CRYPTO QUANTIQUE

Enabling Cloud Connectivity in Resource-Constrained IoT Devices: A Wi-Fi Module and Low-End MCU Approach

Version 1.0

Table of Contents

1 Abstract 2
2 Introduction 2
3 Limitations of super loop architectures in embedded firmware development for connected devices 2
4 The TLS challenge for resource-constrained MCUs 3
5 How hard is it to move from a super loop to an RTOS architecture? 3
6 Beyond firmware: what are the other key considerations for connected embedded devices? 4
7 A new way of tackling the challenges: wireless modules with an integrated security platform 5
8 A deeper dive: how Cordelia-I works with QuarkLink to streamline secure connectivity 6
9 Summary 8

1 Abstract

As IoT devices become integral to industries like home automation, healthcare, and industrial monitoring, the need for cloud connectivity for microcontroller-based products grows. However, developing cloud-connected devices presents technical and business challenges, including power and processing constraints, security concerns, data management, and interoperability. This article explores these issues in detail and introduces an affordable and efficient approach: using a Wi-Fi module with a low-end microcontroller unit (MCU). Such a design allows developers to bring secure, scalable, and reliable cloud connectivity to constrained devices without excessive costs or complexity.

2 Introduction

The Internet of Things (IoT) revolution has expanded the role of embedded systems, particularly microcontrollers, into cloud-connected applications, enabling remote monitoring, real-time data analytics, and seamless user control. However, enabling secure cloud connectivity with limited resources poses numerous technical challenges.

Inexpensive microcontrollers (MCUs) with constrained processing power, memory, and storage are often used in cost-sensitive products that still need secure and reliable cloud connectivity. To meet these demands, pairing a low-power Wi-Fi module with a low-end MCU offers a viable path forward. So, what are the key challenges of cloud connectivity in constrained environments and how can we address these issues effectively?

3 Limitations of super loop architectures in embedded firmware development for connected devices

Low-end MCUs typically use a super loop architecture for their firmware. This is a single central infinite loop which co-ordinates the actions in the firmware without needing to involve an RTOS. The loop can handle tasks like reading inputs, process information (e.g. state machine), outputting results, and introducing delays until the next loop execution.

If tasks run with different time intervals, the super loop must take care which function should be called in which run of the loop. A super loop design can also work in an event-driven way through interrupts. If you have developed the application for your product using this method, moving to a cloud connected product is problematic. TLS is at the heart of the issue. Here's why.

4 The TLS challenge for resource-constrained MCUs

Communication with cloud services requires a TLS connection. However, many parts of a TLS stack, especially cryptographic operations, are computationally intensive and can block the CPU for a significant time. In a super loop, this could prevent the MCU from responding to other events, such as handling interrupts, network events, or sensor data.
TLS may also require precise timing for things like session timeouts, retransmissions, and retries. The super loop, being single-threaded, may introduce unpredictable delays, making it harder to meet real-time requirements. What's more, TLS requires non-blocking network communication to efficiently exchange data. In a super loop, handling network events might involve polling or using interrupts, which can be tricky to synchronize with the TLS state machine, especially when the network latency is variable.

Added to this, a low-cost MCU typically has small memory capacity, and TLS stacks are often memory-intensive due to the need to store encryption keys, certificates, and session data. The super loop must ensure efficient memory management and avoid fragmentation.

As a result of these complexities, a more advanced RTOS with proper multitasking and real-time capabilities may be a better approach for more complex systems or higher security requirements.

5 How hard is it to move from a super loop to an RTOS architecture?

Firstly, RTOS's are commonly shipped with a TLS stack or can be integrated easily with third party stacks. WolfSSL and mbedTLS are a couple of examples, so that's good news. But challenges remain.

For example, at the core of any RTOS, task scheduling must efficiently manage tasks, with real-time priorities. In low-cost MCUs, this means designing a scheduler that doesn't impose too much overhead. Efficient handling of hardware interrupts and context switching is critical, so interrupt service routines (ISRs) must be optimized for low latency, and memory allocation for stacks, heaps, and buffers must be optimized. Added to this, the RTOS must avoid fragmentation and be designed to work in environments with constrained memory. Synchronization primitive mechanisms like semaphores, mutexes, and queues are often implemented to ensure proper synchronization between tasks, though these should be lightweight to minimize overhead.

