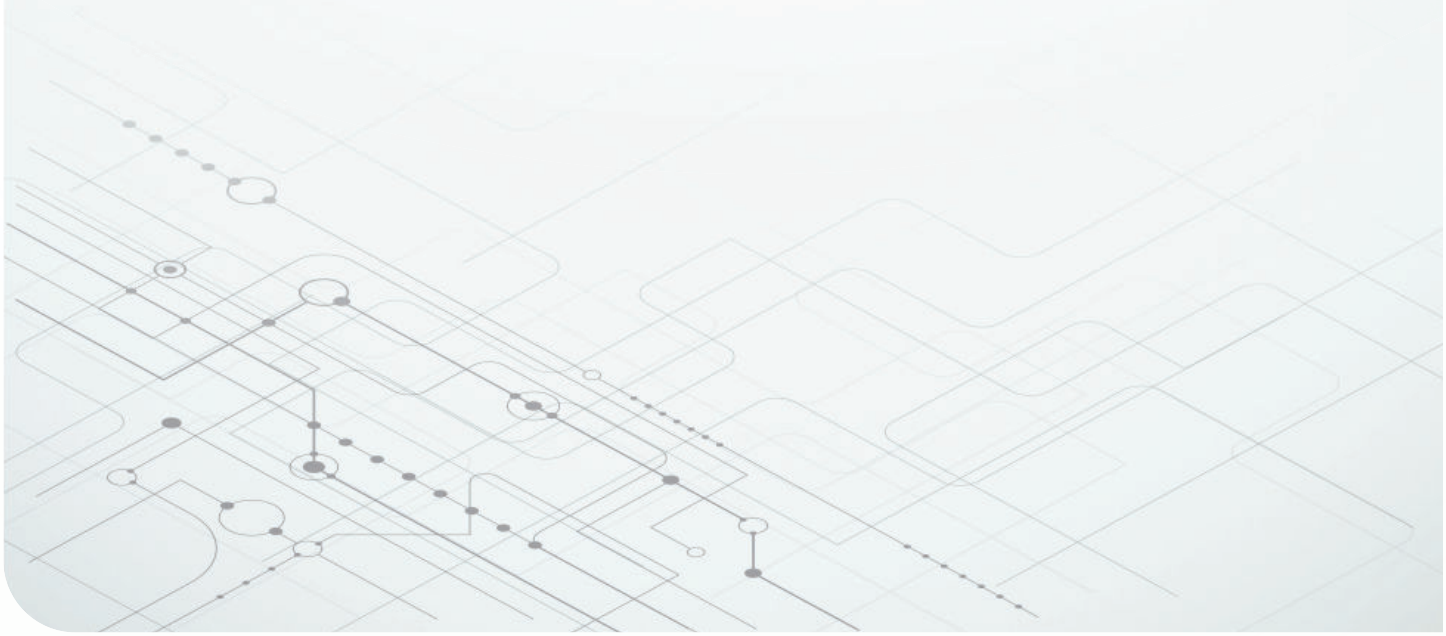
Securing Arm-Based Application Processors

Amrit Mundra
Security Architect and Systems Engineer



Introduction

Computer security once meant annoying viruses on PCs. Then, the stakes increased. Hacking into business and government systems exposed personal and financial information to fraud, theft and embezzlement. Now though, the security of embedded systems—or, more accurately, the insecurity of embedded systems—poses a threat to very critical data.

Today, the world runs on data and every bit or byte should be considered a potential target of attack. At the same time, both software and hardware systems are becoming much more complex, connected and interdependent. And with complexity comes vulnerabilities. The billions or trillions of lines of code and the interrelated hardware modules, subsystems and partitions all crammed on tiny slices of silicon are a hacker's delight.

Of course, hackers are not standing still. Reports of vulnerabilities in embedded systems go on and on: satellite communication systems, wireless base stations, laser printers in residences and businesses, the smart electrical grid, medical devices like defibrillators and many other systems are at risk. There has only been an increased need for security in multicore embedded systems-on-chips (SoCs) as the years have passed. Embedded devices like heart equipment, smartphones and automotive control units rely on multiple components including embedded SoCs to protect the control center.

First, lets introduce these elements that must be present to help secure an Arm®-based application processor with multiple cores in an embedded system. Second, the foundational layer of security for these processors, secure boot, is examined in greater detail because with secure boot the system is protected from "power on." Without secure boot the system has a gap from "power on" to usage. With the ever-changing nature of threats, security will always be a moving target.

