Development and Deployment of TLS Communication Protocol in IoT Devices Using MQTT and CoAP Protocols

Mariam Gevorgyan
National Polytechnic University of Armenia
Yerevan, Armenia
e-mail: mariamgevorgyan.tt055-1@polytechnic.am

***Abstract*— In the rapidly evolving field of Internet of Things (IoT) technologies, secure data exchange has become a significant challenge for devices with limited resources. This paper examines the implementation of the Transport Layer Security (TLS) protocol for IoT devices utilizing the MQTT and CoAP communication standards. It presents a system designed to ensure data confidentiality, integrity, and authentication through the use of Datagram TLS (DTLS) 1.2, a specific handshake model, and Pre-Shared Key (PSK) authentication. The approach leverages the nRF9160 DK and Thingy devices, which are powered by Arm Cortex-M33 processors and equipped with integrated CryptoCell310 security subsystems. MQTT facilitates a publish/subscribe message exchange model through Mutual TLS, while CoAP enables RESTful requests secured by DTLS. Sensor data are encrypted and transmitted over Bluetooth Low Energy (BLE), with handshake validation conducted through traffic analysis using Wireshark and PSK decryption. This research highlights that lightweight TLS modules, configured with CoAP and MQTT protocols and incorporating ARM's TrustZone technology, can afford secure and efficient communication in constrained environments without sacrificing performance.**

***Keywords*—TLS, DTLS, MQTT, CoAP, nRF9160 DK, CryptoCell, Trustzone, CBOR, PSK, Wireshark.**

I. Introduction

With the widespread proliferation of Internet of Things (IoT) devices, secure communication and data protection have become primary concerns. This article explores the development of an adaptive and lightweight security methodology for implementing Transport Layer Security (TLS). We have chosen the Constrained Application Protocol (CoAP) and Message Queuing Telemetry Transport (MQTT) as foundational communication layers because of their suitability for low-power devices. Key components such as ARM TrustZone, the CryptoCell security subsystem, JSON Web Token (JWT) authentication, and Pre-Shared Key (PSK) handshake models have been integrated to ensure both flexibility and system efficiency.

II. ARM Architecture and CryptoCell Security Subsystem with ARM TrustZone

ARM (Advanced RISC Machine) [1] architecture is a family of computer processor architectures based on the RISC (Reduced Instruction Set Computing) model, developed by ARM Holdings. ARM processors are widely used in a variety of electronic devices, including smartphones, tablets, wearable devices, embedded systems, IoT devices, and servers.

ARM architecture integrates security technologies through a Trusted Execution Environment (TEE), enabling sensitive code execution in isolated and secure zones free from interference by other system components. The most commonly used TEE implementation is TrustZone, introduced by ARM in the ARMv8-M profile. TrustZone splits system resources into secure and non-secure domains, establishing a security boundary within a single device. This segmentation supports secure boot, trusted firmware updates, and root-of-trust installations—ensuring secure IoT applications without compromising performance.

Through the TrustZone approach, ARM supports the CryptoCell security subsystem, which serves as a hardware-based root of trust and facilitates cryptographic operations within the device. It is physically isolated and accessed via dedicated software APIs.

CryptoCell [2] resides in the secure domain defined by TEE in the ARM architecture. Developers can interact with it using specialized APIs for encryption tasks. By offering rich cryptographic and security resources, CryptoCell strengthens IoT application resilience against cyber threats and is specifically optimized for energy-constrained devices.

In IoT devices, the inclusion of isolated security subsystems such as CryptoCell provides:

https://doi.org/10.51408/csit2025_48

- **Efficient processor usage:** Delegating encryption to a dedicated subsystem reduces CPU load and streamlines overall system performance.
- **Reduced power consumption:** The subsystem remains mostly in standby mode and activates efficiently when needed.
- **Enhanced security:** Isolation strengthens protection against unauthorized software and physical tampering.
- **Improved performance:** Tasks run faster within the security subsystem due to its streamlined architecture and dedicated functionality.

CryptoCell also acts as a root of trust, supporting the following functional domains:

- **Control domain:** Manages subsystem access and information exchange.
- **Data interface domain:** Handles secure data storage and transmission.
- **Symmetric cryptography domain:** Performs AES or ChaCha20/Poly1305-based encryption/decryption.
- **Asymmetric cryptography domain:** Enables ECC or RSA encryption/decryption.
- **Security resource domain:** Delivers various protective mechanisms, transforming the system from a simple accelerator into a full-featured security processor.

