2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)

Lightweight Security Scheme for MQTT/MQTT-SN Protocol

Ousmane Sadio, Ibrahima Ngom, Claude Lishou
Département Génie Informatique, Ecole Supérieure Polytechnique (ESP), Dakar, Senegal
Email: ousmane.sadio@ucad.edu.sn, ibrahima.ngom@ucad.edu.sn, claude.lishou@ucad.edu.sn

***Abstract*—In the coming years, sensors will likely have a permeated every aspect of our life. Several works explain how the Internet of things (IoT) will have an impact on almost all aspects of our life and why security is at the top of the list of IoT challenges. Constrained nodes constitute a significant portion of devices in IoT. These nodes are characterized by severe constraints on power, memory, and processing resources, therefore, do not support conventional security protocols such as Transport Layer Security (TLS). Message Queue Telemetry Transport (MQTT) is a lightweight communication protocol particularly adapted for constrained nodes. Security solution, in MQTT protocol, can be achieved in multiple layers. To ensure end-to-end encryption, Authenticated Encryption with Associated Data (AEAD) is one of the most recommended solutions. Actually, the Advanced Encryption Standard (AES) is one of the most widely used standard encryption methods. However, constrained nodes processors did not have hardware support for AES and the physical-layer packet size of these nodes is limited. This paper proposes ChaCha20-Poly1305 AEAD as a solution to secure constrained nodes communication over MQTT/MQTT-SN. ChaCha20 and Poly1305 are respectively lightweight stream cipher and one-time authenticator which continue gain popularity from crypto community. A prototype of the proposed solution is implemented on constrained nodes like Arduino UNO. The paper mainly provides results related to memory footprint and execution time. These results indicate that the proposed scheme requires small amount of memory and present low processing time.**

Keywords—ChaCh20-Poly1305, Constrained Nodes, IoT security, MQTT/MQTT-SN

I. Introduction

Internet of Things (IoT) covers a whole range of technologies, from the sensors/actuators devices to cloud storage. MQTT (Message Queue Telemetry Transport) [1] is a lightweight protocol which works on publish and subscribe model to ensure efficient communication across platforms. MQTT is widely used for IoT because of its small footprint and minimal bandwidth consumption. For resource-constrained nodes and nodes that do not have TCP/IP connection, a lightweight version of MQTT named MQTT-SN (MQTT for Sensor Network) [2] is proposed. By its nature, MQTT is a plain protocol but allow implementers to choose the network, privacy, authentication and authorization technologies. However, MQTT specification [1] recommends usage of TLS (Transport Layer Security) to secure transaction between publisher/subscriber (clients) and broker (server). There are practical limitations to this approach. TLS did not ensure end to end encryption and on the other is "too heavy" for resource-constrained devices. Indeed, TLS adds overhead (memory, energy) that cannot be supported by constrained nodes. Therefore, MQTT specification proposes an AES and ChaCha20 encryption algorithm for constrained devices network. However most embedded processors did not have hardware support for AES (Advanced Encryption Standard). ChaCha20 [3] algorithm is based on an ARX (Addition Rotation XOR) construction which is CPU-friendly and is faster than AES in software platform. Authenticated Encryption with Associated Data (AEAD) simultaneously provides confidentiality, integrity, and authenticity. ChaCha20-Poly1305 AEAD is comprised of ChaCha20 stream cipher and Poly1305 authenticator proposed by D. J. Bernstein. Both have received considerable scrutiny from crypto community and literature [4], [5] and are considered safe. ChaCha20-Poly1305 AEAD is currently officially supported ciphers by Google in their Chrome browser for TLS and also support has been added to OpenSSL, as well as OpenSSH. Motivated by the above, this paper proposes to secure MQTT protocol by using ChaCha20-Poly1305 AEAD. The proposed security scheme concern MQTT Payload Encryption and Message Data Integrity. The application data stay encrypted and broker has no way to look into the encrypted data. Therefore, MQTT PUBLISH packet payload is fully protected and do not involve any broker configuration, this is likely very popular method of protecting data.

The paper is organized as follows. Section II presents the background of Constrained Nodes, MQTT protocol, and ChaCh20-Poly1305 AEAD. Section II reviews the state of the art of security schemes for MQTT linked to resource-constrained nodes. Section IV proposes a new security scheme for constrained nodes. Finally, Section V concludes the paper and discusses directions for future extensions.

