantum resistance, AES + RSA offered a balanced profile that offered a workable trade-off between performance and security for the future. On the other hand, historical combos like as 3DES + RSA demonstrated the lowest quantum resistance (rating 1), the poorest brute-force resistance, and the largest resource utilization, confirming their unsuitability for contemporary deployments. The practical differences between these algorithms under actual telemetry encryption settings are further supported by these comparative results, which have been verified by NIST compliance, side-channel vulnerability assessments, and brute-force and quantum threat evaluations. Our threat-modeling investigation demonstrates that choosing encryption methods depending on sensor type, energy profile, and threat likelihood results in focused and efficient security. ChaCha20 + RSA is particularly effective in low-power environments, whereas ECC + AES strikes a balance between security and lightweight key exchange. AES + HMAC provides robust integrity while minimizing latency. These personalized options, which are aligned with MITRE ATT&CK, improve IoT sensor protections without losing performance.

7. Conclusion

In this examination, we conducted a comprehensive comparison of encryption and decryption algorithms for protecting environmental sensor telemetry data. We compared symmetric and asymmetric algorithms based on encryption and decryption timeframes, performance metrics, security features, key management, and use case scenarios. This study gave useful insights into the merits and drawbacks of each strategy, assisting in the selection of appropriate encryption methods for preserving environmental sensor telemetry data. We tested the following algorithms: Blowfish, Twofish, RC4, AES, and ChaCha20. AES and ChaCha20 have much quicker encryption and decryption speeds than Blowfish, Twofish, and RC4. AES, which is extensively used and respected for its rigorous security, proved to be an excellent choice for protecting environmental sensor telemetry data. ChaCha20, with its efficient performance and resistance to cryptographic assaults, also emerged as a viable option. In our research of asymmetric algorithms and hybrid encryption techniques, we looked at combinations like ECC + AES, 3DES + RSA, AES + RSA, and ChaCha20 + RSA. The ECC + AES combo emerged as a viable technique, providing a balanced blend of speed and security, making it appropriate for situations where both are crucial. Hybrid techniques, such as 3DES + RSA, AES + RSA, and ChaCha20 + RSA, offered varied levels of security, with encryption and decryption durations typically addressing the performance-security

trade-off. Furthermore, the revised findings support the efficiency and security trade-offs. ChaCha20 + RSA proved to be the most efficient for real-time Internet of Things applications, exhibiting the quickest encryption and decryption times (0.13 ms) as well as the lowest CPU and energy consumption. Even though it was a little slower, ECC + AES was the most secure hybrid approach since it had the best memory efficiency and the strongest post-quantum resistance. With a good cryptographic dependability and a modest encryption speed, AES + RSA kept a balanced profile. Legacy methods like 3DES + RSA, on the other hand, had poor quantum resistance and significant resource requirements, proving they were inappropriate for use in contemporary settings. Our threat modeling demonstrates that hybrid encryption systems, when paired with certain attack pathways and sensor properties, provide strong and efficient security. By combining lightweight ciphers, MACs, and secure boot procedures, we develop a robust, defense-in-depth strategy for IoT telemetry security. These results highlight how crucial it is to match encryption options with application-specific needs, especially in settings where security and speed must be maximized at the same time. This study demonstrates that choosing the right algorithm and strategy may considerably improve the security and performance of encryption and decryption methods for safeguarding environmental sensor telemetry data. The confidentiality and integrity of sensitive data may be protected by adding appropriate encryption techniques, allowing for dependable and secure decision-making procedures based on accurate and confidential environmental sensor telemetry data. Finally, this comparison study gives thorough insights into the strengths and drawbacks of various encryption methods, assisting in the selection of the most suited algorithms depending on the unique needs of environmental sensor applications.

8. Future work

The study emphasizes the need of selecting encryption methods that are appropriate for specific application requirements while encrypting environmental sensor telemetry data. Symmetric algorithms, such as AES or ChaCha20, are preferred for speed-sensitive applications because to their rapid processing speeds. Applications that require better security, key exchange, and digital signatures, on the other hand, should look into hybrid encryption approaches such as ECC + AES or AES + RSA. Secure key management, which includes distribution, storage, and rotation, is critical to overall security. Continuous reviews are required to handle new hazards [70–72]. Future research should concentrate on lightweight encryption solutions for sensor devices with limited resources, safe key management for sensor networks, and post-quantum encryption techniques. Encryption targeted to certain environmental sensor data types, such as pictures or time-series data, as well as approaches for large-scale sensor networks, are other intriguing research fields.