It's important to remember that time-related services such as timeouts, delays, and periodic task management are integral to the RTOS and that low-cost MCUs often operate in power-constrained environments, so managing power usage through sleep modes and low-power tasks is essential.

The implementation challenges are compounded by the inherent performance limitations of low-end MCUs. These typically have constrained RAM and flash storage, sometimes in the range of a few kilobytes to a few megabytes. This means that every optimization in the TLS stack and RTOS must prioritize low memory usage.

Many low-cost MCUs run at modest clock speeds of 8 MHz to 100 MHz, meaning that cryptographic operations and real-time scheduling need to be highly efficient.

Power is a key concern in embedded applications, so both the TLS stack and the RTOS must be designed to minimize power consumption, using techniques like efficient scheduling and low-power modes.

Last, but not least, implementing robust security mechanisms in the TLS stack, such as secure key storage, and avoiding vulnerabilities like side-channel attacks, requires significant attention.

Fortunately, there's now a plethora of tools and libraries available with popular RTOSs such as FreeRTOS (now Amazon FreeRTOS). Cryptographic libraries and TLS stacks can help in reducing the development time needed to integrate all this new functionality into an unfamiliar architecture.

By leveraging existing libraries and frameworks, development time can be shortened, but custom optimizations will likely still be necessary to achieve a secure, efficient solution on a constrained platform. The development effort could span several months to over a year, depending on the complexity of the system and the security requirements, not least because not all engineers have the required expertise in embedded systems, cryptography, and real-time systems.

6 Beyond firmware: what are the other key considerations for connected embedded devices?

Security and regulation are becoming ever-more critical because cloud-connected devices are vulnerable to cyber threats, including data breaches and unauthorized access. Security challenges are magnified in low-power devices, as robust encryption and authentication processes can be resource-intensive. Also, the new Cyber Resilience Act (CRA) means that products will have to comply with the new regulations if they are to be sold within Europe. Certification costs add a further burden.

For applications in consumer electronics, healthcare, and agriculture, cost-effectiveness is critical. Cloud connectivity can require significant investment in hardware, software (as we have seen above), cloud infrastructure, and ongoing maintenance. Ensuring that robust security does not come at an unacceptable cost premium is a growing issue for development teams.

Data management is an ongoing operational cost. Cloud-connected devices sometimes generate and transmit large amounts of data, depending on the application. Storage, processing, and transmitting this data in real-time can be challenging, especially in cost-sensitive applications where data usage directly impacts service costs.

All these issues are amplified when you consider that scaling an IoT network may involve reliably managing thousands of devices across diverse locations. Devices must remain resilient to network fluctuations, interference, and connectivity disruptions. Low-end devices may lack the resources to implement sophisticated recovery and reconnection mechanisms.

