IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Impact Factor: 7.301

Volume 3, Issue 2, January 2023

Review Paper on the uses of Digital Signature in MQTT Protocol

Mr. Pradeep Nayak[1], Ashwini M[2], Monisha N. S.[3], Moollya Gautami[4], Bhaskar Sahana[5]

Assistant Professor, Department of Information Science and Engineering[1]

Students, Department of Information Science and Engineering[2,3,4,5]

Alva's Institute of Engineering and Technology, Mijar, Mangalore, Karnataka, India

Abstract: *The Message Queue Telemetry Transport (MQTT) protocol for publish/subscribe middleware is proposed in this paper as a way to secure messages. In which the end-to-end method employs the Advanced Encryption System (AES) and Secure Hash Algorithm (SHA), and analyses the overhead associated with the usage of digital signatures Because there is no encryption method applied to the payload, MQTT has this drawback. Which enables one to discover the payload content that results in no data privacy. MQTT also has issues with data integrity. This digital signature's function is to confirm the payload's authenticity, that it doesn't alter during transmission, and that the payload is secret.The proposed solution can be evaluated and tested after which the programme can secure the MQTT payload. The addition of a security mechanism to MQTT, such as the encryption and decryption processes and verification outcomes, results in overhead in many areas. The overhead employed in this study is used to calculate the payload size, message sending time, process of digital signature security mechanism, memory consumption, and CPU utilisation. In an overhead analysis, overhead is performed by looking at many AES key types and numerous SHA key types. Upon closer inspection, it is seen that the digital signature system has resulted in a size increase for a number of the previously listed elements.*

Keywords: AES, SHA, digital signature, payload, MQTT, publish, end-to-end, subscription, overhead

I. INTRODUCTION

The Internet of Things (IoT) is a method of machine-to-machine communication. The idea is that every device may communicate via Internet access, to elaborate [1]. Message Queue Telemetry Transport (MQTT) is a communication protocol that is commonly used in the Internet of Things [2]. Because it consumes a minimal amount of bandwidth, this protocol is frequently used in the Internet of Things .So, it works well for IoT [3].

The MQTT protocol has been used extensivelySeveral systems, including the Internet of Things for Health [4], GPS Tracking [5], and Automated Home Automation [6], employed MQTT to transmit sensitive and secret data. However, the Internet of Things (IoT) is not without flaws. According to the Open Web Application Security Project (OWASP), IoT has targeted network service devices when it discovered that the messages lacked load verification and integrity checks. IoT devices occasionally seek to deliver confidential communications that can only be seen by trusted devices [1].

The problem with the MQTT protocol is that it does not have an encryption process on the payload that raises security and integrity issues in the payload data, which allows the attacker to know the content of the free charge [1]. Based on the above problems to deal with data security and integrity issues a digital signatures can be used to solve this. The digital signatures aim to authenticate the payload data and to ensure that information is not updated during transmission [8]. Digital signatures will give confidence to the recipient that the data sent is correct from the authorized publisher. Digital signatures can also be used to identify whether the sender's message meets the requirements to be received. and read by the recipient or not. A digital signatures can be used to secure a sensitive data [9]whether the receiver read it or not. Sensitive data can be secured with a digital signature [9].

To transfer data securely over the Internet, Digital Signature additionally enables a shared secret key encryption [10]. The AES and SHA algorithms are used in the study's digital signature application. Because it is easy to use and effective in hardware and software [11] and can also be used for digital signatures [12], AES encryption was chosen for this work. Additionally, SHA digest is used to look for unauthorised changes made by unreliable parties [9].

Of course, adding security measures will increase the system's computational load [13].The amount of additional resources required for the implemented security method is determined using an overhead analysis [14]. Several factors that rise with the addition of mechanisms, such as the encryption process, delivery latency time, CPU load [13], memory usage, and security procedure [14], can be examined to perform an overhead analysis in MQTT.

We shall examine the cost of employing digital signatures in the MQTT Protocol in this essay. Due to the addition of additional procedures, using this digital signature mechanism involves overhead and has led to advances in a number of areas. In order to determine how much overhead is generated in the suggested system, an overhead analysis is carried out.

