Safe MQTT-SN: a lightweight secure encrypted communication in IoT

T L Kao¹*, H C Wang² and J E Li²

¹Department of Communication Engineering, I-Shou University, No. 1, Sec. 1, Syuecheng Rd., Dashu Dist., Kaohsiung City 84001, Taiwan
²Department of Electronics Engineering, National Taiwan University of Science and Technology, No. 43, Keelung Rd., Sec. 4, Da’an Dist., Taipei City 10607, Taiwan

*Email: tlkao@isu.edu.tw

Abstract. The Internet of Things (IoT) is a technology trend that has been evolving in recent years. With the development of sensing technology and communication technology, the various applications of IoT are integrated into our lives gradually. Various security issues also need to be resolved with the convenience brought to us by IoT. MQTT or MQTT-SN is a lightweight communication protocol in IoT, which is more compactly on sensor nodes with limited resources. This paper provides a secure connection and end-to-end encrypted communication method which is based on MQTT-SN. By using digital signature (ECDSA), hash function, key exchange (ECDHE), AEAD (ChaCha20-Poly1305) to achieve a safe version of MQTT-SN called Safe MQTT-SN. The Safe MQTT-SN enables an end-to-end encrypted communication between the subscriber and publisher. The implementation of these algorithms enable the Safe MQTT-SN to have good performance on sensor nodes with limited computing power. Comparing with mosquitto MQTT-TLS, the handshake time of MQTT-TLS is about 56.12ms and Safe MQTT-SN is about 24.78ms on Zedboard.

1. Introduction

With the development of sensing technology and communication technology, various applications of the Internet of Things have gradually entered our lives. It is possible to transmit more confidential information such as confidential sensor data, fingerprints, or iris information through the Internet of Things. At this time, the transmission protocol of Internet of Things using general plaintext transmission is very insecure. The plaintext transmission can easily enable a third party to retrieve the transmission information. At this time, authentication of transmission nodes and encryption of messages will be necessary to protect confidential information.

Currently, Eclipse Mosquitto performs authentication and encryption for MQTT [1][2], but it is only limited to the encrypted communication between the BROKER and the client, so that messages may leak from the BROKER. There is a lack of point-to-point protection between the client and the client. However, to establish a secure communication, identity authentication and message encryption are basically required. Identity authentication is implemented by an asymmetric encryption algorithm. Asymmetric encryption algorithms usually require higher system computing power. There will be a big computing load on the IoT nodes. How to choose a suitable Digital Signature, key agreement, and message encryption method to establish a secure connection and transmission of the Internet of Things, there are many conditions that need to be considered. This paper uses the asymmetrically encrypted method which is Elliptic Curve Digital Signature Algorithm (ECDSA) digital signature to achieve identity authentication [3-5]; Elliptic Curve Diffie–Hellman Ephemeral (ECDHE) [6] to make secure
key exchange; and ChaCha20-Poly1305 to realize symmetric encryption and message authentication. Compared with other similar algorithms, the above algorithms can perform much better on devices with low computing performance.

In this paper, the Linux operating system is ported to the ZedBoard (Xilinx Zynq-7000 AP SoC XC7Z020-CLG484) development board which be used as the execution environment of the message protocol MQTT-SN in IoT. ZedBoard executes the programs of Subscriber, Publisher, and BROKER in MQTT-SN, and implements the connection transmission method of simple and secure authentication based on MQTT-SN by adding ECDSA, ECDHE, and ChaCha20-Poly1305 algorithms in the BROKER to establish a security certification connection with the Client node. Compared with the original MQTT-SN, the simple and safe authentication connection transmission method of the Internet of Things (Safe MQTT-SN) has better authentication, message confidentiality and integrity confirmation effects, and can reduce the occurrence of information security problems in IoT.

Next, the second section of the paper compares the algorithms required to establish a secure connection, such as: digital signature algorithms RSA and ECDSA [7], key agreement ECDH and ECDHE, authentication encryption AES-GCM and ChaCha20-Poly1305 [8][9]. The third section describes the realization of the Safe MQTT-SN method [10]; the fourth section is the experimental results and analysis, and finally the conclusion and prospects.