The nRF9160 DK includes the CryptoCell 310 subsystem, supporting the following cryptographic features: [3]

- True Random Number Generation (TRNG)
- Pseudo-Random Number Generation (PRNG) using AES core
- RSA public-key cryptography
- ECC-based cryptography, including:
  - NIST FIPS 186-4 curves (using pseudo-random parameters)
  - SEC 2 curves (using pseudo-random parameters)
  - Koblitz curves (using fixed parameters)
  - Edwards/Montgomery curves
- ECDH/ECDSA support
- Secure Remote Password (SRP) protocol
- Hashing functions:
  - SHA-1
  - SHA-2 (up to 256-bit)
  - HMAC (Keyed-hash Message Authentication Code)
- Symmetric encryption:
  - AES
  - ChaCha20/Poly1305

III. Secure Communication Protocols

TLS protocol: TLS (Transport Layer Security)[4] is a transport layer protocol designed to secure communication over computer networks. It ensures the confidentiality, integrity, and authenticity of data exchanged between clients and servers.

To enable TLS on a website or application, an SSL/TLS certificate must be installed on the server. This certificate is issued by a Certificate Authority (CA) to the individual or organization that owns the domain. The certificate contains essential information about domain ownership as well as the server's public key, both of which are critical for verifying the server's identity.

During the TLS handshake, a cryptographic suite is established for each communication session. A cipher suite is a set of algorithms that defines the keys and encryption techniques to be used throughout the session.

TLS 1.2 does not inherently support Zero Round-Trip Time (0-RTT)[5], a feature introduced in TLS 1.3. This feature, also known as "early data," allows the client to start sending application-layer data—such as HTTP requests—without waiting for the handshake to complete, thereby reducing connection latency.

However, some implementations of TLS 1.2 may include extensions that mimic 0-RTT behavior. These typically rely on Pre-Shared Keys (PSKs) or session resumption mechanisms to achieve similar efficiency. The system we designed uses PSKs to implement the concept of 0-RTT communication.

Since IoT devices are resource-constrained, with limited memory, storage, processing power, battery capacity, and bandwidth, it is more effective to secure them using the CoAP protocol, which incorporates TLS through its datagram variant, DTLS.

CoAP protocol: CoAP (Constrained Application Protocol)[6] is a transport layer protocol based on UDP, specifically designed for communication between constrained devices. Its connectionless nature allows endpoints to communicate without prior negotiation, making it especially useful in systems that monitor status changes and exchange state information between clients and servers.

CoAP employs a request-response mechanism with a one-to-one architecture, functioning without any intermediary components. The protocol architecture of CoAP is illustrated in Figure 1.

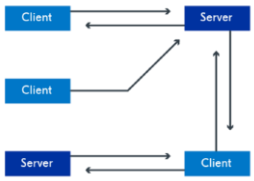

Figure 1. CoAP protocol architecture

CoAP minimizes network traffic by transmitting compact messages using CBOR (Concise Binary Object Representation) formatting rather than JSON.
Each CoAP endpoint supports CBOR to optimize traffic by reducing or compressing payload sizes. JSON is also supported, but using CBOR with CoAP can decrease data usage by up to 75%.

CBOR is a compact serialization format designed to represent structured data efficiently. It aims to achieve the same goals as JSON but employs a more succinct binary encoding.

All CoAP communication with nRF Cloud occurs over DTLS (Datagram Transport Layer Security) on port 5684 for server authentication. The server presents its X.509

certificate, while the device does not. The device verifies the server's certificate and then sends its JWT (JSON Web Token) to establish its identity. Once authenticated, the device and server can continue secure communication.

MQTT protocol: MQTT[6] is a lightweight publish/subscribe protocol for message exchange.

The structure of the MQTT network is illustrated in Figure 2.

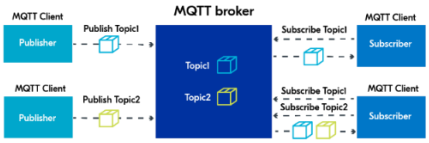

Figure 2. MQTT network structure

nRF Cloud is hosted on AWS and uses the AWS IoT Core MQTT broker to process messages sent and received through MQTT topics.

The MQTT publish/subscribe process works as follows:

- The device or modem-based MQTT client connects to the nRF Cloud MQTT endpoint.
- It publishes a JSON message to a specified topic.
- The MQTT broker receives the message. If a rule is defined and its criteria are met, the rule triggers an action—such as storing the message in a database or republishing a subset of its data to another topic.

All communication with the AWS IoT MQTT broker must use Mutual TLS on port 8883. This requires MQTT-enabled devices to possess an X.509 certificate and be registered with nRF Cloud.

MQTT (Message Queuing Telemetry Transport) is a widely used IoT protocol operating at the application layer over TCP, primarily suited for backend cloud platforms. It excels in event-driven systems with multiple nodes interacting, often based on sensor readings.

