INVOTEK

Jurnal Inovasi Vokasional dan Teknologi

http://invotek.ppj.unp.ac.id/index.php/invotek

Vol. 24, No. 1, Feb 2024

ISSN: 1411 – 3411 (p)
ISSN: 2549 – 9815 (e)

Secure Communication ECC-Based between IoT Device and Server

Maretha Ruswiansari[1] [*], Febrianto Surya Kusumah[1], Sigit Wasista[1], Mohamad Ridwan[2]

[1] Department of Informatics and Computer Engineering, Politeknik Elektronika Negeri Surabaya
Jl. Raya ITS, Kampus PENS, Sukolilo, Surabaya, Indonesia-60111

[2] Department of Electronics Engineering, Universitat Politècnica de Catalunya
Carrer de Jordi Girona, 31, Les Corts, Barcelona, Spain-08034

**Corresponding author: maretha@pens.ac.id*

Doi: https://doi.org/10.24036/invotek.v24i1.1165

Abstract

The rapid development of the IoT (Internet of Things) demands speed and security of communication between connected entities via the Internet. A suitable communication protocol for this communication in terms of speed optimization is MQTT (Message Queuing Telemetry Transport). However, it has security limitations that make it vulnerable to third-party attacks. This research proposes an IoT communication system and server using the MQTT protocol and Elliptic Curve Cryptography (ECC) algorithms to secure communications. ECC efficiently uses computing resources and has a short key size compared to Rivest Shamir Adleman (RSA), so it is suitable for mutual authentication. In addition, data encryption uses the 128-bit Advanced Encryption Standard (AES), which has good security and computing efficiency. The study included testing the mutual authentication speed of ECC and RSA across different key sizes, demonstrating that ECC consistently outperformed RSA in execution time. This study also compared the speed of mutual authentication between ECC and RSA with key sizes of 256 and 3072 bits, respectively; ECC achieved an average speed of 117.33 ms, whereas RSA took 241.92 ms. Furthermore, this study was also tested for the level of security using ECC as a cryptographic algorithm. The system is tested for security by performing sniffing attacks, brute force attacks, replay attacks, and fingerprint matching accuracy by measuring the False Rejection Rate (FRR) and False Acceptance Rate (FAR). The most suitable threshold value is between 30 and 40 within an Equal Error Rate (ERR) between 20% and 30%. The overall testing results show that the system is time-efficient and resilient to attacks.

Keywords: IoT, MQTT, ECC, Mutual Authentication, Security.

1. Introduction

The development of IoT today has increased by leaps and bounds. This IoT development is supported by technological advances, increased connectivity, and the need for intelligent systems to help human life [1]. Globally, the number of internet-connected devices or IoT will reach 30 to 70 billion units by 2025 [2]–[4]. Security and privacy are essential factors in IoT networks. The sheer number of connected IoT devices is fueling an increase in cyberattacks from adversaries [1], [5].

Good security can protect user privacy and reduce potential or prevent data leakage [1], [6]. With so many connected IoT devices, communication between connected devices can be strengthened by mutual authentication because this method provides a more robust layer of security than one-way authentication [7]. In addition, maintaining the confidentiality of the data communicated is also very important, so it is necessary to carry out an encryption process on the messages sent [8]. Communication and high privacy security need many security requirements, so communication must consider memory efficiency and time because IoT devices perform processes with limited resources.

Previous research has proposed securing Two-Factor Authentication (2FA) for IoT devices by applying mutual authentication and message encryption to communication via the MQTT protocol between client and server [9]. However, the study conducted mutual authentication by applying the RSA algorithm, which is less effective for use on IoT devices [10].

Received date 28/03/2024, Revised date 19/06/2024, Accepted date 27/06/2024

Mutual authentication can be performed using certificates generated by signing ECC public keys [11] using a Certificate Authority (CA) [12]. ECC is a cryptographic algorithm based on elliptic curves, while CA is a third party responsible for issuing, managing, and certifying digital certificates. The entity that receives the certificate will verify it using the same CA. The ECC algorithm is used because it is considered better. After all, with the same level of security between RSA and ECC, the parameters used by ECC have a smaller size, so a more suitable algorithm for IoT devices with limited resources is ECC [10], [13], [14]. Data encryption is performed using symmetric keys generated from the Elliptic Curve Diffie-Hellman (ECDH) algorithm [15] and the AES algorithm [16], [17]. ECDH is an ECC-based secret key exchange protocol, while AES is a symmetric encryption algorithm. The encryption key used the ECDH secret key combined with AES to encrypt data.

Mutual authentication using ECC on IoT devices is needed to accelerate and strengthen communication security between client devices and servers so that problems from previous studies can be solved. Thus, this research built a communication system between IoT devices and servers using the ECC method as a basis for mutual authentication and data encryption using symmetric keys from ECDH. This system was tested based on speed and security aspects. In the speed aspect, mutual authentication speed testing was carried out. Meanwhile, in the security aspect, testing was carried out, which included a sniffing attack, brute force attack, replay attack, and fingerprint matching accuracy.

