White Paper

Security for the Connected World

Kimberly Dinsmore, Sr. Engineer, IoT Infrastructure Business Unit, Renesas Electronics Corp.

September 2019

Abstract

“Security” is a word that never fails to elicit a response, but what it means varies greatly depending on context, even when narrowed down to the world of electronic devices. For the consumer, “security” usually means that personal data is not available to anyone other than the specific intended recipients, but for products powered by a microcontroller, there often isn’t much end-customer data to protect. Let’s look at some other definitions of the word “security.” For software developers, “security” might mean that no one can steal their code. For OEMs, “security” could mean that no one can create clones of their device. For service providers that offer a service via an electronic device, “security” often means that no one can use their service without proper authorization or payment. For governments, “security” can mean that the device cannot be infiltrated and used as a weapon as part of a DDoS attack. All of these definitions definitely apply to microcontrollers and products based on them, regardless of market segment.

As soon as you enter the security realm, you are inundated by new terms and acronyms. The sheer quantity of jargon is mind-numbing. Let’s take it one step at a time, while avoiding the difficult vocabulary. How do you go about incorporating security into your MCU-based design?

Step 1: Commit to designing in security from the beginning.

When starting a new design, upper management wants to see progress, and to them that means a working prototype, but prototypes don’t have to be secure, and security doesn’t make a flashy demo to impress management and investors. As enticing as it is to defer security until the end of the design, do not succumb to the temptation. Yes, it will take some time up-front, and often require some management re-education, but it has been proven many times that security cannot be retrofitted into a design. Security is not an add-on; it is fundamental to the foundation of the architecture. Trying to add it later almost invariably results in a complete redesign – a data stream that was generated and transmitted byte-by-byte cannot be converted to an encrypted data block, hard-coded plaintext private keys can’t magically become securely-stored device-unique keys, and so on.

Step 2: Does your product fall under industry or government regulation?

This requirement overrules any other thoughts about security. If your industry requires a specific suite of cryptographic functions, you must use that exact suite, even if there are alternatives that offer the same level of protection. Financial transactions and energy meters are just two examples where specific

regulations exist and must be adhered to. Make sure you research and understand any regulatory requirements for your product, being sure to include general government regulations for the areas where your product will be used.

Step 3: What do you want to protect?

Most companies determine this list by looking at cost, which is why GDPR fines are so hefty. Governments realize that without a financial incentive, companies are more likely to manage security breaches than proactively try to prevent them. The first pass of creating this list usually focuses on the device itself and data: firmware, keys, and customer information, but look beyond the device and consider the system as a whole. You may not consider the data to be sensitive, but can someone exploit your device during normal operation to have a detriment affect, like a DDoS attack? If your device is responsible for a critical function, you need to protect that function against anything that can improperly modify it. If keys are used to enable a service, the service itself is what needs to be protected.

Step 4: What are your vulnerabilities?

Again, look at both the device and its deployed infrastructure. When considering the device, the obvious technical vulnerability is internet connectivity. This is often discounted for MCU-based products, since non-Linux IP connections aren't generally a target of attackers and the value of the transmitted data is debatable, but if you perform firmware updates over the internet and you want to protect your code, then the IP connection becomes a vulnerability. Even if your device is not IP connected, consider all connections to the outside world. When you look at the product's operating environment here, don't forget to consider the human element as well. It is a sad reality that cutting-edge security can sometimes be entirely circumvented by a well-placed bribe and you might want to consider what a malicious, or simply mischievous, actor could do.

Step 5: Whom do you trust?

The adage is, "trust no one," but there is no getting around the fact that security solutions will add cost, so do not waste money protecting your product from a non-existent threat. If you have on-site manufacturing, it probably isn't necessary to invest in a secure programming solution; however, if your device programming is done off-site, or if you need to program the device with keys or other sensitive data, a secure programming solution might be required.

Step 6: Define your limitations.

Given enough time and resources, anything can be broken. You must decide the scope of your required protection. Protecting against unwanted debugger access is one thing; protecting against someone decapping the MCU and analyzing the chip with an electron microscope is something else. Sometimes these limitations are defined by your product's regulatory authority, but often they are simply common sense. For example, if your main goal is to protect your firmware IP, you don't need an MCU that has immunity to side-channel analysis.

Step 7: Make a plan.