II. Background

A. Constrained-Node Networks

Constrained-Node Networks is largely consisting of constrained nodes operating in a constrained network. RFC 6574 [6] gives the description of constrained objects and network by focusing on three points: energy constraints, bandwidth constraints, and memory constraints. RFC 7102 [7] specify common terminology to be used in ROLL (Routing Over Low power and Lossy network) Work Group documents. Thereby, an LLN (Low-Power and Lossy Network) is defined as a set of embedded devices with limited power, memory, and processing resources interconnected by a variety of links. LLN is therefore subject to high packets loss, low data rates or

978-1-7281-2949-5/19/$31.00 ©2019 IEEE

asymmetric connection. RFC 7228 [8] try to quantify some constraints commons to all constrained nodes, namely code size (ROM/Flash), data size (RAM), computing power (CPU). RFC 7228 also characterized constrained network with following constraints: low achievable bitrate/throughput, high packet loss and high variability of the packet loss, highly asymmetric link characteristics, severe penalties for using larger packets, limits on reachability over time… RFC 6574 gives some values, which characterize a constrained network: data transmission rates vary from 20 kbps to 900 kbps and the physical-layer packet size limited to ~100 bytes. Table 1 summarizes the set of constraints for this kind a node.

B. *MQTT protocol*

MQTT is a Client-Server publish/subscribe messaging transport protocol. MQTT is adopted and published as an official OASIS standard [1]. MQTT is a lightweight protocol with low complexity, low power and low footprint implementations, low overhead and is suitable for constrained-node networks. The protocol runs over TCP/IP (port 1883 and port 8883 for MQTT over TLS/SSL). An MQTT client, which can be a publisher or subscribers, always publishes or subscribes to a specific topic. A central server, known as broker, receive subscriptions from clients on topics, receive messages from clients and forward these, based on clients' subscriptions, to interested clients. The MQTT topics are hierarchical with the form of file paths, e.g., garden/grass/humidity. The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. MQTT focuses on reliable messaging, therefore it includes Quality of Service (QoS) levels (level 0: "at most once", level 1: "at least once", and level 2: "exactly once"). MQTT requires a TCP/IP connection, nevertheless, this makes it harder to implement in simpler sensor networks essentially composed of constraint nodes. If the client chooses to use a durable connection, then the broker will store undelivered messages if the client disconnects and the broker try subsequently to deliver these saved messages as soon as the client connects again.

TABLE I. CONSTRAINED DEVICES CLASSIFICATION

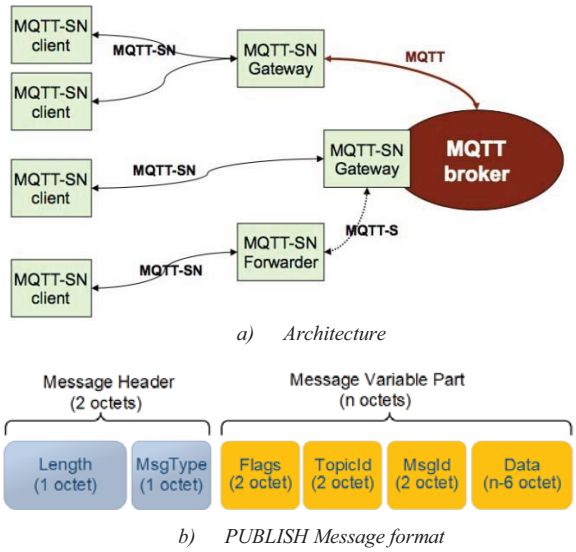
Classes of Constrained Devices (KiB = 1024 bytes)		
Name	***data size (e.g., RAM)***	***code size (e.g., Flash)***
Class 0, C0	<< 10 KiB	<< 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB
Classes of Energy Limitation		
Name	***Type of energy limitation***	***Example Power Source***
E0	Event energy-limited	Event-based harvesting
E1	Period energy-limited	Battery that is periodically recharged or replaced
E2	Lifetime energy-limited	Non-replaceable primary battery
E9	No direct quantitative limitations t	Mains-powered
Strategies of Using Power for Communication		
Name	***Strategy***	***Ability to communicate***
P0	Normally-off	Reattach when required
P1	Low-power	Appears connected, perhaps with high latency
P9	Always-on	Always connected