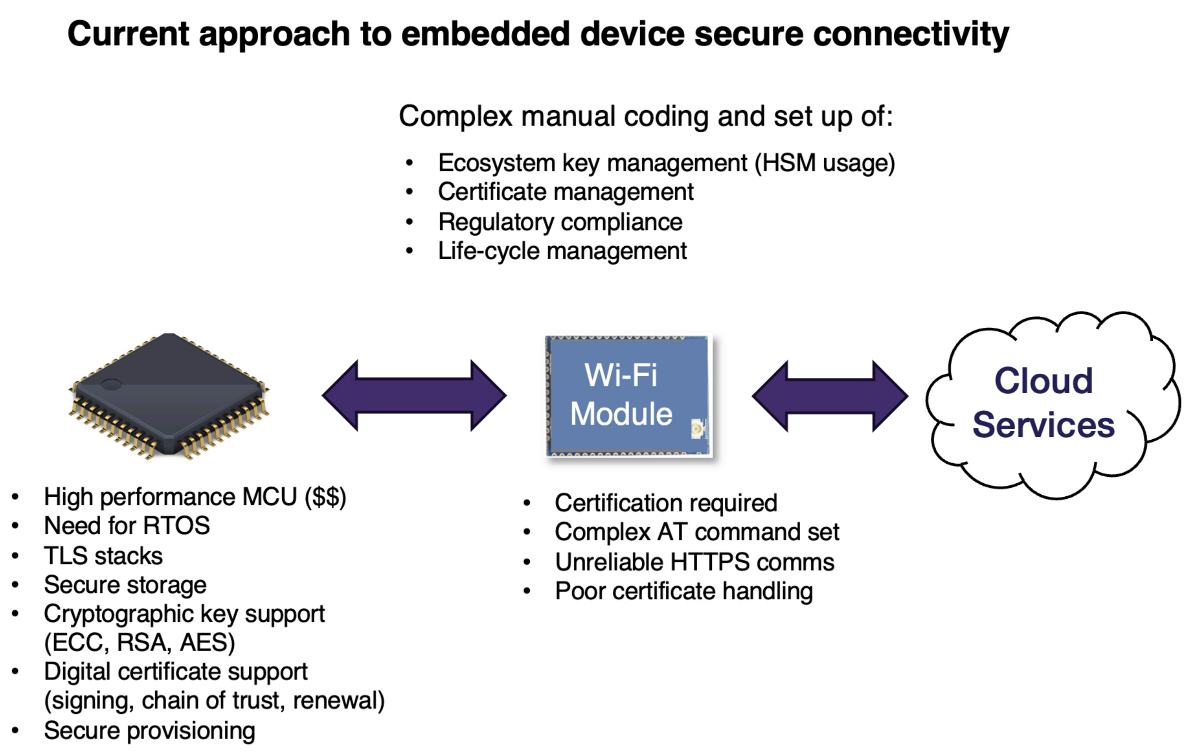

Current approach to embedded device secure connectivity
Complex manual coding and set up of:
• Ecosystem key management (HSM usage)
• Certificate management
• Regulatory compliance
• Life-cycle management
Wi-Fi Module
Cloud Services
• High performance MCU ($$)
• Need for RTOS
• TLS stacks
• Secure storage
• Cryptographic key support (ECC, RSA, AES)
• Digital certificate support (signing, chain of trust, renewal)
• Secure provisioning
• Certification required
• Complex AT command set
• Unreliable HTTPS comms
• Poor certificate handling

Figure 1: Embedded device development is made infinitely more complex when cloud connectivity is required

It's also worth bearing in mind that IoT devices must operate across diverse ecosystems, which often means complying with multiple communication protocols and standards. Without interoperability, devices may struggle to interact with cloud platforms or other IoT implementations, leading to functionality and deployment limitations.

7 A new way of tackling the challenges: wireless modules with an integrated security platform

Rather than trying to squeeze a lot more functionality from a resource-constrained MCU platform or having to resort to the more expensive and powerful MCU, much of the load can now be handled by wireless modules like the Cordelia-I unit from Würth Elektronik. The module comprises certified Wi-Fi hardware that is configured to work with a cloud-based device security platform from Crypto Quantique, called QuarkLink. The module connects to the embedded device via a simple UART connection. It houses the TCP/IP and HTTPs stacks, secure memory storage for cryptographic keys, certificates (including the QuarkLink Root Certificate), and the QuarkLink API.

The firmware challenges are addressed by the Cordelia-I module and QuarkLink delivers device security and management. It facilitates automatic generation and renewal of secret keys and digital certificates, and compliance with secure communication protocols. This helps users achieve and maintain compliance with regulations and enables management of the connected devices throughout their lifecycle.

