Hybrid Cryptographic End-to-End Encryption Method for Protecting IoT Devices Against MitM Attacks

Nader KARMOUS, Moez HIZEM, Yassmine BEN DHIAB, Mohamed OULD-ELHASSEN AOUEILEYINE, Ridha BOUALLEGUE, Neji YOUSSEF

Innov'COM Laboratory, SUPCOM, University of Carthage, Route de Raoued 3.5 Km, Ariana, Tunis, Tunisia

{nader.karmous, moez.hizem, yassmine.bendhiab mohamed.ouldelhassen, ridha.bouallegue, neji.youssef}@supcom.tn

Submitted May 5, 2024 / Accepted September 16, 2024 / Online first October 31, 2024

Abstract. *End-to-End Encryption (E2EE) plays an essential role in safeguarding user privacy and protecting sensitive data across various communication platforms, including messaging applications, email services, and Internet of Things (IoT) devices. This paper presents a Hybrid Cryptography-Based E2EE method implemented on a Software Defined Networking (SDN) infrastructure, to strengthen bidirectional data security between hosts and IoT devices via the non-secure Message Queuing Telemetry Transport (MQTT) port. By addressing the threat of Man-in-the-Middle (MitM) attacks, the proposed system ensures that only authorized users can decrypt transmitted messages. This paper thoroughly analyzes the implementation and advantages of our Hybrid Cryptography-Based E2EE method by comparing RSA and ECC encryption techniques. ECC-256 is favored for key generation, owing to its high efficiency and speed, measured at 0.4009 ms. Additionally, through a comparison of RSA, AES, and ChaCha20 algorithms, AES-256 emerges as the optimal encryption choice, demonstrating the fastest encryption and decryption times for publishing 0.2758 ms and 0.1781 ms, respectively and for subscribing, with encryption at 0.2542 ms and decryption at 0.1577 ms. Along with its minimal packet size and low resource consumption, our proposed Hybrid Cryptography-Based E2EE method, implemented on SDN infrastructure, validate it's effectiveness in securing digital communications within SDN environments compared to existing solutions.*

Keywords

Software-Defined Networking (SDN), cyber security, Man-in-the-Middle (MitM), end-to-end encryption, Internet of Things (IoT)

1. Introduction

In the context of IoT devices [1], MQTT [2] facilitates efficient and reliable communication between IoT devices and other systems via a central broker, enabling real-time data exchange in IoT applications [3]. An IoT device transmits a message payload to a subscriber through an MQTT broker on port 1883. The device, acting as an MQTT client, establishes a TCP connection with the broker on port 1883. Once connected, the device publish to a specific topic of interest, with topics acting as channels for message transmission. The device sends a publish message to the broker, specifying the topic and desired Quality of Service (QoS) level. The broker then forwards the message payload to all subscribers interested in that topic.

In the context of an SDN-IoT environment [4], where the control plane [5] is distinct from the data plane [6], and communication involves network devices orchestrated by a centralized controller, the process of subscribing and publishing in MQTT remains largely unchanged. The SDN controller plays a central role in managing the network infrastructure and facilitating communication between MQTT clients and the broker. By leveraging SDN capabilities such as centralized control and programmability, organizations can achieve more efficient and dynamic management of MQTT communication in their IoT deployments. Nevertheless, this communication may pose cybersecurity threats, such as an MitM attack [7] and [8], due to the vulnerability of port 1883 MQTT. When MQTT traffic is transmitted over port 1883 without encryption, an attacker positioned between the client and the broker can intercept and manipulate the communication, as shown in Fig. 1. This allows the attacker to eavesdrop on the messages exchanged between the client and the broker, modify the content of the messages, or even impersonate either party, thereby compromising the confidentiality and integrity [9] of the communication. To mitigate this vulnerability, it's recommended to use encryption [10] method to secure MQTT communication and prevent MitM attacks. Therefore, securing the MQTT port is crucial to mitigate such threats. E2EE [11] is an effective method that ensures transmitted information remains confidential and secure. When a device publishes data, it encrypts the information using the public key of the intended recipient(s). Subscribing devices receive the encrypted data and use their private keys to decrypt and access the original information. The encryption and decryption processes occur exclusively at the endpoints,

DOI: 10.13164/re.2024.0583

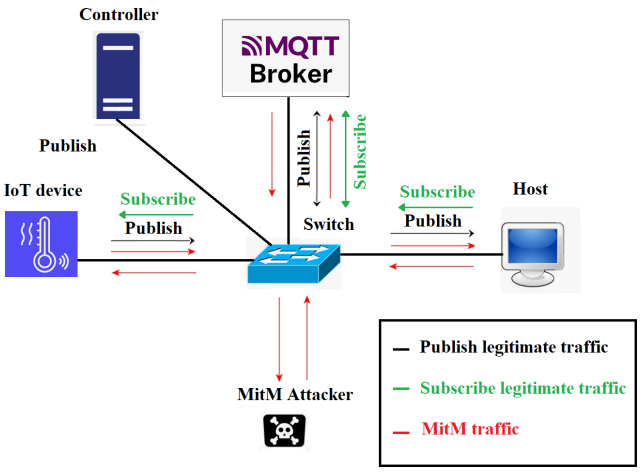

Fig. 1. MitM attacks in SDN.

i.e., the publishing and subscribing devices. Even if the data passes through intermediary communication channels, it remains encrypted and secure. The integrity of the data is maintained as any tampering or unauthorized access would result in unreadable, garbled information.

The main emphasis of this paper lies on port 1883, which currently lacks encryption, whereas port 8883 utilizes TLS/SSL to safeguard MQTT payloads. Our objective is to bolster the security of data transmitted via MQTT port 1883. We propose a Hybrid Cryptography [12] Based E2EE method to secure communications among resource-constrained IoT devices across the host network within an SDN infrastructure. This method involves employing one algorithm for key generation and management, coupled with another algorithm for efficient encryption and decryption of data.

To structure our paper effectively, we divide it into six sections. The first section reviews related works relevant to our research, highlighting their advantages and limitations. In the second section, we delineate our system model, which involves bidirectional transmission of MQTT packet data between IoT devices and hosts via the unsecured MQTT protocol. Moving to the third part, We conduct a comparative study among various algorithms to determine the optimal key generation and encryption/decryption method for MQTT packet data. This analysis is crucial for implementing our Hybrid Cryptography-Based E2EE method. In the fourth section, we implement our Hybrid Cryptography-Based E2EE method. In the fifth section we deploy our Hybrid Cryptography-Based E2EE method within an SDN network to evaluate its effectiveness. Finally, we compare our system with related works to demonstrate its efficacy and relevance.