However, this study does not cover all possible types of cyberattacks. Other relevant attack types, such as side-channel and denial-of-service (DoS) attacks, are not discussed, which does not provide a complete picture of the proposed system's resilience. While critical, denial-of-service (DoS) attacks primarily target network availability and are often mitigated through network management strategies rather than cryptographic methods, which are the focus of this study.

2. Method

2.1 System in General

The system built an infrastructure using the Raspberry Pi 3 as a client device, which acts as the front end of the system, and a laptop with a 2.8 GHz Dual-Core Intel Core i7 specification as a server, which functions as a party that verifies and stores user data. The details of the software specifications are shown in Table 1. The client device and the server were connected over a local Wi-Fi network using a mobile phone as a hotspot with a 2.4 GHz frequency band and WPA2 network security.

Table 1. Software Specifications

Descriptions	Client Device (Raspberry Pi 3)	Server (Laptop)
Operating System	Raspbian OS	Ubuntu 22.04 LTS
Programming Language	Python 3.10	Python 3.10
Library	Paho-MQTT, Cryptography	Paho-MQTT, Cryptography
Additional Software	-	MQTT Broker Mosquitto

Communication between client devices run by the Raspberry Pi 3 and servers based on laptops is done via the Message Queuing Telemetry Transport (MQTT) protocol [18]. The MQTT Broker that bridges between client devices and servers is located on the server and has been configured to be publicly accessible. The infrastructure of this system is described in detail in Figure 1, which illustrates the relationship between the main components in the system architecture.

In the communication structure that is compiled, three main topics become the backbone of the interaction between client devices and servers:

- The topic of authentication enables mutual authentication between client and server, ensuring that both parties can trust each other's identity before starting the data exchange.
- The registration topic is an entry point for new users to register and join the system.
- The verification topic plays a crucial role in verifying the user's identity and validating any access requests or actions performed to ensure the legitimacy and security of the operations performed.

The mutual authentication process occurs automatically when the client device is turned on, initializing the connection between the device and the server by verifying each other's identity. After the mutual authentication, the device can only be used for registration and verification. Client devices are connected with a Radio Frequency Identification (RFID) module [19] and a fingerprint module [20] to record user data and collect important information about users, which is then used in authentication and verification processes. User data sent between client devices and servers is encrypted using symmetric keys and AES 128 to maintain security and confidentiality. On the server side, the system will determine whether the user is verified; if so, the server will respond that the user has been verified, and the client device can grant access to the user.

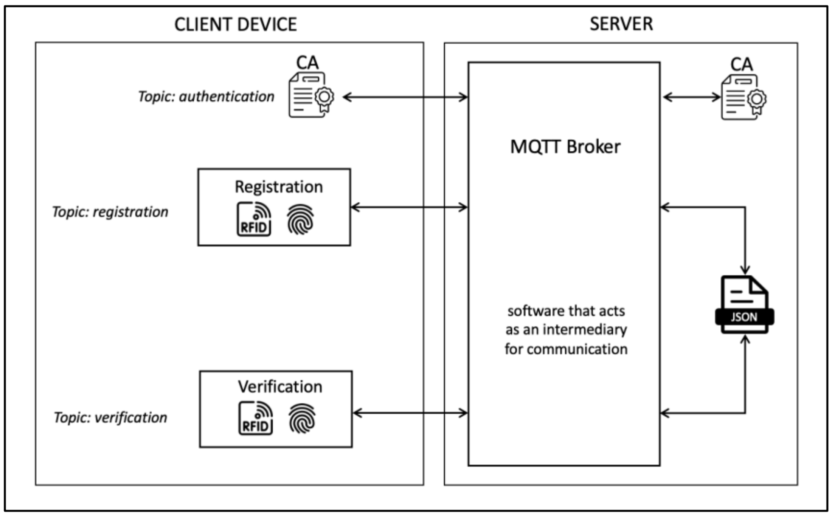


Figure 1. System Diagram in General

2.2 *Mutual Authentication Process*

Mutual authentication is performed using the ECC and RSA algorithms. The mutual authentication process using ECC begins with the client device sending its public key and certificate to the server. The two are combined into one delivery payload with an explicit delimiter, dividing part of the certificate and the public key. This data is transmitted to the server without encryption due to its public nature. Next, the server receives the public key and certificate from the client device, which will be verified using the CA owned by the server. If the certificate is valid, the server creates the shared key. This shared key is created by combining the server's private key and the client device's public key. When the client device receives the public key and certificate from the server, the client device also verifies the certificate using the CA. The client device creates the shared key if the server certificate is valid. The mutual authentication process can be seen in Figure 2.

