Comparative Performance Analysis of Cryptographic Techniques for Securing the Physical Layer in Internet of Medical Things (IoMT) Systems

Eterigho Okpomo Okpu, Onate Egerton Taylor, Nuka Dumle Nwiabu and Daniel Matthias
Dept. of Cyber Security, Delta State University of Science & Technology, Ozoro, Nigeria
Dept. of Computer Science, Rivers State University, Port-Harcourt, Nigeria
Dept. of Computer Science, Rivers State University, Port-Harcourt, Nigeria
Dept. of Computer Science, Rivers State University, Port-Harcourt, Nigeria
Contact: +2348065518636 Corresponding author email: okpuoe@dsust.edu.ng

DOI: 10.56201/ijcsmt.v10.no2.2024.pg157.169

Abstract
Sensitive medical data can be exchanged and collected via connected devices because to the rapidly expanding Internet of Medical Things (IoMT). Protecting patient privacy and the dependability of medical applications requires ensuring the security and integrity of this data. This study examines and contrasts two cryptographic strategies for IoMT system physical layer security. The first method combines the HMAC-SHA-256 hashing technique to assure data integrity with the AES-256 encryption algorithm to maintain data confidentiality. The second strategy makes use of the AES-GCM (Galois/Counter Mode) technique, which offers assurance of integrity and confidentiality in a single, integrated process. The study assessed and contrasted the performance characteristics of the two techniques with respect to the duration required for the encryption and decryption of identical data samples. The outcomes show that, in terms of encryption and decryption times, the AES-GCM technique performs better than the AES-256 + HMAC-SHA-256 strategy. The research's conclusions offer insightful information to IoMT system designers and developers, empowering them to choose the best cryptographic methods for protecting the integrity and confidentiality of private medical data in IoMT applications' physical layer.

Keywords: *Advanced Encryption Standard (AES-256) encryption; Hash-based Message Authentication Code (HMAC); Advanced Encryption Standard - Galois/Counter Mode (AES-GCM); Internet of Medical Things (IoMT); Cryptography; Encryption; Decryption; hashing; wearable device manometer; physical layer of IoMT.*

1. INTRODUCTION

Cryptography is the study and use of techniques to shield information and communications from adversaries [1]. The science of information and communication security is known as cryptography. By prohibiting illegal alteration, it provides protection against unauthorized parties [2]. Hashing functions allow data to be converted into unique fixed-length strings (hash values), which allow recipients to compare hashes and verify the content's integrity [3]. Two of the most common forms of cryptography are symmetric encryption, which uses one key for both encryption and decryption, and asymmetric encryption, which uses two keys—a public key for encryption and a private key for decryption—for both operations [3]. On a variety of digital platforms, the Advanced Encryption Standard (AES) ensures secure communication and data security. [4]AES-256 employs a 256-bit key in its symmetric-key encryption technique. By utilizing the AES cipher to encrypt data, it offers

confidentiality. AES-256 is an encryption method that has gained widespread use and confidence due to its high level of security. [5] There are three variants of the popular block cipher AES (Advanced Encryption Standard): AES-128, AES-192, and AES-256. The versions vary in the number of rounds and key size. Parallel architectures are essential for good performance with AES algorithms because they require substantial computational power for information security [6].

The trustworthy Message Authentication Code (HMAC), which is hash-based, can be used to verify the integrity and authenticity of data transfer. [7] A hash function's job is to create a message or set of data's "fingerprint" for authentication. The length of the hash code that the algorithm generates determines how resilient it is to brute-force attacks. Software can execute Message Authentication Codes (MAC) more quickly when they are constructed from Cryptographic hash functions (SHA-256) rather than symmetric block ciphers like Data Encryption. [8] Unlike encryption/decryption algorithms, the widely used Message Authentication Code (MAC) algorithm is not reversible for message authentication. HMAC creates a unique fixed-size hash value known as a tag by combining the message that has to be verified with a secret key using cryptographic hash methods like Secure hash algorithm (SHA-256). HMAC safeguards the data integrity by generating a tag that is dependent on both the message content and the secret key [9]. This tag is sent with the message so that the recipient can use the shared secret key and the message they received to recalculate the HMAC and verify its authenticity [10]. While HMAC is a subset of MAC that verifies the authenticity of a message using a private key and a cryptographic hash function, MAC is used to confirm the authenticity of messages. SHA-256 and other hashing algorithms can be inserted into the HMAC algorithm's framework; the term "SHA256+" refers to the combined use of these methods [11].

[12] Data encryption and authentication on the physical layer of the Internet of Things using AES-GCM (Advanced Encryption Standard - Galois/Counter Mode) can be done. An effective encryption technique for Internet of Things applications is AES-GCM. [13] To provide data secrecy while being sent over the IoT physical layer, AES-GCM employs the AES block cipher. AES-GCM offers robust encryption capabilities and supports key sizes of 128 bits, 192 bits, and 256 bits. The Galois/Counter Mode (GCM) tag, which is produced by AES-GCM, is a Message Authentication Code (MAC) that is used to confirm the validity and integrity of encrypted data [14]. The recipient can confirm the data's integrity by recalculating the GCM tag and comparing it with the received tag after receiving the encrypted data and GCM tag [15]. The recipient can be sure that the data hasn't been altered after transmission if the tags match [16]. One can adjust the GCM tag length (usually 16 bytes) to offer strong protection against attempts at forging. The encrypted data, the IV, and any additional authenticated data (AAD) are used to calculate this GCM tag. [17] Additional Authenticated Data (AAD), which can be used to secure headers or metadata in addition to the encrypted payload, can be included with AES-GCM. This is especially helpful in Internet of Things instances where you might need to safeguard both the sensitive payload and some non-sensitive data. Every encryption operation performed by AES-GCM generates a distinct Initialization Vector (IV), preventing replay attacks, in which a malicious party tries to reuse previously transmitted data [17]. To guarantee that the IV is distinct for each message, it is usually generated from a counter value or a nonce, which is an integer that is used just once. Because of its speed and efficiency, AES-GCM is well suited for Internet of Things applications that demand high-performance cryptographic processes [18]. The algorithm can benefit from parallelization for increased throughput and can be implemented in either software or hardware.

