Implementation and Performance Evaluation of Elliptic Curve Cryptography over SECP256R1 on STM32 Microprocessor

Onur İŞLER
R&D
TÜRKTRUST
Ankara, TÜRKİYE
onur.isler@turktrust.com.tr

***Abstract*—The use of Internet of Things (IoT) devices in embedded systems has become increasingly popular with advancing technologies. These devices become vulnerable to cyber attacks as they gain popularity. The cryptographic operations performed for the purpose of protection against cyber attacks are crucial to yield fast results in open networks and not slow down network traffic. Therefore, to enhance communication security, studies have been conducted in the literature on using asymmetric encryption and symmetric encryption together in IoT devices for activities such as key sharing, encryption, decryption, data signing, and verifying signed data. In this study, we first propose a cryptographic system engaging of IoT devices operated from a server. Then we do performance analysis of our proposal. In particular, we evaluate the elliptic curve Diffie-Hellman key exchange and elliptic curve digital signature algorithms on the Secp256r1 elliptic curve and AES symmetric encryption via the Micro uECC library conducted with the 32-bit STM32F410RB Nucleo development board microprocessor running at 48 MHz.**

***Index Terms*—Key Exchange, Digital Signature, Elliptic Curve, Secp256r1, IoT**

I. Introduction

The usage of embedded systems, increasingly prevalent in our homes, not only makes our lives more convenient but also brings along security risks. These embedded systems have made our homes smarter through Internet of Things (IoT) devices connected to home networks. However, these devices can be vulnerable to cyber attacks, putting home security at risk [3], [11]. Cyber attackers can infiltrate IoT devices on home networks, gaining access to devices such as cameras, audio devices, or internet-connected kitchen appliances. This situation can jeopardize user privacy and security.

Attackers, by compromising these devices over the internet, can impersonate users, steal, record sensitive data, or harm the devices. Therefore, ensuring the security of messages sent in the communication of embedded systems is of critical importance. Messages or commands used in communication between devices must be protected against external malicious interventions. This protection should be based on features such as confidentiality and non-repudiation in communication. Confidentiality in communication between devices means that messages or commands sent cannot be understood by external malicious actors. This is critical for information security, preventing malicious individuals from monitoring or understanding communication. Non-repudiation, on the other hand, is another important factor that secures communication between devices. Malicious actors may perform harmful actions by impersonating users or other devices. Therefore, it is crucial for devices to authenticate themselves and their communication securely. However, during the implementation of these security measures, there should be no disruption in communication between devices. Security measures in communication should enable smooth interaction between devices without impacting performance.

To ensure the security of data transmission between devices, cryptographic algorithms and protocols are employed. However, these cryptographic algorithms generally require more powerful processors to perform mathematical calculations. IoT devices operate with lower power consumption but lower performance and processing limits. Therefore, cryptographic operations in IoT devices may lead to communication delays. To address this, lightweight algorithms that do not compromise security but do not hinder device performance should be preferred. For instance, Diffie-Hellman (DH) key exchange algorithm is used for generating and sharing a common cryptographic key during symmetric encryption, especially in Device-to-Device (D2D) data transfer over insecure networks. If the DH key exchange algorithm is defined on an elliptic curve, it tends to be faster than finite field version [22]. Elliptic curve cryptography (ECC) offers a lower key size compared to finite field public key cryptography, providing similar security levels, see [17] for security levels, and [5] for their faster implementation. As the equal security level is satisfied by lower key sizes, ECC requires less number of field operations [8, Section 5]. This makes it more appealing for use in IoT devices with low performance and limited resources than RSA and DH variants. For instance, in [13] it was shown that RSA exponentiation takes about 10x longer than an equivalently

This study is supported by TÜBİTAK under the project number TEYDEP-3192166

secure scalar multiplication over an elliptic curve on resource constrained devices.

The Standards for Efficient Cryptography (SEC) academic committee is a well known non-profit community for elliptic curve cryptography (ECC) and they proposed a set of parameters for implementing ECC [4]. We preferred a randomly selected curve for a security level of 128-bits only. Therefore, in this work, we implemented our system on the recommended curve Secp256r1. But other non-random curves and curves from other security levels would be considered in another work.