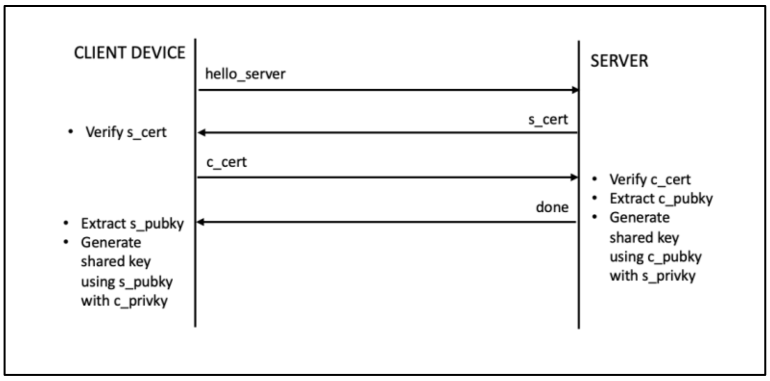


Figure 2. ECC Mutual Authentication

The process of mutual authentication and key exchange using the RSA algorithm begins with the client device contacting the server. The server that receives the message sends the public key and certificate that the server has. The client device verifies the certificate. If the certificate is valid, the client encrypts the client's certificate using the server's public key. Next, the client device will combine the ciphertext and public key of the client device with an explicit delimiter. The server that receives the message will share the ciphertext and public key of the client device. Then, the server decrypts the message with the private key from the server and verifies the client device's certificate. If the certificate is valid, the server encrypts the shared key and sends it using the client device's public key. The client device receives the ciphertext and decrypts it using the client device's private key to obtain the shared key. The mutual authentication process using RSA can be seen in Figure 3.

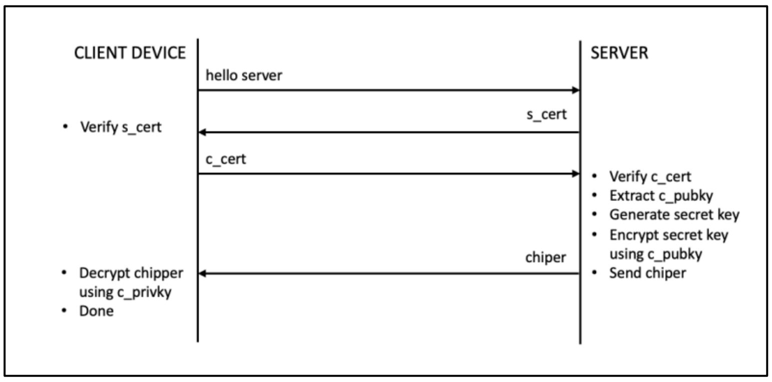


Figure 3. RSA Mutual Authentication

2.3 *Registration Process*

The recorded fingerprint and RFID data will be encrypted using ECDH symmetric keys and AES algorithms. The ciphertext will be combined with a random Initialization Vector (IV) [17] with a delimiter and then sent to the server. After the server receives data from the client device, the received symmetric and IV keys decrypt the data. The data will be stored in JSON, where RFID data becomes the key and fingerprint data becomes the value. Next, the server will notify the client device that the registration has been successfully carried out by encrypting the message. The client device will display a message based on the response from the server to indicate whether the registration was successful or failed. The overall registration process is shown in Figure 4.

2.4 *Verification Process*

The recorded fingerprint and RFID data are combined into one plaintext and limited by a delimiter. The plaintext is then encrypted using its symmetric key, random IV, and the AES-128 algorithm. The ciphertext is then combined into one with the IV and bounded by a delimiter. Then, that payload is sent to the server. Once the server receives the encrypted message, the payload will be separated between the ciphertext and IV. The server will decrypt the ciphertext using an IV that is sent with the server's symmetric key. Then, the RFID data plaintext will be separated from the fingerprint data plaintext. The following process matches the data with the registered data. If the matching results indicate a match, the server responds to the client device by sending a message that the user is verified and encrypting the message before sending it to the client device in the same way that the client device sends a message to the server. The overall verification process is shown in Figure 4.