Protecting a system from hackers, those that would like to steal data or take over a system to use it differently than it was intended, is the goal of the security aspects of the system. This is different than the related concept of functional safety. Safety is more focused on making sure the system responds to a wide variety of situations in an organized fashion, failing gracefully if needed. The combination of these concepts implies the system will operate as intended out in the real world where things break and bad actors exist.

Risk management

Security threats are always present and, with the rapid proliferation of the Internet of Things (IoT), those threats can come from anywhere, even inconspicuous and low-cost end-node devices. The basic security question is not whether a system will be attacked, but rather, when it will be. This leads to the conclusion that security is just as much about risk management as it is protection.

Given that the system may come under attack, how can system designers reduce the risk of a security breach to the absolute lowest level?

What to protect?

Anything of value could be subject to attack. And, of course, depending on the perspective and intent of the hacker, just about everything could be perceived as valuable. At the crudest level, the mere thrill of breaking into a system has value for a large portion of the hacker community. Most hackers are not innocuous thrill seekers. Many hackers would not hesitate to dip into an electronic wallet or steal financial information like credit card and bank account numbers for fraudulent use. IP can be stolen for sale or competitive advantage, while government secrets could be misappropriated and applied to disrupt, damage or destroy transportation systems, water suppliers, energy distribution networks, nuclear power plants and other aspects of a country's public infrastructure.

Of course, all of these valuables must be protected, but before that can happen, the security system itself must be secure. For embedded systems, the security elements within the system and what it protects must be safeguarded. At the most basic level, this means securing the cryptographic keys and identity that are used to validate software, users and connectivity links. It also means ensuring the integrity of the software running on every system or node in a network. This requires visibility into and control over the boot-up and run-time software on even the most unassuming node in a network or on the Internet.

How much security?

Security, like everything else, comes with a cost. The cost of security for system developers includes the cost of designing and integrating security measures into the system, as well as the toll on the system's performance that those security measures will exact. Given the constantly changing nature of security threats, as well as the continuing ubiquity of embedded systems through initiatives like the IoT, the design of a new system should include the development of a set of metrics that will measure the cost of security against its benefits. Embedded devices can be taken over and used as a launching pad for attacks on other systems where more valuable resources may be located. For example, hacking into a printer/copier may not yield much value to the hacker, but if every document the printer prints or copies is captured and sent to hackers, the damage could be immense.

Embedded systems have an advantage when it comes to the cost of security as many of the products based on embedded systems are produced in great numbers. As a result, the cost of the security subsystem developed for these products can be amortized over large production runs, lowering the per unit cost of security. In addition, a versatile, scalable and portable security architecture developed for a new design can often be transferred to closely related systems or the architecture might be modified slightly to suit the needs of other products.

Architectural considerations

Many security subsystems are architected in layers and take advantage of compartmentalization. Deploying security measures in layers has a cumulative effect on the security of the system because each layer can certify the security of the layer below or above it before any action is taken. Compartmentalization is important for ensuring run-time security of software running on the system and it gives designers the ability to tailor security measures depending on the relative value of the resource or process being protected.

Embedded security starts in hardware. Coupling software and hardware security features together enables a more secure layer of protection than either solution working independently. In addition, the tools provided by vendors can streamline the development of security subsystems and ensure that the resulting architecture meets the developers' requirements. For example, hardware-based security accelerators can mitigate performance cost of a security subsystem.

Of course, the strength of a security architecture will depend on the foundation upon which it is built. Three aspects of the foundational layer are essential: a secure boot process, hardware-based device ID/keys and cryptographic acceleration.

The security pyramid

The security pyramid (see **Figure 1**) illustrates the various layers and constituent parts of a comprehensive security subsystem for a multicore SoC embedded processor.