Researchers can secure the communication channel at the bottom of the IoT stack by employing AES-GCM at the IoMT physical layer to preserve the confidentiality and integrity of your IoMT data. [19] In addition to ensuring confidentiality through encryption, IoT device manufacturers and system designers can guarantee the integrity of the data being transmitted over the physical layer by

utilizing AES-GCM. All things considered, AES-GCM, with its distinct IV and GCM tag mechanism, strengthens the overall security of the IoT infrastructure by offering a strong protection against replay assaults in IoT systems.

AES-256 offers confidentiality, HMAC-SHA-256 offers data integrity and authenticity, and AES-GCM offers both [19]. These are the main distinctions between AES-256, HMAC-SHA-256, and AES-GCM. AES-GCM is an encryption method that combines GHASH authentication and AES encryption, while HMAC-SHA-256 is a message authentication code. AES-256 is a symmetric-key encryption algorithm. AES-GCM is frequently used in secure communication protocols and applications that require both encryption and authentication, while AES-256 is frequently used for encryption and HMAC-SHA-256 is frequently used for data signing and verification [20]. In general, AES-GCM outperforms AES-256 and HMAC-SHA-256 combined in terms of speed and efficiency. High security is offered by AES-GCM and the AES-256 + HMAC-SHA-256 method.

A wearable manometer designed exclusively for blood pressure measurement is a device that combines Internet of Things (IoT) capabilities with manometer capability to provide continuous, linked blood pressure monitoring [21]. Many medical conditions that are sometimes hard to diagnose or have symptoms that are not immediately apparent can be predicted with the use of a heart rate monitoring gear. Security and privacy are the main drawbacks of IoMT healthcare [22]. Attacks on internet-enabled and connected medical devices have the potential to seriously impair patients' physical and mental health, possibly even resulting in death. Man-in-the-middle (MitM) attacks, wormhole attacks, sinkhole attacks, distributed denial of service (DDoS) attacks, eavesdropping, ransomware, and many more are examples of common IoT attacks [23]. Attacks similar to denial-of-service attacks can be launched by exploiting security flaws in Internet of Things devices or by getting into a large number of devices to establish a botnet. Data stored on Internet of Things devices can be encrypted by ransomware, preventing authorized users from accessing it [24]. [25] In 2019, the medical industry reported 41.4 million patient data breaches, or approximately 49% more attacks than in 2018. 32% of all reported data breaches between 2015 and 2022 occurred in the healthcare industry. 707 healthcare data breaches occurred in 2022 [26].

A few of the IoMT systems now in use have poor encryption and no physical layer data integrity. Based on the aforementioned problems, the study's goals are to apply HMAC hashing to confirm the data's integrity at the physical layer, adding another degree of security and privacy, and to deploy AES-256 encryption to guarantee the confidentiality of data transferred through Internet of Things smart healthcare devices. The AES-GCM technique was also used in the research to guarantee data integrity and security at the physical layer of the Internet of Medical Things. The amount of time needed to encrypt and decrypt the same data was also compared for both algorithms in this paper. Python was used as the programming language to create the system. The goal of this research is to create a safe flow of data from the physical layer to the application layer in wearable fitness trackers, smart watches, temperature sensor devices, wearable manometers, and smart clothing that monitors blood pressure for hypertension.

2. RELATED WORK

[5] In order to increase computational capacity for high-security cryptography, the investigator's primary objective is to implement the AES cryptographic algorithm using parallel computing architectures, notably field-programmable gate arrays (FPGAs). For high-security cryptography applications, parallelized architecture on FPGA platforms enables the AES algorithm's implementation, providing more processing capacity. [7] The author wants to use Verilog to develop Advanced Encryption Standard (AES) encryption. Cryptographic methods are used to safeguard data, such as that found in electronics. [8] The researchers employed Secured Hash Algorithm 256 and