2. The algorithms required for a secure communication

The most way to establish a secure communication is to encrypt the message. The transmitter node and the receiver node use the same key to encrypt and decrypt the message. However, the transmission of the key is usually over an insecure channel, which will allow the key to be intercepted by a third party, causing the encryption to lose its effect. Therefore, how to securely exchange keys between two communicating nodes is an important part of establishing a secure communication.

At first, Asymmetric encryption is applied to digital signatures so that both nodes in communication can confirm each other’s identities, while key agreement is to exchange key material between the two parties so that both parties can obtain the shared key. Finally, the symmetric encryption algorithm can establish a basic secure communication by using the shared key to encrypt the messages communicated by both nodes.

Due to the limitations of low power consumption and low computing power for resource-constrained IoT devices, appropriate the asymmetric encryption algorithm, the key agreement algorithm, and the symmetric encryption algorithm must be selected.

2.1. Digital signature

RSA and ECDSA (Elliptic Curve Digital Signature Algorithm) are commonly used algorithms for digital signatures. RSA and ECDSA are cryptographic and signature algorithms based on different mathematical problems and have different computational complexity. This paper uses Zedboard development boards to emulate nodes or devices with limited computing power in the Internet of Things. Openssl-1.0.2l is ported to Zedboard to implement RSA and ECDSA algorithms.

| Key strength and signature length. |
|-----------------------------------|
| Key length (bits) | Sign length (bits) |
| RSA | ECDSA | RSA | ECDSA |
| 512 | 112 | 64 | 28 |
| 1024 | 160 | 128 | 40 |
| 2048 | 224 | 256 | 56 |

Table 1 is a comparison table of encryption strength and signature length. Under the same encryption strength, the key length of ECDSA is significantly smaller than that of RSA, and as the encryption strength increases, the key length ratio of RSA to ECDSA also increases from 4.5:1 to 12:1. This trend shows that the higher the encryption strength, the key length of ECDSA will be more advantageous than that of RSA, which can save more key storage space. In addition, under the condition of the same key
strength, the signature size of ECDSA is smaller than that of RSA signature. The signature length of RSA is increased from 2.3:1 to 4.6:1 than ECDSA. With a larger key length, ECDSA can save more signature size than RSA and can save more bandwidth during transmission.

Table 2 shows the performance comparison of RSA and ECDSA on Zedboard. Since RSA having a lot of large number multiplications and modular operations in key generation, the performance of the operations is very low. For example, when the key lengths are RSA 1024bits and ECDSA 160bits in table 2, the time consumed of signature by RSA is about 3 times that of ECDSA. When the key length is larger, the ratio of the key length between the RSA and the ECDSA of the same security level will be larger, so the difference in the influence of the key length on the operation of the signature will increase. In the verification of table 2, the signature performance of RSA is better than ECDSA under different key lengths. This is because the public key is used to perform operations when verifying in RSA. Compared with RSA digital signature, ECDSA has the following advantages: (1).With the same security strength, the required key length is shorter. (2).The amount of calculation is reduced, and the processing speed is faster. ECDSA will be much faster than RSA in the case that private keys are required to participate in the calculation. (3).Less memory space required for storing parameters. (4).Less required bandwidth for network loads. For the nodes of the Internet of Things, the hardware resources are usually limited. Therefore, this paper uses ECDSA to realize the digital signature algorithm for identity authentication.

| Key generation time(s) | RSA   | ECDSA |
|------------------------|-------|-------|
| 512                    | 364.54| 907.58|
| 1024                   | 2.67  | 4.68  |
| 112                    | 3.97  | 15.44 |
| 160                    | 3.08  | 5.51  |

2.2. Key agreement

ECDH (Elliptic Curve Diffie–Hellman key exchange) is a variant of DH (Diffie–Hellman key exchange). The key pair replaces the large number parameters used by the original DH, and the mathematical principle characteristics of the elliptic curve are used to replace the large number operations and modular operations required by the original DH. Therefore, it can have better performance and security. ECDH only needs the two nodes to exchange each other's public key to obtain the same shared key through their private key and the other node's public key.