C. *MQTT-SN protocol*

MQTT-SN [2] can be considered as a version of MQTT, which is adapted to deal with constrained nodes, both from a footprint/complexity standpoint, and to adapt to the fact that constrained nodes may not have TCP/IP support. MQTT-SN can work on top of any unordered, lossy and bidirectional network protocols. MQTT-SN uses UDP connection mainly because UDP is much lightweight than TCP over a wireless link. To reduce packet size, MQTT-SN support topic ID (2 bytes) instead of topic name encoded in UTF-8. When a client sends a registration request to a broker, it indicates the topic ID with the associated topic name. Thus, for a further request, only topic ID will be used. The architecture of MQTT-SN is detailed in Fig. 1.a. MQTT-SN gateways main function is the translation between MQTT and MQTT-SN. It uses conventional MQTT over TCP/IP to publish/subscribe message to/from MQTT broker. MQTT-SN support slipping client, so constrained devices can go to sleep state will receive buffered messages from the server once they wake up. MQTT-SN clients can also access MQTT-SN gateways via an MQTT-SN forwarder in case the gateway is not directly attached to their network. An MQTT-SN message consists of two parts: a 2- or 4-bytes long header and an optional variable part. The PUBLISH message, Fig. 1.b, is used by both clients and gateways to publish data for a certain topic. The Data field corresponds to a payload of an MQTT PUBLISH message. It has a variable length and contains the application data that is being published.

D. *MQTT security*

Security in MQTT protocol is left to the implementer the responsibility to provide appropriate security features. The security solution can be achieved in multiple layers and must ensure various security options in terms of authentication, authorization and data confidentiality. The MQTT specification [1] lists the security concerns to consider when implementing MQTT. Authentication can be achieved using CONNECT Packet, which contains Username and Password fields. Authenticating credentials are sent in plaintext and some form of encryption should be used [9]. MQTT clients can also provide



a) *Architecture*

b) *PUBLISH Message format*

Fig. 1. MQTT-SN Architecture and packet format

client ID when it registers at the broker. A Virtual Private Network is another option for end-host authentication. MQTT client can authenticate the broker by using SSL certificates sent from the server. Authorization in MQTT is used to restrict access to broker resources based on information provided by the client. To verify data integrity, a hash function can be used. TLS can provide such algorithms. For confidentiality of MQTT messages, encryption algorithm can be used. However, these algorithms must be lightweight and required client support. Encryption can be implemented either as end-to-end (publisher and subscribers) or just client-to-broker. Encryption in lower layers can be used when transparent encryption has opted. They can be achieved in the transport layer with TLS or in the link layer with AES. MQTT-SN does not define any security mechanism at protocol level. However, for MQTT-SN over UDP, DTLS can be used for transport layer security.

E. ChaCh20-Poly1305 AEAD

ChaCha20 and Poly1305 have been designed for high-performance software implementations and to minimize leakage of information through side channels.

1) ChaCha20

ChaCha, based on Salsa20, is a stream cipher developed by D. J. Bernstein in 2008 and is published by the RFC7539 [3]. ChaCha20 is a variant of ChaCha which has 20 rounds, a 96-bit nonce N, and a 256-bit key K. ChaCha20 encryption function transform an arbitrary-length plaintext P to a ciphertext C of the same length. This function is defined as follows in [10]:

$$ChaCha20 : \{0,1\}^{256} \times \{0,1\}^{96} \times \{0,1\}^{*} \rightarrow \{0,1\}^{*} \quad (1)$$
$$(K, N, P) \rightarrow C$$

The basic operation of Chacha20 algorithm is the quarter round, which consists of integer addition, bitwise Exclusive OR and n-bit left rotation. Chacha20 block function generates a keystream block to XOR with a plaintext. More detail can be found in [3].