Decide how you will protect your assets from the vulnerabilities, leveraging the elements you trust, within the limits that you set. Sometimes referred to as a threat model, threat analysis, security assessment, or security policy, this effort is time well spent to ensure that you are properly focusing your efforts and your budget. This final step is also important in case anything does go wrong. If you can prove that you performed due diligence to assess the security needs of your product, it can help you argue against claims of negligence on your part.

The combination of threats, vulnerabilities, and trusted parties is as varied as the number of embedded devices, but luckily, there are some common themes and solution overlap.

Securing Data at Rest

A fundamental requirement for device security is to be able to store data securely on the device, but as with all things in the security realm, secure storage has different aspects. If your device has no external connectivity, an MCU is quite simple to secure by disabling or protecting all debugger and programmer access. If you need to ensure that the device cannot inadvertently corrupt itself by reprogramming its flash, many MCUs have the capability to designate part, or all of the flash, as OTP, preventing it from being erased and/or reprogrammed even via self-programming. If, however, your device has external connectivity, you might want to consider logically separating your code and data into “trusted” and “non-trusted” categories, and limiting access of the “trusted” data to only the “trusted” code. Optimally, this isolation should be hardware-enforced via a mechanism such as a Memory Protection Unit (MPU) or Arm® TrustZone®. While this isolation is not complete security, it does serve to reduce the attack surface for the “trusted” area.

Device Identity

If your product is going to be connected to an infrastructure, you are going to need some way of uniquely identifying it. There are a variety of ways to give each device a unique identity. Some MCUs already have a unique identity built-in, but these tend to be simple serialization, which then needs to be mapped so the centralized command center knows which device is deployed where. While this can be useful, a cryptographic unique identity is even more useful, as it enables additional security solutions, like securing data in flight.

A cryptographic identity leverages the properties of various encryption schemes, with the device’s identity being the encryption key. There are two basic options:

- Symmetric encryption, where the same key both encrypts and decrypts the data. The command center needs to know the symmetric key for each device.

- Asymmetric encryption, where two keys are required – one to encrypt the data, and one to decrypt the data. The functionality is interchangeable, enabling the device to keep one key private.

The tricky bit is how to get the keys on the device in the first place, such that the command center either knows the key, or knows that the key can be trusted. This is called provisioning. If your security

assessment concludes that your product can be installed and provisioned by a trusted technician, then your solution might consist of injecting or even generating keys onsite. If, however, your product is intended to be installed by a general consumer, you will need to provision your devices securely. This can be done during secure programming.

Securing Data in Flight

There are five goals for securing data in flight: confidentiality, data integrity, data origin, entity authentication, and non-repudiation. The scope of that discussion is beyond this overview. The primary consideration here is, what is the communication infrastructure? If your device is communicating over a proprietary bus within a closed infrastructure, you will have very different solutions for meeting these requirements than if your device is connected to the internet over a Wi-Fi connection. The latter is a worst-case scenario, but it is also the most common use case for IoT devices. The fundamental building block to securing data in flight over an IP connection is a cryptographic identity. While it may seem that a cryptographic identity might be a bit overkill, it becomes a requirement if your product is IP connected.

Secure Programming

Like most security solutions, secure programming can solve multiple problems, and there are multiple options available. A secure deployment solution can solve the problems of IP theft, cloning, and overproduction by enabling you to deliver your firmware image in an encrypted format, which can be decrypted only within a specific device programmer, and to receive an audit report of exactly how many devices were programmed. There are also secure programming solutions where device-unique keys can be injected into the MCU. There are even advanced solutions where the MCU itself can generate an asymmetric key pair, export the public key, and receive and store a signed certificate including that public key, generating both an audit report and a group of approved certificates that can be used for authenticating the final product. This aspect, in particular, requires strong support of the MCU by ecosystem partners.

“Security” is not a “one size fits all” proposition. It’s vital to determine your specific requirements for your product at the beginning of the development phase, and select an MCU with the necessary hardware capabilities, software support, and solution demonstration, supported by a committed silicon vendor with a strong partner network. Security may seem daunting, but choosing the right MCU with solid ecosystem support will enable you to create a secure product for today’s connected world.

Renesas offers multiple MCUs[1] that address the concerns discussed in this white paper. Please visit renesas.com to learn more.

References

[1] RA Family of 32-bit Arm Cortex-M MCUs
RX Family of 32-bit MCUs
Synergy Platform of 32-bit Arm Cortex-M MCUs + qualified software