II. ASSOCIATED WORK

Securing messages between publishers and subscribers during a data exchange must ensure that attacks from unauthorised parties are prevented [9]. For different IoT usage, MQTT requires the addition of a scalable and lightweight security mechanism [7] since it uses in memory computing and the Central Processing Unit, which are low resource devices (CPU) [14].

Digital signatures are intended to be used generally for payload or data authentication [12].A security mechanism will need some more resources when it is implemented. Therefore, the adoption of a flexible and lightweight mechanism is necessary to reduce or eliminate the utilisation of resources.

The overhead analysis can document the alterations to extra resources brought on by the inclusion of security mechanisms [14]. The overhead analysis in the study [14] was done by assessing a number of extra resource-demanding factors. Memory and CPU use also has the impact of enhancing security measures. According to the overhead study in research [14], memory and CPU usage have increased, though not significantly.

In the study [13], the authors examined MQTT open source authentication through overhead analysis. The MQTT system developed for research purposes [13] achieves the lowest feasible overhead to minimise the least amount of resource utilisation while still meeting the system's expected overhead.

III. SYSTEM PROPOSED

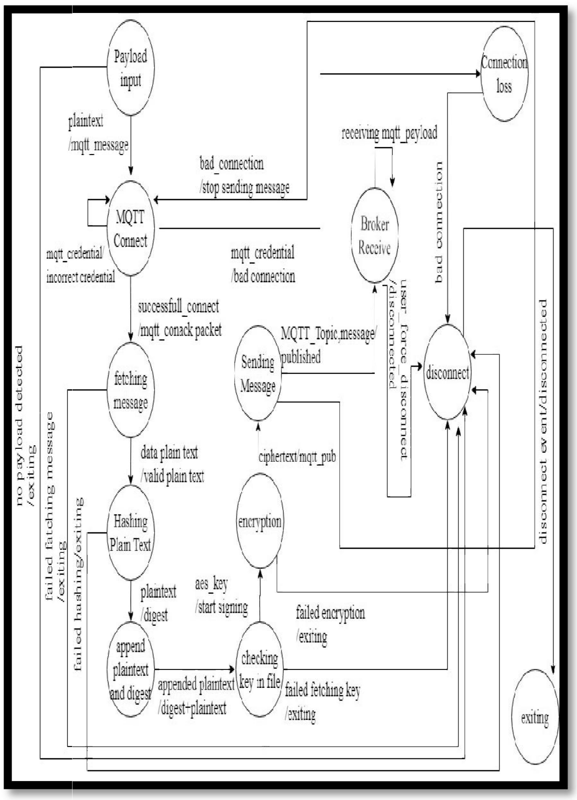

Payload input
plaintext /mqtt_message
bad_connection /stop sending message
Connection loss
receiving mqtt_payload
MQTT Connect
mqtt_credential/ incorrect credential
mqtt_credential /bad connection
Broker Receive
bad connection
user_force_disconnect /disconnected
successfull_connect /mqtt_conack packet
MQTT_Topic,message/ published
Sending Message
disconnect
disconnect event/disconnected
no payload detected /exiting
failed fatching message /exiting
failed hashing/exiting
fetching message
data plain text /valid plain text
ciphertext/mqtt_pub
Hashing Plain Text
encryption
plaintext /digest
aes_key /start signing
failed encryption /exiting
append plaintext and digest
appended plaintext /digest+plaintext
checking key in file
failed fetching key /exiting
exiting

Fig. 1. Publisher FSM

AES and SHA are used as the plaintext hashing algorithms in this digital signature scheme. This AES technique is employed for this purpose because hardware can utilise it effectively [3]. The methods that can be used to provide a secret key to the authorities include providing the key physically, using a reliable third party, or using the Diffie-Helman method for the key agreement [12]. The hash algorithm used is SHA-224,256,384,512. The scenario used by this system uses an end-to-end mechanism, where the publisher initiates the encryption process and the subscriber performs the decoding

The publisher's definition of a finite state machine (FSM) is shown in Fig. 1. If the condition does not hold, the programme will terminate, but if it does, it will enter the state of connecting to the MQTT server and begin fetching messages to retrieve the message that the connected user has inputted. The first process in terms of the publisher is to receive input from the connected user, and then the programme will connect to the MQTT server.