2. Related Works

The Armstrong Number Encryption Standard (ANES) method [13] is used in [14] to enhance security in IoT devices using the MQTT protocol. ANES employs circular shifts and XOR operations for encryption, making it suitable for resource-constrained devices. The algorithm involves dividing data into blocks, performing shifts, XORing with an Armstrong number and an initialization vector, and chaining blocks. While ANES is efficient in ensuring data confidentiality over MQTT, the paper reports that the encryption and decryption time for an MQTT packet of 100 bytes is 67 ms, highlighting the need to reduce this time. The algorithm's scope includes simulating processes on a microcontroller and evaluating its robustness through MitM attacks.

In [15], the author used the Robust Security Scheme (RSS) to enhance the security of the MQTT protocol in IoT environments. RSS combines a dynamic variant of the Advanced Encryption Standard (D-AES) with Key-Policy Attribute-Based Encryption (KP-ABE) to strengthen security while reducing computational overhead. D-AES improves the standard AES by making key expansion and transformations more robust, while KP-ABE securely manages encryption keys. The scheme ensures confidentiality, access control, collusion resistance, and efficient encryption for MQTT communications, with the paper reporting an encryption and decryption time of 5.73 ms for an MQTT packet. However, the paper's limitation is that it does not focus on testing the scheme against cyber-attacks in real environments, such as SDN and actual MQTT broker servers, to evaluate performance.

H. Li-Wen, et al. [16] introduced a dynamic encryption algorithm for Internet of Vehicles (IoV) systems, balancing security and real-time performance. The algorithm combines AES, PRESENT, and TEA encryption methods, dynamically selecting the appropriate one based on the message's Quality of Service (QoS) level. This approach optimizes system efficiency, improving encryption and decryption performance compared to using AES alone. Simulation experiments demonstrate enhanced security and performance for MQTT communications in IoV, with encryption taking 3463 ms and decryption 3557 ms for a 128-bit MQTT packet. However, the paper notes that the method's low security requirements, aimed at minimizing resource consumption, should be tested in a live network to evaluate its effectiveness against vulnerabilities.

The paper [17] highlighting that ECC offers an equivalent security level to RSA, with significantly smaller key size. This reduction in key size improves processing speed and lowers memory usage, making ECC more efficient, especially for high-security applications such as SSL for web communications. The paper concludes that ECC is preferable for modern cryptographic needs due to its superior performance and security compared to RSA, which has the disadvantage of larger key sizes. Specifically the key generation step which takes only 121 ms with ECC, as against the AES encryption and decryption steps which require an additional latency of 423 ms. A noted limitation of the paper is the lack of real-world beds testing in order to validate performance in real-time scenarios.

The authors of [18] examined privacy protection in vehicular ad-hoc networks (VANETs) and compared elliptic curve cryptography (ECC) with genus 2 and genus 3 hyperelliptic curve cryptography (HECC). It finds that ECC is broadly the better choice for most metrics, but under certain conditions HECC could be more energy efficient. The paper notes that ECC may not fully address the security-performance trade-off in various scenarios, although it has a data encryption and decryption time of 3500 micro-seconds.

3. System Model

Our system model comprises a smart home ecosystem, consisting of IoT devices equipped with DHT11 sensors and smart lights. Additionally, It includes an MQTT broker, specifically the Mosquitto server [19], which was installed on a Raspberry Pi 4 running the Raspbian OS. The purpose of this system model is to encrypt the MQTT payload messages sent by the publisher via 1883' port to the broker before being decrypted by IoT devices or SDN host server.

3.1 IoT Device Subscribe

The subscription architecture in SDN is illustrated in Fig. 2. Host 1 publishes a payload message that indicates either an 'on' or an 'off' status. This message is transmitted to a single topic named iot/device2 using the MQTT protocol via port 1883 to the Mosquitto MQTT broker. Subsequently, the broker forwards the MQTT message to the subscriber IoT devices, in this instance, the smart light. In this scenario, the payload message has a size of 10 bytes. The IoT device, which is an ESP8266, controls the opening or closing of the light based on the payload message sent by the MQTT broker.

3.2 IoT Device Publish

The publishing architecture in SDN is shown in Fig. 3. The IoT device, in particular an ESP8266 combined with a DHT11 sensor, publishes a payload message indicating the temperature and humidity values in the room. This message is transmitted to a single topic named iot/device1 using the MQTT protocol to the Mosquitto MQTT broker through 1883 port. Subsequently, the broker forwards the MQTT message to the subscriber hosts, Host 1 and Host 2. In this scenario, the payload message has a size of 30 bytes. If an IoT device needs to securely communicate with multiple hosts, it must have a public key for each host. It should encrypt each host data separately using the corresponding public key for each one, to ensure that only the intended recipient could decrypt the data.

3.3 Encrypting and Decrypting MQTT Message Payloads

Generating keys for both IoT devices and hosts before encrypting and decrypting data aims to enhance the security of MQTT payloads exchanged between them. To demonstrate this, we applied our Hybrid Cryptography-Based E2EE method in the scenarios outlined in Sec. 3.1 and Sec. 3.2. In these cases, data from Host 1 or the IoT device is encrypted before being sent to the MQTT broker and decrypted only after it is received by the intended recipient, either the host or the IoT device.

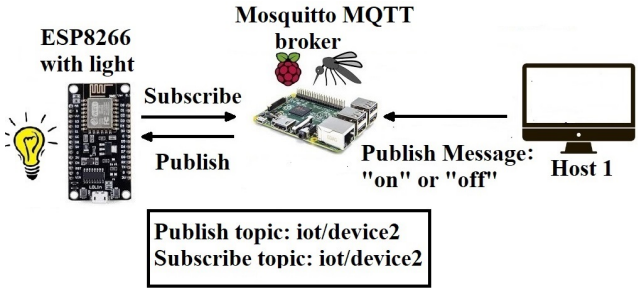

Fig. 2. Subscription architecture for IoT devices in SDN.

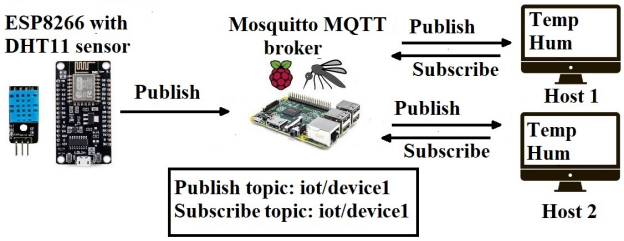

Fig. 3. Publishing architecture for IoT devices in SDN.