Keyed-Hash Message Authentication Code (HMAC-SHA256) to provide a safe information-sharing environment. Additionally, a trust-based mechanism was added that can distinguish between trusted and malicious nodes in the network. [27] A high-throughput, compact SHA-256 hash function FPGA design was reported in this study, along with the matching HMAC FPGA design. [9] This author examines a quick implementation of the Hash-based Message Authentication Code (HMAC), which makes use of the SHA-256 algorithm to maximize hardware efficiency and design efficacy by ensuring the validity and integrity of data.
[11] A high-performance HMAC processor built on the SHA-2 family of hash algorithms was presented by the authors. Specifically, four HMAC hardware modules—SHA224, SHA-256, SHA-384, and SHA-512—are implemented. [28] The primary objective of the suggested approach is to minimize superfluous block operations and enhance the internal workings of the underlying pseudo random function (PRF). Stated differently, the author has combined multiple superfluous processes and fully utilizes the constant values present in PBKDF2. [29] The purpose of this master's thesis is to evaluate the Solo key's security against various side channel attacks. Solo is an open source security token. To provide a strong second factor authentication, the Fido Alliance's U2F authentication protocol is used by the Solo key. This study, which is based on the concept of session protection as put forth by [30], stores the shared keys required for HMAC-SHA256 encryption by utilizing the session Storage feature of HTML5. [12] This paper describes a constant-time version of AES that requires just 7.59 cycles/byte on an Intel Core 2 Q9550. The costs associated with this implementation include the expenses associated with converting the input data into bitsliced format and the output data back to normal format. [15]The AES-GCM authenticated encryption (AE) crypto-core design described in this study is appropriate for Internet of Things security applications. The AES-GCM core provides integrity and authenticity using GHASH and confidentiality through the Counter (CTR) mode of the AES block cipher. [17] The authors utilized this (already-fixed) problem as a concrete illustration of how AES-GCM's authentication technique (GHASH) is vulnerable. [31] Several high throughput network protocols have been allowed to use the Advanced Encryption Standard (AES) in conjunction with the Galois Counter Mode (GCM) of operation to offer authenticated encryption. [32] The authors of this paper discussed the effective FPGA architecture of the GCM in conjunction with the AES block cipher. [33] This article presents an efficient Galois Counter Mode of operation (GCM) implementation using the Advanced Encryption Standard (AES) on low-end microcontrollers. [34]Two effective hardware implementations of the AE schemes, AES-GCM and AEGIS, are also shown in this article. Because of the innate computation feedback in AES-GCM, system efficiency is always dictated by the Galois Hash (GHASH) architecture. [35] The writer's assessments on Deoxys, a third-round contender in the current Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR), are included in this article.

3. METHODOLOGY

Description of the Proposed System:

Advanced Encryption Standard (AES)-enabled wearable health data monitoring devices ensure the integrity and confidentiality of sensitive data. AES encryption enables safe communication between the blood pressure monitor and other connected devices, such as smartphones or cloud servers, where health data is processed or stored. Message Authentication Codes (MACs) are used to ensure data integrity and authenticity instead of encrypting data. The methods listed below can be used to implement MAC and AES encryption in wearable technology in order to monitor related health data:
1. Determine Which Data to Encrypt: Determine which health data collected by wearable technology needs to be encrypted. "TCD_data" is the name of the dataset that the system utilizes. The Transcranial Doppler (TCD) Data non-invasive ultrasound technique is used to measure the

major arteries in the brain's blood flow velocity. A "TCD_data" dataset may contain measurements made during TCD exams, such as pulsatility indices, blood flow velocities, and other hemodynamic parameters. Table 1 shows the dataset TCD_data, and Table 2 shows an example of SUBJ003, which is a dataset TCD_data. These datasets on blood pressure were gathered from physionet.org.

2. Select the AES Configuration: For this investigation, the AES-256 has been used. With a 256-bit key length, this arrangement provides the highest level of safety among all AES versions. AES-256 offers a stronger resistance against brute-force attacks.

3. Produce Keys for Encryption: Generate strong encryption keys to use with AES-256. These keys ought to be securely stored by the wearable device and rendered unreadable by everyone but approved users or apps. The AES-256 operating mechanism is based on CBC (Cipher Block Chaining).

4. Encrypt Health Data: Encrypt the wearable device's collected private health data using AES-256 encryption and the created encryption keys. Encrypt data before transferring it across wired or wireless communication routes.

5. MAC Computation: A secure MAC technique, such as HMACSHA256, is employed in addition to encryption to create a MAC over the encrypted data. The MAC ensures the integrity and validity of the encrypted data. The wearable device determines the MAC using a secret key that is unique to it and the server. The PyCryptodome framework was used in this investigation. Many cryptographic techniques, including hash functions, digital signatures, symmetric and asymmetric encryption, and HMAC, are available through this Python wrapper for the Crypto++ package. Use the HMAC-SHA256 method to find the message's HMAC value. Provide the secret key and the message to the HMAC algorithm as inputs.

6. Secure Data Transmission: Verify that the wearable technology securely sends encrypted health data to other platforms, cloud servers, or mobile devices.

7. Install Data Decryption: To unlock the protected health data, install a decryption capability in the recipient systems or applications using the appropriate AES-256 keys. After acquiring the data, the central server recomputes the MAC using the same secret key and the encrypted data it received. If both the computed and acquired MACs match, it indicates that the data was not changed during communication.

8. Manage Key Management: Use suitable key management protocols to stop unauthorized access to or exposure of AES-256 encryption keys. Establish procedures for key production, distribution, rotation, and revocation in order to safeguard encrypted data.

9. Test and Validate Security: Carefully test and validate the AES encryption deployment to ensure its effectiveness and security.