After receiving the message, the publisher will perform the hashing process to preserve the integrity of the payload. Then, after creating a digest of the message, a process of combining digest with messages is used to ensure that the message is not altered during the process of transmission over the network. Finally, the programme will check the AES key in the configuration file, and if the key is present, the signing process will begin.

The publisher can move on to the sending message state after the signing process is complete by sending the ciphertext to MQTT Brokers. Brokers will continue to receive messages from the publisher.

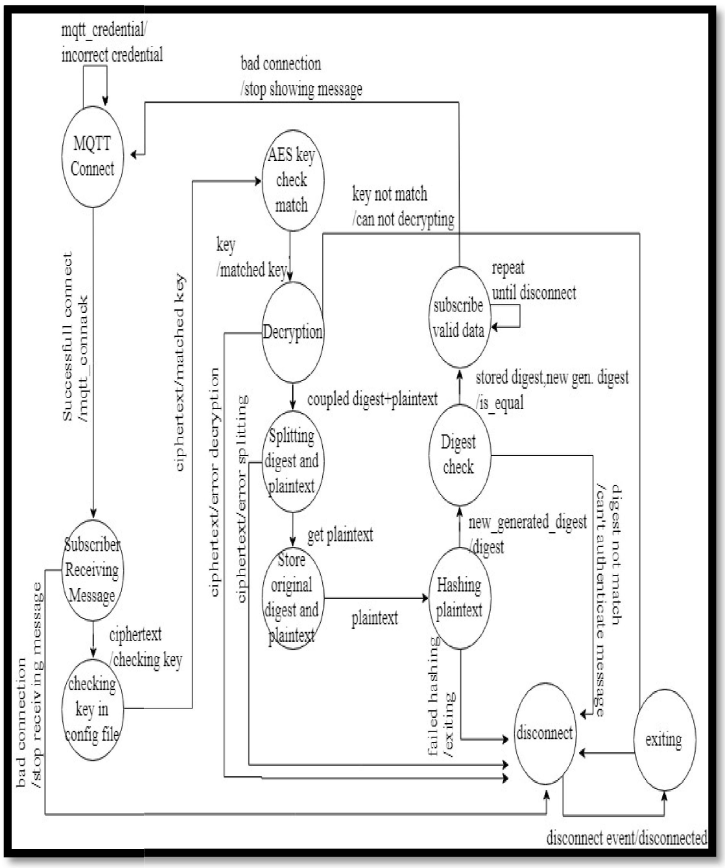

mqtt_credential/ incorrect credential
bad connection /stop showing message
MQTT Connect
AES key check match
key not match /can not decrypting
key /matched key
repeat until disconnect
subscribe valid data
Decryption
Successfull connect /mqtt_connack
ciphertext/matched key
ciphertext/error decryption
ciphertext/error splitting
coupled digest+plaintext
stored digest,new gen. digest /is_equal
Splitting digest and plaintext
Digest check
get plaintext
new_generated_digest /digest
Subscriber Receiving Message
Store original digest and plaintext
plaintext
Hashing plaintext
digest not match /can't authenticate message
ciphertext /checking key
checking key in config file
bad connection /stop receiving message
failed hashing /exiting
disconnect
exiting
disconnect event/disconnected

Fig. 2. Subscriber FSM

A FSM diagram in terms of subscribers is shown in Figure 2. The subscriber will connect to the MQTT server in the first state, and if that connection is successful, the programme will attempt to reconnect it. In the second state, the subscriber will receive raw payload in the form of ciphertext that has not yet been displayed on the terminal. After that, the programme will check the AES key in the configuration file, and if it can, it will determine whether the ciphertext can be decrypted.

After the decryption process has entered the state splitting plaintext, which successfully separates the message from the digest, the digest will be temporarily saved. The plaintext produced by the state decryption contains both the digest and the original messages that the publisher has merged. On the decrypted text hashing state, the output from hashing is a SHA digest; after getting a new digest, going to the state "digest check," which functions to compare the temporary digest stored with the digest generated by the subscriber; if the digest values match, the subscriber will display the message and consider it to be valid data; otherwise, the programme will exit.