In this study, a protocol to secure a client's communication with an IoT device thorough a central server is established by using cryptographic methods. Elliptic curve DH (ECDH) key exchange mechanism is established for a secure key establishment between IoT device and the server. Elliptic curve digital signature algorithm (ECDSA) is used for authenticating both sides. In addition, the symmetric encryption AES [1] in cipher-block-chaining (CBC) mode [18] is used for securing the communication between IoT device and server. The performance evaluation of the ECDH key exchange, ECDSA signature generation and verification on the randomly generated Secp256r1 elliptic curve, and AES-CBC encryption are conducted on an IoT device STM32F410RB Nucleo development board microprocessor [19] for cryptographic processing times during the communication. The performance of the protocol on the IoT device is calculated through time measurements.

II. Related Work

Research into the implementation of cryptographic algorithms on MCUs primarily emphasizes performance, memory usage, and energy consumption. For instance, a study by Liu et al. [10] implements elliptic curve cryptography on the 8-bit ATmega128 processor (used in the MicaZ) and on the MSP430 (utilized in the Tmote Sky), focusing on optimizing execution speed and memory usage. Similarly, Wenger et. al. [21] evaluates three popular microprocessors (Cortex-M0+, MSP430 and ATmega) based on their runtime, chip area, power, and energy consumption in standard side-channel protected elliptic curve cryptography. Results indicate that the Cortex-M0+ excels in speed and energy efficiency, ideal for Wireless Sensor Nodes; the MSP430 enables compact and low-power designs, suitable for RFID tags; and the ATmega performs best with instruction-set modifications, making it suitable for long-lived products requiring ECC.

Later, Ledwaba et al. [9] examined the costs associated with cryptographic software on contemporary end-point devices, analyzing the performance of AES-CTR, SHA256, and ECDSA algorithms across various ARM Cortex-M devices.

Kane et. al. [14] analyzed performance metrics of microcontroller devices commonly used in IoT applications. In particular, AES, ChaCha, and Acorn ciphers were evaluated on three microcontroller devices STM32F103C8T6, ATmega328, and ESP8266 Wi-Fi Witty Cloud Development Board with measurements taken for power consumption, time cost, energy cost, peak RAM usage, and flash usage. Results were compared, and the STM32F103C8T6 emerged as a balanced choice for IoT deployments, offering good performance and speed.

A detailed analysis was conducted in [20] to estimate the computation and communication energy expenses of algorithms for end-to-end security and digital signatures. Measurements were taken across three platforms (STM32L073RZT6-ARM Cortex-M0+, MSP432P401R-ARM Cortex-M4F, and MAX32620-ARM Cortex-M4F) utilizing three distinct wireless communication protocols (BLE, Wi-Fi, and LoRaWAN).

Apart from these studies, we in this paper focus on the algorithms required to establish an end-to-end secure communication. In particular, we implemented elliptic curve cryptography algorithms including key generation, signing, verification and shared secret calculation, and AES-CBC for symmetric encryption on the STM32F410RB-ARM Cortex-M4F Nucleo development board.

III. Background

A. Finite Fields

Finite fields, also known as Galois fields, constitute a fundamental concept in algebra and cryptography. A finite field is a mathematical structure that consists of a finite set of elements along with two binary operations, addition and multiplication. The order of a finite field, denoted as q, represents the number of elements in the field, which can be only a prime power integer e.g. $q = p^n$ for some prime number p. Finite fields find extensive applications in various areas, particularly in error-correcting codes, cryptography, and algebraic coding theory. In cryptography, they play a crucial role in algorithms such as elliptic curve cryptography, where the arithmetic operations are performed within a finite field to ensure security and efficiency.

Let p be prime number. A finite field $\mathbb{F}_p$ containing p elements is called a prime finite field. While there is a unique finite field $\mathbb{F}_p$ for each odd prime p, there are many different ways to represent the elements of $\mathbb{F}_p$. The elements of $\mathbb{F}_p$ are represented here as $\{0, 1, \ldots, p-1\}$, with addition and multiplication operations performed using the standard addition and multiplication operations under modulo p. The set $\mathbb{F}_p^*$ denotes the multiplicative group of elements in $\{1, \ldots, p-1\}$ under multiplication modulo p.

B. Elliptic Curve Cryptography (ECC)

The discrete logarithm problem is a cryptographic challenge based on the difficulty of computing a certain element's specified exponent within a mathematical group G. This problem plays a fundamental role in cryptographic protocols such as El-Gamal Encryption, Diffie-Hellman Key Exchange, and the Digital Signature Algorithm (DSA), where $G = F_p^*$ is often considered. Specifically, when examining the discrete logarithm problem on a group of points on an elliptic curve over a finite field, it refers to the difficulty of calculating the result of multiplying a certain element by a predetermined scalar on the curve. For instance, given a pair P, Q on an elliptic curve and knowing that Q is a scalar multiple of P, finding this scalar is

known as EC discrete logarithm problem and it is believed to be one of the hard problems in mathematics. Currently, a 256-bit finite field provides a sufficient security level for securing data transferring on the web. For greater security or long-term durability, curves defined on larger finite fields can be chosen. For example, Secp256r1 is an elliptic curve proposed by The Standards for Efficient Cryptography Group (SECG) [4]. This parameter set is defined for use in digital signature algorithms and key exchange protocols.