Table 1: TCD_data Dataset

Participant	Sex	Dominant Hand	Body Mass Index	Systolic Blood Pressure	Diastolic Blood Pressure	Education	0-back(1) Left Blood Flow Velocity	1-back Left Blood Flow Velocity
SUBJ001	M	R	28.4	125	85	MD, PhD	54.2806	53.4231
SUBJ002	M	R	26.95	115	70	PhD	23.74456	22.19077
SUBJ003	M	R	25.98	114	70	MD	44.98957	45.51511
SUBJ004	F	R	20.51	98	60	MD	45.51381	44.84977
SUBJ005	M	R	26.93	132	80	MD	28.75367	29.50703

Table 2: SUBJ003 Dataset

Sample Rate: 400				
Relative Time	Date	Time Stamp UTC	Left MCA	Right MCA
0	9/13/2018	1:59:40 PM	0.600586	0.611267
0.01	9/13/2018	1:59:40 PM	0.593872	0.611267
0.02	9/13/2018	1:59:40 PM	0.583801	0.610352
0.03	9/13/2018	1:59:40 PM	0.574951	0.610046
0.04	9/13/2018	1:59:40 PM	0.570984	0.610046
0.05	9/13/2018	1:59:40 PM	0.57251	0.610657

Algorithm

The steps of the AES-256 algorithm are as follows:

AES-256 Encryption:

1. Take the 128-bit plaintext block and the 256-bit secret key.
2. Run through the plaintext through a first key-dependent permutation.
3. Carry out the ensuing 14 rounds:
    a. Use S-boxes to carry out a replacement layer.
    b. Use a linear transformation to carry out a linear mixing layer.
    c. Include the round key that the secret key yielded.
4. Complete one last permutation that depends on a key.
5. The output is the generated 128-bit ciphertext.

The following is the expression for the HMAC-SHA256 formula:

$HMAC\text{SHA-256}(K,)=\text{SHA-256}((K\oplus\text{opad})\|\text{SHA-256}((K\oplus\text{ipad})\|M))HMAC\text{SHA-256}(K,M)=\text{SHA-256}((K\oplus\text{opad})\|\text{SHA-256}((K\oplus\text{ipad})\|M))$

Where:

- K (padded to a block size if necessary) represents the secret key.
- M is the message that has to be confirmed.
- The symbol $\oplus$ indicates bitwise XOR.
- The key provides the padding constants padopad and ipadipad.
- $\|$ indicates concatenation.
- SHA-256 (SHA-256() is the representation of the SHA-256 hash function.
- The building of the HMAC consists of two steps:

    There are two steps in the construction of the HMAC:

The building of the HMAC consists of two steps:

1. **Key Padding:** If a key is larger than the block size (64 bytes for SHA-256), it can be hashed using the hash function to create a fixed-size key. If the key is shorter than the block size, it is padded with zeros to the block size.
2. **Inner and Outer Padding:** The key is XORed with the inner and outer padding constants, ipadipad and opadopad. Subsequently, the message is hashed twice: once with the concatenation of the message and the inner padded key, and once with the outer padded key and the result of the first hash.

The HMAC-SHA256 authentication code, which can be used to verify the message's validity and integrity, is the result of applying a single secret key to a hash. It is important to keep in mind that HMAC-SHA256's security features and, when used effectively, resilience to known threats make it a popular choice for message authentication in a wide range of cryptographic protocols and applications.

The procedures for encrypting and decrypting a CSV file using the AES-GCM (Advanced Encryption Standard with Galois/Counter Mode) algorithm. The techniques employed and the algorithm are broken out as follows:
Methods Used:
1**.** The function encrypt_data(data, key, aad) is in charge of utilizing AES-GCM to encrypt the input data. The following specifications are required. The additional authenticated data (AAD), which is also a 256-bit key, the 256-bit encryption key, and the data to be encrypted—which in this case is a string. Using the AESGCM class from the cryptography package, the method creates a random 12-byte nonce, encrypts the data, and then returns both the encrypted data and the nonce.
2. The function decrypt_data(nonce, encrypted_data, key, aad) is in charge of utilizing AES-GCM to decrypt the encrypted data. The following specifications are required. The additional authenticated data (AAD), which is likewise a 256-bit key, the encrypted data, the nonce used during the encryption procedure, and the 256-bit encryption key**.** The function decrypts the data using the AESGCM class and outputs the resultant string.
The algorithm
1. To begin the encryption or decryption process, the user is required to provide the path to the CSV file.
2. After reading the CSV file, the information is saved as a list of rows.
3. The first 5 rows of the original data are printed to the console.
4. The secrets.token_bytes() function is used to generate a random 256-bit AAD key and a random 256-bit encryption key.
5. The encryption procedure has a time limit, and for every data row:
- The encryption process is timed, and for each row in the data:
- The row is converted to a comma-separated string.
- The encrypt_data() function is called with the row data, the encryption key, and the AAD key as arguments.
- The returned nonce and encrypted data are stored in a list.

6. Each nonce and encrypted data combination in the encrypted data list has a timed decryption process.
- The nonce, encrypted data, encryption key, and AAD key are passed as parameters to the decrypt_data() method.
- After decrypting, the row is divided into separate columns and kept in a list.

7. The console is printed with the first five rows of both the encrypted and decrypted data.

4. RESULTS

Figure 1 displays both the original and encrypted data after hashing and encrypting it using the AES 256 and HMAC algorithms. The original data and the encrypted data are displayed in Figure 2, which displays the outcomes of hashing and encryption using the AES-GCM technique. A graphical depiction of the amount of time required to encrypt and decrypt data using both techniques is shown in Figure 3. Table 3 displays the encrypted output using the AES 256 and HMAC approaches. The encrypted file, file size in megabytes (MB), encryption key, and HMAC key are all listed in this table.

The encrypted output using the AES-GCM method is shown in Table 4. The encrypted file, file size in megabytes (MB), encryption key, and AAD key are all listed in this table. Table 5 displays the amount of time required to encrypt and decrypt data using AES-256/HMAC techniques. Time to encrypt CSV data with AES-256/HMAC is shown in this table as TEDAH, and Time to decrypt CSV data encrypted using AES-256/HMAC is shown as TDEDAH. Table 6 shows the Time Consumption Demonstration for Encrypting and Decrypting Data Using AES-GCM Method. The Time to Encrypt CSV Data with AES-GCM is shown in this table as TEDAG, while the Time to Decrypt CSV Data with AES-GCM is shown as TDED.