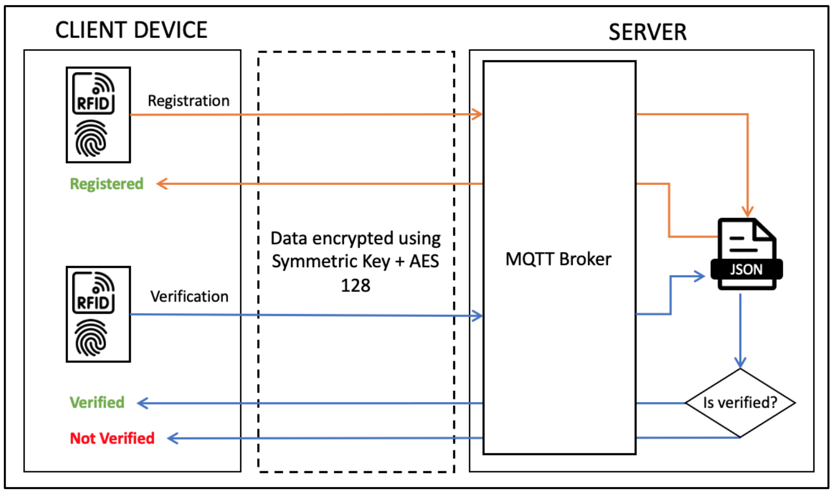


Figure 4. Registration and Verification Process

3. Result and Evaluation

3.1 Speed of Mutual Authentication

At this stage, speed testing is performed on the mutual authentication process until the client and server devices have shared keys using ECC and RSA. The first test was conducted to test the speed of mutual authentication using different key sizes by NIST recommendations for minimum legacy systems [21], as shown in Table 2. Then, the second test was conducted using public and private keys, with a key length of 256 bits for ECC and 3072 bits for RSA by NIST recommendations for minimum real-time authentication [21]. This test was conducted one hundred times. The sample speed results are shown in Table 3.

Table 2. Time Execution by Key Size

Key Size (bit)		Time Execution (ms)	
ECC	RSA	ECC	RSA
224	2048	153.77	219.69
256	3072	167.10	248.03
384	7680	189.01	586.40
512	15360	235.66	2988.13

In the speed measurement using different key sizes, the system shows different execution time results between mutual authentication processes using ECC and RSA algorithms. ECC consistently shows faster execution times than RSA for all key sizes tested. This also happened in speed measurement testing using 256-bit ECC and 3072-bit RSA. The average execution time of mutual authentication using ECC was 153.24 ms, while the average execution time using RSA was 216.27 ms.

Table 3. Time Execution 256-bit ECC and 3072-bit RSA

No.	ECC (ms)	RSA (ms)
1.	138.84	193.09
2.	152.39	210.57
3.	131.13	215.08
4.	127.22	202.29
5.	135.87	393.41
6.	139.49	229.95
7.	140.39	202.10
8.	133.67	208.29
9.	139.47	205.36
10.	193.29	216.99
…	…	…
Average.	153.24	216.27

3.1.1 Execution Time Comparison

The average execution time of mutual authentication with ECC is faster than with RSA. This significant difference is due to the difference in key length between ECC and RSA, where ECC tends to have shorter keys. In addition, the key exchange process using RSA also takes longer because the RSA algorithm requires sending symmetric keys from the server to the client device, which adds complexity and overall execution time.

P-ISSN: 1411-3414
E-ISSN: 2549-9815

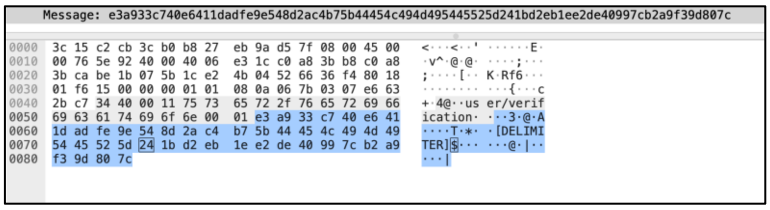
3.1.2 Computing Efficiency

ECC uses scalar multiplication operations on elliptic curves, which are computationally less intensive than the modular multiplication and large integer factorization required by RSA. This efficiency in ECC allows it to perform cryptographic operations quicker, making it more suitable for devices with limited processing power and memory.

This analysis shows that ECC offers significant advantages in speed and efficiency over RSA, making it a better choice for many modern applications that require strong and efficient cryptographic security.

3.2 Sniffing Attack

A sniffing attack is a technique attackers use to steal or spy on data sent over a network. In this test scenario, the attacker is already on the network between the client device and the server. Attackers use Wireshark software to monitor network traffic. This software is installed on a computer device connected to the same network as the system to be tested. After that, the attacker activates the sniffer software and monitors network traffic. The software automatically filters and records the data packets sent over the network.