Secp256r1 represents an elliptic curve over a prime field, given by the equation $y^2 = x^3 - 3x + b \mod p$, where $p = 2^{224}(2^{32}-1)+2^{192}+2^{96}-1$ is a prime, and a, b are coefficients defining the curve, where $a = -3$ and b is randomly selected integer. The Secp256r1 curve provides a 256-bit security level and is particularly favored in digital signature algorithms like ECDSA (Elliptic Curve Digital Signature Algorithm) and key exchange protocols like ECDH (Elliptic Curve Diffie-Hellman). Secp256r1 is also standardized by NIST as P-256, and is widely used in various cryptographic applications e.g. in Transport Layer Security (TLS) of web communication.

C. Diffie-Hellman Key Exchange

The Diffie-Hellman (DH) protocol is the first example of asymmetric cryptography, used to securely facilitate key exchange. Typically, two parties, often referred to as Alice and Bob, create a mutual agreement to share secret keys. This protocol allows the parties to determine a shared secret key before communicating with each other. It was proposed by W. Diffie and M. Hellman in 1976 [6]. In this study, the Elliptic Curve Diffie-Hellman (ECDH) key exchange scheme based on the DH scheme is employed.

D. Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA [7] is essentially a version of the digital signature algorithm, where operations are performed on an elliptic curve. It is used in many applications as it performs the signing process more efficiently than DSA and RSA, primarily due to providing the same level of security with smaller key sizes [22]. In this study, we implement ECDSA on Secp256r1 curve for authentication.

In ECDSA, a private key and a public key are generated on an elliptic curve, where private key is an integer and public key is a point on the curve. The private key is kept secret, while the public key is shared with others. To sign a message, the private key is used to generate a digital signature. This signature is then attached to the message and sent to the recipient. To verify the authenticity of a signed message, the recipient uses the sender's public key to validate the digital signature. If the signature is valid, it means that the message has not been tampered with and that it was indeed signed by the sender.

E. Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES), also known as Rijndael, is a symmetric block cipher that was adopted as a U.S. Federal Information Processing Standard (FIPS) in 2001 [1]. It is one of the most widely used encryption algorithms in the world, and is used to protect sensitive data in a variety of applications, such as secure communication over the internet, data storage on any kind of devices, encrypting software programs to prevent unauthorized access or modification. AES uses a combination of mathematical operations, including substitution, permutation, and diffusion, to encrypt and decrypt data. It is considered to be a very secure algorithm and extensively analyzed by cryptographers around the world, see the selection procedure [2], and it is still an active standard available in [1]. AES is a block cipher, which means that it operates on blocks of data of a fixed size. The block size for AES is 128 bits, which means that each block of data that is encrypted or decrypted is 128 bits long. AES uses a key to encrypt and decrypt data. The key is a sequence of bits that is used to control the encryption and decryption process. The key size for AES can be 128, 192, or 256 bits. The larger the key size, the more secure the encryption is. AES is a very efficient algorithm, and can be implemented in embedded devices. In this study, AES-128 is implemented to secure the communication between IoT devices and server.

IV. MATERIAL AND METHOD

A. Development Board

The STM32F410RB series Nucleo development board [19] from STMicroelectronics has been used in this study. It is a part of the STM32 series and operates on the ARM Cortex-M4 processor core, providing a high-performance, energy-efficient, and reliable platform. It is particularly suitable for embedded systems, industrial automation, smart home applications, medical devices, portable devices, and similar application areas. However, this series does not have a Trusted Platform Module (TPM) module or Secure Module. Therefore, it is known that implementing key storage and prevention of external exposure may require additional effort with this model. Its main specifications are given below:

- Processor: It has ARM Cortex-M4 core, a 32-bit processor offering high-performance computational capabilities.
- Memory: It has a comprehensive memory configuration, including 128 KB Flash memory and 64 KB SRAM, expandable with external memory interfaces.
- Peripherals: It provides a range of integrated peripherals, including GPIO (General-Purpose Input/Output), USART, SPI, I2C, ADC (Analog-Digital Converter), Timers, and others.
- Communication: STM32F410RB supports popular communication protocols for serial communication, such as USART, SPI, and I2C. This can be used for communication with sensors, displays, memory cards, and other external devices.