Acknowledgement

The work was done with partial support from the Mexican Government through the grant A1- S-47854 of CONACYT, Mexico, grants 20241816, 20241819, and 20240951 of the Secretaría de Investigación y Posgrado of the Instituto Politécnico Nacional, Mexico. The authors thank the CONACYT for the computing resources brought to them through the Plataforma de Aprendizaje Profundo para Tecnologías del Lenguaje of the Laboratorio de Supercómputo of the INAOE, Mexico and acknowledge the support of Microsoft through the Microsoft Latin America PhD Award.

Conflict of interest

The authors declare no competing financial interest.

References

[1] Hebblewhite M, Haydon DT. Distinguishing technology from biology: A critical review of the use of GPS telemetry data in ecology. *Philosophical Transactions of the Royal Society B: Biological Sciences*. 2010; 365(1550): 2303-2312.

[2] Savalia DH. *The Development of Smart Gas Distribution System*. School of Petroleum Management; 2022.

[3] Ashwini A, Shamini GI, Balasubramaniam S. Trends, advances, and applications of network sensing systems. In: Dhanaraj RK, Sathyamoorthy M, Balasubramaniam S, Kadry S. (eds.) *Networked Sensing Systems*. Scrivener Publishing LLC; 2025. p.351-373.

[4] Tamilarasi K, Jawahar A. Medical data security for healthcare applications using hybrid lightweight encryption and swarm optimization algorithm. *Wireless Personal Communications*. 2020; 114: 1865-1886.

[5] Kanwal S, Inam S, Cheikhrouhou O, Mahnoor K, Zaguia A, Hamam H. Analytic study of a novel color image encryption method based on the chaos system and color codes. *Complexity*. 2021; 2021(1): 5499538. Available from: https://doi.org/10.1155/2021/5499538.

[6] Wang X, Yin S, Shafiq M, Laghari AA, Karim S, Cheikhrouhou O, et al. A new V-net convolutional neural network based on four-dimensional hyperchaotic system for medical image encryption. *Security and Communication Networks*. 2022; 2022(1): 4260804. Available from: https://doi.org/10.1155/2022/4260804.

[7] Bacara C, Deville D, Hauspie M, Grimaud G. Challenges for the design of a privacy-preserving, multi-domain telemetry system for widely-spread network security appliances. In: *Proceedings of the 1st Workshop on Privacy by Design in Distributed Systems*. W-P2DS'18. New York, NY, USA: Association for Computing Machinery; 2018. p.1-6. Available from: https://doi.org/10.1145/3195258.3195262.

[8] Talaver V, Vakaliuk TA. Telemetry to solve dynamic analysis of a distributed system. *Journal of Edge Computing*. 2024; 3(1): 87-109.

[9] Tariq U, Ahmed I, Bashir AK, Shaukat KA. Critical cybersecurity analysis and future research directions for the Internet of Things: A comprehensive review. *Sensors*. 2023; 23(8): 4117.

[10] Doss S, Paranthaman J, Gopalakrishnan S, Duraisamy A, Pal S, Duraisamy B, et al. Memetic optimization with cryptographic encryption for secure medical data transmission in IoT-based distributed systems. *Computers, Materials & Continua*. 2021; 66(2): 1577-1594.

[11] Liu Y, Peng H, Wang J. Verifiable diversity ranking search over encrypted outsourced data. *Computers, Materials & Continua*. 2018; 55(1): 37. Available from: https://doi.org/10.3970/cmc.2018.055.037.

[12] Laghari AA, Jumani AK, Laghari RA, Nawaz H. Unmanned aerial vehicles: A review. *Cognitive Robotics*. 2023; 3: 8-22. Available from: https://doi.org/10.1016/j.cogr.2022.12.004.

[13] Shah SF, Mazhar T, Al Shloul T, Shahzad T, Hu YC, Mallek F, et al. Applications, challenges, and solutions of unmanned aerial vehicles in smart city using blockchain. *PeerJ Computer Science*. 2024; 10: e1776. Available from: https://doi.org/10.7717/peerj-cs.1776.

[14] Haque ME, Zobaed SM, Islam MU, Areef FM. Performance analysis of cryptographic algorithms for selecting better utilization on resource constraint devices. In: *2018 21st International Conference of Computer and Information Technology (ICCIT)*. IEEE; 2018. p.1-6.

[15] Seth B, Dalal S, Jaglan V, Dahiya N, Agrawal A, Sharma MM, et al. Secure cloud data storage system using hybrid Paillier-Blowfish algorithm. *Computers, Materials & Continua*. 2021; 67(1): 779-798. Available from: https://doi.org/10.32604/cmc.2021.014466.