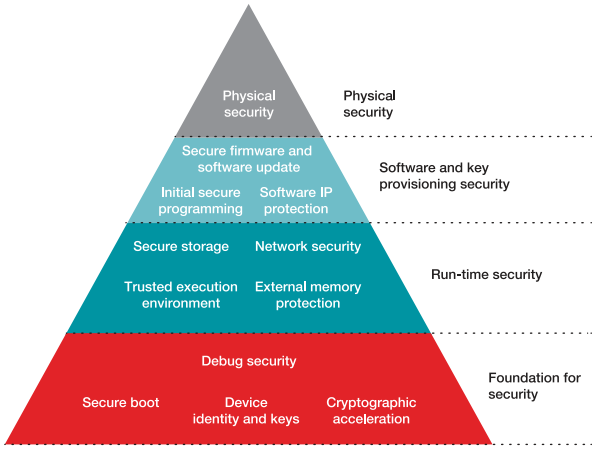


Figure 1. *Security pyramid.*

Secure boot

A secure boot process establishes a root-of-trust for the embedded system. Even when booting is initiated from external Flash memory, a secure boot process verifies the integrity of the boot firmware through any number of mechanisms, including embedded cryptographic keys and others. The secure boot layer safeguards against takeover of the system by malware, any possible cloning of the in-system IP, inadvertent execution of unwanted applications and other security risks.

Secure boot also assists in providing an additional layer of protection by encrypting the IP and copying it securely to protect internal memories. Having the ability to encrypt also provides additional security for code base as it prohibits carrying out directed exploration attacks.

Bottom-line, secure boot assists in establishing a foundation for embedded system security.

Cryptographic acceleration

Cryptographic processing, involving the generation, verification and certification of various public and private keys, can take a toll on the performance and throughput of an embedded system. Some muti-core application processors are equipped with hardware-based accelerators or co-processors that speed up the coding/decoding processes tremendously. Software-based acceleration is also available, but, as software, it is not as inherently secure as hardware-based cryptographic acceleration.

Common cryptographic elements	
Random number generator (RNG)	Used by cryptographic algorithms and hashing functions. Hardware-generated random numbers are more secure than software-generated RNG.
Cryptographic algorithms	
3Data encryption standard (3DES)	3DES performs DES encryption three times to strengthen the protection of the encrypted data and overcome some of vulnerabilities of the DES algorithm.
Public Key Algorithms (PKA)	Accelerated PKA using RSA or ECC asymmetric encryption using public/private keys. Helps with authentication used in secure boot.
Advanced encryption standard (AES)	AES is one of the most advanced cryptographic algorithms in widespread use today.
Hashing functions (for signatures, authentication, and so forth)	
Message digest algorithm (MD5)	Although this hashing function has been widely deployed,it has certain vulnerabilities in some applications.
Secure hash algorithm 2 (SHA2)	Processes large hash, so more secure than SHA1.

Table 1. *Examples of common cryptographic functions.*

Device-ID and keys

In order to trust communications over a local-area network (LAN), wide-area network (WAN) or the Internet, devices must have a unique identity which can be shared. Communicating devices can then decide the authenticity or trustworthiness of the other devices participating in the conversation.

Application processors often come with a unique identification (ID) code of some sort. Alternatively, or in addition to the ID code, devices might identify themselves through a signature or certificate key with

a corresponding public key that is accessible through a cloud service, for example.

Figure 2. *Device ID helps prevent theft.*

Debug security

During system development, designers need access to embedded multicore application processors in order to debug firmware and software, and to troubleshoot possible hardware problems. In most cases, the port that provides this access is the JTAG port. In an operating environment, the debug port must either be sealed closed by some sort of fuse, or it should only be accessible through certified cryptographic keys. Otherwise, the debug port could provide an easy way into the system for hackers (see **Figure 3**).

Figure 3. *MSP430™ MCU debug port.*

Trusted execution environment

The run-time security layer is comprised of several distinct capabilities which all play a part in protecting the system following the boot-up process and while the system's operating system (OS) is executing. An important aspect of run-time security is to monitor all aspects of the system to determine when an intrusion has either occurred or been attempted.

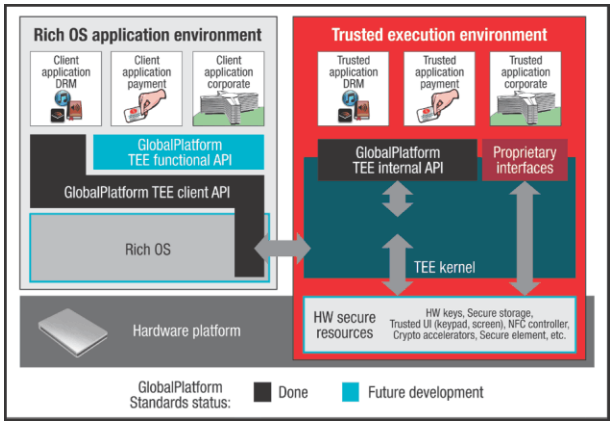


Figure 4. *Trusted execution environment (TEE).*

Trusted execution environment security provides the ability for a system to host secure and non-secure applications concurrently and maintain the partition through the system such that there is no leak of data. It is important to run sensitive applications where the application and associated code/data base is fully sand-boxed from other applications.