2) Poly1305

Poly1305 is a one-time authenticator designed by D. J. Bernstein in 2008 and is published by the RFC7539 [3]. An arbitrary-length l message M is broken up into 16-byte chunks and fed into a polynomial $mod 2^{130} - 5$ based on the 16-byte authentication key (r, s). The final polynomial value is then combined with a 16-byte nonce to create the authentication token T used to authenticate the message. In [10], the Poly1305 function is defined as follows:

$$Poly1305 : \{0,1\}^{128} \times \{0,1\}^{128} \times \{0,1\}^{8l} \rightarrow \{0,1\}^{128} \quad (2)$$
$$(r, s, M) \rightarrow T$$

The ChaCha20 can be used to generate the one-time 32-byte secret key (r, s) using a 32-byte secret key K and a 12-byte nonce N. More detail can be found in [3].

3) AEAD_CHACHA20_POLY1305

AEAD_CHACHA20_POLY1305 [3] is an Authenticated Encryption with Associated Data (AEAD) based on the combination of ChaCha20 stream cipher and the Poly1305 authenticator. AEAD_CHACHA20_POLY1305 takes as input a 256-bit key K, a 96-bit nonce N, an arbitrary length plaintext P, and an arbitrary length Additional Authenticated Data (AAD). This function returns a ciphertext C and a 128-bit authentication tag T which is the output of the Poly1305 function. AEAD_CHACHA20_POLY1305 is defined as follows in [10]:

$$CP: \{0,1\}^{256} \times \{0,1\}^{96} \times (\{0,1\}^{*})^{2} \rightarrow \{0,1\}^{*} \times \{0,1\}^{128} \quad (3)$$
$$(K, N, P, AAD) \rightarrow (C, T)$$

Decryption works in a similar way with some differences: the roles of ciphertext and plaintext are reversed. The Poly1305 function is still run on the AAD and the ciphertext and the generated tag must be bitwise compared with the received tag in order to verify the authenticity of data.

III. RELATED WORKS

In literature, we can find some usage of MQTT protocol for constrained nodes. Prada et al. [11] propose to use MQTT for communication with resource-constrained devices for educational applications. Interactive user interfaces based on web standards use MQTT to parametrize and communicate with Arduino that, in turn, controls actuators. Ahmed et al. [12] use MQTT to remotely control robots equipped with relevant sensors, actuators, and manipulators. The authors propose an architecture based on 6LoWPAN robotized mesh network connected to cloud service using MQTT. Security schemes for MQTT are proposed in several works, however, only some papers address an issue linked to resource-constrained nodes. Pittoli et al. [13] discuss the use of DTLS to secure MQTT-SN protocol. The authors describe the interactions between the entities in MQTT-SN architecture and observe the first drawback of DTLS. The latter implies more than ten messages at the connection of a broker to MQTT-SN GW and MQTT-SN client. Niruntasukrat et al. [14] propose an authorization mechanism by taking into account the limited resources of nodes. The authors base their design on OAuth 1.0a protocol with several modifications to accommodate the restrictions of IoT devices. Security analysis shows that their solution protects MQTT protocol against eavesdropping, man-in-the-middle, replay, rogue AuthServer... Performance evaluation consists of measure the authorization delay of three types of devices including Arduino Mega. Singh et al. [15] propose SMQTT and SMQTT-SN security schemes based on Key/Ciphertext Policy-Attribute Based Encryption (KP/CP-ABE) using lightweight Elliptic Curve Cryptography. The authors also give a security analysis of their proposed scheme and show that their protocols are secure under several attacks. The performance result did not include constrained nodes, therefore the usage of their security scheme for these classes of nodes is not proved. Katsikeas et al. [9] present a secure deployment of MQTT for industrial domain. A constrained node, Zolertia Z1, is used for performance evaluation. The authors conclude that payload authenticated encryption with AES-OCB is the most suitable for industrial applications. However, if the node is limited by payload size, the payload encryption with AES-CBC could be used. De Santis et al. [10] present one of the rare works discussing the usage of chacha20-poly1305 AEAD for lightweight IoT applications. The authors propose an optimized implementation of ChaCha20 for ARM Cortex-M4 processors. Performance evaluation shows that ChaCha20-Poly1305 ciphers are promising candidates to secure emerging IoT applications with tight speed and space constraints.