B. ECC Library

There are many public libraries supporting elliptic curve cryptography, one of which is Micro uECC [16]. The Micro ECC library offers a robust defense against known side-channel attacks, ensuring the security of cryptographic operations. Implemented in C, with the option for GCC inline

assembly tailored for AVR, ARM, and Thumb platforms, it ensures efficient performance across various architectures, including 8, 32, and 64-bit systems. Notably, it boasts a compact code size and eliminates the need for dynamic memory allocation, enhancing its suitability for resource-constrained environments. Additionally, the library provides support for five standard SEC curves: Secp160r1, Secp192r1, Secp224r1, Secp256r1, and Secp256k1, catering to diverse cryptographic requirements. Its BSD 2-clause license promotes flexibility and ease of integration into different projects. See its web page for further details [16]. These features are important reasons for us to choose Micro ECC library in this work. STM32CubeIDE version 1.13.2 for STM32 family has been used in order to implement the libraries and develop code scripts.

C. The protocol

The protocol in this study involves data transmission from the client to the center, from the center to the device, from the device to the center, and from the center to the client. The STM32F410RB has been selected, and it is operated at 48 MHz to make a fair comparison with the microcontrollers with lower speeds. This setting is done through STM32CubeMX. In the scenario, for decrypting incoming data or encrypting data to be sent, devices need a symmetric key and distribution of this key in an insecure network. In this work, ECDH is applied for this key distribution, and AES-CBC is used for symmetric encryption. In order to authenticate the sender, ECDSA is applied at both sides. The general scenario is depicted in Figure 1.

Various stages of the system, including generating and distributing a shared key, encrypting and decrypting data, signing for sender and recipient authentication, are evaluated for the time consumption in this study. Different key pairs have been selected for key establishment and signing/verification processes, which creates an additional security layer even if the key is compromised.

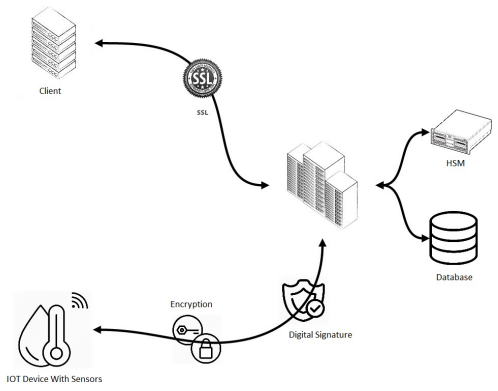

Fig. 1. General Scenario

In the scenario depicted in Figure 1, our objective is to establish secure communication among devices situated at remote locations from the central server, with each IoT device possessing a unique shared key with the server. Additionally, at intervals specified by the server, each device regenerates its key pair to update the shared key with the server and notifies the central server accordingly. It's important to note that the key pair utilized for key establishment differs from the one employed for signing, and a server-specific certificate is generated for the signing key pair, valid for the duration specified in the certificate.

All messages directed to the devices from the client are routed through the TÜRKTRUST server before reaching the device, and similarly, the response messages from the devices also pass through TÜRKTRUST to reach the client. Each command issued by the client undergoes encryption and signing at the TÜRKTRUST server before transmission to the device, with each signing operation executed by the HSM using securely stored keys. On the device end, incoming messages are initially verified, then decrypted, and subsequently, the requisite commands (such as temperature or humidity measurements) are executed. Upon readiness to transmit data, the device encrypts the message and employs a distinct key pair for signing. The server verifies, decrypts, and forwards the message to the client, thus safeguarding against interception, alteration, or capture by unauthorized parties.

V. Results

The project's initial settings were done with STM32CubeMX, a screenshot can be seen in Figure 2. Then, in STM32CubeIDE, relevant settings and project files were opened, and evaluation operations were coded into the appropriate steps of the protocol. Measurements of the operations were done with a time counter embedded into the code.

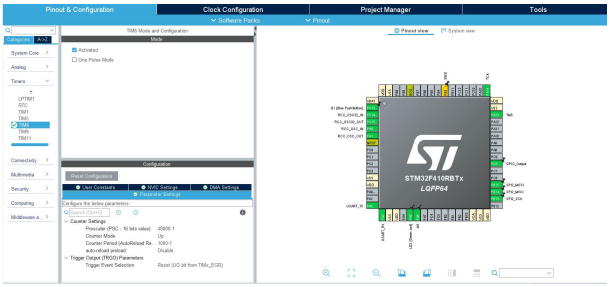

Fig. 2. Timer Setup on STM32CubeMX

In measurement with a timer, all operations were run 1000 times and the minimum time, maximum time and average time were calculated in these cycles.