The suggested method uses symmetric encryption to deal with computing that exceeds resources since IoT devices have constrained resources. There is no protection against repudiation, but the repudiation problem can be solved if both the publisher and the subscriber use an arbitrator or trusted third party. Symmetric keys have several drawbacks, including that each entity that has the same key can make a signature. However, we can solve these problems by using a trusted third party that is safe to make key exchanges.

During the implementation phase depicted in Table 1, the environment is made up of the software. The Raspberry Pi 3 is the device environment used, and the Google Compute Engine (VPS) cloud server has the
following specifications: 1 Virtual CPU, 3.75GB Memory, and a 10GB Standard Persistent Disk.

Table: Software Environment

SL. No	Title	Explanation
1	CentOS 7 64 bit	OS for a cloud server.
2	Raspbian Strecth with Desktop, Kernel version 4.14	OS for Raspberry Pi3
3	Mosquitto 1.4.15	MQTT Server Broker
4.	Python 3	Publish/Subscribe Middleware made by python.

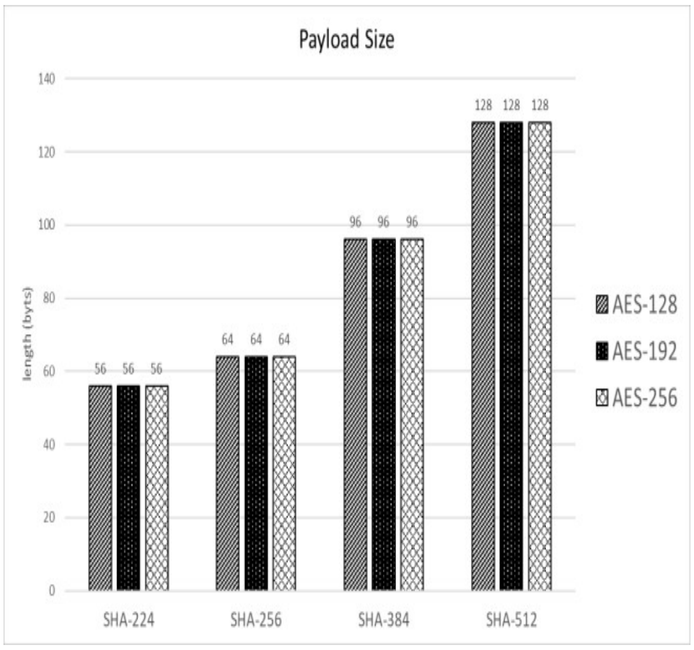


Fig.4

The payload size is the first overhead analysis to be performed. The payload size is determined by sending a payload "heartbeat: 66bpm" from the publisher to the broker for 5 minutes, after which the subscriber receives data from the broker. Figure 4 depicts the overhead of various payload sizes from the AES and SHA implementations for digital signatures. According to the results of the analysis, the AES key has no effect on the overhead of the payload length; rather, the effect occurred within the SHA itself. According to the analysis results, SHA-224 has an overhead of 56 bytes, SHA256 has an overhead of 64 bytes, SHA-384 has an overhead of 96 bytes, and SHA-512 has an overhead of 128 bytes.

The message delivery time is measured in the second overhead analysis. The system is tested by calculating how long it takes the system to send the payload from the publisher to the subscriber. Publishers will deliver payloads to subscribers via cloud brokers or VPS with Quality of Service (QoS) = 2. Because the QoS level is set to 2, each message is only received once by the intended recipient.

Fig. 5 depicts the delivery time overhead, SHA-224 with AES128 has an overhead of 50 ms, then a key increase from AES to 192, an additional increase of 32%, and finally, a key increase from AES to AES-256, an increase of 4.55% of SHA-192.