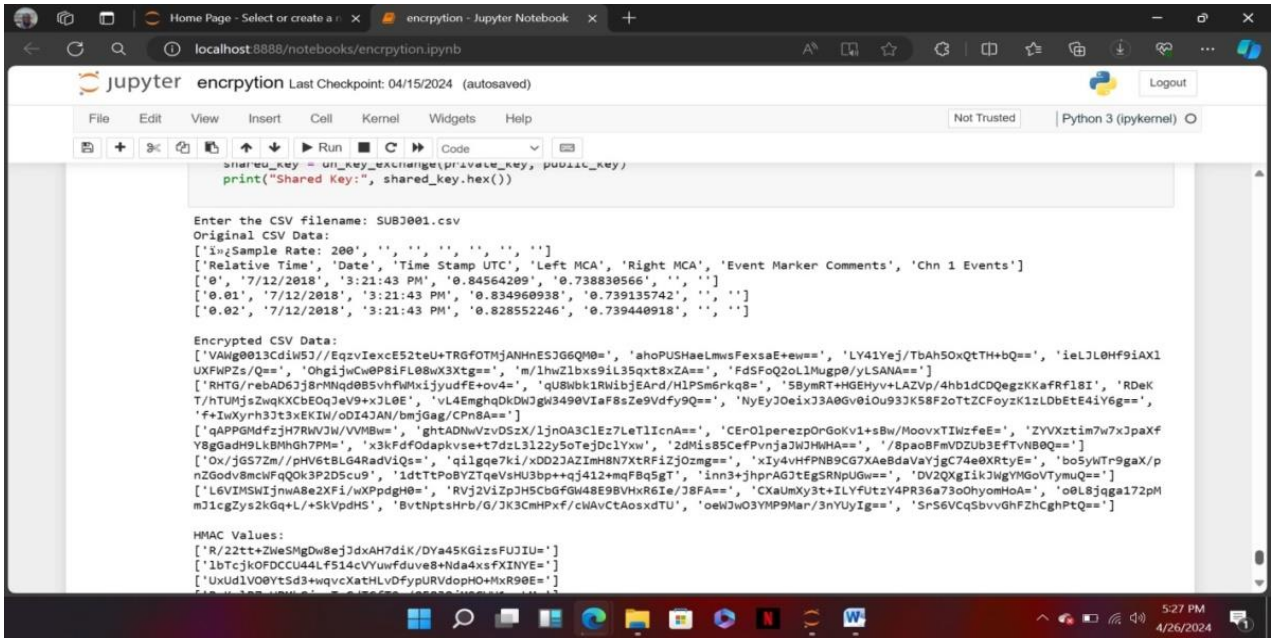

Figure 1: Results of Encryption and Hashing using the AES 256 and HMAC Techniques