4. Comparative Study

In this section, we first specify the hardware and software environment of our work. Then, to implement our Hybrid Cryptography-Based E2EE method, we conduct a comparative study of key generation algorithms such as RSA [20], and ECC [21] across multiple key sizes (128, 192, and 256 bits) to measure the generation time (in ms) and identify the algorithm with the shortest generation time. Subsequently, we perform a comparative study of data encryption algorithms with MQTT payload data of 10 bytes and 30 bytes, including RSA [22], AES [23], and ChaCha20 [24]. We evaluate these algorithms using three different key sizes for AES, and ChaCha20 (128, 192, and 256 bits), and two key sizes for RSA (2048 and 4096 bits). The objective is to determine the best algorithms in terms of encrypted packet size (in octets), encryption time (in ms), and decryption time (in ms). Finally, we select the optimal key generation algorithm among RSA and ECC, as well as the most suitable encryption and decryption algorithms from RSA, AES, and ChaCha20 to implement our Hybrid Cryptography-Based E2EE method deployed in the SDN architecture to protect IoT devices from MitM attacks.

4.1 Hardware and Software Tools

This project utilizes specific hardware and software tools. The hardware includes a virtual machine running Ubuntu 22.04 with the following specifications: Intel Core i5-1235U processor (4.40 GHz Turbo max, 12 MB cache, 10 cores), 16 GB DDR4 RAM, and 256 GB SSD storage. The software tools used are Mininet, a free and open-source tool for creating virtual networks and running networking applications at high speeds; Ryu Controller, an open-source SDN controller that manages network devices via the OpenFlow protocol for dynamic control and centralized management; Mosquitto, an open-source message broker implementing the MQTT protocol for publishing and subscribing to messages in IoT applications; Ettercap, a tool for performing man-in-the-middle attacks, which allows traffic interception and analysis; and Wireshark, an open-source network protocol analyzer for capturing and inspecting data packets in real-time.

4.2 Comparative Analysis

To measure the time taken for key generation, the comparison focuses on RSA ,and ECC key algorithms, chosen for their encryption performance [25] and [26]. We have excluded the 1024-bit key size due to its susceptibility to cyber attacks [27], opting instead for key sizes of 2048 and 4096 for RSA. For ECC, key sizes of 128, 192, and 256 are employed. For the comparison of MQTT payload data encryption sizes, encryption and decryption times, CPU usage, and memory consumption, we utilized RSA with key sizes of 2048 and 4096, as well as AES, and ChaCha20 with key sizes of 128, 192, and 256.

4.2.1 Key Generation

Key generation occurs initially and may also be triggered subsequently by events such as device reboot or session initiation, or based on the expiration of previous keys.

Table 1 compares key generation times for RSA and ECC algorithms in SDN infrastructure. RSA, with key sizes of 2048 and 4096 bits, exhibits longer generation times at 1.73303 and 3.1699 ms, respectively, owing to its reliance on larger key lengths for security. ECC offers key lengths of 128, 192, and 256 bits, with corresponding generation times of 0.2150, 0.3262, and 0.4009 ms, showcasing faster key generation compared to RSA due to its shorter key lengths while maintaining robust security. This makes ECC advantageous for the rapid establishment of secure communication channels, particularly in real-time IoT devices in SDN environments, due to its key agreement nature and reduced computational overhead.

4.2.2 Encrypted Packet Size

Figure 4 illustrates the analysis of encrypted packet sizes for RSA, AES, and ChaCha20 algorithms within an SDN infrastructure, considering packet sizes of 10 bytes and 30 bytes. The results highlight significant observations.

Algorithm	**Key size** [octets]	**Key generation time** [ms]
RSA	2048	1.73303
RSA	4096	3.1699
ECC	128	0.2150
ECC	192	0.3262
ECC	256	0.4009

Tab. 1. Key generation time comparison for RSA and ECC algorithms in SDN infrastructure

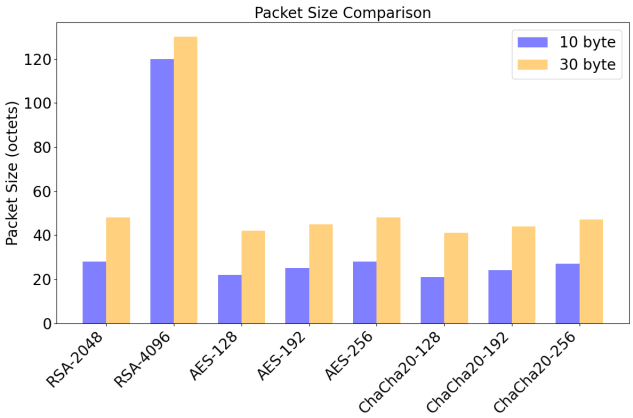

Fig. 4. Comparison of encrypted packet sizes using RSA, AES, and ChaCha20 algorithms for 10 and 30 bytes in SDN infrastructure.

As the packet size increases due to the application of encryption algorithms, there is a consistent escalation in the encrypted packet size across all algorithms, indicating a direct correlation between packet size and encryption overhead. Moreover, upon comparison within each packet size category, RSA consistently yields larger encrypted packet sizes compared to AES and ChaCha20, whereas AES and ChaCha20 exhibit packet sizes closely aligned with each other. This emphasizes the considerable impact of algorithm selection on the resulting encrypted packet size. These findings suggest that in resource-constrained environments like SDN infrastructure, the selection of AES may strike a balance between encryption efficiency and security.