TABLE I
Generating key pair for encryption on Secp256r1

Generating key pair for encryption on Secp256r1					
	Time (ms)			***Average Value***	***Median Value***
	283	284	285	284,244	284
Frequency	22	712	266		

As seen in Table 1, the key pair to be used in the calculation of the public key is generated 22 times in 283 ms, 712 times in 284 ms, and 266 times in 285 ms. We conclude that the minimum and maximum key generation time of the key pair are 283 ms and 285 ms, respectively. We illustrated the timings in Figure 3 and Figure 4. From Table 1 we obtain that the average time is 284,244 ms and the median value is 284 ms.

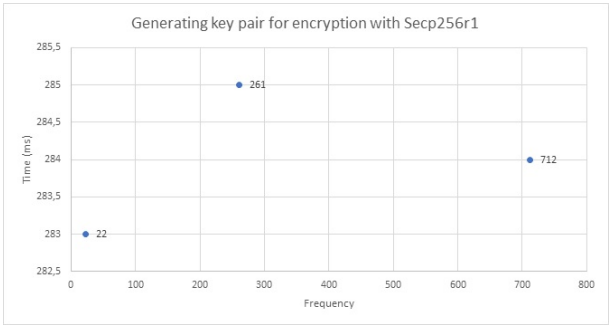

Fig. 3. Generating key pair for encryption on Secp256r1

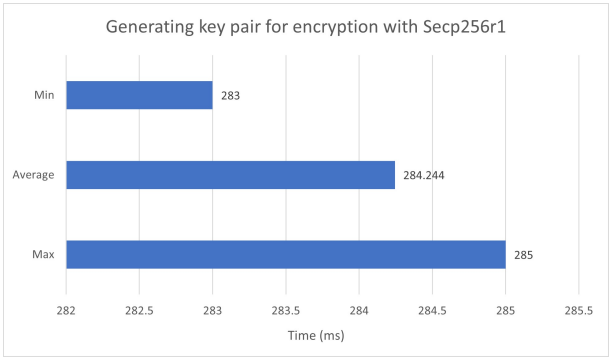

Fig. 4. Generating key pair for encryption on Secp256r1

Similarly, Table 2 tabulates generation time of the key pair which are used in signing and verification is 16 times 283 ms, 705 times 284 ms, 273 times 285 ms, 1 time 286 ms, 1 time 346 ms, 1 time 347 ms, 2 times 348 ms, 1 time in 361 ms. Figure 5 and Figure 6 show that the generation time of the key pair ranges between 283 ms and 361 ms, and the average is 284,589 ms.

TABLE II
GENERATING KEY PAIR FOR SIGNATURE ON SECP256R1

Generating key pair for signature on Secp256r1										
	Time (ms)								*Average Value*	*Median Value*
	283	284	285	286	346	347	348	361	284,589	284
Frequency	16	705	273	1	1	1	2	1		

As seen in Table 3, the calculation of the shared secret is produced 10 times in 283 ms, 707 times in 284 ms, and 283 times in 285 ms. Figure 7 and Figure 8 gives the calculation time of the public key with minimum 283 ms, maximum 285 ms, and average is 284,273 ms.

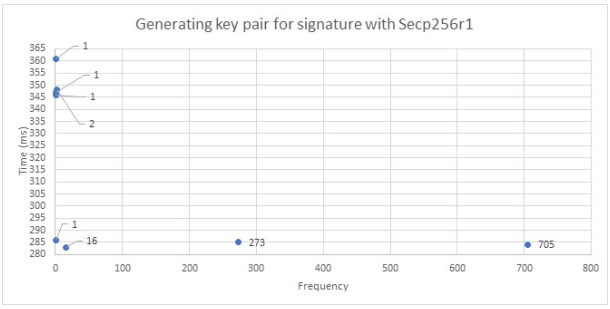

Fig. 5. Generating key pair for signature on Secp256r1

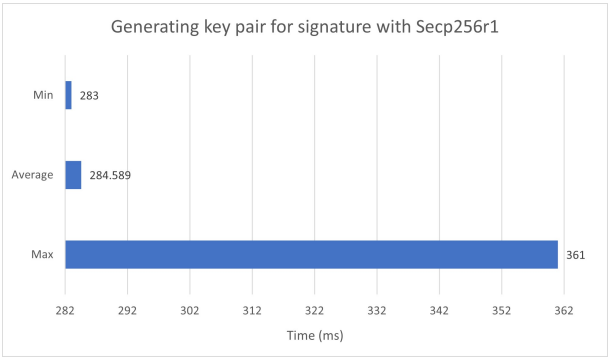

Fig. 6. Generating key pair for signature on Secp256r1

TABLE III
SHARED SECRET KEY CALCULATION

Shared Secret Key Calculation					
	Time (ms)			*Average Value*	*Median Value*
	283	284	285	284,273	284
Frequency	10	707	283		