A trusted execution environment essentially provides a secured partition within a multicore system where only certified secure firmware, software and applications can execute, and certified data can be stored.

Walling off the trusted execution environment from the rest of the multicore/multiprocessing system prevents suspect code, applications and data that may pass through the system from contaminating mission-critical software, data and other IP.

External memory protection

When designers must add another application or subsystem to the system, they usually are faced with adding memory that is external to the main processor and connected to it by a memory bus. Designers must protect the data stored in external memory against tampering or replacement so they can be ensured that only trusted data or application code are stored in external memory. A number of methods can

be employed to safeguard the contents of external memory, such as secured execute-in-place directly from external memory without loading data into the processor's integrated memory, decrypt-on-the-fly which can maintain confidentiality while allowing applications to run on the main processor and other methods.

Figure 5. *Secure memory.*

Network security

Hackers are quite adept at intercepting wireless or wired network communications. In fact, some communication protocols have known security weaknesses that have been exploited. Deploying only highly secure communication protocols often involves a significant number of processing cycles to encrypt and decrypt the communication stream, as well as verify the authenticity of the sender or receiver. Designers are sometimes faced with balancing communication throughput and security, but some embedded processors avoid this dilemma by integrating hardware-based accelerators for the cryptographic algorithms that are used in conjunction with standard communication protocols.

Figure 6. *Secure storage.*

Secure storage

Cryptographic keys and security data must be stored in system memory in locations that are impervious to unwanted access. A number of capabilities can be used to provide secure storage, including encrypted blob of keys, anti-tamper protection that can only be unlocked by a master key, a private key bus between non-volatile memory and cryptographic engines, and others.

Initial secure programming

In today's era of globalization where the design, key provisioning and manufacturing are disjoint, and sometimes occur oceans apart, it creates a challenge to keep security assets like keys safe. To make things more complex the business model may involve ODM with completely un-trusted manufacturing setup.

Security enablers like initial secure programming provides a methodology that customers can evaluate and elect to use to strengthen the confidentiality, integrity and authenticity of initial firmware or keys programmed in an untrusted facility or during the first boot of the application.

Secure firmware and software updates

Ability to update the system is essential part of security framework, this provides the opportunity for customers to patch or update the software remotely to combat identified vulnerability in the system, however the biggest challenge during update is to deter against spy, impersonate and replays.

Security framework provides additional keys and mechanism like authentication, encryption and integrity checks that can be deployed to ensure the genuineness of the updates.

Software Intellectual Property (IP) protection

Customers make significant investments to create Intellectual Property (IP) that may represent the critical value proposition to end users in the market, hence it becomes imperative that security framework provides mechanisms like encrypted boot, the ability to carry isolated processing, and fire walling to allow customers to protect their IP.

Physical security

Sophisticated and not-so-sophisticated hacking organizations have been known to remove chips from a system or a silicon die from a chip package to access the embedded assets.

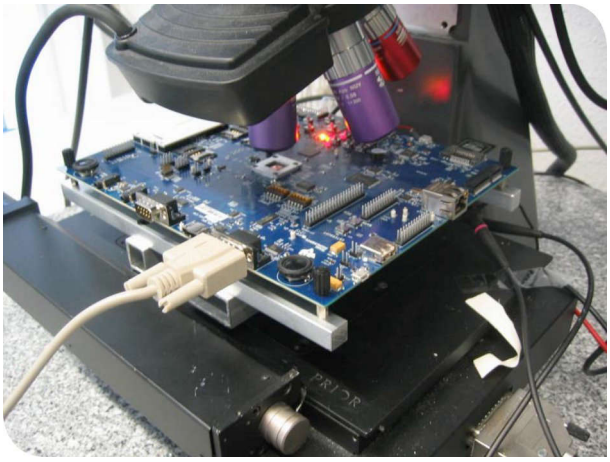


Figure 7. *A system under physical attack.*

Once the device or die have been removed, hackers can bombard them with lasers, power them up beyond their specified power limits or employ other means. Their objective is to observe how the device reacts to the stimulus because this response may betray vulnerabilities that the hackers can then exploit to access the device.