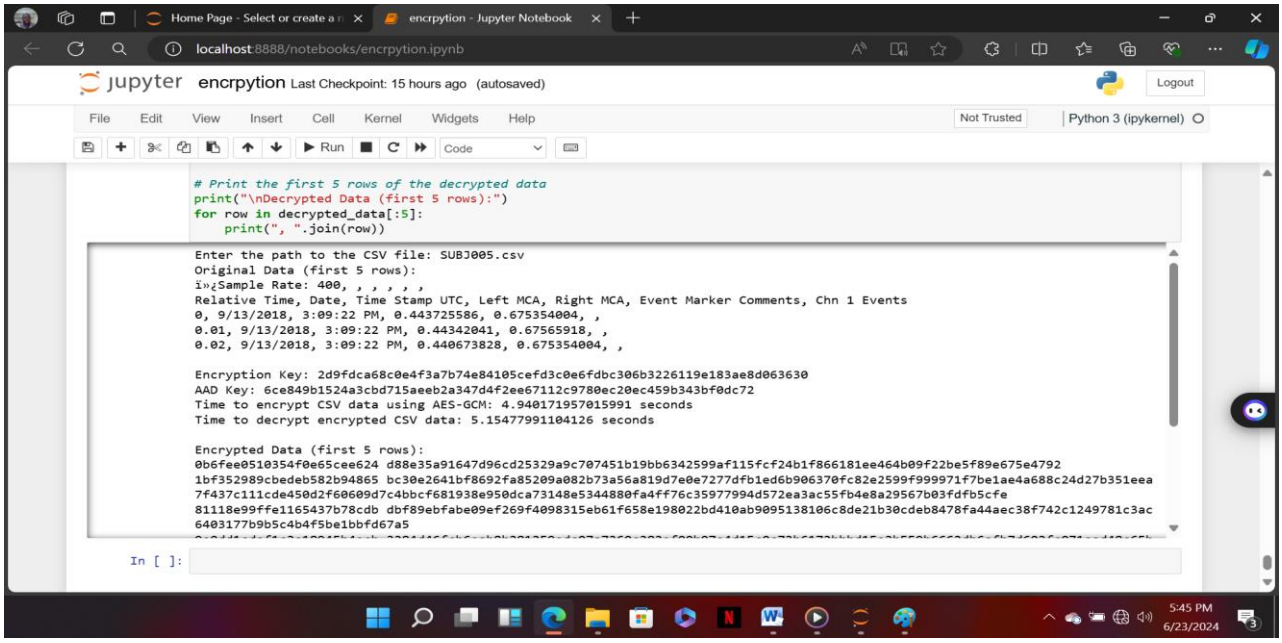

Figure 2: Results of Encryption and Hashing using the AES-GCM Method

Table 3: Encrypted result using AES 256 and HMAC techniques

N/S	Encrypted file	File Size MB	Encryption Key	HMAC Key
1	SUBJ001_encrypted.csv	4.89	b'w\xb6"\xc7\xe5@\x05\xc4\xb4!\x89\xab\x05\xde\x9c"\'\xf8\xfe\xd9\xd3w\xc0\xd6\x03\xbd/\xea,\x9d\x13,'	177d39f67cac68281708d0144558a4d3401296a588f0d1e7131b119f637c604c
2	SUBJ002_encrypted.csv	3.36	b'\xeb\xda\xd5\xd4\xdew\xf4x\xe6\x7f\xa5B\xf1\xd7\xa6\xe9@\xc4>\xea_%\x86\xeb\xaf\x00\xe1\xc6\xcb\xd5D\x82'	57f2126c3d98d4cfd7277de45c8a245d74ec9c2cae5546458de1605edb75d36e
3	SUBJ003_encrypted.csv	3.31	b'&s\x1f\x9a\xda]\\\x19)\x15\x14\xb5\xbe(\xc9\xc5\x1c\xebr#\xd9r\x12\x9a\xd0\xec\xc4va\x96e_'	1c902aa02a1a8e5ab00588c2ea1e52766e5f5a64dc11390523af8d3fc44c9a44
4	SUBJ004_encrypted.csv	3.70	b'b{\xc0\xab\x86\xb5\x80|\xb6\xd7\xd8\x82\xe7o&c\xdawk\x8e\xbe\x9b\xf6W(U\x81\x17\xf0s\x8a\x1e'	31f0e6928602bf12580c61a4634843b341e97fdffd5f738553ce2461a79cc261

5	SUBJ005_encrypted.csv	3.37	b'\xbb\x99i\xd9\xfc\xce`\x8a\xc3\x00r\x1d&<x\x9c\xad\xd4_\x90{\x80\xc8\xc5\x10\x12\xdf>\x17\xa5\x8b\xcd'	63d532e9658d48913a12f3b33a01430a2dc0978ef4f26af8b4edef13ff73d169

Table 4: Encrypted result using AES-GCM Method

N/S	Encrypted file	File Size MB	Encryption Key	AAD Key
1	SUBJ001_encrypted.csv	4.89	b51dcb733f09746429b90e2936975f8c22165994b041b61895424ef3537596da	ed881e933734a1e10d905e2aab2da2095ed9963aaab29f507920e11ed974548a
2	SUBJ002_encrypted.csv	3.36	13c6c5f41b13272468f06fe9722f1aea7541d6fda64326db3c8f0a677750f3ba	db07c473bd9cc95262874bfbe220af004e495fe6b56aa24f6190dd3a7f69f4a6
3	SUBJ003_encrypted.csv	3.31	e4584291284b6119b5306df03d7faf2ae89d3c212da367d66f1d5231c9489eb8	20f3fe05230fa98d8117e743d9a7571e8d48c80e627254cf545b2b1906568d6d
4	SUBJ004_encrypted.csv	3.70	a6045904db5e1462585cf75df151e830a3c9f5022a570c1efb46755d72d5441d	b22eddcddc6dbd86fb9ff133f24b37b89089c3be7aea9a91915c5bec45e1c990
5	SUBJ005_encrypted.csv	3.37	2d9fdca68c0e4f3a7b74e84105cefd3c0e6fdbc306b3226119e183ae8d063630	6ce849b1524a3cbd715aeeb2a347d4f2ee67112c9780ec20ec459b343bf0dc72

Table 5: Proves Time Consumption to Encrypt and Decrypt Data using AES-256/HMAC Techniques

N/S	File Size MB	TEDAH	TDEDAH
1	4.89	48.9	31.5
2	3.36	20.7	21.9
3	3.31	19.4	20.7
4	3.70	24.9	23.2
5	3.37	21.8	22.4

Table 6: Demonstrates Time Consumption to Encrypt and Decrypt Data using AES-GCM Method

N/S	File Size MB	TEDAG	TDED
1	4.89	5.660716533660889	5.709982872009277
2	3.36	5.620843410491943	5.454753160476685
3	3.31	5.571392297744751	6.3335487842559814
4	3.70	6.45686674118042	6.066409349441528
5	3.37	4.940171957015991	5.15477991104126

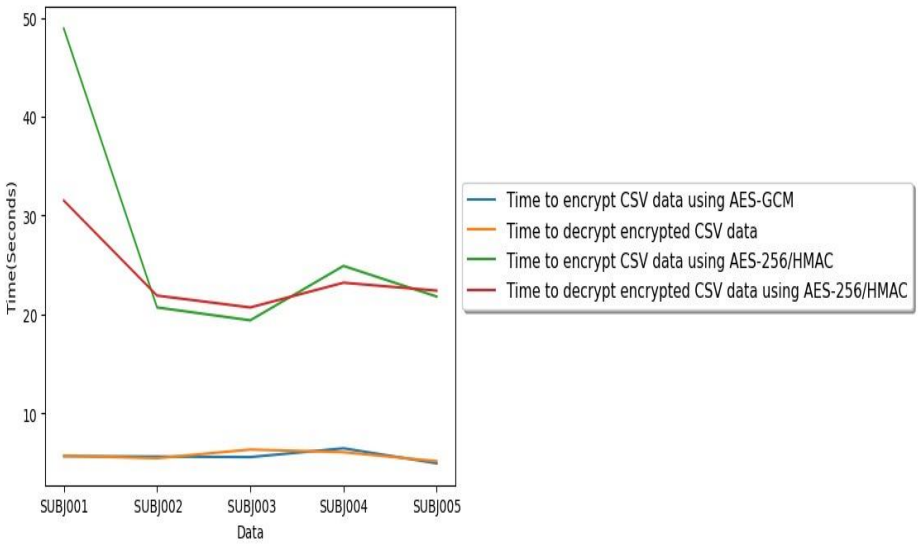

Figure 4: Graphical Representation of Time consumption to Encrypt and Decrypt data using both methods