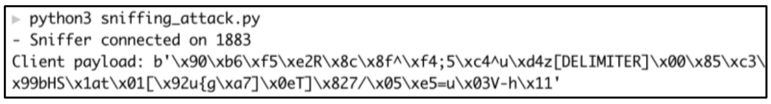


```
Message: e3a933c740e6411dadfe9e548d2ac4b75b44454c494d495445525d241bd2eb1ee2de40997cb2a9f39d807c

0000  3c 15 c2 cb 3c b0 b8 27  eb 9a d5 7f 08 00 45 00   <···<··' ······E·
0010  00 76 5e 92 40 00 40 06  e3 1c c0 a8 3b b8 c0 a8   ·v^·@·@· ····;···
0020  3b ca be 1b 07 5b 1c e2  4b 04 52 66 36 f4 80 18   ;····[·· K·Rf6···
0030  01 f6 15 00 00 00 01 01  08 0a 06 7b 03 07 e6 63   ········ ···{···c
0040  2b c7 34 40 00 11 75 73  65 72 2f 76 65 72 69 66   +·4@··us er/verif
0050  69 63 61 74 69 6f 6e 00  01 e3 a9 33 c7 40 e6 41   ication· ···3·@·A
0060  1d ad fe 9e 54 8d 2a c4  b7 5b 44 45 4c 49 4d 49   ····T·*· ·[DELIMI
0070  54 45 52 5d 24 1b d2 eb  1e e2 de 40 99 7c b2 a9   TER]$··· ···@·|··
0080  f3 9d 80 7c                                        ···|
```

Figure 5. Sniffing Attack Raw

If an attacker captures the message, the next step is to analyze the message structure, which will look like the one depicted in Figure 5. If an attacker creates a program to capture the message and decodes the captured message, the message will look like in Figure 6. This message is shared by a "[DELIMITER]" sign, which indicates a delimiter between IV on the left and ciphertext on the right, encrypted using AES-128.

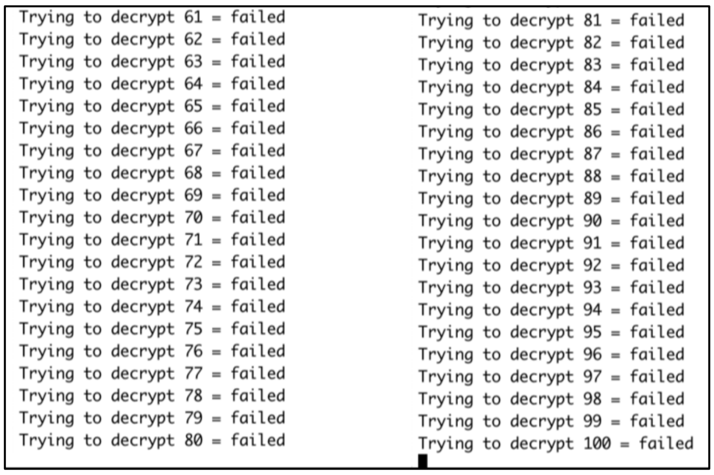


```
▶ python3 sniffing_attack.py
- Sniffer connected on 1883
Client payload: b'\x90\xb6\xf5\xe2R\x8c\x8f^\xf4;5\xc4^u\xd4z[DELIMITER]\x00\x85\xc3\
x99bHS\x1at\x01[\x92u{g\xa7]\x0eT]\x827/\x05\xe5=u\x03V-h\x11'
```

Figure 6. Sniffing Attack Decoded

3.3 Brute Force Attack

In this scenario, a security system test involving a brute force attack was performed to decrypt the captured message. First, the attacker prepares an automated script to perform a brute-force attack on the captured message.



```
Trying to decrypt 61 = failed        Trying to decrypt 81 = failed
Trying to decrypt 62 = failed        Trying to decrypt 82 = failed
Trying to decrypt 63 = failed        Trying to decrypt 83 = failed
Trying to decrypt 64 = failed        Trying to decrypt 84 = failed
Trying to decrypt 65 = failed        Trying to decrypt 85 = failed
Trying to decrypt 66 = failed        Trying to decrypt 86 = failed
Trying to decrypt 67 = failed        Trying to decrypt 87 = failed
Trying to decrypt 68 = failed        Trying to decrypt 88 = failed
Trying to decrypt 69 = failed        Trying to decrypt 89 = failed
Trying to decrypt 70 = failed        Trying to decrypt 90 = failed
Trying to decrypt 71 = failed        Trying to decrypt 91 = failed
Trying to decrypt 72 = failed        Trying to decrypt 92 = failed
Trying to decrypt 73 = failed        Trying to decrypt 93 = failed
Trying to decrypt 74 = failed        Trying to decrypt 94 = failed
Trying to decrypt 75 = failed        Trying to decrypt 95 = failed
Trying to decrypt 76 = failed        Trying to decrypt 96 = failed
Trying to decrypt 77 = failed        Trying to decrypt 97 = failed
Trying to decrypt 78 = failed        Trying to decrypt 98 = failed
Trying to decrypt 79 = failed        Trying to decrypt 99 = failed
Trying to decrypt 80 = failed        Trying to decrypt 100 = failed
```

Figure 7. Brute Force Attack

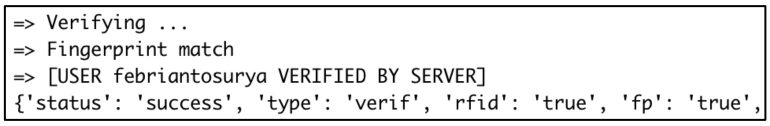
This script is designed to automatically try different password or encryption key combinations until successfully decrypting the message or gaining access to the desired information. After that, the attacker runs a brute-force script on the captured message to try out all possible encryption key or password combinations. This script repeatedly tries many combinations 100 times, as shown in Figure 7.