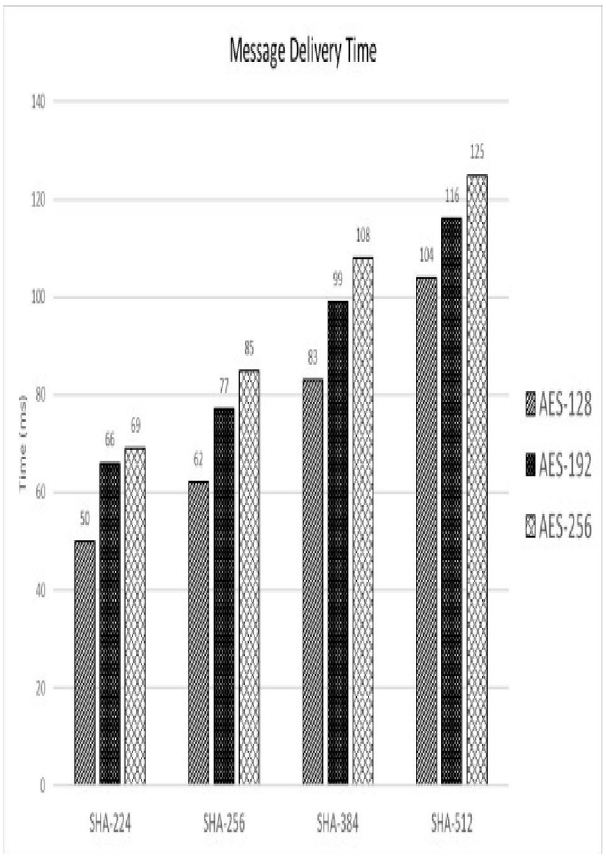


Fig.5

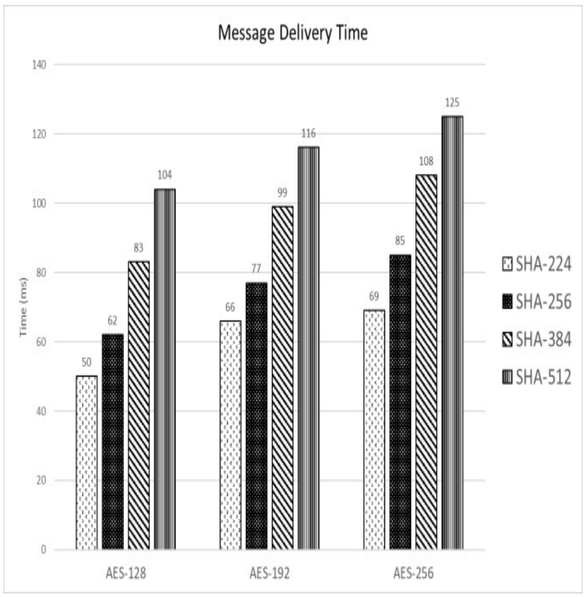


Fig.6

Whereas SHA-256 with AES-128 has an overhead of 62 ms, increasing the AES key to 192 results in a 24.19% increase. When the key is increased to 256, there is a 10.39% increase. The overhead on SHA-384 with AES-128 is 83 ms, and if the AES key is increased to 192, the gain is 19.28%, and if the AES key is increased to 256, the gain is 9.09%. At SHA-512 with AES-128 and an overhead of 104 ms, increasing the AES key size to 192 results in an 11.54% increase, while increasing the key size to 256 results in a 7.76% increase.

Figure 6 depicts data when the SHA key size is changed to the same AES key size. According to Fig. 6, as the key size of AES and SHA grows larger, the value of the message delivery time decreases.

The measurement of the process of the digital signature security mechanism is the third overhead analysis. The digital signature security mechanism with AES and SHA implementation has different encryption and decryption times. In other words, the length of processing time is affected by the key sizes of AES and the SHA version.

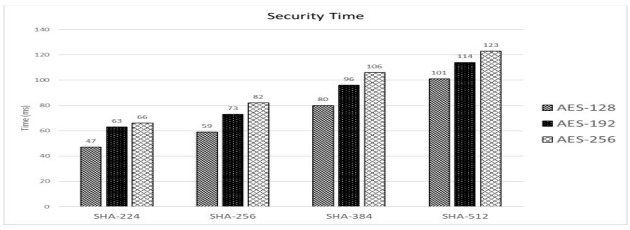


Fig.7

The length of the digital signature process by using AES and SHA varies due to the key differences applied, and testing is done by calculating how the system does a long encryption and decryption. The longer the process runs, the larger the key. On Fig. 7, it can be seen that SHA-224 with AES-128 has a 47 ms overhead. If the AES key is increased to 192, the increase is 34.04%; if it is increased again to a larger key, the increase is 4.76% for AES-256.