IV. Proposed Security Schemes

A. Architecture description

The proposed architecture, Fig. 2, is composed of several entities. MQTT-SN client is a constrained node based on Arduino Uno. The Arduino Cryptography Library[1] is used to encrypt, decrypt and authenticate MQTT PUBLISH message data. Libsodium is also a cryptographic library. However, it does not propose a compatible version of CHACHA20_POLY1305 for Arduino. An MQTT-SN client is able to serialize and deserialize MQTT-SN packets thank ESIBot/MQTT-SN-Arduino library[2]. Therefore, these packets can be sent to the UART serial interface without additional configuration. Another advantage is that most of wireless transceiver modules like BLE and XBEE propose a UART interface for data transmission. Since these kinds of wireless modules do not offer IP connectivity, MQTT-SN client cannot communicate directly with MQTT-SN Gateway. It is possible to integrate on MQTT-SN Gateway a Serial Port Bridge that is able to convert UART frames into UDP packets. A Single Board Computer (SBC), Raspberry Pi, is used as MQTT-SN Gateway. Based on MQTT-SN Tools[3], a Serial Port Bridge is implemented on the SBC. The Eclipse Paho MQTT-SN library[4] is used to implement the functionality of MQTT-SN Gateway on the SBC. The Mosquitto/RSMB is an eclipse project that proposes an MQTT Broker with MQTT-SN capability. The forked version[5] of Mosquitto/RSMB is used as MQTT Broker in the proposed architecture. MQTT client[6] is a node which is not necessarily constrained and have IP connectivity (UDP or TCP/IP).

Before publishing data, an MQTT-SN client firstly encrypts and generate authentication token based on AEAD_CHACHA20_POLY1305 cryptographic functions. MQTT-SN ID can be used as ADD. The encrypted data and tag constitute the MQTT PUBLISH packet payload. After serialized, this packet is sent via MQTT-SN over BLE or XBEE. When receiving the serialized packet, the MQTT-SN Gateway via the Serial Port Bridge, deserialize it, then encapsulate it in UDP packet. This packet is relayed to the MQTT Broker which is responsible for sending it to all subscribed clients. For MQTT-SN client, the packet passes firstly to the MQTT-SN Gateway. After the packet is de-encapsulated and serialized, it finally relayed to the MQTT-SN client. The later decrypt the MQTT PUBLISH packet payload, then verify the authenticity of data. As regards the MQTT client, the procedure of exchanging data is the same, with the exception that the MQTT client communicates directly with the MQTT Broker without having to go through MQTT-SN Gateway. The proposed architecture ensures end-to-end encryption for the MQTT-SN and MQTT clients. Only the clients that have the secret key can decrypt the MQTT PUBLISH packet payload. Therefore, the MQTT-SN Gateway, MQTT Broker and other unauthorized entities cannot read the message content.

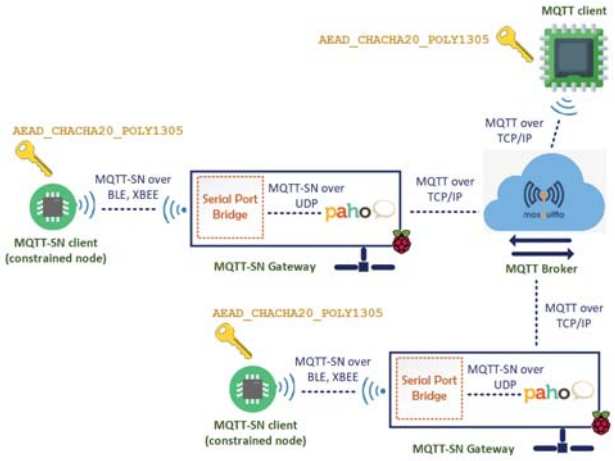
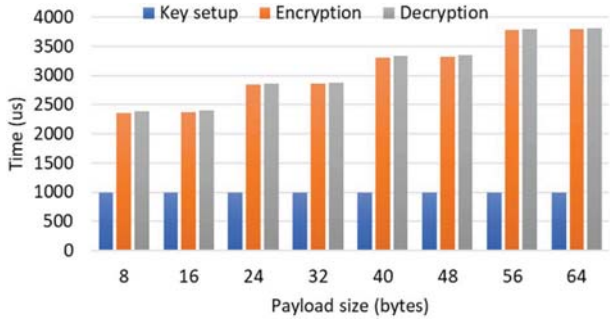