MQTT includes a keep-alive mechanism that maintains a persistent connection between the client and the broker—but this increases energy consumption. The client sets a keep-alive interval during connection initialization, which defines the maximum time they may remain connected without sending a message. The broker may enforce a maximum keep-alive duration (typically 60 seconds). If the client's interval exceeds this, the broker may terminate the connection.

MQTT transmits credentials in plain text and doesn't include built-in authentication or security mechanisms—but it does support encryption via TLS. Enabling TLS secures communication between the MQTT client and broker.

IV. Results

TLS implementation was performed on nRF9160 DK and Thingy devices powered by the ARM Cortex-M33 architecture, both of which integrate the CryptoCell310 security subsystem. This subsystem supports symmetric and asymmetric cryptographic operations, TRNG/PRNG mechanisms, and ECC and AES algorithm acceleration.

The TLS module is deployed using DTLS 1.2, suitable for UDP-based protocols such as CoAP and MQTT. The 0-RTT connection model enables bypassing traditional handshake phases using pre-shared keys (PSKs), significantly reducing connection latency.

Traffic encryption and decryption were validated via Wireshark, confirming successful DTLS handshake results, the cipher suites listed in the Client Hello packet, and the server's response certificates.

MQTT operated over Mutual TLS, establishing a secure channel from the device to the broker and the designated server at home.arevatech.am. CoAP utilized DTLS 1.2 with Connection ID (CID), enabling persistent and secure sessions—even when the device's IP address changed.

Environmental data from the Thingy device's built-in sensors was securely transferred to the host device via BLE, then forwarded to cloud systems using MQTT/CoAP APIs.

Within Wireshark, cipher suite negotiation and session key validation were carried out during the DTLS handshake using the supplied PSK.

The nRF9160DK Thingy includes sensors capable of collecting environmental parameters such as humidity, temperature, brightness, and more.

Temperature acquisition is shown in Figure 3.

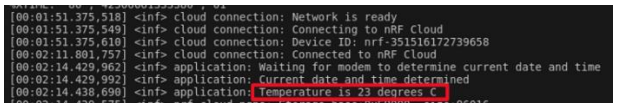

Figure 3. Temperature acquisition

To send data to home.arevatech.am, first assign a topic name to the topic field, followed by a specific message categorized under that topic. Once this is configured, clicking the Publish button will transmit the message to the designated host. home.arevatech.am acts as the server, functioning as an MQTT Broker for the MQTT protocol. The figure below illustrates that the message has been successfully delivered to the specified host.

Figure 4 shows the sent message.

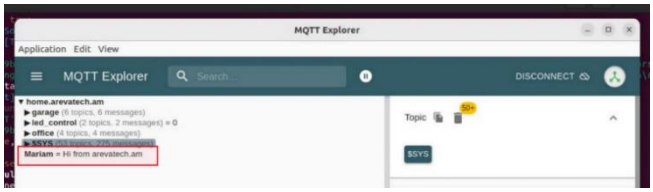

Figure 4. Sent message

The nRF9160 DK is configured to periodically check the incoming data and output it to the terminal. By pressing the 1 button on the keypad, the GET request result is obtained, which is shown in Figure 5.

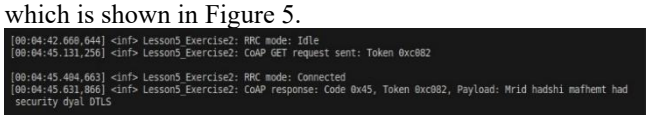

Figure 5. GET request and response

Using both buttons, we get the response to the PUT request, as shown in Figure 6.

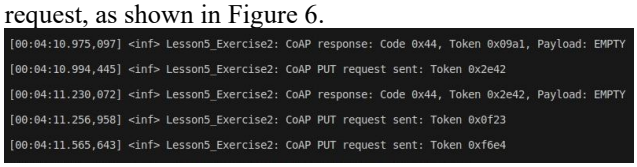

Figure 6. PUT request and response

To decrypt traffic, it is first necessary to capture trace data. The Cellular Monitor provides the capability to record a modem trace and save it as a raw file.

Using the ip keyword, we filter the traffic of a newly connecting device. In these packets, IP traffic can be divided into 3 parts:

- DNS lookup
- DTLS handshake
- Encrypted application data

Figure 7 shows traffic from a newly connected device.