ECDH does not have forward secret key agreement. In order to prevent this from happening, changing ECDH to ECDHE can effectively solve this problem. ECDHE no longer uses the private key fixed by both nodes but randomly generated each time. Therefore, even if the private keys of both nodes are accidentally leaked, hackers cannot obtain the shared key through the private key. Therefore, ECDHE improves the ECDH problem while maintaining the advantages of ECDH. It should be combined with a digital signature to achieve the most complete security protection. Therefore, this paper also uses ECDHE as a key agreement method.

2.3. AEAD authentication encryption

For symmetric encryption, there is no way to confirm whether the key used for decryption and the decrypted data match each other during decryption, which means that even if the wrong key is used to decrypt, a set of data will be solved. However, the receiver node does not know whether the solved data is correct. Therefore, in addition to encryption and decryption, the transmitter node need to add some verification methods to help the receiver node determine whether the decryption is correct when decrypting. Usually, the method of verification is to use Hash Function. AEAD (Authenticated Encryption with Associated Data) is a commonly used algorithm that combines encryption and authentication. AES-GCM and ChaCha20-Poly1305 are commonly used in AEAD.

AES-GCM (AES Galois/Counter Mode) is a kind of AEAD. AES-GCM is an encryption mode of AES (Advanced Encryption Standard), using CTR mode encryption, and then using GMAC method to obtain
MAC. After the encryption is completed, a cipher and an authentication tag (Tag) will be generated. However, AES-GCM has a large amount of calculation, and the calculation performance cannot be maximized if it is implemented only by software. Because AES itself is a commonly used symmetric encryption algorithm. Therefore, hardware chips will provide AES-NI (AES New Instructions) to improve the performance of AES.

ChaCha20-Poly1305 is a kind of AEAD, a new authentication encryption method adopted by Google. It uses ChaCha20 as a symmetric encryption algorithm and Poly1305 as a message authentication code. ChaCha20-Poly1305 uses ARM reduced instruction set and the computing effect on the platform is remarkable. ChaCha20-Poly1305 algorithm is relatively streamlined and the security is also strong. Figure 1 shows the test results on ZedBoard. The performance of ChaCha20-Poly1305 is significantly higher than that of AES-GCM on ZedBoard that does not support AES-NI. The reduction of the processing time of the calculation can also reduce the power consumption. If the large data needs to be encrypted, ChaCha20-Poly1305 can save power and time. Since the implementation platform of this paper is ARM and does not support AES-NI, ChaCha20-Poly1305 is chosen for the AEAD authentication encryption algorithm.

![Comparison of Encryption and Decryption Time](image)

**Figure 1.** Performance of AES-GCM and ChaCha20-Poly1305 on Zedboard.

### 3. Establish MQTT-SN transmission with authentication and encryption

Currently, MQTT uses TLS (Transport Layer Security) to establish a secure connection. When the Publisher side wants to establish a secure connection with the subscriber side, it is not that the publisher and the subscriber establish a TLS connection. Instead, the publisher, the subscriber individually authenticate with BROKER to establish a TLS connection.

Although TLS is a very secure connection method, the TLS connection establishment between Publisher and Subscriber in the architecture is not a client-to-client secure connection. Even though the encrypted connection is established, the BROKER can still know the message sent by both clients.

The messages published by MQTT and MQTT-SN are in plaintext and are not encrypted. The bottom layer of MQTT is TCP, and MQTT-SN runs on UDP, which can save power consumption and memory usage. To improve the connection security problem, this paper proposes Safe MQTT-SN which is a new method of simple identity authentication and encrypted communication based on MQTT-SN without changing the original connection process.

Safe MQTT-SN attaches the data required for authentication or encryption to the connect, connack, subscribe, and suback packet that MQTT-SN transmits to each other. Therefore, the client can authenticate with the BROKER and negotiate a symmetric encryption key. Safe MQTT-SN is executed
in a BROKER environment trusted by users, and BROKER will only provide its own certificate to trusted users. Figure 2 is depicted as Safe MQTT-SN protocol.

![Figure 2. Safe MQTT-SN protocol.](image)

When entering the publish stage, both Publisher and Subscriber have successfully completed connect and subscribe. At this time, Publisher and Subscriber both have the same Key\textsubscript{topic}, and use Key\textsubscript{topic} and topic through HMAC-based Key Derivation Function (HKDF) to generate the session\_key [11]. The session key can be used for encrypted communication between Publisher and Subscriber.