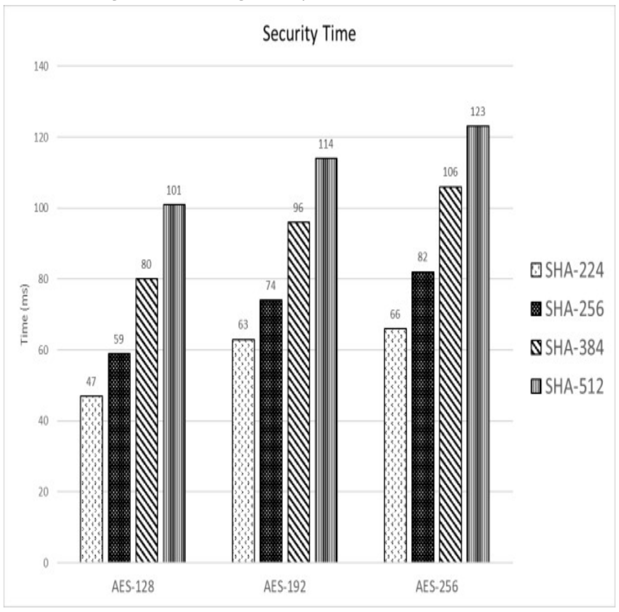


Fig.8

The overhead for the SHA-256 paired AES-128 is 59 ms; if the AES key is increased to 192, the overhead increases by 23.73%; if the AES key is increased to 256, the overhead increases by 12.33%. The overhead for SHA-384 is 80 ms, but if the AES key is changed to 192, the overhead increases by 20%. Furthermore, an increase of 10.42% occurs when the key is increased to 256 at SHA-512 with AES-128 with overhead of 101 ms, an increase of 12.87% occurs when the key is changed to 192, and an increase of 7.89% occurs when the key is increased to 256 bits.

Figure 8 depicts the AESSHA data. If the key size of AES is increased, the security mechanism time always increases. The length of time in the digital security mechanism is increased for each use of AES and SHA. It always increases in proportion to the size of the key used. There is no decrease in the security mechanism process because the key size affects the program's execution process. Figures 7 and 8 demonstrate this. The longer the key, the more overhead it generates and the longer it takes to issue security.

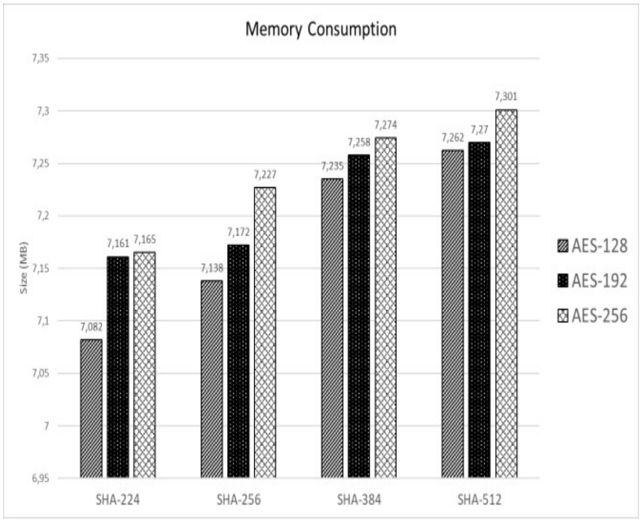
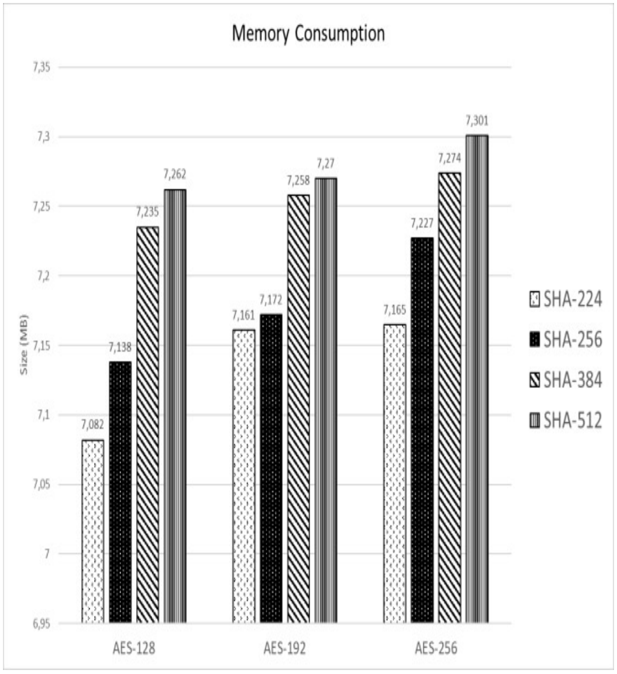


Fig.9

The fourth overhead analysis is a measurement of the proposed system's memory consumption. In this case, the system is tested using Linux's process status (ps) tool and measured according to the appropriate id process.