4.2.3 Encryption and Decryption Time

Table 2 presents encryption and decryption times for RSA, AES, and ChaCha20 algorithms across various key sizes for 10-byte and 30 byte MQTT payload messages in SDN infrastructure. For 10-byte payloads, RSA encryption and decryption times are relatively higher, with RSA-2048 encryption taking approximately 0.9372 ms and decryption about 0.3839 ms, increasing to 1.597 ms for RSA-4096 encryption and 0.5546 ms for decryption. Conversely, AES demonstrates efficient performance, with AES-128 encryption and decryption times at 0.1504 and 0.5927 ms, respectively. As key sizes increase, AES maintains consistently low times, with AES-256 encryption and decryption times at 0.2542 and 0.1577 ms, respectively. ChaCha20, while slightly slower than AES, offers competitive performance, with encryption and decryption times ranging from 0.3091 to 0.4116 ms for key sizes 128 to 256 bits. For 30-byte notably, RSA encryption and decryption times are the highest,

Algorithm	Encryption time for 10B [ms]	Decryption time for 10B [ms]	Encryption time for 30B [ms]	Decryption time for 30B [ms]
RSA-2048	0.9372	0.3839	0.9666	0.4412
RSA-4096	1.597	0.5546	1.772	0.6560
AES-128	0.1504	0.05927	0.1826	0.07110
AES-192	0.2138	0.1062	0.2391	0.1369
AES-256	0.2542	0.1577	0.2758	0.1781
ChaCha20-128	0.3091	0.2118	0.3200	0.2366
ChaCha20-192	0.3598	0.2575	0.3697	0.2782
ChaCha20-256	0.4116	0.3013	0.4595	0.3479

Tab. 2. Comparison of encryption and decryption times for 10-byte and 30-byte MQTT payload messages using RSA, AES, and ChaCha20 algorithms in SDN infrastructure.

with RSA-2048 encryption taking approximately 0.9666 ms and decryption about 0.4412 ms, increasing to 1.772 ms for RSA-4096 encryption and 0.6560 ms for decryption. In contrast, AES demonstrates superior efficiency, with AES-128 encryption and decryption times at 0.1826 and 0.07110 ms, respectively. As the key size increases, AES still maintains relatively low times, with AES-256 encryption and decryption times at 0.2758 and 0.1781 ms. ChaCha20, is slightly slower than AES, offers competitive performance, with encryption and decryption times ranging from 0.3200 to 0.4595 ms for key sizes 128 to 256 bits.

AES offers fast and efficient encryption and decryption, particularly for symmetric encryption, making it suitable for scenarios requiring high throughput and low latency.

4.2.4 CPU Usage and Memory Consumption

The analysis of CPU usage and memory consumption for our hardware machine, when using RSA, AES, and ChaCha20 algorithms with 10-byte and 30-byte MQTT payloads, reveals distinct performance characteristics. RSA exhibits higher CPU and memory consumption compared to the other algorithms. In contrast, ChaCha20 shows lower CPU and memory usage relative to RSA. AES, however, performs the best in terms of both CPU and memory utilization for both 10-byte and 30-byte MQTT payloads. This performance comparison is illustrated in Figs. 5 and 6.

4.3 Selecting the Optimal Algorithms

Based on the results discussed in Sec. 4.2, our Hybrid Cryptography-Based E2EE method is designed to utilize the ECC 256 algorithm for key generation due to its efficient key generation process compared to other algorithms. Additionally, we opt for the AES 256 algorithm for encrypting and decrypting MQTT payload data. This choice is motivated by its ability to encrypt and decrypt data swiftly with minimal memory usage and CPU utilization on our operating system, making it a favorable option when compared to alternative algorithms. We used AES256-GCM as AES mode. By using AES-GCM [28], we benefit from its robust security features and efficient performance, making it a preferred choice for modern encryption needs, especially when both confidentiality and data integrity are crucial.

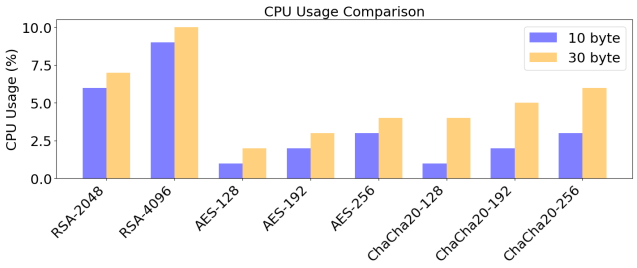

Fig. 5. Comparison of CPU usage using RSA, AES, and ChaCha20 algorithms in SDN infrastructure.

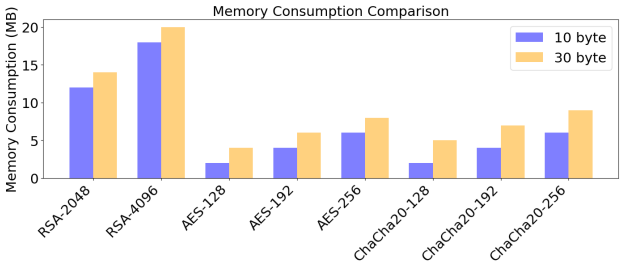

Fig. 6. Comparison of memory consumption using RSA, AES, and ChaCha20 algorithms in SDN infrastructure.

4.4 Algorithms

Before implementing our Hybrid Cryptography-Based E2EE method, which is based on Sec. 4.3, we define the two algorithms that form the basis of our Hybrid Cryptography-Based E2EE method. Algorithm 1 involves publishing and encrypting at the sender's side (IoT device) using ECC-256 and AES-256, while Algorithm 2 entails subscribing, decrypting, and receiving at the receiver's side (Host 2) using ECC-256 and AES-256.

Algorithm 1 on the sender's side, the IoT device captures the plaintext data and generates a shared secret using its private key and the receiver's (host) public key via Elliptic Curve Diffie-Hellman (ECDH) [29]. This shared secret is then used to derive a 256-bit encryption key through Hash-based Key Derivation Function (HKDF) [30], which encrypts the plaintext data with AES-256, producing the encrypted message.

Algorithm 1. Encrypting at sender's side (IoT device).

1:<**Input:** published_data, public_key_receiver>
2:<**Output:** encrypted_message>
3.pm <–published_data
4. shared_secret <–perform_ECDH(public_key_receiver, private_key_sender)
5. derived_key <–HKDF(shared_secret, key_length=256)
6. c_text <–encrypt_AES256(pm, derived_key)
7. encrypted_message <– ciphertext
return (encrypted_message)

Algorithm 2. Decrypting at receiver's side (host).

1:<**Input:** encrypted_message, public_key_sender>
2:<**Output:** received_data>
3. ciphertext <– encrypted_message
4. shared_secret <– perform_ECDH(public_key_sender, private_key_receiver)
5. derived_key <–HKDF(shared_secret, key_length=256)
6. pmr <–decrypt_AES256(ciphertext, derived_key)
7. received_data <–pmr
return (received_data)

Algorithm 2 on the receiver's side, upon receiving the encrypted message and the IoT device's public key, the host device extracts the ciphertext. It then performs an ECDH operation using the IoT device's public key and its own private key to generate a shared secret. This shared secret is processed with a HKDF to derive a 256-bit encryption key. The host device uses this derived key to decrypt the ciphertext with AES-256, revealing the original plaintext message, which is then stored in received_data and returned. This ensures secure communication between the IoT device and the host device by combining ECDH for secure key exchange and AES-256 for strong encryption.

