Creating a defence

Embedded systems security is a growing threat; attacks can happen anywhere and we should all be concerned. By **Christophe Tremlet**.

An embedded system can be attacked by injecting malware at some point. Once installed, this can collect confidential data, change a system's behaviour or induce unpredictable actions.

These threats can be combatted using a properly secured boot process that allows only trusted software to run. To sustain a high level of trust, secure boot must rely on proven cryptographic algorithms. While this makes sense, a secure boot has its own challenges:

- The most appropriate algorithms are asymmetric, which requires intensive computing power
- The keys associated with these algorithms must be protected and their integrity retained
- The implementation must be flawless.

Authentication

To ensure the target runs only authorised software, firmware needs to be authenticated. This process – a digital signature – verifies that a piece of software is genuine and approved.

The software loaded during manufacturing must be signed digitally and this process should apply to each firmware update. A digital signature enables trust during the device's lifetime.

A strong digital signature must be computed by a public and well proven cryptographic algorithm. If system firmware is authenticated using an elliptic curve digital signature algorithm (ECDSA) and RSA, both combined with SHA, users can have a high level of trust.

Asymmetric cryptography

The fundamental principle of asymmetric cryptography is the software developer holds the private key, used for signing, while the embedded device stores the public key for verification. The importance of this cannot be overstated. The major advantage is that the confidential element – the private key – is never stored in the end product. Hence, if an ECDSA or RSA algorithm is used, an attacker cannot retrieve the private key, even with the most sophisticated invasive methods; all they can get is the public key and, by definition, it is impossible to retrieve the private key, even when the public key is known.

Fig 1 shows the process flow of a secure boot based on asymmetric cryptography. The ECDSA and RSA algorithms are supported by the SHA-256 hash algorithm, which provides the highest level of secure authentication.

Will the attack be professional or amateurish? What will be the financial impact of a successful attack? Who might be hurt and how badly?

Why do we also need SHA-256? For performance reasons, it would be impractical to sign the full firmware digitally, so SHA-256 is used to compute a unique digest (a 'hash' value) which cannot be forged. This digest is then signed through ECDSA or RSA. These same processes are applicable to firmware updates. For those updates, software is downloaded, rather than programmed during manufacture, but the digital signature generation and verification processes remain the same.

Fig 1: The secure boot process

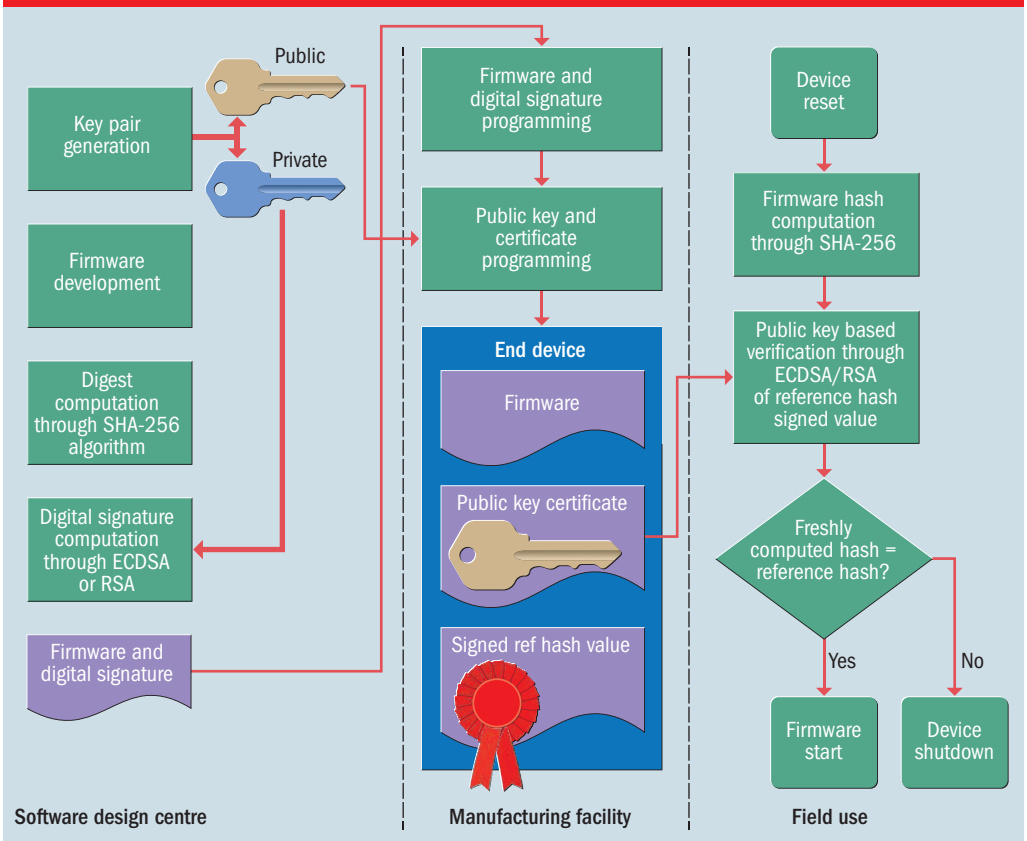
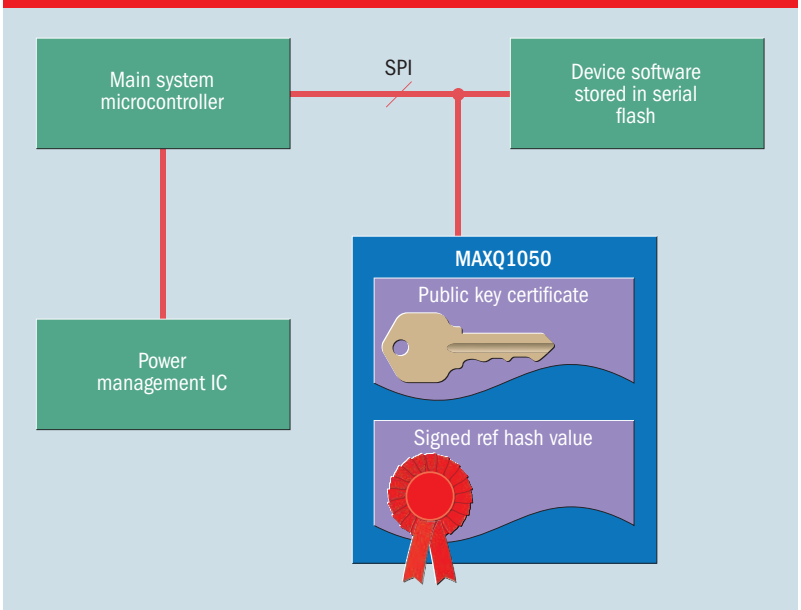