Brute-force attempts were only made 100 times because performing a brute-force attack to decrypt AES 128-encrypted ciphertext had 2^{128} possibilities, many of which were significantly time-consuming to execute effectively. In addition, IVs for encrypting messages are also randomly generated so that each message sent has a different pattern and increases the overall level of system security.

3.4 Replay Attack

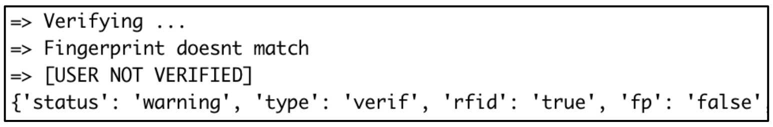
In this scenario, the attacker tries to exploit a vulnerability in a communication protocol that does not have a strong authentication or timestamp verification mechanism. This replay attack aims to grant the client device access to the user by displaying a successful verification condition. This attack is done by resending the message sent by the server to the client device when verification is successful. There are three conditions when the client device operates: verification successful, verification failed, and a replay attack detected. The status will display success when successful verification is shown in Figure 8. However, if verification fails, either RFID only, fingerprint only, or both are incorrect, the status will display a warning, as shown in Figure 9. Additionally, if a replay attack is detected, the status will display hazards, as shown in Figure 10. A sniffing attack records message traffic so a verification message can be obtained from the server and sent to the client device.

The recorded message, as shown in Figure 11, is sent to the broker so that the client device can provide access to the user through this attack. However, the result of this attack failed, as shown in Figure 10. This attack is due to a system that applies a timestamp to each message received and ensures that the message's sender is a trusted party. Thus, even if the attacker manages to record and transmit a verification message, the system can detect and reject the message.



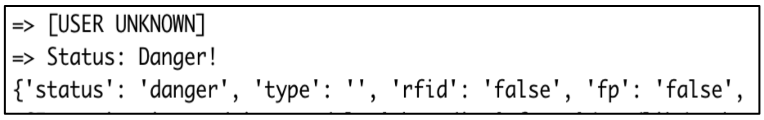
```
=> Verifying ...
=> Fingerprint match
=> [USER febriantosurya VERIFIED BY SERVER]
{'status': 'success', 'type': 'verif', 'rfid': 'true', 'fp': 'true',
```

Figure 8. Client Device on Verification Failed



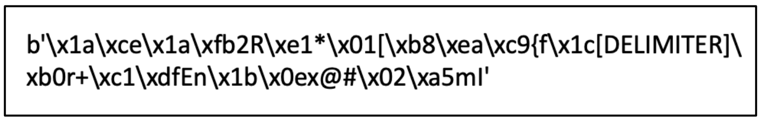
```
=> Verifying ...
=> Fingerprint doesnt match
=> [USER NOT VERIFIED]
{'status': 'warning', 'type': 'verif', 'rfid': 'true', 'fp': 'false'
```

Figure 9. Client Device on Verification Failed



```
=> [USER UNKNOWN]
=> Status: Danger!
{'status': 'danger', 'type': '', 'rfid': 'false', 'fp': 'false',
```

Figure 10. Client Device on Handling Replay Attack



```
b'\x1a\xce\x1a\xfb2R\xe1*\x01[\xb8\xea\xc9{f\x1c[DELIMITER]\
xb0r+\xc1\xdfEn\x1b\x0ex@#\x02\xa5ml'
```

Figure 11. Replay Attack Massage

3.5 Fingerprint Accuracy

Accuracy testing of fingerprint scanning uses FAR and FRR to measure the performance of the fingerprint-matching system. The test was conducted 80 times using valid fingerprints (which should have been accepted by the system) and fake fingerprints (which should have been rejected by the system). Next, each threshold, with every multiple of 10 from 0 to 100, is evaluated to find the appropriate accuracy for fingerprint matching. The threshold used is the similarity of each fingerprint data. The results of fingerprint data matching tests for each threshold are shown in Table 4.

Table 4. FRR and FRR based on Threshold

Threshold	FRR	FAR
0	0%	100%
10	0%	98.33%
20	3.39%	91.67%
30	16.95%	56.67%
40	32.20%	10.00%
50	50.85%	0%
60	50.85%	0%
70	62.71%	0%
80	98.31%	0%
90	100%	0%
100	100%	0%

The results of the fingerprint accuracy testing in Table 4 show that the higher the threshold value, the higher the False Rejection Rate (FRR) increases. At the same time, the False Acceptance Rate (FAR) tends to decrease. This result suggests that as the threshold increases, systems tend to be more selective in accepting correct fingerprints and increase the risk of rejecting legitimate fingerprints. Based on the data in Table 4, the FRR and FAR results are depicted in a graph, as shown in Figure 12.