4. Testing and results analysis
The platform used in the test is the Zedboard (Dual ARM Cortex-A9 MPCore @667MHz) development board. The test architecture is shown in Figure 3. The Linux system is ported to the Zedboard development board and the Subscriber, Publisher and BROKER programs are executed on Linux. Under the environment, compare the performance of Safe MQTT-SN and mosquitto MQTT-TLS.

![Figure 3. Test architecture.](image)

Mosquitto MQTT-TLS uses RSA for identity authentication and key agreement, while Safe MQTT-SN uses ECDSA and ECDHE. Due to the large amount of calculation and computing performance required by the RSA, the time required for the handshake will be longer. In addition, due to the complexity of
the TLS protocol and the need to exchange certificates, mosquitto MQTT-TLS has more handshake overhead. In contrast, Safe MQTT-SN is simpler and lighter and does not need to exchange certificates. Therefore, a smaller overhead is generated when the Safe MQTT-SN handshake is established, which can save bandwidth.

| Handshake time (ms) | Mosquitto MQTT-TLS | Safe MQTT-SN |
|---------------------|--------------------|--------------|
| 56.12               | 24.78              |
| Handshake overhead (bytes) | 2391            | 212          |

Table 3. Performance of MQTT-TLS and safe MQTT-SN on Zedboard.

Safe MQTT-SN uses ChaCha20-Poly130 as a symmetric encryption algorithm, and here is a comparison of power consumption with the commonly used AES-GCM. The test platform Zedboard is 2500 DMIPS, 5 DMIPS/mW, and the power consumption on Zedboard can be estimated as time (μs) x 0.5μW. From the performance test results in Figure 1, the power consumption of ChaCha20-Poly130 encrypted 50bytes data is estimated to be 7μW. The power consumption of AES-256-GCM encrypted 50bytes data is 27μW. The results show that ChaCha20-Poly130 has faster encryption and decryption speeds and lower power consumption on platforms without AES-NI support, which is a better choice for resource-constrained devices like IoT nodes.

5. Conclusions

This paper implements a simple and safe authentication connection transmission method (Safe MQTT-SN) based on MQTT-SN. Compared with the general use of RSA, this paper chooses to use ECDSA, ECDHE, and ChaCha20-Poly130 algorithms. IoT nodes do not need high computing power. In Safe MQTT-SN, because the client can encrypt and decrypt the published message, the BROKER cannot snoop on the content of the published message. In addition, through the use of ChaCha20-Poly130, the published message can be authenticated without misuse of the error message. When transmitting large amounts of data on platforms that do not support AES-NI, ChaCha20-Poly130 has relatively lower power consumption than AES-GCM. When the Internet of Things begins to transmit more confidential or sensitive information, such as retina information, fingerprints, or body data, it is extremely important to transmit with authentication and encryption. It is believed that information security issues such as authentication and encryption of the Internet of Things transmission will be an essential research and development direction.

References
[1] Light R A 2017 Mosquitto: server and client implementation of the MQTT protocol J. Open Source Software 2 10.21105/joss.00265
[2] Cohn R J and Coppen R J 2014 MQTT Version 3.1.1 IBM OASIS Standard, ed Banks A and Gupta R
[3] Frimpong E and Michalas A 2020 Proc. of the 35th Annual ACM Symposium on Applied Computing p 767
[4] Blake I, Seroussi G and Smart N 1999 Elliptic Curves in Cryptography (Cambridge: Cambridge University Press)
[5] Govindan K and Azad A P 2015 Proc. Int. Conf. on IEEE Consumer Communications and Networking p 290
[6] Yildiz K, Buldu A and Saritas H 2016 J. Security and Communication Networks 9 4242
[7] Rivest R L, Shamir A and Adleman L 1978 J. Communications of the ACM 21(2) 120
[8] Bernstein D J 2008 Workshop Record of SASC (http://cr.yp.to/papers.html#chacha) p 6
[9] Bernstein D J 2005 Fast software encryption (Berlin: Springer-Verlag), ed Gilbert H and Handschuh H p 32
[10] Stanford-Clark A and Truong H L 2013 MQTT-SN Version 1.2 IBM OASIS Standard
[11] Krawczyk H 2010 Proc. of the 30th Annual International Cryptology Conference (Santa Barbara) vol 6223 p 631