Tables 3 and 4 present the encrypted result display for both strategies. Encrypted file, file size in megabytes, encryption key, HMAC key, and AAD key are all listed in this table. A graphical depiction of the amount of time required to both encrypt and decrypt data is shown in Fig. 3. These findings demonstrate that the 4.89 MB SUBJ001 file was encrypted and decrypted in 48.9 and 31.5 seconds, respectively, using AES-256/HMAC techniques. The findings also confirm that the 4.89 MB SUBJ001 file was encrypted and decrypted in 5.660716533660889 seconds and 5.709982872009277 seconds, respectively, using the AES-GCM Method. This demonstrates that the AES-GCM Method can ensure data integrity and encrypt and decode data more quickly than the AES-256/HMAC Techniques.

5. CONCLUSION

This study applies the HMAC hashing approach to assure data integrity at the physical layer of IoMT and integrates the AES-256 encryption technique to secure data secrecy there as well. The AES-GCM technique was also used in the research to guarantee data integrity and security at the physical layer of the Internet of Medical Things. The amount of time needed to encrypt and decrypt the same data was also compared for both algorithms in this paper. On a variety of digital platforms, the Advanced Encryption Standard (AES) ensures secure communication and data security. The trustworthy Message Authentication Code (HMAC), which is hash-based, can be used to verify the integrity and authenticity of data transfer. The AES cipher operates in the Galois/Counter Mode, or AES-GCM, which offers both authenticity and confidentiality. To offer authenticated encryption, it combines the GHASH function with AES encryption in counter mode (AES-CTR). Python was used as the programming language to create the system. These findings demonstrate that the 3.36 MB SUBJ002 file was encrypted and decrypted in 20.7 and 21.9 seconds, respectively, using AES-256/HMAC techniques. The findings also confirm that the 3.36 MB SUBJ002 file was encrypted and decrypted in 5.454753160476685 seconds and 5.620843410491943 seconds, respectively, using the AES-GCM Method. Better hardware support, performance and throughput, power and energy

consumption, security needs, ease of implementation, and compatibility are all benefits that AES-GCM may offer. This is especially important for Internet of medical devices. The results of this study will improve safe IoT smart healthcare systems and offer insightful information to practitioners and researchers alike.

REFERENCES

1. Biryukov, A., & Khovratovich, D. (2009). *Related-key cryptanalysis of the full AES-192 and AES-256*. In Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15 (pp. 1-18). Springer Berlin Heidelberg.
2. Andriani, R., Wijayanti, S. E., & Wibowo, F. W. (2018, November). *Comparision of AES 128, 192 and 256 bit algorithm for encryption and description file*, In 2018 3rd International Conference on Information Technology, Information System and Electrical Engineering (ICITISEE) (pp. 120-124). IEEE.
3. Lanjewar, R., & Pande, G. (2015). *Implementation of AES-256 Bit: A Review*, Inventi Rapid: Information Security.
4. Mohammed, N. Q., Amir, A., Ahmad, B., Salih, M. H., Arrfou, H., Thalji, N., ... & Abdulhassan, M. M. (2023, April). *A Review on Implementation of AES Algorithm Using Parallelized Architecture on FPGA Platform*, In 2023 IEEE International Conference on Advanced Systems and Emergent Technologies (IC_ASET) (pp. 1-6). IEEE.
5. Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., & Shamir, A. (2010). *Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds*. In Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29 (pp. 299-319). Springer Berlin Heidelberg.
6. Chandu, G. M., Abhishek, K., Lokesh, S., Ramalingeswararao, V., & Sarma, R. (2022). Implementation of AES Algorithm using Dynamic S-box on FPGA, *Futuristic Sustainable Energy & Technology* (pp. 137-145). CRC Press.
7. Shet, G. G., Jamuna, V., Shravani, S., Nayana, H. G., & Kumar, P. (2020). Implementation of AES Algorithm Using Verilog. *JNNCE Journal of Engineering & Management (JJEM)*, 4(1), 17.
8. Azeez, N. A., & Chinazo, O. J. (2018). Achieving Data Authentication With Hmac-Sha256 Algorithm, *Computer Science & Telecommunications*, 54(2).
9. Suhaili, S., Julai, N., Sapawi, R., & Rajaee, N. (2024). Towards Maximising Hardware Resources and Design Efficiency via High-Speed Implementation of HMAC based on SHA-256 Design. *Pertanika Journal of Science & Technology*, 32(1).
10. Kelly, S., & Frankel, S. (2007). Using hmac-sha-256, hmac-sha-384, and hmac-sha-512 with ipsec (No. rfc4868).
11. Juliato, M., & Gebotys, C. (2011). *FPGA implementation of an HMAC processor based on the SHA-2 family of hash functions*. University of Waterloo, Tech. Rep.
12. Käsper, E., & Schwabe, P. (2009, September). *Faster and timing-attack resistant AES-GCM*. In International Workshop on Cryptographic Hardware and Embedded Systems (pp. 1-17). Berlin, Heidelberg: Springer Berlin Heidelberg.
13. Gueron, S., Langley, A., & Lindell, Y. (2017). *AES-GCM-SIV: specification and analysis*. Cryptology ePrint Archive.