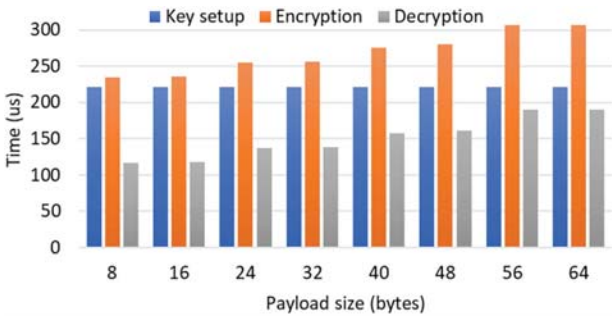
Fig. 2. Proposed secured MQTT/MQTT-SN architecture

B. Performance results

The performance results of the proposed architecture concern only the end devices, i.e. the MQTT-SN clients (constrained nodes) and MQTT clients (non-constrained nodes). Arduino Uno (AVR 8bits/16MHz, 32Ko Flash, 2Ko SRAM, UART interface) is used as a constrained node and ESP-8266 12E module (Xtensa 32bits/80MHz, 4Mo Flash, 50Ko SRAM, WiFi interface) as a non-constrained node. Fig. 3 and Fig. 4 show the performance results of the proposed security scheme. Based on the results of Fig. 3, the first observation is that cryptographic operations take much more time when the size of MQTT PUBLISH packet payload growing. However, key setup takes constant time because the key size is constant (256 bit) and remain unchanged during the whole performance test. For Arduino Uno, Fig. 3a, which is considerate as a constrained



a) Arduino Uno



b) ESP8266 - 12E

Fig. 3. Cryptographic operations duration with different payload size

[1] https://rweather.github.io/arduinolibs/crypto.html
[2] https://github.com/ESIBot/MQTT-SN-Arduino
[3] https://github.com/njh/mqtt-sn-tools
[4] https://github.com/eclipse/paho.mqtt-sn.embedded-c
[5] https://github.com/MichalFoksa/rsmb
[6] https://github.com/knolleary/pubsubclient

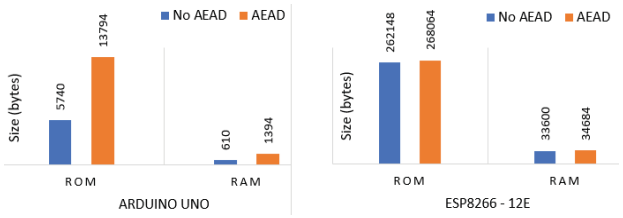


Fig. 4. Memory occupation in MQTT-SN and MQTT clients

node, the key setup takes less more time (988µs) compared to encryption and decryption procedures. It is interesting to note that encryption and decryption duration is almost the same and varies linearly by step of 16 bytes. This linearity is justified because during, the authentication tag computing, the input message M is chopped into $q = \lceil l/16 \rceil$ 17 -byte chunks $(m_i)_{0 \le i \le q}$ and cryptographic operations are applied on each block m_i. In addition, during the encryption and decryption procedure, the text P is encrypted by simply chopping it into 64-byte blocks $(p_i)_{0 \le i \le 2^n}$ and applying on each block p_i cryptographic operations. When the size of the payload is growing, the number of blocks is also growing, therefore the encryption and decryption procedure take more time. For ESP8266-12E Fig. 3b, which is not a constrained node, the cryptographic operations also take much more time when the payload size is growing. Linearity between computing time and payload size by step of 16 bytes is also noted. However, the key setup time remains constant (221µs) but is greater than the decryption time. Fig. 4 shows the memory occupation of MQTT-SN client (Arduino Uno) and MQTT client (ESP8266-12E) when AEAD is applied and not applied. RAM and ROM usage are both much higher when the chacha20-poly1305 AEAD is implemented on clients. The ROM and RAM size are increased respectively to 140.31% and 128.52% for Arduino Uno. For ESP8266-12E, the ROM and RAM size are increased respectively to 2.26% and 3.23%. The difference in memory usage on Arduino Uno and ESP-8266 is due to the fact that the MQTT-SN client library is much lightweight than MQTT client.