Fig 2: Implementing secure boot



Implementation challenges

While asymmetric cryptography offers essential benefits, it has a resource cost. Computing an SHA algorithm on a large piece of software is time consuming when done through software. RSA or ECDSA signature verification also requires overhead, especially if the main system MCU does not have a multiplier.

Another challenge is ensuring the integrity of the public key and its certificate. While a public key does not need to be kept confidential; it can be disclosed because it only allows verification, an attacker might want to substitute a different public key. If successful, the device would authorise fraudulent software signed by the attacker's private key. To avoid this, we must ensure the public key cannot be modified or replaced.

Many systems running on medium range MCUs cannot implement these basic requirements easily. Rather than changing the main system MCU, which might require a full redesign, a secure MCU can be added to implement a secure boot efficiently, handle the power and performance criteria and protect the public key, while guaranteeing a high level of security.

With an integrated secure hash engine, the MAXQ1050 is one such secure MCU that has the power to accelerate the computation of the firmware hash. This is important as hash time impacts the system's boot time directly. Because it has a modulo arithmetic accelerator, the MAXQ1050 can also perform fast ECDSA or RSA signature verification.

In the field use phase of a secure boot implementation (see fig 1), the MAXQ1050 will execute all steps and then inform the main system MCU and/or the power management IC (PMIC) of the authentication status. It can also provide application flexibility: for example, one of the MAXQ1050's GPIOs can enable the PMIC to power the main system MCU or one of its GPIOs could be connected to the main MCU's reset pin so reset happens only when firmware is verified. Optionally, the MCU's startup could be initiated by a specific sequence on the MAXQ1050's GPIOs.

The right security

It is often difficult to define the necessary level of system security. The highest possible level often results in high development and manufacturing costs. Hence, designers and users seek a trade off between cost and security.

Many issues affect these decisions. Will the attack be professional or amateurish? What will be the financial impact of a successful attack? Who might be hurt and how badly?

There are three potential levels of attack.

- **Basic**. The system is attacked using software. The attacker is unable, or not tooled to perform any physical attack or to modify any physical characteristic of the system.

Here, using MAXQ1050 as shown in fig 2 provides sufficient protection.

- **Moderate**. In addition to software, the hardware can be attacked by probing a PCB track to read a signal, forcing the level of a digital pin and/or removing an IC from the board.

While this level of threat is more complex, the standard MAXQ1050 implementation in fig 2 protects against some physical attacks, such as an attempt to replace the serial flash with a fake. The secure boot sequence using SHA-256 and ECDSA or RSA would detect any fake software.

To increase resistance to hardware attacks without increasing costs, additional layout precautions are recommended, including: routing the tracks connecting the MAXQ1050 to the PMIC or main system MCU in the PCB's inner layers; using pulses or sequences of pulses to indicate successful boot; and using at least two pins bearing different dynamic signals to inform the main controller that boot is successful.

- **High**. Here invasive attacks, such as microprobing signals on the bonding wires of an IC, are used. Protecting against these sophisticated attacks requires an implementation that is compliant with FIPS 140-2 level 3 or 4.

Such implementations detect any physical tamper attempt and react immediately by destroying sensitive information and rendering the system inoperable. Since restoring device operation would require maintenance, this level of protection should be implemented only when security overrides availability.

Because the MAXQ1050 incorporates self destruct inputs and instantly erasable NVSRAM, it can support these requirements.

"An attacker might want to substitute a different public key. If successful, the device would authorise fraudulent software. To avoid this, we must ensure the public key cannot be modified or replaced."
Christophe Tremlet

Christophe Tremlet is a marketing manager with Maxim Integrated.