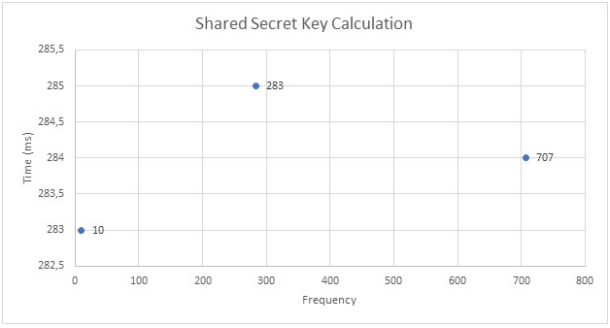

Fig. 7. Shared Secret Key Calculation

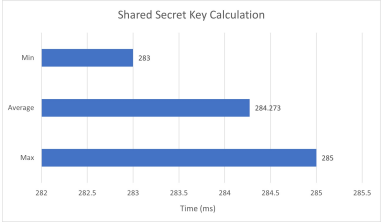

Fig. 8. Shared Secret Key Calculation

TABLE IV
AES ENCRYPTION

AES Encryption				
	Time (ms)		*Average Value*	*Median Value*
	2	3	2,816	3
Frequency	184	816		

The encryption process with AES 128 takes 2 ms 184 times and 3 ms 816 times, they are tabulated in Table 4. It lasts minimum 2 ms, maximum 3 ms, and the average is 2,816 ms, see Figure 9 and Figure 10.

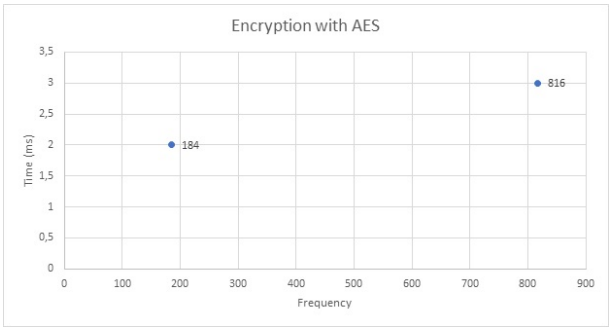

Fig. 9. AES Encryption

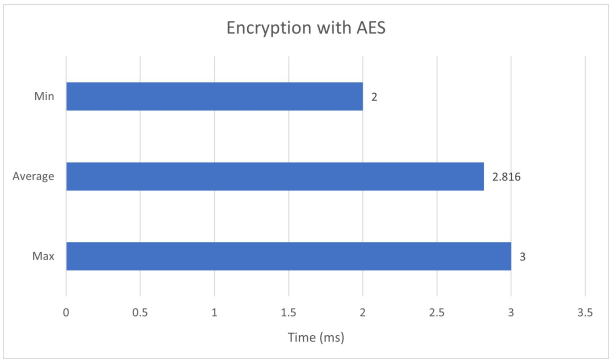

Fig. 10. AES Encryption

TABLE V
SIGNING MESSAGE

Sign						
	Time (ms)				*Average Value*	*Median Value*
	336	337	338	415	336,861	337
Frequency	254	708	37	1		

In our experiment, the signature generation on a fixed message takes 336 ms 254 times, 337 ms 708 times, 228 ms 37 times, and 415 ms once given in Table 5. Figure 11 and Figure 12 show the minimum, maximum and average time of signing process as 336 ms, 415 ms and 336,861 ms, respectively. On the other hand, the signature verification process takes 368 ms 446 times and 369 ms 554 times, and hence the minimum is 368 ms, the maximum is 369 ms, and the average is 368,554 ms, see Table 6, Figure 13 and Figure 14.