Our Hybrid Cryptography-Based E2EE method ensures data confidentiality and secure transmission between the IoT device and the host.

5. Implementation

In this section, we describe the implementation of our Hybrid Cryptography-Based End-to-End Encryption (E2EE) method within a Software-Defined Networking (SDN) environment. The process for securely distributing the host's Elliptic Curve Cryptography (ECC) public key to an IoT device over MQTT on port 1883 using our method involves several key steps:

- **Key Management:** The SDN controller, acting as a central authority, oversees key management. The host generates an ECC key pair and registers the public key with the Ryu SDN controller.
- **Public Key Distribution:** The Ryu SDN controller publishes the host's ECC public key to a predefined MQTT topic, host/key/public. The IoT device subscribes to this topic and receives the public key.
- **Key Verification and Sharing:** Upon receiving the public key, the IoT device verifies its integrity using a Message Authentication Code (MAC) provided by the SDN controller.
- **Key Derivation:** After successful verification, the IoT device uses the ECC public key to derive a shared secret key. This shared secret is then employed to generate an AES-256 key for encrypting data.
- **Data Encryption and Transmission:** The encrypted data is sent to the host. The host uses its private ECC key to derive the same AES-256 key and subsequently decrypts the received data.

This approach ensures secure key distribution and data encryption, leveraging ECC for key management and AES-256 for data encryption within the SDN framework.

5.1 Steps of our Methodology

Figure 7 illustrates the methodology of our Hybrid Cryptography-Based E2EE method within the SDN interface, establishing a secure connection between the IoT device and the host. The detailed steps are outlined below.

- **Generate Keys**

  ECC involves creating key pairs consisting of a private key and a corresponding public key. Here's a detailed explanation of the key generation process for both parties (IoT device and host device) using a 256-bit elliptic curve:

  Key Generation For IoT Device (Sender): Generate a private key I_{pr} which is a randomly selected integer. For a 256-bit elliptic curve, this integer should be in the range $[1, n-1]$, where n is the order of the elliptic curve. Calculate the public key I_{pu} as:

$$I_{\text{pu}} = I_{\text{pr}} * G \tag{1}$$

  where G is the generator point on the elliptic curve.

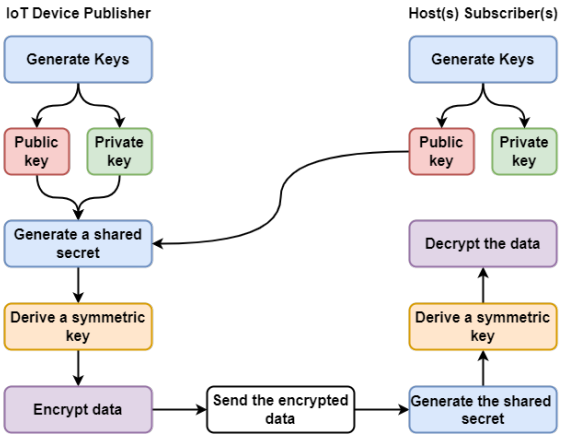

Fig. 7. Integrating our hybrid cryptography-based E2EE method within SDN infrastructure.

Key Generation For Host Device (Receiver): Generate a private key h_{pr} which is a randomly selected integer. For a 256-bit elliptic curve, this integer should be in the range $[1, n-1]$, where n is the order of the elliptic curve. Calculate the public key h_{pu} as:

$$h_{pu} = h_{pr} * G. \tag{2}$$

- **Generate a Shared Secret**

  The IoT device wants to send an encrypted MQTT message to the host's subscribers. It obtains the host subscriber's public key. To generate a shared secret key, we used the ECDH key exchange protocol, where both parties (IoT device and host) use their own private key and the other party's public key. This shared secret is typically a point on the elliptic curve multiplication written as:

$$S = I_{pr} * h_{pu} \tag{3}$$

  where S is the shared secret point, I_{pr} is the IoT device's private key, and h_{pu} is the host's public key.

- **Derive a Symmetric Key from the Shared Secret**

  Next, the IoT device derives a symmetric key from the shared secret. The symmetric key has a length of 32 bytes (256 bits), using HKDF with SHA-256 [31] as the hash function.

- **Encrypt the Data with the Derived Symmetric Key**

  The sender then uses the derived symmetric key to encrypt the actual data. We used the AES-256 algorithm for this step.

- **Send the Encrypted Data and Public Key**

  The host subscriber receives the encrypted data and the IoT device sender's public key via a secure channel.

- **Generate the Shared Secret (On Host Side)**

  The host device wants to decrypt the MQTT message sent by the IoT device. It obtains the IoT device's public key. To generate a shared secret key, we use the ECDH key exchange protocol. Both parties (host and IoT device) will use their own private key and the other party's public key. This shared secret is typically a point on the elliptic curve multiplication written as:

$$S = I_{pu} * h_{pr} \tag{4}$$

  where S is the shared secret point, I_{pu} is the IoT device's public key, and h_{pr} is the host's private key.

- **Derive the Same Symmetric Key**

  The same step as in Sec. 5.1.3 is applied here to derive the symmetric key from the shared secret.

- **Decrypt the Data with the Symmetric Key**

  The host subscriber uses the derived symmetric key to decrypt the encrypted data. We used the AES-256 algorithm for this step. The host subscriber now has the decrypted data, which was originally encrypted by the IoT device.

6. Deployment in SDN

In Fig. 8, we integrate our Hybrid Cryptography-Based E2EE method into an SDN network with 1 Ryu controller, 1 switch, and 4 hosts. Among the hosts, h1 represents an IoT device sensor for temperature and humidity, publishing data every 1 second. h2 is the subscriber host, and h3 represents an MitM attacker. Using Ettercap to launch MitM attacks allows interception of data exchanged between an IoT device and h2. Figure 9 shows the initial Mininet setup with 4 hosts, 1 switch, and 1 controller.