14. Bellare, M., & Tackmann, B. (2016). *The multi-user security of authenticated encryption: AES-GCM in TLS 1.3*. In Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I 36 (pp. 247-276). Springer Berlin Heidelberg.
15. Sung, B. Y., Kim, K. B., & Shin, K. W. (2018, January). *An AES-GCM authenticated encryption crypto-core for IoT security*. In 2018 International Conference on Electronics, Information, and Communication (ICEIC) (pp. 1-3). IEEE.
16. Wang, S. (2006). *An architecture for the AES-GCM security standard* (Master's thesis, University of Waterloo).
17. Gueron, S., & Krasnov, V. (2014, April). *The fragility of AES-GCM authentication algorithm*. In 2014 11th International Conference on Information Technology: New Generations (pp. 333-337). IEEE.
18. Arunkumar, B., & Kousalya, G. (2018). Analysis of AES-GCM cipher suites in TLS. *In Intelligent Systems Technologies and Applications*, Springer International Publishing, (pp. 102-111).
19. Rodríguez, M., Astarloa, A., Lázaro, J., Bidarte, U., & Jiménez, J. (2018, November). *System-on-Programmable-Chip AES-GCM implementation for wire-speed cryptography for SAS*. In 2018 Conference on Design of Circuits and Integrated Systems (DCIS) (pp. 1-6). IEEE.
20. Liu, Y., Guo, F., & Wang, C. (2019). Performance Evaluation of AES-GCM and AES-CBC with HMAC-SHA256 for IPsec Encryption Offload, *IEEE Access*, 7.
21. Norsuriati, M. S., Sobri, N. M., Hafiszah, H. Z., Nazib, A. M., Suhaimizan, W. Z., Ashok, V., & Mahadi, A. (2021). Development of IoT Based Cuffless Blood Pressure Measurement System, *Journal of Physics: Conference Series, International Conference on Biomedical Engineering (ICoBE)*, 2071, 1-7.
22. Nia, A. M., & Jha, N. K. (2017). A comprehensive study of the security of Internet-of-Things. *IEEE Trans. Emerging Top. Computer*, 23(12), 586–602.
23. Jain, R., Dhand, G., Bansal, H., Shiksha, S., Sonepat, S., & Jain, P. (2023). Detection Mechanism in IoT framework using Artifcial Neural Networks, *Research Square*.
24. Yaqoob, I., Ahmed, E., Rehman, M. H., Ahmed, A. I. A., Al-garadi, M. A., Imran, M., & Guizani, M. (2017). The rise of ransomware and emerging security challenges in the Internet of Things, *Computer Networks*, 129, 444–458.
25. Rasool, R. U., Ahmad, H. F., Rafique, W., Qayyum, A., & Qadir, J. (2022). Security and privacy of internet of medical things: A contemporary review in the age of surveillance, botnets, and adversarial ML, *Journal of Network and Computer Applications*, 201, 103332.
26. Murray-Watson, R. (2024, January 26). Healthcare Data Breach Statistics, *Hipaajournal*, https://www.hipaajournal.com/healthcare-data-breach-statistics/.
27. Michail, H. E., Athanasiou, G. S., Kelefouras, V., Theodoridis, G., & Goutis, C. E. (2012). On the exploitation of a high-throughput SHA-256 FPGA design for HMAC. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 5(1), 1-28.
28. Choi, H., & Seo, S. C. (2021). Optimization of PBKDF2 using HMAC-SHA2 and HMAC-LSH families in CPU environment. *IEEE Access*, 9, 40165-40177.
29. Collin, S., & Standaert, F. X. (2020). *Side channel attacks against the Solo key-HMAC-SHA256 scheme* (Doctoral dissertation, Ph. D. thesis, UCL-Ecole polytechnique de Louvain).
30. Lin, L., Chen, K., & Zhong, S. (2017). Enhancing the session security of zen cart based on HMAC-SHA256. *KSII Transactions on Internet and Information Systems (TIIS)*, 11(1), 466-483.

31. Buhrow, B., Fritz, K., Gilbert, B., & Daniel, E. (2015, December). *A highly parallel AES-GCM core for authenticated encryption of 400 Gb/s network protocols*. In 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig) (pp. 1-7). IEEE.
32. Henzen, L., & Fichtner, W. (2010, September). *FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications.* In 2010 Proceedings of ESSCIRC (pp. 202-205). IEEE.
33. Kim, K., Choi, S., Kwon, H., Kim, H., Liu, Z., & Seo, H. (2020). PAGE—Practical AES-GCM Encryption for Low-End Microcontrollers. *Applied Sciences*, 10(9), 3131.
34. Abdellatif, K. M., Chotin-Avot, R., & Mehrez, H. (2017). AES-GCM and AEGIS: efficient and high speed hardware implementations. *Journal of Signal Processing Systems*, 88, 1-12.
35. Koteshwara, S., Das, A., & Parhi, K. K. (2017, May). *FPGA implementation and comparison of AES-GCM and Deoxys authenticated encryption schemes.* In 2017 IEEE International symposium on circuits and systems (ISCAS) (pp. 1-4). IEEE.