Figure 9 depicts the overhead of memory consumption caused by using various types of AES and SHA sizes in the environment. Furthermore, Fig. 9 depicts the data obtained graph. When the AES key size is increased, there is an increase.

Memory usage overhead on the SHA-224 when combined with AES128 is 7.082 MB. If the AES key is increased to 192, the increase is 1.12%, resulting in an increase of 0.079 MB. When the AES key is increased to 256, the expansion is 0.06%, with an additional 0.004 MB of memory usage. This increase is not significant. It has an overhead of 7.138 MB when using SHA-256 and AES-128. If the AES key is increased to 192, the result is a 0.47% increase. Furthermore, if the AES key is 256, the resulting inflation is 0.76%, with a 0.055 MB increase in memory. The overhead for SHA-384 combined with AES-128 is 7.235 MB.



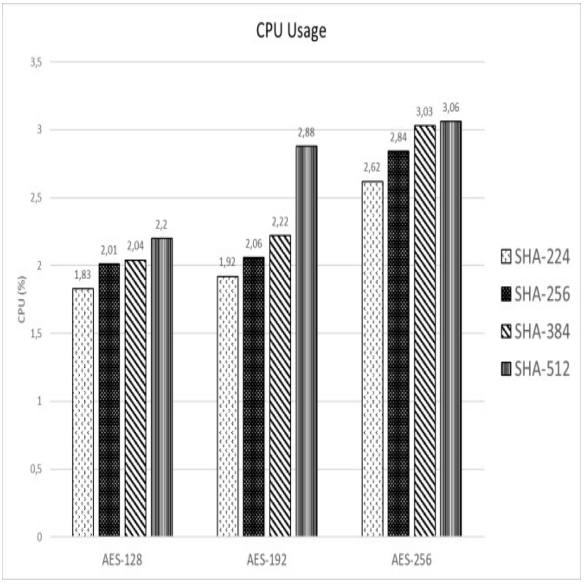


Fig.10

Fig.11

Then, if the key is increased to 256, there is a 0.22% or 0.016 MB increase. The overhead value for SHA-512 paired with AES-128 is 7.262 MB, which is the highest overhead value of all SHA key sizes. When the AES key is increased to 192 on SHA-512, it increases by 0.11% or equals 0.008 MB.

Following that, the key is changed by increasing the AES key to 256, a 0.43% or 0.031 MB increase.Figure 10 depicts data when the SHA key size is changed to the same AES key size. In addition, see Fig. 10. AES-128SHA-224 and AES256SHA-512 use the least amount of memory, while AES-128SHA-512 and AES256SHA-512 use the most.

According to the results of the preceding experiments, SHA-512 and/or AES-256 require the most resources when compared to the others. In Linux, the system is tested using the process status (ps) tool and measured according to the appropriate id process. Figure 11 depicts the CPU overhead caused by applying a digital signature with various AES and SHA key sizes. The overhead of SHA-224 in conjunction with AES-128 CPU usage is 1.83%. If the AES key is increased to 192, the increase is 4.92%, which is equal to 0.09. If the AES key is increased to 256, the increase is 36.46%, or 0.7. The CPU overhead for SHA-256 with AES 128 is 2.01%.

If the AES key is increased to 192, CPU usage will rise. That is, the resulting increase is 2.49%, or 0.05. If the key is increased to 256, the growth occurs again. The resulting improvement is 37.86%, or 0.78. The CPU usage overhead for SHA-384 with AES-128 is 2.04%. When the key 128 is increased to 192, there is an increase. In this case, the increase is 8.82%. There is also an increase if the key is raised to 256. Notably, the resulting expansion is 36,49%, or 0.81. The overhead for SHA-512 with AES-128 is 2.2%. An increase occurs when the AES key is increased by 30.91% to 192.