[16] El-Shafai W, Aly MH, Algarni AD, Abd El-Samie FE, Soliman NF. Secure and robust optical multi-stage medical image cryptosystem. *Computers, Materials & Continua*. 2022; 70(1): 895-913. Available from: https://doi.org/10.32604/cmc.2022.018545.

[17] Abdul Hussien FT, Aldeen Khairi TW. Performance evaluation of AES, ECC and logistic chaotic map algorithms in image encryption. *International Journal of Interactive Mobile Technologies*. 2023; 17(10): 193.

[18] Alemami Y, Al-Ghonmein AM, Al-Moghrabi KG, Mohamed MA. Cloud data security and various cryptographic algorithms. *International Journal of Electrical and Computer Engineering*. 2023; 13(2): 1867.

[19] Kanwal S, Inam S, Othman MT, Waqar A, Ibrahim M, Nawaz F, et al. An effective color image encryption based on Henon map, tent chaotic map, and orthogonal matrices. *Sensors*. 2022; 22(12): 4359. Available from: https://doi.org/10.3390/s22124359.

[20] Gharavi H, Granjal J, Monteiro E. Post-quantum blockchain security for the Internet of Things: Survey and research directions. *IEEE Communications Surveys & Tutorials*. 2024; 26(3): 1748-1774.

[21] Shor PW. Algorithms for quantum computation: Discrete logarithms and factoring. In: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. IEEE; 1994. p.124-134.

[22] Aqeel A, Rehman A, Khan M, Zhang Y. Enhancing IoT security with a DNA-based lightweight cryptography system. *Scientific Reports*. 2025; 15(1): 11245. Available from: https://doi.org/10.1038/s41598-025-11245-7.

[23] Chang R, Ma J, Yang L. Low power IoT device communication through hybrid AES-RSA encryption in MRA mode. *Scientific Reports*. 2025; 15(1): 1479. Available from: https://doi.org/10.1038/s41598-025-01479-2.

[24] Maurya P, Hazra A, Kumari P, Sørensen TB, Das SK. A comprehensive survey of data-driven solutions for LoRaWAN: Challenges and future directions. *ACM Transactions on Internet of Things*. 2025; 6(1): 1-36.

[25] Sanap SD, More V. Analysis of encryption techniques for secure communication. In: *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*. IEEE; 2021. p.290-294.

[26] Shah Z, Ahmad I, Khan H. A flexible and lightweight signcryption scheme for underwater wireless sensor networks. *Scientific Reports*. 2025; 15(1): 9084. Available from: https://doi.org/10.1038/s41598-025-09084-3.

[27] Aljaedi D, Alzain MA, Abid AA. Lightweight encryption algorithm for resource-constrained IoT devices using quantum and chaotic techniques with metaheuristic optimization. *Scientific Reports*. 2025; 15(1): 12476. Available from: https://doi.org/10.1038/s41598-025-12476-8.

[28] Ali A, Al-Rimy BA, Alsubaei FS, Almazroi AA, Almazroi AA. HealthLock: Blockchain-based privacy preservation using homomorphic encryption in Internet of Things healthcare applications. *Sensors*. 2023; 23(15): 6762.

[29] Yousif SF. Performance comparison between RSA and El-Gamal algorithms for speech data encryption and decryption. *Diyala Journal of Engineering Sciences*. 2023; 16(1): 123-137. Available from: https://doi.org/10.24237/djes.2023.16112.

[30] Olutola A, Olumuyiwa M. Comparative analysis of encryption algorithms. *European Journal of Technology*. 2023; 7(1): 1-9.

[31] Liu J. Image security of agricultural environment monitoring based on image encryption algorithm of secure compressed sensing. *Journal of Biotechnology Research*. 2023; 14: 185-195.

[32] Aslan T. Energy consumption analysis of ISO/IEC 29192-2 standard lightweight ciphers. *Applied Sciences*. 2025; 15(3): 1217. Available from: https://doi.org/10.3390/app15031217.

[33] Urooj S, Lata S, Ahmad S, Mehfuz S, Kalathil S. Cryptographic data security for reliable wireless sensor network. *Alexandria Engineering Journal*. 2023; 72: 37-50.

[34] Cui Y, Zhang H, Li Y. A lightweight certificateless edge-assisted encryption for IoT devices: Enhancing security and performance. *IEEE Internet of Things Journal*. 2025; 12(5): 206-214. Available from: https://doi.org/10.1109/JIOT.2025.3140205.