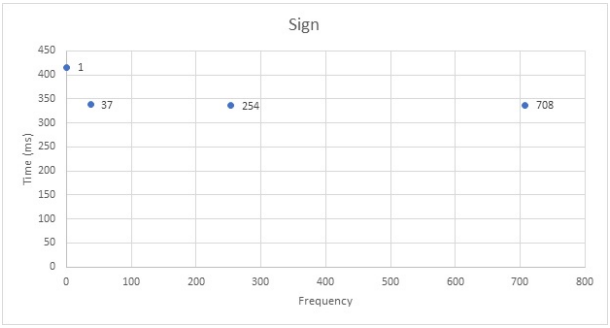

Fig. 11. Signing Message

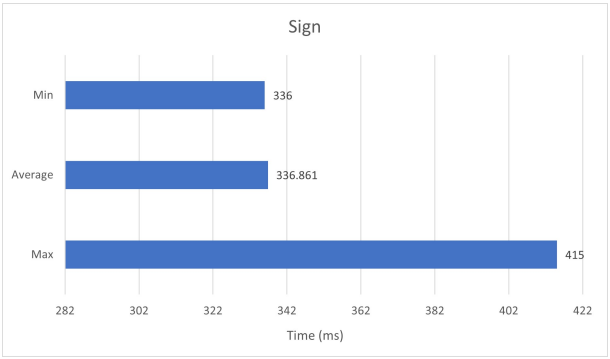

Fig. 12. Signing Message

TABLE VI
VERIFYING MESSAGE

Verify				
	Time (ms)		*Average Value*	*Median Value*
	368	369	368,554	369
Frequency	446	554		

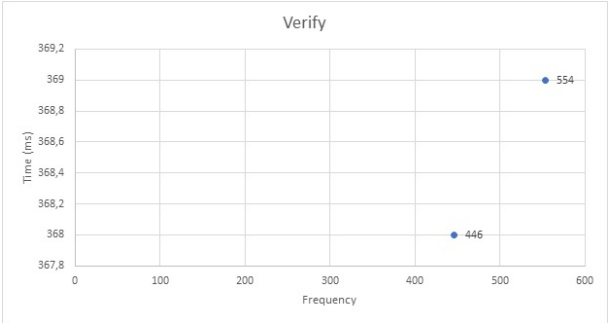

Fig. 13. Verifying Message

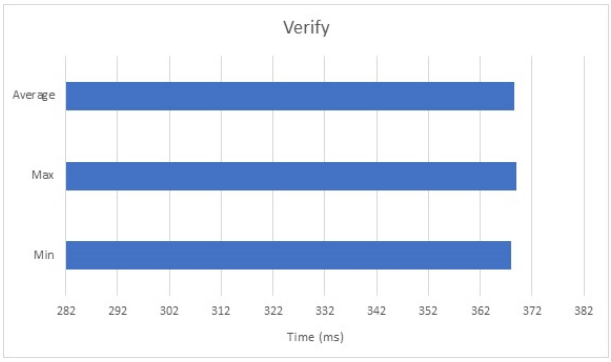

Fig. 14. Verifying Message

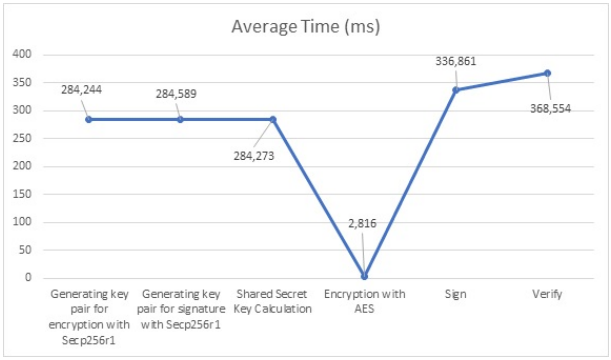

Fig. 15. Average of All Processes

Figure 15 shows the average duration of all transactions together. According to this graph, the total average time of all operations from start to finish = 1.561,337 ms, and again according to this graph, the time for the device to process a routine command (decrypting the message, verifying, encrypting and signing the new message, respectively) is 711,047 ms. Since the routine operation will be performed every time information is requested from the device, taking less than 1 second is considered a successful result in our project. In this way, the speed of information exchange is not affected much.