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Impact Factor: 7.301

Volume 3, Issue 2, January 2023

IV. CONCLUSION

The paper discusses an overhead analysis of digital signature implementations in MQTT in this chapter. The investigation AES and SHA are symmetric encryption algorithms that were used. End-to-end hashing requires publishers to encrypt and subscribers to decrypt. This method's digital signature can safely secure the MQTT payload. The attacker can view the payload using Wireshark based on penetration and overhead testing. However, the payload that would be visible is in the ciphertext. Because the message is cut off and the ciphertext is in byte form, it is difficult to determine the ciphertext content. Furthermore, the payload size grows due to the digital signature security mechanism that secures the payload.Additionaly, This paper's overhead analysis includes payload size, message delivery time, security time, memory consumption, and CPU usage. Furthermore, the overhead is performed by checking the various types of AES keys and SHA. According to the findings of this study's overhead analysis, the larger the key size of AES and SHA, the greater the overhead generated, and vice versa. The lightest digital signature implementation in this study was AES-128SHA-224, and the heaviest was AES-256SHA-512.

REFERENCES

[1]. Syaiful Andy, Budi Rahardjo, and BagusHanindhito. Attack scenarios and security analysis of mqtt communication protocol in iot system. International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), 4(September):600–604, 2017.

[2]. Syed Naeem Firdous, Zubair Baig, Craig Valli, and Ahmed Ibrahim. Modelling and evaluation of malicious attacks against the IoT MQTT protocol. Proceedings - 2017 IEEE International Conference on Internet of Things, IEEE Green Computing and Communications, IEEE Cyber, Physical and Social Computing, IEEE Smart Data, iThingsGreenComCPSCom-SmartData 2017, 2018-Janua:748–755, 2018. K. Elissa, "Title of paper if known," unpublished.

[3]. Manish Parmar Lochan Bisne. Composite Secure MQTT for Internet of Things using ABE and Dynamic S-Box AES. pages 1–5, 2017.

[4]. Yuri F Gomes, Danilo F S Santos, Hyggo O Almeida, and Angelo Perkusich. Integrating MQTT and ISO / IEEE 11073 for Health Information Sharing in the Internet of Things. 2015 IEEE International Conference on Consumer Electronics (ICCE), pages 200–201, 2015

[5]. Nut Aroon. Study of using MQTT cloud platform for remotely control robot and GPS tracking. 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, ECTI-CON 2016, 2016.

[6]. Yuvraj Upadhyay, Amol Borole, and D. Dileepan. MQTT based secured home automation system. 2016 Symposium on Colossal Data Analysis and Networking, CDAN 2016, 2016.

[7]. Singh Meena, MA Rajan, VL Shivraj, P Baramuralidhar P. Secure MQTT for Intenet of Things (IoT). 2015 Fifth International Conference on Communication Systems and Network Technologies

[8]. RashiDhagat and Purvi Joshi. New approach of user authentication using digital signature. 2016 Symposium on Colossal Data Analysis and Networking (CDAN), pages 1–3, 2016.

[9]. Muhammad Arif Mughal, Xiong Luo, and A T A Ullah. A Lightweight Digital Signature Based Security Scheme for Human-Centered Internet of Things. 2018.

[10]. Don Jomar S Hombrebueno, Ma Gracia Corazon E Sicat, Jasmin D. Niguidula, Enrico P. Chavez, and Alexander A. Hernandez. Symmetric cryptosystem based on data encryption standard integrating HMAC and digital signature scheme implemented in multi-cast messenger application. 2009 International Conference on Computer and Electrical Engineering, ICCEE 2009, 2:327–334, 2009.

[11]. Sourabh Chandra, SmitaPaira, B Tech Student, SkSafikul, Alam Assistant, and Goutam Sanyal. A comparative survey of symmetric and asymmetric key cryptography. 2014 International Conference on Electronics, Communication and Compytational Engineering (ICECCE), pages 83–93, 2014.

[12]. Alizai, Zahoor Ahmed. Improved IoT Device Authentication Scheme Using Device Capability and Digital Signatures. 2018 International Conference on Applied and Engineering Mathematics.