C. Evaluation

The performance results show that more latency will be added when securing MQTT Message data payload. However, this latency is rather low and does not affect the way the MQTT-SN client operates because MQTT-SN is designed to support high latency and low bandwidth (delay tolerant). Indeed, based on the "best practice" values for the timers and counters defined in the MQTT-SN specification [2], the recommended value of retry timer, T_{retry}, is fixed between $10 - 15s$. If T_{retry} time out and the expected GW's reply is not received, the client retransmits the message. Therefore, the delay (maximum 3812µs) caused by AEAD_CHACHA20_POLY1305 is insignificant to cause a timeout. The chacha20-poly1305 AEAD add a 16-byte authentication tag T on MQTT Message data payload. When, in ESIBot/MQTT-SN-Arduino library, the maximum message buffer size is fixed to 80 bytes, it remains 64 bytes to be used for data publishing. Most connected objects sent sensor data which use less than 64 bytes. In short, chacha20-poly1305 AEAD has a low impact on MQTT-SN Message overhead. The free memory space after implementing MQTT-SN client and chacha20-poly1305 AEAD on Arduino Uno is about 18462 bytes for ROM and 654 bytes for RAM. This available memory is sufficient to implement additional features. Based on Table 1, only Class 1 and Class 2 constrained nodes can support MQTT-SN with chacha20-poly1305 AEAD. For Class 0 constrained nodes, the chacha20-poly1305 AEAD memory occupation must be optimized.

V. Conclusion

In this paper, we proposed a lightweight security scheme for MQTT/MQTT-SN protocol based on chacha20-poly1305 AEAD. This scheme ensures end-to-end encryption for nodes. The implementation of MQTT-SN client and MQTT client use different existing Arduino libraries. The proposed architecture does not require IP connection for MQTT-SN client. In addition, performance evaluation shows that the proposed security scheme is suitable for constrained nodes in terms of CPU and memory usage. The proposed security scheme needs a best secret key management when the number of nodes increases. In future work, we propose to implement lightweight key exchange scheme.

VI. References

[1] OASIS MQTT Technical Committee, "MQTT Version 5.0," OASIS, Committee Specification 02, May 2018.
[2] A. Stanford-Clark and H. L. Truong, "MQTT For Sensor Networks (MQTT-SN)," Protocol Specification Version 1.2, Nov. 2014.
[3] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," IETF, RFC7539, May 2015.
[4] G. Procter, "A Security Analysis of the Composition of ChaCha20 and Poly1305," *IACR Cryptol. EPrint Arch.*, vol. 2014, p. 613, 2014.
[5] B. Jungk and S. Bhasin, "Don't fall into a trap: Physical side-channel analysis of ChaCha20-Poly1305," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1110–1115.
[6] H. Tschofenig and J. Arkko, "Report from the Smart Object Workshop," IETF, RFC RFC 6574, Apr. 2012.
[7] J. P. Vasseur, "Terms Used in Routing for Low-Power and Lossy Networks," IETF, RFC 7102, Jan. 2014.
[8] C. Bormann, M. Ersue, and A. Keränen, "Terminology for Constrained-Node Networks," IETF, RFC 7228, May 2014.
[9] S. Katsikeas *et al.*, "Lightweight secure industrial IoT communications via the MQ telemetry transport protocol," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, 2017, pp. 1193–1200.
[10] F. D. Santis, A. Schauer, and G. Sigl, "ChaCha20-Poly1305 authenticated encryption for high-speed embedded IoT applications," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 692–697.
[11] M. A. Prada, P. Reguera, S. Alonso, A. Morán, J. J. Fuertes, and M. Domínguez, "Communication with resource-constrained devices through MQTT for control education," *IFAC-Pap.*, vol. 49, no. 6, pp. 150–155, Jan. 2016.
[12] S. Ahmed, A. Topalov, and N. Shakev, "A robotized wireless sensor network based on MQTT cloud computing," in *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, 2017, pp. 1–6.
[13] P. Pittoli, P. David, and T. Noël, "Security Architectures in Constrained Environments: a Survey," *Ad Hoc Netw.*
[14] A. Niruntasukrat, C. Issariyapat, P. Pongpaibool, K. Meesublak, P. Aiumsupucgul, and A. Panya, "Authorization mechanism for MQTT-based Internet of Things," in *2016 IEEE International Conference on Communications Workshops (ICC)*, 2016, pp. 290–295.
[15] L. Bisne and M. Parmar, "Composite secure MQTT for Internet of Things using ABE and dynamic S-box AES," in *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, 2017, pp. 1–5.