Some application processors have been integrated with hardware and software features to thwart these physical intrusions into both the digital and analog sections of SoCs. Tamper-protection modules integrated into multicore application processors can contain power and temperature monitors, reset functionality, frequency monitors and programmable tamper-protection capabilities.

Enclosure protection

Enclosure protection features are physical measures that safeguard the enclosure which encases a system. These can range from locking mechanisms to electronic switches, break-away wire tripping mechanisms and others (see **Figure 8**).

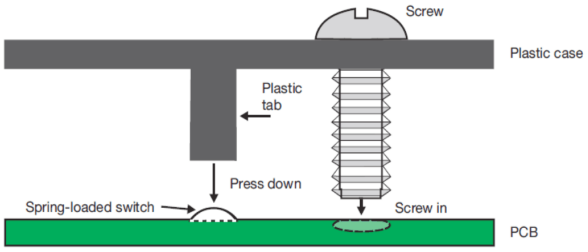


Figure 8. *Enclosure protection.*

Where to start with embedded security?

The fundamental basis for the security of an embedded multicore application processor begins in hardware. If the hardware is not secure, no amount of security software will assist in making it so. Assuming security features are built into the hardware, the first place to look to begin building a security subsystem is in the first software that will execute following power up, the boot code. If the booting process cannot be authenticated, then no other software running on the system can be

either. So, securing the boot process is the fulcrum upon which all of the security in the system depends.

A secure boot process establishes the root-of-trust, which is the goal of every security subsystem. Establishing a root-of-trust through a secure boot process helps to ensure the integrity of the system and guards against hackers taking over any part of the system. This also helps protect customer software in the system and acts as an anti-cloning barrier so the system or any part of it cannot be copied.

Usually, a secure boot process involves programming a public cryptographic key into non-volatile, one-time-programmable memory somewhere in the system. Then, this public key must be matched up with private/public keys associated with the boot code to authenticate the validity of the encrypted boot code before execution begins. Booting firmware can either be loaded into the embedded processor's RAM or, for added security; can be secured and executed-in-place out of memory external to the embedded processor. Some firmware images are made up of various components or modules. Requiring authentication before decrypting and executing each module enhances boot security.

Security enablers for TI application processors

Our Arm-based application processors offer a comprehensive set of security enablers to help developers implement their security measures to protect their assets (data, code, identity and keys).

Security enabler	AM335x	AM437x	AM438x	AM570x/ AM574x	AM64x/AM65x	AM62/ AM62L/ AM62P	AM68x/AM69x	DRA821/ DRA829/ TDA4VM
Cryptographic acceleration	✓	✓	✓	✓	✓	✓	✓	✓
Device identity/ Keys	✓	✓	✓	✓	✓	✓	✓	✓
Secure boot	✓	✓	✓	✓	✓	✓	✓	✓
Debug security	✓	✓	✓	✓	✓	✓	✓	✓
External memory protection			✓		✓	✓	✓	✓
Trusted execution environment (TEE)		✓	✓	✓	✓	✓	✓	✓
Network security					✓	✓	✓	✓
Secure storage		✓	✓	✓	✓	✓	✓	✓
Software IP protection	✓	✓	✓	✓	✓	✓	✓	✓
Initial secure programming	✓	✓	✓	✓	✓	✓	✓	✓
Secure firmware update	✓	✓	✓	✓	✓	✓	✓	✓
Physical security			✓					
Request Access	Contact TI Representative	More Info	More Info	More Info	More Info	More Info	More Info	More Info

Table 2. *Security enablers for TI application processors.*

Conclusion

Embedded processor security is a multifaceted, complex subject. With the ascent of the IoT and the ubiquity of embedded systems, hackers, now more so than ever, have an abundance of prime targets.

Of course, fundamental security features must already be present in the hardware, but building a security subsystem for an embedded multicore SoC should start at the foundational layer of secure boot. Without a root-of-trust derived from a secure boot process, no other security measures matter. Once this root-of-trust is established, other facets of system security, such as debug security, run-time security and networking security, have a solid footing. Otherwise,every security measure is built on sand.

References

1. Texas Instruments: *E-book: Building your application with security in mind* (SWPB021)
2. Texas Instruments: ***Enable security and amp up chip performance with hardware-accelerated cryptography***
3. Texas Instruments: ***Secure Boot on embedded Sitara™ processors***
4. Texas Instruments: ***Sitara AM438x processor: tamper protection***

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265