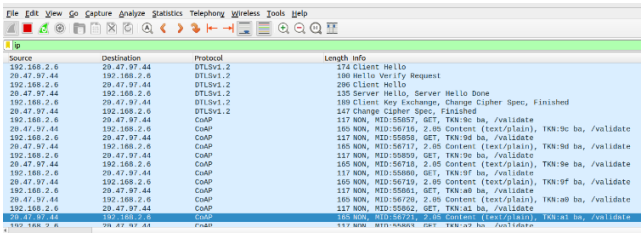

Figure 7. Traffic from a newly connected device

Examining these three scenarios is particularly helpful when the device fails to connect to the server. Often, the most insightful packet is the Client Hello. This is the first packet sent from the device to the server and contains numerous details about the device's supported capabilities.

In our case, we can see that the device uses PSK as the method for authentication and encryption, and intends to initiate a connection with the server at home.arevatech.am.

By clicking on the Client Hello packet, we can view the list of supported cipher suites, which is shown in Figure 8.

Figure 8. Cipher suites

In this protocol, we can see the response message sent by the server, as shown in Figure 9.

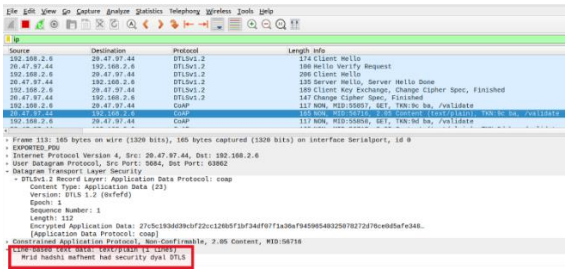

Figure 9. Decrypted message

Another way to decrypt DTLS is to use PSKs. The DTLS handshake ends with an "Encrypted Handshake Message," after which only encrypted application data is transmitted. However, by providing Wireshark with the PSK used in the DTLS connection, we can decrypt the traffic, which is shown in Figure 10.

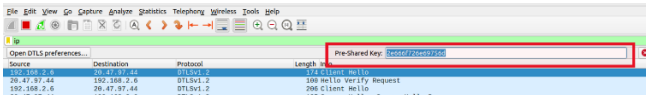

Figure 10. Traffic decryption with PSK

The DTLS handshake using the 0-RTT PSK method reduced connection establishment time by more than 25%, providing significant relief for low-power microcontrollers.

It's important to note that traffic decryption is only possible when PSK-based cipher suites are used. In cases where asymmetric key cipher suites are applied, a symmetric session key is generated and exchanged via the Diffie-Hellman key exchange protocol. Since the modem does not expose the generated symmetric key, decrypting the traffic becomes impossible.

V. Conclusion

Examining these three scenarios is particularly helpful when the device fails to connect to the server. Often, the most insightful packet is the Client Hello. This is the first packet sent from the device to the server and contains numerous details about the device's supported capabilities.

In our case, we can see that the device uses PSK as the method for authentication and encryption, and intends to initiate a connection with the server at home.arevatech.am.

The security challenges of IoT devices can be effectively addressed through the implementation of TLS/DTLS protocols—particularly using PSK and 0-RTT techniques. Deploying CryptoCell310 within an ARM TrustZone environment ensures a reliable platform for cryptographic functionality, supporting ECC, AES, SHA, and RNG.

Decryption tests conducted via Wireshark confirm that the PSK model enables not only a lightweight handshake process but also full traffic visibility and control.

MQTT and CoAP protocols, featuring Mutual TLS and DTLS support, enable efficient communication across various application-layer environments. Data exchange from device to cloud, secured through JWT authentication and CBOR formatting, helps optimize data flow and reduce power consumption.

References

[1] Preventing a $500 Attack Destroying Your IoT Devices, Axiado Corporation & Global Semiconductor Alliance, November 2021.

[2] A. Malashanka, "Arm TrustZone and CryptoCell: The Easy Way to Accelerate Cryptography," 1 October 2021. [Online]. Available: https://klika-tech.com/blog/2021/10/11/arm-trustzone-and-cryptocell-the-easy-way-to-accelerate-cryptography.

[3] J. Wallance, "Arm CryptoCell-312: Simplifying the design of secure IoT systems," Arm Community Blogs, 1 November 2016. [Online]. Available: https://community.arm.com/arm-community-blogs/b/embedded-and-microcontrollers-blog/posts/arm-trustzone-cryptocell-312-simplifying-the-design-of-secure-iot-systems.

[4] W. contributors, "Transport Layer Security," [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security

[5] K. G. T. J. Nimrod Aviram, "Session Resumption Protocols and Efficient Forward Security for TLS 1.3 0-RTT," Journal of Cryptology, 18 May 2021. [Online]. Available: https://link.springer.com/article/10.1007/s00145-021-09385-0.

[6] N. S. Academy, "Cellular IoT Fundamentals," [Online]. Available: https://academy.nordicsemi.com/courses/cellular-iot-fundamentals/.