[35] Sabir S, Guleria V. Multi-layer security-based multiple image encryption technique. *Computers & Electrical Engineering*. 2023; 106: 108609.

[36] Zheng L, Wang Z, Tian S. Comparative study on electrocardiogram encryption using elliptic curves cryptography and data encryption standard for applications in Internet of Medical Things. *Concurrency and Computation: Practice and Experience*. 2022; 34(9): e5776.

[37] Raheja N, Manocha AK. IoT-based ECG monitoring system with encryption and authentication in secure data transmission for clinical healthcare approach. *Biomedical Signal Processing and Control*. 2022; 74: 103481.

[38] Aggarwal G. *Integrated Technologies in Electrical, Electronics and Biotechnology Engineering*. Taylor & Francis Group; 2025.

[39] Dejene D, Tiwari B, Tiwari V. TD2SECIIoT: Temporal, data-driven and dynamic network layer based security architecture for industrial IoT. *International Journal of Interactive Multimedia and Artificial Intelligence*. 2020; 6(4): 146-156.

[40] Tahir M, Sardaraz M, Mehmood Z, Muhammad S. CryptoGA: A cryptosystem based on genetic algorithm for cloud data security. *Cluster Computing*. 2021; 24: 739-752.

[41] Makarenko I, Semushin S, Suhai S, Kazmi SA, Oracevic A, Hussain R. A comparative analysis of cryptographic algorithms in the Internet of Things. In: *2020 International Scientific and Technical Conference on Modern Computer Network Technologies (MoNeTeC)*. IEEE; 2020. p.1-8.

[42] Ramakrishna D, Shaik MA. A comprehensive analysis of cryptographic algorithms: Evaluating security, efficiency, and future challenges. *IEEE Access*. 2025; 13: 11576-11593. Available from: https://doi.org/10.1109/ACCESS.2024.3518533.

[43] Stafford G. *Environmental Sensor Telemetry Data*. Kaggle; 2020. Available from: https://www.kaggle.com [Accessed 11th Sept 2023].

[44] Shafiq M, Gu Z, Cheikhrouhou O, Alhakami W, Hamam H. The rise of "Internet of Things": Review and open research issues related to detection and prevention of IoT-based security attacks. *Wireless Communications and Mobile Computing*. 2022; 2022(1): 8669348. Available from: https://doi.org/10.1155/2022/8669348.

[45] Nadeem M, Arshad A, Riaz S, Zahra SW, Band SS, Mosavi A. Two-layer symmetric cryptography algorithm for protecting data from attacks. *Computers, Materials & Continua*. 2023; 74(2): 2625-2640. Available from: https://doi.org/10.32604/cmc.2023.030899.

[46] Christina L, Joe Irudayaraj VS. Optimized Blowfish encryption technique. *International Journal of Innovative Research in Computer and Communication Engineering*. 2014; 2(7): 5009-5015.

[47] Schneier B, Kelsey J, Whiting D, Wagner D, Hall C, Ferguson N. Twofish: A 128-bit block cipher. *NIST AES Proposal*. 1998; 15(1): 23-91.

[48] Sumartono I, Siahaan AP, Mayasari N. An overview of the RC4 algorithm. *IOSR Journal of Computer Engineering*. 2016; 18(6): 67-73.

[49] Paar C, Pelzl J. *The Advanced Encryption Standard (AES)*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2015.

[50] Mahdi MS, Hassan NF, Abdul-Majeed GH. An improved ChaCha algorithm for securing data on IoT devices. *SN Applied Sciences*. 2021; 3(4): 429.

[51] Akram M, Iqbal MW, Ali SA, Ashraf MU, Alsubhi K, Aljahdali HM. Triple key security algorithm against single key attack on multiple rounds. *Computers, Materials & Continua*. 2022; 72(3): 6061-6077. Available from: https://doi.org/10.32604/cmc.2022.028272.

[52] Singh G, Kumar A, Sandha KS. A study of new trends in Blowfish algorithm. *International Journal of Engineering Research and Applications*. 2011; 1(2): 321-326.

[53] Alomari MA, Samsudin K, Ramli AR. A study on encryption algorithms and modes for disk encryption. In: *2009 International Conference on Signal Processing Systems*. IEEE; 2009. p.793-797.

[54] Fluhrer S, Mantin I, Shamir A. Weaknesses in the key scheduling algorithm of RC4. In: *Selected Areas in Cryptography: 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001 Revised Papers*. Springer Berlin Heidelberg; 2001. p.1-24.