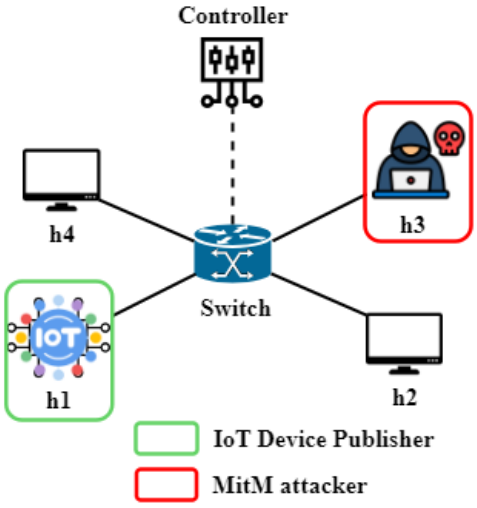

Fig. 8. Our SDN topology utilized.

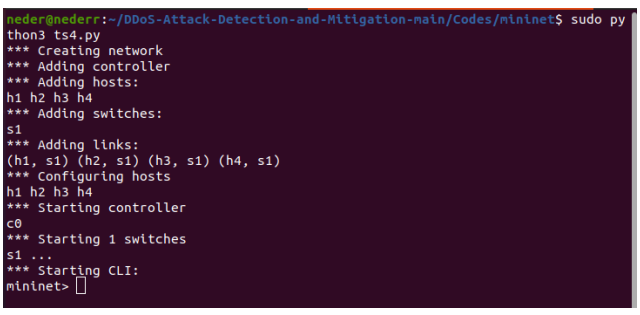

Fig. 9. Creating our virtual SDN network using Mininet.

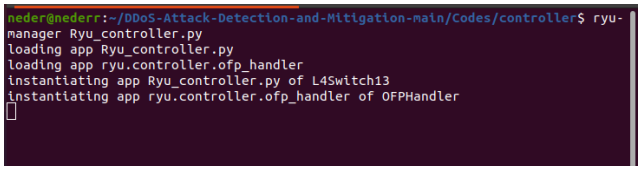

Fig. 10. Starting the Ryu controller.

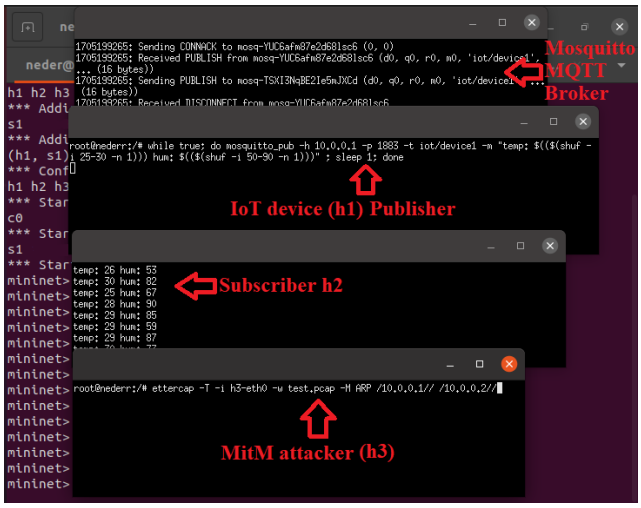

Fig. 11. Launching an MitM attack using Ettercap tools.

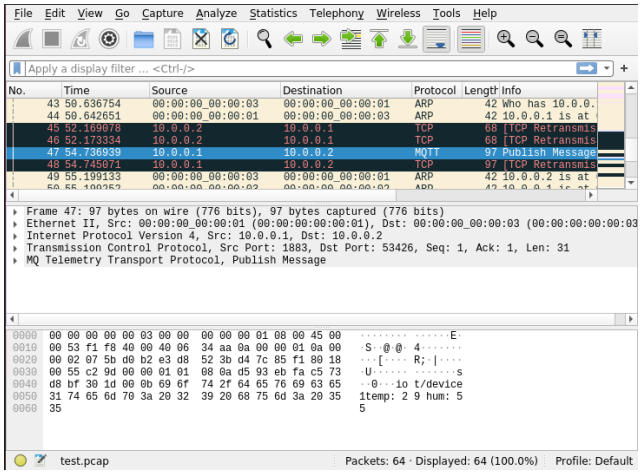

Fig. 12. Wireshark capture file of plaintext traffic collected by an MitM attacker.

6.1 Before Applying Our Hybrid Cryptography-Based E2EE Method

Figure 10 depicts the start of a simple Ryu controller. As shown in Fig. 11, we launched an MitM attack using the Ettercap tool by h3 to collect data exchanged between h1, the IoT device publisher, and h2, the host subscriber. As shown in Fig. 12, the attacker h3 could easily collect the data exchanged between h1 and h2, which includes the temperature and humidity values sent by h1 to h2 using wireshark tools.

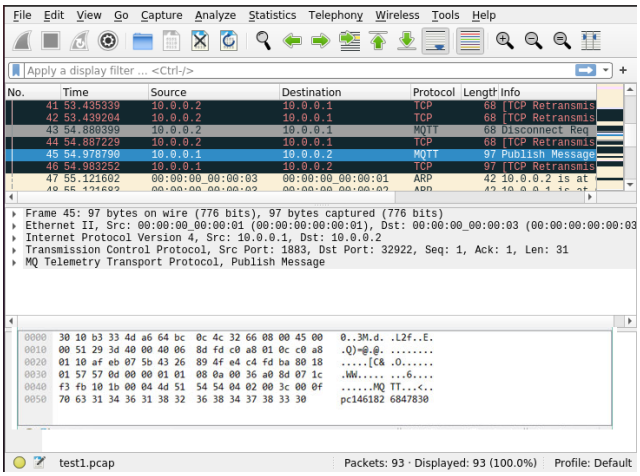

Fig. 13. Wireshark capture file of encrypted traffic collected by an MitM attacker.

6.2 After Applying Our Hybrid Cryptography-Based E2EE Method

After integrating our E2EE method, as explained in Sec. 5.1, we launched an MitM attack using the Ettercap tool by h3 to collect data exchanged between h1, the IoT device publisher, and h2, the host subscriber, as shown in Fig. 11. The attacker could not read the temperature and humidity data sent by h1 to h2. h3 could only read the encrypted data, as shown in Fig. 13.

7. Discussion

This section presents a comparative study of three related works that are closely related to my enhanced E2EE method.

P. Sushma in [14] used the MQTT protocol in an IoT application and implemented Armstrong number encryption to maintain the confidentiality of data over port 1883. From the work, it can be inferred that decryption is faster than the encryption process, taking 0.33 ms and 0.34 ms, respectively.