VI. Conclusion

The use of IoT devices in embedded systems is gaining popularity, but it also makes them vulnerable to cyber attacks. To protect against these attacks, cryptographic operations are crucial, and studies have explored using both asymmetric and symmetric encryption in IoT devices. This study proposes a novel approach for securing IoT devices in the wild operated form a server by using elliptic curve Diffie-Hellman (ECDH) key exchange, elliptic curve digital signature algorithm (ECDSA) and AES symmetric encryption. In addition, we analyzed the performance of DH key exchange, ECDSA and AES algorithms on the Secp256r1 elliptic curve in the Micro uECC library using a 32-bit STM 32F410RB Nucleo development board microprocessor from STMicroelectronics running at 48 MHz. As a result of our analysis, it is seen that ECDH key generation and key establishment take 284 ms approximately while ECDSA signing and verification take 336 ms and 368 ms respectively. On the other hand, AES algorithms takes only 2.816 ms approximately. The experiment analysis also shows that our implementation on the all cryptographic operations does not vary for different sessions, in other words they are constant time implementations.

References

[1] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," NIST FIPS 197-upd1, May 9, 2023. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.197-upd1.

[2] AES Development. [Online]. Available: https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development. [Accessed: May 8, 2023].

[3] M. Abomhara and GM Køien, "Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks," J. Cyber Secur. Mobil., vol. 4, no. 1, pp. 65-88, Jan. 2015.

[4] D. Brown, "SEC 2: Recommended elliptic curve domain parameters," [Online]. Available: https://www.secg.org/sec2-v2.pdf. [Accessed: Jan. 2013].

[5] M. Brown, D. Hankerson, J. López, and A. Menezes, "Software implementation of the NIST elliptic curves over prime fields," in Topics in Cryptology—CT-RSA 2001: The Cryptographers' Track at RSA Conference 2001 San Francisco, CA, USA, April 8–12, 2001 Proceedings, 2001, pp. 250-265.

[6] W. Diffie and M. E. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, 1976.

[7] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," International journal of information security, vol. 1, pp. 36-63, 2001.

[8] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," Designs, codes and cryptography, vol. 19, pp. 173-93, Mar. 2000.

[9] L.P.I. Ledwaba, G.P. Hancke, and H.S. Venter, "Performance costs of software cryptography in securing new-generation internet of energy endpoint devices," IEEE Access, vol. 6, pp. 9303, 2018, doi: 10.1109/ACCESS.2018.2793301.

[10] Z. Liu, X. Huang, Z. Hu, M. K. Khan, H. Seo and L. Zhou, "On emerging family of elliptic curves to secure Internet of Things: ECC comes of age," IEEE Trans. Depend. Sec. Comput., vol. 14, no. 3, pp. 237-248, Jun. 2017.

[11] Y. Lu and L. Da Xu, "Internet of Things (IoT) cybersecurity research: A review of current research topics," IEEE Internet of Things Journal, vol. 6, no. 2, pp. 2103-15, Sep. 2018.

[12] D. Hankerson, A. J. Menezes, and S. Vanstone, Guide to Elliptic Curve Cryptography. Springer, 2004.

[13] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-Bit CPUs," in Cryptographic Hardware and Embedded Systems – CHES 2004, 6th International Workshop, Cambridge, MA, USA, August 11-13, 2004, Proceedings, vol. 3156, pp. 119–132, 2004.

[14] L. E. Kane, J. J. Chen, R. Thomas, V. Liu and M. Mckague, "Security and Performance in IoT: A Balancing Act," IEEE Access, vol. 8, pp. 121969-121986, 2020, doi: 10.1109/ACCESS.2020.3007536.

[15] A. Menezes, "Implementation of elliptic Curve cryptosystems," in Elliptic Curve Public Key Cryptosystems, vol. 234, 1993.

[16] MICRO-ECC repository. [Online]. Available: https://github.com/kmackay/micro-ecc. [Accessed: Jan. 26, 2024].

[17] E. Barker and Q. Dang, "NIST Special Publication 800-57 Part 3 Revision 1: Recommendation for Key Management: Application-Specific Key Management Guidance," NIST, Jan. 22, 2015. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-57pt3r1.

[18] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques," NIST Special Publication 800-38A, Final, [Online]. Available: https://csrc.nist.gov/pubs/sp/800/38/a/final.

[19] STMicroelectronics, "STM32F410RB Product overview specification." [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f410rb.htmldocumentation.

[20] J. Winderickx, A. Braeken, and D. Singelée, "In-depth energy analysis of security algorithms and protocols for the Internet of Things," J Cryptogr Eng, vol. 12, pp. 137–149, 2022, doi: 10.1007/s13389-021-00274-7.
[21] E. Wenger, T. Unterluggauer, and M. Werner, "8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors," in Progress in Cryptology – INDOCRYPT 2013, vol. 8250, 2013.
[22] E. De Win, S. Mister, B. Preneel, and M. Wiener, "On the performance of signature schemes based on elliptic curves," in Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings, vol. 3, pp. 252-266, 1998.