Figure 12 shows that FAR experiences a significant decrease in the threshold range of 10 to 40, while FRR increases from the threshold range of 10 to 50. Looking at the FAR and FRR curves, they intersect at the threshold range of 30 to 40 with a percentage of 20 to 30 percent. This intersection point is the Equal Error Rate (ERR), where the system experiences the same error in accepting false fingerprints and rejecting valid fingerprints. Therefore, a threshold around 40 could be a good choice because, at the threshold value of 40, there is a better balance between the likelihood of recognizing valid fingerprints and maintaining security against false fingerprints. The ERR value can still be optimized by improving the dataset quality, sensor quality, and matching algorithms.

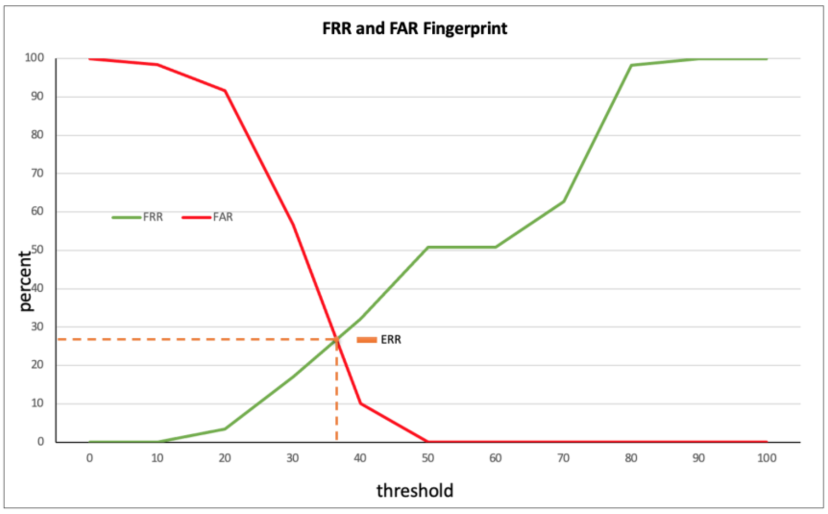


Figure 12. FRR and FAR Fingerprint

4. Conclusion

Based on the test results and analysis, several significant findings are related to the security of communication systems and data protection. First, the comparison of average times for mutual authentication between ECC and RSA shows that ECC is faster, with an average execution time of 153.24 ms, compared to RSA, which requires an average execution time of 216.27 ms. This result is due to the influence of the ECC algorithm, which uses shorter key lengths and faster computation. Additionally, the tests indicate that ECC maintains a significant speed advantage over RSA as key sizes increase, making it a more efficient choice for cryptographic operations. Second, testing against sniffing attack threats shows that even though attackers successfully record messages sent between the client device and the server, the messages remain protected because they have been encrypted using AES-128. Third, brute force attacks on the security system were also conducted, but the very high complexity of AES-128 and randomly generated IVs increased the system's security. Fourth, attackers attempted to exploit vulnerabilities in the communication protocol by recording and resending verification messages. Still, this attempt failed because the system implemented a time limit for each received message. Finally, fingerprint accuracy testing shows that the threshold value between 30% and 40% is the best choice as it provides a good balance between the likelihood of recognizing valid fingerprints and maintaining security against false fingerprints. The results demonstrate that the proposed system is time-efficient and resilient to certain types of attacks. While this study tested with Raspberry Pi 3, the system's design applies to IoT devices with lower specifications, such as the MCU ESP32. Compatibility across different IoT devices and robust network infrastructure are crucial for large-scale implementation. Future research should explore the applicability to a broader range of devices and diverse network environments. These insights provide a clearer roadmap for enhancing IoT security on a larger scale.

References

[1] W. Yang, S. Wang, N. M. Sahri, N. M. Karie, M. Ahmed, and C. Valli, “Biometrics for internet-of-things security: A review,” *Sensors*, vol. 21, no. 18. MDPI, Sep. 01, 2021. doi: 10.3390/s21186163.

[2] K. L. Lueth, “State of the IoT 2020: 12 Billion IoT Connections Surpassing Non-IoT for the First Time,” IoT Analytics. [Online]. Available: https://iot-analytics.com/author/knud-lasse-lueth/

[3] A. Shamsoshoara, A. Korenda, F. Afghah, and S. Zeadally, “A survey on physical unclonable function (PUF)-based security solutions for Internet of Things,” *Comput. Networks*, vol. 183, p. 107593, 2020, doi: https://doi.org/10.1016/j.comnet.2020.107593.

[4] Z. Hussain, A. Akhunzada, J. Iqbal, I. Bibi, and A. Gani, “Secure IIoT-enabled industry 4.0,” *Sustainability (Switzerland)*, vol. 13, no. 22, Nov. 2021, doi: 10.3390/su132212384.