A. J. Hintaw and S. Manickam in [15] proposed a new security solution called RSS for adoption on IoT devices. This solution augments the existing MQTT protocol with enhanced security features using two separate cryptosystems: enhanced D-AES and KP-ABE. These cryptosystems distribute the publisher's secret key to the subscriber and provide confidentiality of the MQTT payload, broadcast encryption, fine-grained access control, and collision resistance. The proposed scheme increases the average total processing time by only 2.16 and 3.21 ms for encryption and decryption, respectively, compared to related works, where encryption takes 188 ms and decryption 232 ms.

H. Li-Wen in [16] proposed a multi-level dynamic encryption algorithm for an Internet of Vehicles (IoV) system using MQTT. This algorithm encrypts and decrypts the MQTT messages, taking 1143 ms for encryption and 1309 ms

Paper	Framework	Method	Key generation time [ms]	Average time [ms]
[12]	Standard MQTT	Armstrong Number Encryption Standard	-	67
[14]	Standard MQTT	Enhanced D-AES and KP-ABE	-	5.73
[15]	MQTT model	Multi-level Dynamic Encryption Algorithm	-	7020
Our Hybrid Cryptography-Based E2EE	SDN	ECC 256 with AES 256	0.1841	0.4539

Tab. 3. Comparative study of our work with related works.

for decryption. The algorithm improves the overall efficiency of the system and realizes high efficiency and energy saving.

Our proposed Hybrid Cryptography-Based E2EE method secures MQTT payload data on port 1883 for publishing and subscribing. The selected key generation algorithm, ECC-256, requires 0.184 ms for key generation. The encryption of MQTT payload data is oriented to AES-256, with an encryption time of 0.2542 ms and a decryption time of 0.1577 ms for subscribing, while the encryption time is 0.2758 ms and the decryption time is 0.1781 ms for publishing. Our proposed Hybrid Cryptography-Based E2EE method is adaptable to SDN environments, has minimal CPU usage and memory consumption, and outperforms other related works.

Table 3 presents a comparative study of our work with related works. The comparison is based on the framework, method, key generation time, and average encryption and decryption time. This comparative study shows that our hybrid E2EE method performs better than the other works and is flexible and adaptive with SDN networks.

8. Conclusion

In this paper, we present a Hybrid Cryptography-Based End-to-End Encryption (E2EE) method designed to enhance the security of data transmitted between IoT devices and subscribing hosts within an SDN network. Our approach effectively mitigates man-in-the-middle (MitM) attacks by ensuring that data remains confidential even when transmitted over unencrypted channels such as port 1883. The method leverages Elliptic Curve Cryptography (ECC-256) for key generation and employs the AES-256 algorithm for both encryption and decryption, using symmetric keys derived from the publisher IoT device and subscriber hosts.

Our Hybrid Cryptography-Based E2EE method represents a significant advancement over existing solutions, providing improved efficiency and security for MQTT payload exchanges. It is particularly well-suited for practical implementation in IoT systems within SDN environments.

However, a current limitation is that the same encrypted data is transmitted consistently over extended periods. To address this, future work will involve regenerating keys on an hourly basis and updating data transmission intervals to either every minute or every five minutes. Additionally, we are considering incorporating timestamp or sequence number-based mechanisms into our encryption protocol to uniquely identify and validate each data transmission session. These enhancements aim to prevent malicious actors from reusing intercepted ciphertexts to impersonate legitimate data.

References

[1] MENEGHELLO, F., CALORE, A., ZUCCHETTO, D., et al. IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices. *IEEE Internet of Things Journal*, 2019, vol. 6, no. 5, p. 8182–8201. DOI: 10.1109/JIOT.2019.2935189

[2] SONI, D., MAKWANA, A. A survey on MQTT: A protocol of Internet of Things (IoT). In *Proceedings of the International Conference on Telecommunication, Power Analysis and Computing Techniques (ICTPACT)*. Chennai (India), 2017, p. 1–20.

[3] BEN DHIAB, Y., OULD-ELHASSEN AOUEILEYINE, M., ABDELKADER, M., et al. Edge-based human activity recognition: A novel approach using spectral analysis and deep learning. In *Proceedings of the International Wireless Communications and Mobile Computing (IWCMC)*. Ayia Napa (Cyprus), 2024, p. 1734–1739. DOI: 10.1109/IWCMC61514.2024.10592539

[4] SARICA, A. K., ANGIN, P. Explainable security in SDN-based IoT networks. *Sensors*, 2020, vol. 20, no. 24, p. 1–30. DOI: 10.3390/S20247326

[5] BHUIYAN, Z. A., ISLAM, S., ISLAM, M. M., et al. On the (in)security of the control plane of SDN architecture: A survey. *IEEE Access*, 2023, vol. 11, p. 1–33. DOI: 10.1109/ACCESS.2023.3307467

[6] SHAGHAGHI, A., KAAFAR, M. A., BUYYA, R., et al. Software-defined network (SDN) data plane security: Issues, solutions, and future directions. Chapter in *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*. Cham: Springer, 2020, p. 341–387. DOI: 10.1007/978-3-030-22277-2_14

[7] CONTI, M., DRAGONI, N., LESYK, V. A survey of man-in-the-middle attacks. *IEEE Communications Surveys & Tutorials*, 2016, vol. 18, no. 3, p. 2027–2051. DOI: 10.1109/COMST.2016.2548426

[8] FEREIDOUNI, H., FADEITCHEVA, O., ZALAI, M. IoT and man-in-the-middle attacks. *arXiv*, 2023, p. 1–11. DOI: 10.48550/arXiv.2308.02479

[9] YEE, C. K., ZOLKIPLI, M. F. Review on confidentiality, integrity and availability in information security. *Journal of Information and Communication Technology in Education*, 2021, vol. 8, no. 2, p. 34–42. DOI: 10.37134/JICTIE.vol8.2.4.2021

[10] THAMBIRAJA, E., RAMESH, G., UMARANI, R. A survey on various most common encryption techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2012, vol. 2, no. 7, p. 226–233.