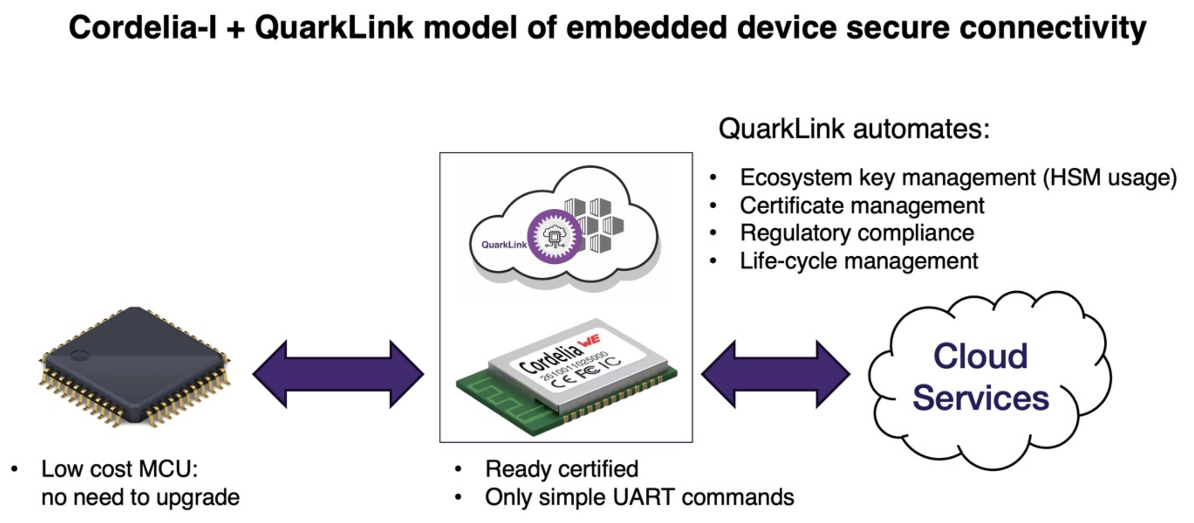

Cordelia-I + QuarkLink model of embedded device secure connectivity
QuarkLink automates:
• Ecosystem key management (HSM usage)
• Certificate management
• Regulatory compliance
• Life-cycle management
QuarkLink
Cordelia
Cloud Services
• Low cost MCU: no need to upgrade
• Ready certified
• Only simple UART commands

Figure 2: A more capable Wi-Fi module designed to work with a cloud-based device security platform enables developers to use familiar low-cost MCUs

8 A deeper dive: how Cordelia-I works with QuarkLink to streamline secure connectivity

The embedded firmware running Cordelia-I Wi-Fi module has minimized the AT Command set needed to connect to a cloud service. Once the Cordelia-I has been provisioned using QuarkLink, only two commands are needed to start communication with an MQTT broker, whether that's a QuarkLink local broker or one provided by a cloud service provider (e.g. Amazon AWS).

- **AT+iotenrol** (to initiate a secure connection to the assigned QuarkLink)
- **AT+iotconnect** (to initiate the connection to the MQTT broker)

All the secure credentials have been received by Cordelia-I from QuarkLink so communication through the cloud can begin.

- **AT+publish=1**, “Hello World!”

The subtopic that the message will be sent to is configured as part of the QuarkLink configuration described below. During provisioning, Cordelia-I is also configured to subscribe to topics. Any messages published by other entities to these specific topics and subtopics will be automatically received by the module.

