Electronic Design

By KIMBERLY DINSMORE, Security Solutions Manager,
Renesas Electronics, www.renesas.com/iot-security

Top Strategies to Improve Microcontroller Security (Part 1)

Microcontrollers are small and inexpensive. Security solutions have the reputation for being neither. With increasingly strict regulation and legislation, the need to balance the two is critical.

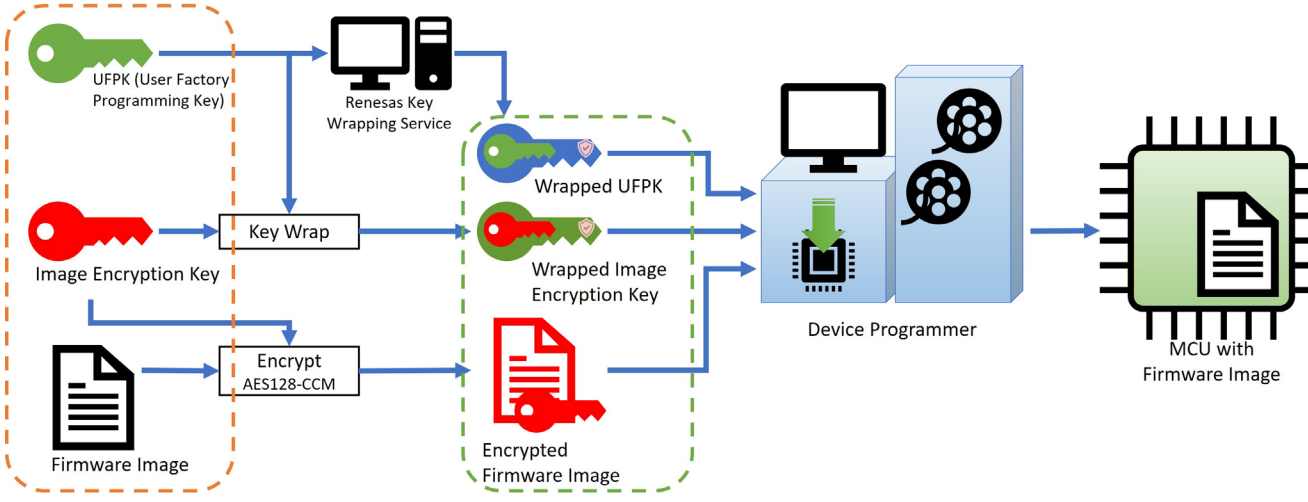
Microcontrollers (MCUs) have come a long way since their early days. With 4-bit data and all assembly language instructions verbosely described in a handful of pages in the device's datasheet, the most advanced peripheral was an analog-to-digital converter (ADC), and communication had to be bit-banged over I/O pins.

Today's 32-bit MCUs with gigabytes of internal program memory storage not only support a wide variety of communication interfaces, but they're also expected to use those interfaces to provide high-end solutions such as artificial intelligence and machine learning.

Security was not much of a concern for 4-bit MCUs. Typically, they performed a minor task in a larger system. And with code in masked ROM or PROM, it wasn't possible to alter that code. Now, however, the MCU is an integral part of the system, often performing so many tasks that an operating system is required, blurring the line between a microcontroller and a microprocessor.

However, as the saying goes, "with great power comes great responsibility." It's now the responsibility of the microcontroller to provide security for the overall product.

Strong security requires a strong foundation. Before the MCU can secure the entire system, it must secure itself. While it's vital to determine the relevant threat model and security policy for a specific product, some security solutions are very broadly applicable. Let's see what questions you can ask to help determine which solutions might be



Secure factory programming guarantees genuine silicon and provides IP protection.

suitable for your application. Like any good firmware implementation, let's try to look at these bottom-up.

In the Beginning: Secure Factory Programming

Long before a product is used by an end customer, it's vulnerable to attack. Unfortunately, these threats aren't usually considered by the engineers designing the product. That's partially due to their lack of exposure to the production process, but also because they're more focused on the final application, often with the added pressure of being required to produce an early prototype.

Some questions to ask include:

- How do you know that the specific MCU you specified (and paid for!) is actually being installed on your circuit board?
- How is your binary code being delivered to the programming house? Is it encrypted or in plaintext? If it's in plaintext, someone can easily copy and/or reverse-engineer it.
- If the binary is encrypted, at what point in the programming process is it decrypted? If it's decrypted outside the MCU, the plaintext can be sniffed by attaching wires to the programming pins.

Secure factory programming solutions can protect against these threats. As shown in the *figure*, the strongest solutions are those where the binary is encrypted using a method that's supported only by a genuine MCU. In these cases, the binary is decrypted on the MCU itself, such that only encrypted data is transferred on the programming pins.

Many of these solutions require the use of MCUs that have been pre-programmed with a special bootloader, but some MCUs now support this capability in blank chips. The beauty of the latter is that it doesn't require additional hardware or complicated infrastructure to transfer decryption keys, doesn't need special programming equipment, and doesn't require pre-programmed MCUs.

For MCUs that don't have this feature built into their silicon, secure factory programming solutions are available from experienced programming houses. These companies will offer assurances (certifications or other) that your data will be handled in a secure manner, and they will work with you to provide the necessary hardware, software, and logistics.

Simple Things First: Disabling External Interfaces

One security item that can be easily overlooked is disabling the external debugging and programming interfaces. For simple devices that will never be updated or debugged after deployment, this is usually as simple as flipping a bit or burning a fuse.

However, with MCUs becoming more powerful, updating firmware and re-enabling debug access are becoming essential requirements. The flexibility needed to implement these requirements tends to complicate the process for disabling these external interfaces, often requiring extra steps or even extra MCU pin connections, so be sure to study this aspect of the MCU before designing your PCB.

If unauthorized people gain access to these interfaces, they may be able to perform a variety of possible attacks:

- Extract all of the device's code and data, to reverse-engineer or clone your product.
- Extract or modify sensitive information, such as cryptographic keys and product operating parameters.
- Completely reprogram the device.

Some questions to ask include:

- How will firmware in the end-product be updated? Will the updater have physical access to the device and use standard programming tools? If yes, it's important that only authorized people be able to perform this update.
- Will the product need to be debugged after it has been released to the end-customer? Again, it's important that only authorized people re-enable this feature.
- Are any special external connections required to support the Device Lifecycle Management features of the chip?

Part 2 will delve into isolation mechanisms, secure and first-stage bootloaders, secure key storage, and decryption on-the-fly.

Kimberly Dinsmore is the Security Solutions Manager for Arm-core microcontrollers at Renesas Electronics.