[11] HALE, B., KOMLO, C. On end-to-end encryption. *Cryptology ePrint Archive*, 2022, p. 1–20. [Online] Available at: https://eprint.iacr.org/2022/449

[12] DIXIT, P., GUPTA, K., TRIVEDI, M. C., et al. Traditional and hybrid encryption techniques: A survey. *Networking Communication and Data Knowledge Engineering*, 2018, vol. 4, p. 31–46. DOI: 10.1007/978-981-10-4600-1_22

[13] SUSHMA, P., GOPAL, V. V. H. Armstrong number encryption standard for smart devices - An IoT based encryption algorithm. *Dogo Rangsang Research Journal*, 2022, vol. 12, no. 12, p. 36–42. ISSN: 2347-7180

[14] SUSHMA, P. Smart devices security with Armstrong number encryption standard algorithm using MQTT protocol-An IoT application. *International Journal of Intelligent Systems and Applications in Engineering*, 2024, vol. 12, no. 10, p. 45–51. ISSN: 2147-6799

[15] HINTAW, A. J., MANICKAM, S., KARUPPAYAH, S., et al. A robust security scheme based on enhanced symmetric algorithm for MQTT in the internet of things. *IEEE Access*, 2023, vol. 11, p. 43019–43040. DOI: 10.1109/ACCESS.2023.3267718

[16] LI-WEN, H., YANG, K., FU, L., et al. Dynamic encryption method for MQTT communication. *Journal of Physics: Conference Series*, 2024, vol. 2717, no. 1, p. 1–8. DOI: 10.1088/1742-6596/2717/1/012011

[17] KHAN, M. R., UPRETI, K., ALAM, M. I., et al. Analysis of elliptic curve cryptography & RSA. *Journal of ICT Standardization*, 2023, vol. 11, no. 4, p. 355–378. DOI: 10.13052/jicts2245-800X.1142

[18] ROUTIS, G., DAGAS, P., ROUSSAKI, I. Enhancing privacy in the internet of vehicles via hyperelliptic curve cryptography. *Electronics*, 2024, vol. 13, no. 4, p. 1–29. DOI: 10.3390/electronics13040730

[19] LIGHT, R. A. Mosquitto: Server and client implementation of the MQTT protocol. *Journal of Open Source Software*, 2017, vol. 2, no. 13, p. 1–2. DOI: 10.21105/joss.00265

[20] GALLA, L. K., KOGANTI, V. S., NUTHALAPATI, N. Implementation of RSA. In *Proceedings of the International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. Kumaracoil (India), 2016, p. 730–733. DOI: 10.1109/ICCICCT.2016.7987922

[21] NIMBHORKAR, S. U., MALIK, L. G. A survey on elliptic curve cryptography (ECC). *International Journal of Advanced Studies in Computers, Science and Engineering*, 2012, vol. 1, no. 1, p. 1–5.

[22] GARG, S., RANA, M. K. A review on RSA encryption algorithm. *International Journal of Engineering and Computer Science*, 2016, vol. 5, no. 7, p. 17148–17151. DOI: 10.18535/ijecs/v5i7.07

[23] ABDULLAH, A. M. Advanced encryption standard (AES) algorithm to encrypt and decrypt data. *Cryptography and Network Security*, 2017, vol. 16, no. 1, p. 1–12.

[24] THARA, K. S. T., VALLALA, P. G. A survey of encryption algorithms in IoT. In *Proceedings of the 17th SC@RUG 2019-2020*. Groningen (Netherlands), 2020, p. 9–14. ISBN: 978-94-034-2766-9

[25] VAHDATI, Z., YASIN, S., GHASEMPOUR, A., et al. Comparison of ECC and RSA algorithms in IoT devices. *Journal of Theoretical and Applied Information Technology*, 2019, vol. 97, no. 16, p. 4293–4301. ISSN: 1992-8645

[26] SUAREZ-ALBELA, M., FERNANDEZ-CARAMES, T. M., FRAGA-LAMAS, P., et al. A practical performance comparison of ECC and RSA for resource-constrained IoT devices. In *Proceedings of the Global Internet of Things Summit (GIoTS)*. Bilbao (Spain), 2018, p. 1–6. DOI: 10.1109/GIOTS.2018.8534575

[27] YU, H., KIM, Y. New RSA encryption mechanism using one-time encryption keys and unpredictable bio-signal for wireless communication devices. *Electronics*, 2020, vol. 9, no. 2, p. 1–10. DOI: 10.3390/electronics9020246

[28] KIM, K., CHOI, S., KWON, H., et al. PAGE-practical AES-GCM encryption for low-end microcontrollers. *Applied Sciences*, 2020, vol. 10, no. 9, p. 1–14. DOI: 10.3390/app10093131

[29] HAAKEGAARD, R., LANG, J. *The Elliptic Curve Diffie-Hellman (ECDH)*. [Online] Cited 2023-08-18. Available at: https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Haakegaard+Lang.pdf

[30] KRAWCZYK, H. Cryptographic extraction and key derivation: The HKDF scheme. In *Proceedings of the Annual Cryptology Conference (CRYPTO)*. Berlin (Germany), 2011, p. 631–648. DOI: 10.1007/978-3-642-14623-7_34

[31] IETF *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*. RFC 6234, 2011. DOI: 10.17487/RFC6234

About the Authors ...

Nader KARMOUS is pursuing a Ph.D. at Innov'COM, SUP'COM University, specializing in cybersecurity for IoT devices using AI. He earned his Master's degree in Computer Science from the University of Carthage in 2016.

Moez HIZEM is an Assistant Professor and a researcher at Innov'COM, SUP'COM, focused on AI, wireless systems, and cognitive radio. He received his M.S. in Electronics in 2004, an M.Sc. in Telecommunications in 2006, and a Ph.D. in Telecommunications from ENIT in 2011. He is currently pursuing an H.D.R. degree.

Yassmine BEN DHIAB is a Ph.D. candidate at the Innov'COM Laboratory at SUP'COM, University of Carthage, where she focuses on edge AI for optimizing IoT performance. She obtained her engineering degree in Microelectronics from ISIMM, University of Monastir, in 2019.

Mohamed OULD-ELHASSEN AOUEILEYINE is an Assistant Professor at the University of Carthage, with expertise in IoT, TinyML, and eHealth. He has a strong background in electrical engineering and is active in professional organizations and research in smart systems, security, and nanosatellites.

Ridha BOUALLEGUE is a Full Professor at ENIT and has been teaching since 1990. He is currently the General Director of SUP'COM and has founded several research labs and conferences. He earned his doctorate in 1998 and his H.D.R. in 2003, focusing on telecommunications, digital communications, and next-generation wireless networks.

Neji YOUSSEF is a Full Professor at SUPCOM, Tunisia. He received his B.E. in Telecommunications in 1983, the D.E.A. in Electrical Engineering in 1986, and his M.E. and Ph.D. in Communication Engineering from The University of Electro-Communications, Tokyo, in 1991 and 1994, respectively. His research interests include noise theory, wireless communications, and multipath fading channels.