[5] Q. A. Al-Haija and S. Zein-Sabatto, “An efficient deep-learning-based detection and classification system for cyber-attacks in iot communication networks,” *Electronics (Switzerland)*, vol. 9, no. 12, pp. 1–26, Dec. 2020, doi: 10.3390/electronics9122152.

[6] K. E. Balto, M. M. Yamin, A. Shalaginov, and B. Katt, “Hybrid IoT Cyber Range,” *Sensors*, vol. 23, no. 6, Mar. 2023, doi: 10.3390/s23063071.

[7] J. C. Yang, H. Pang, and X. Zhang, “Enhanced mutual authentication model of IoT,” *Journal of China Universities of Posts and Telecommunications*, vol. 20, no. SUPPL-2, pp. 69–74, Dec. 2013, doi: 10.1016/S1005-8885(13)60218-6.

[8] S. D. Mohammed, A. M. S. Rahma, and T. M. Hasan, “Maintaining the integrity of encrypted data by using the improving hash function based on GF (28),” *TEM Journal*, vol. 9, no. 3, pp. 1277–1284, Aug. 2020, doi: 10.18421/TEM93-57.

[9] R. R. Pahlevi, V. Suryani, H. H. Nuha, and R. Yasirandi, “Secure Two-Factor Authentication for IoT Device,” in *2022 10th International Conference on Information and Communication Technology, ICoICT 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 407–412. doi: 10.1109/ICoICT55009.2022.9914866.

[10] D. Mahto and D. Kumar Yadav, “RSA and ECC: A Comparative Analysis,” 2017. [Online]. Available: http://www.ripublication.com

[11] M. Al Saadi, “A Review on Elliptic Curve Cryptography,” *International Journal of Future Generation Communication and Networking*, vol. 13, no. 3, pp. 1597–1601, 2020, [Online]. Available: https://www.researchgate.net/publication/350048546

[12] X. Wang and M. El-Said, “DomainPKI: Domain Aware Certificate Management,” in *SIGITE 2020 - Proceedings of the 21st Annual Conference on Information Technology Education*,

Association for Computing Machinery, Inc, Oct. 2020, pp. 419–425. doi: 10.1145/3368308.3415401.

[13] H. N. Almajed and A. S. Almogren, “SE-Enc: A Secure and Efficient Encoding Scheme Using Elliptic Curve Cryptography,” *IEEE Access*, vol. 7, pp. 175865–175878, 2019, doi: 10.1109/ACCESS.2019.2957943.

[14] E. Taiwo Oladipupo and O. Christiana Abikoye, “Improved authenticated elliptic curve cryptography scheme for resource starve applications,” *Computer Science and Information Technologies*, vol. 3, no. 3, pp. 169–185, 2022, doi: 10.11591/csit.v3i3.pp169-185.

[15] A. Saepulrohman, A. Denih, Sukono, and A. T. Bon, “Elliptic Curve Diffie-Hellman Cryptosystem for Public Exchange Process,” in *5th North American International Conference on Industrial Engineering and Operations Management*, IEOM Society International, 2020.

[16] S. Heron, “Advanced Encryption Standard (AES),” *Network Security*, vol. 2009, no. 12, pp. 8–12, Dec. 2009, doi: 10.1016/S1353-4858(10)70006-4.

[17] H. K. S. Premadasa and R. G. N. Meegama, “Extensive compression of text messages in interactive mobile communication,” in *2013 International Conference on Advances in ICT for Emerging Regions (ICTer)*, IEEE, Dec. 2013, pp. 80–83. doi: 10.1109/ICTer.2013.6761159.

[18] Y. Im and M. Lim, “E-MQTT: End-to-End Synchronous and Asynchronous Communication Mechanisms in MQTT Protocol,” *Applied Sciences*, vol. 13, no. 22, p. 12419, Nov. 2023, doi: 10.3390/app132212419.

[19] O. Gaikwad, P. SP, M. Kantimahanti, M. Kamthe, and L. Kumar, “RFID Attendence using RC522,” *Int J Res Appl Sci Eng Technol*, vol. 8, no. 5, pp. 2386–2392, May 2020, doi: 10.22214/ijraset.2020.5392.

[20] J. Linggarjati, “Raspberry Pi Zero Door Locking System with Face Recognition using CNN (Convolutional Neural Network) and Fingerprint Sensor,” in *Proceedings of the International Conference on Industrial Engineering and Operations Management*, Michigan, USA: IEOM Society International, 2022, pp. 1147–1152. doi: 10.46254/SA03.20220247.

[21] E. Barker, “Recommendation for key management: Part 1 - General,” Gaithersburg, MD, May 2020. doi: 10.6028/NIST.SP.800-57pt1r5.