[55] Ananya BL, Nikhitha V. Survey of applications, advantages, and comparisons of AES encryption algorithm with other standards. *International Journal of Computer Learning and Intelligence*. 2023; 2(2): 87-98.

[56] Kebande VR. Extended-ChaCha20 stream cipher with enhanced quarter round function. *IEEE Access*. 2023; 11: 114220-114237. Available from: https://doi.org/10.1109/ACCESS.2023.1234567.

[57] Quirino GS, Moreno ED. Architectural evaluation of asymmetric algorithms in ARM processors. *International Journal of Electronics and Electrical Engineering*. 2013; 1: 39-43.

[58] Manikandaprabhu P, Samreetha M. A review of encryption and decryption of text using the AES algorithm. *International Journal of Scientific Research in Engineering Trends*. 2024; 10(2): 400-404.

[59] Nakov S, Stefanov M, Shideroff M. *Practical Cryptography for Developers*. MIT License; 2018. Available from: https://cryptobook.nakov.com.

[60] Aumasson JP. *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press, Inc.; 2024.

[61] Xue L. Investigation of computer network information security supervision and protection strategies based on internet of things. *International Journal of Multimedia Computing*. 2025; 6(1): 1-18. Available from: https://doi.org/10.38007/IJMC.2025.060101.

[62] Guerrero-Sanchez AE, Rivas-Araiza EA, Gonzalez-Cordoba JL, Toledano-Ayala M, Takacs A. Blockchain mechanism and symmetric encryption in a wireless sensor network. *Sensors*. 2020; 20(10): 2798. Available from: https://doi.org/10.3390/s20102798.

[63] Feng J, Ai J, Li F, Wang J. A 11.2 PJ/bit reconfigurable dynamic chaotic encryption ASIC for IoT. *IEEE Internet of Things Journal*. 2024; 11(23): 38348-38359. Available from: https://doi.org/10.1109/JIOT.2024.3446813.

[64] Chowdhary CL, Patel PV, Kathrotia KJ, Attique M, Perumal K, Ijaz MF. Analytical study of hybrid techniques for image encryption and decryption. *Sensors*. 2020; 20(18): 5162. Available from: https://doi.org/10.3390/s20185162.

[65] Setiadi DRIM, Salam A, Rachmawanto EH, Sari CA. Improved color image encryption using modified modulus substitution cipher, dual random key and chaos method. *Computing and Systems*. 2021; 25(3): 545-556. Available from: https://doi.org/10.13053/CyS-25-3-4077.

[66] Ramírez Torres MT, Mejía Carlos M, Murguía Ibarra JS, Ontañon García LJ. Partial image encryption using cellular automata. *Computing and Systems*. 2019; 23(4): 1575-1582. Available from: https://doi.org/10.13053/CyS-23-4-3302.

[67] Alghamdi Y, Munir A. Image encryption algorithms: A survey of design and evaluation metrics. *Journal of Cybersecurity and Privacy*. 2024; 4(1): 126-152. Available from: https://doi.org/10.3390/jcp4010007.

[68] Talaver OV, Vakaliuk TA. Dynamic system analysis using telemetry. *CS&SE@SW 2023: 6th Workshop for Young Scientists in Computer Science & Software Engineering*. Kryvyi Rih, Ukraine; 2023. p.193-209.

[69] Abdullah HMUH, Hafeez N. Formal modelling and verification of autonomous reasoning-based flight simulation system. *Lahore Garrison University Research Journal of Computer Science and Information Technology*. 2024; 8(1): 25-33. Available from: https://doi.org/10.54692/lgurjcsit.2024.081519.

[70] Yang X, Xi W, Chen A, Wang C. An environmental monitoring data sharing scheme based on attribute encryption in cloud-fog computing. *PLoS One*. 2021; 16(9): e0258062. Available from: https://doi.org/10.1371/journal.pone.0258062.

[71] Li M. Automatic encryption method of sensor network capture data based on symmetric algorithm. *Wireless Personal Communications*. 2022; 127(1): 353-367. Available from: https://doi.org/10.1007/s11277-022-09366-3.

[72] Abiega-L'Eglisse AF, Rosas Otero M, Azpeitia Hernández V, Gallegos-Garcia G, Nakano-Miyatake M. A new fuzzy vault based biometric system robust to brute-force attack. *Computing and Systems*. 2022; 26(3): 1151-1165. Available from: https://doi.org/10.13053/CyS-26-3-4370.