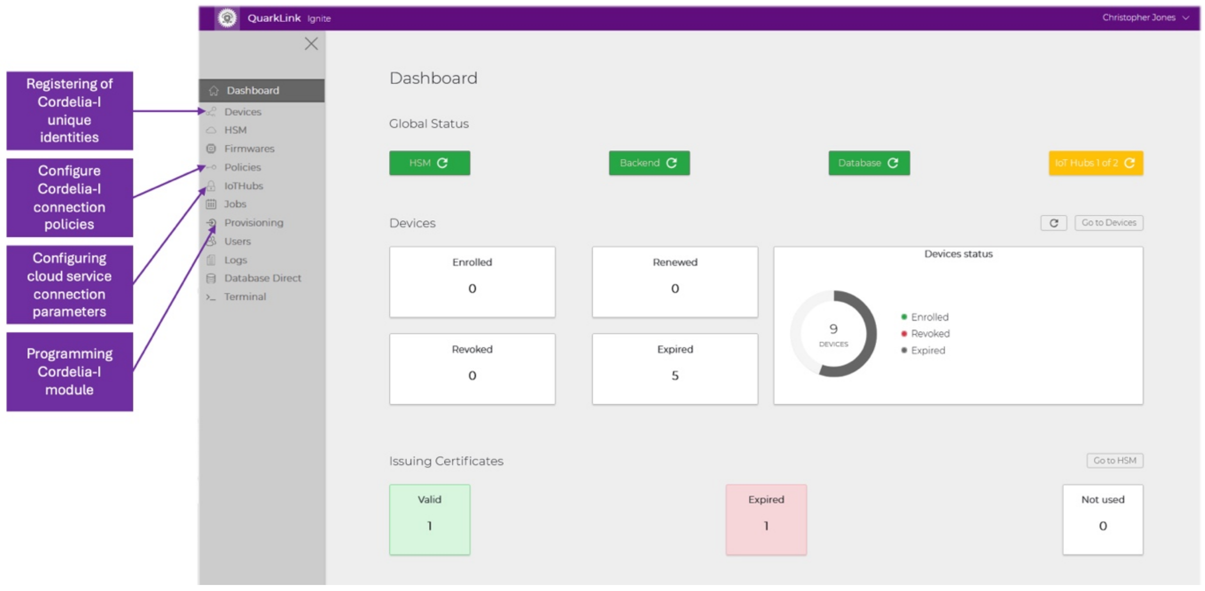
Remember that these commands are sent to Cordelia-I from the low-end microcontroller over a UART, making everything simple to implement. What's more, there are several other features that Cordelia-I offers to further reduce development effort and ensure security. For example, the module is fully compliant with The European Union's Radio Equipment Directive (RED) including the delegated regulation for cybersecurity 2022/30 as per EN18031-1 norm. This makes it a secure basis upon which to build any embedded IoT application.

Cordelia-I supports WEP, WPA/WPA2PSK, WPA2 Enterprise (802.1x) and WPA3, and Würth has implemented a Firmware Over-The-Air (FOTA) update service, as required by the European Cyber Resilience Act (CRA), removing any barriers to regulation compliance.

The main things left to resolve are now the cryptographic keys and certificates, and the lifetime management of these with respect to the embedded device. That's where QuarkLink comes in.

The QuarkLink security platform resides in the cloud. However, it has the capability to communicate directly with hardware through a Chromium based browser (e.g. Chrome, MS Edge etc). This is the key to preparing the Cordelia-I module to communicate with QuarkLink. Using a Provisioning Task created in QuarkLink, Cordelia-I is directly connected and programmed with the QuarkLink credentials (Root Certificate and URL) needed by the module to connect to the cloud. Cryptographic keys are generated securely within Cordelia-I during provisioning. They are then linked to QuarkLink to enable TLS communication whenever Cordelia-I needs to communicate with QuarkLink.

Provisioning tasks are set up via a simple GUI interface. No expertise is needed in the use of cryptographic keys or certificates. This is all handled securely and automatically by QuarkLink.



QuarkLink's simple GUI requires no cryptographic expertise for secure management of IoT device networks

Once the provisioning task is complete, the Cordelia-I module has the following:

- A unique identity (DeviceID)

- A securely stored private key that only Cordelia-I has access to internally to prove its identity (this key is not accessible by any firmware)
- The Root Certificate of its assigned QuarkLink
- The URL of its assigned QuarkLink
- The secure credentials to enable it to create TLS connections to either QuarkLink or any other cloud service provider

- The security credentials of the QuarkLink MQTT Broker (default configuration)
- A reduced AT command set to allow simple control from a low-end microcontroller
- Wi-Fi credentials to access the network (these can be changed later through simple AT commands)

With the Cordelia-I connected to a microcontroller via a UART, connection to the cloud IoT Hub or MQTT broker is simple. The Cordelia-I is now linked to a Public Key Infrastructure (PKI) that is automatically managed by QuarkLink. This offloading of complex tasks, carried out by QuarkLink and Cordelia-I, allows the MCU to focus on core tasks, allowing developers to achieve robust performance with minimal MCU investment.

9 Summary

As IoT adoption grows, the need for reliable, secure, and cost-effective cloud connectivity for constrained devices will only increase. Using a low-end MCU paired with a Wi-Fi module provides a practical and scalable way to address security, cost, power efficiency, and interoperability challenges. This configuration delivers reliable cloud connectivity, empowering businesses to deploy innovative connected devices in a cost-effective and efficient manner.