AN12278

LPC55S00 Security Solutions for IoT

Rev. 1 — 05/2020 Application Note

Contents

1 Introduction

Security should be a concern for all creators of connected devices. Depending on the application and target market, there are governmental requirements for security.

The LPC55S00 is a family of Arm® Cortex®-M33-based MCUs for embedded applications. It features a rich peripheral set leveraging the new ARMv8M architecture. The LPC55S6x introduces new levels of performance and advanced security capabilities including TrustZone-M and multiple co-processor extensions. The family adds LPC55S2x and LPC55S1x. The LPC55Sxx family provides protection for the end products from unexpected threats over their entire lifecycle. This application note provides the differences and advances in the security systems for each LPC55Sxx MCU.

The LPC55Sxx is secure-by-design and supported by secure software driving the secure System on a Chip (SoC). This application note describes the SoC designed-in security features of the LPC55Sxx MCU as they relate to how the state of the art security features address the security goals over the life-cycle of a device. The security sub-systems and SoC specific details of the LPC55Sxx are discussed and an overview of secure boot is provided. The deeper dive into these features can be found in The *LPC55Sxx User Manual* and other application notes, referenced within where possible within this document. See the LPC5500 documentation page at www.nxp.com for most references. A rich set of supporting information is listed in the Resources section of this application note and a Glossary of terms is provided to define and explain the numerous acronyms used in this application note.

1.1 Security landscape

When designing a secure system, it is important to start with a security model. A security model is built from policies, the threat landscape, and methods described below. Applying a security model provides a framework for understanding and designing to the security goals of the device. The methods, or how the security policies are enforced to achieve product security goals, are made possible by the technology that is integrated into the embedded controller. The following introduces a basic security model and provides examples of policies, threat landscape, and methods policies. The rules in place that identify the data that should be protected (for example; the management of firmware, secret keys, user and application data passwords, personal information, and network credentials).

The threat landscape is the definition of the attacks and attackers that the end device faces and provides protection against. It considers the access to the device and the cost of the attack. For example, expert attackers use off-the-shelf tools to gain access and insert malware. The methods are the means by which the policies for the device are enforced. This involves the application of security technology to achieve product goals (for example; disabling the debug access to restrict the availability of secret data on a processor).

1.2 Exploits of secure IoT

All IoT devices need protection. The exploitation of these devices, these nodes we call “things” can lead to a large-scale disruption of internet services or bring harm to the user of the thing.

As a designer, you are tasked with designing a secure thing and finding all the vulnerabilities through a threat analysis. An attacker only needs to find one. How can you succeed? If you don’t succeed, what is the plan to recover from the attack? Do you have a

way of securely updating the firmware of your thing to fight the attack? This must not only be a trusted thing but it must maintain a trusted connection to the overall application network called the cloud or the back-end.

As more and more things are deployed, we want the level of trust to scale with the growth of IoT endpoint. Security cannot be made an optional item, it must be architected from the beginning of the design concept regardless of the thing you are designing. To authenticate everything is the mantra of the design of the IoT security landscape. Your thing must be authenticated along with the gateway, the compute and storage, and the application and analytics layers. In this application note, we focus some effort on the security of the thing itself and security as it communicates with the back-end layers. We discuss the high-level security goals of IoT devices along with the security protections needed over the devices' entire life-cycle.

1.3 Life cycle of an IoT device using LPC55Sxx

A thing, delivered into a world of IoT devices, adheres to a standard life cycle. In the next subsection, we identify the life cycle stages and recommendations to create a secure trusted device. Tasks for each life stage provide the context for requirements. Within each stage, tasks are focused on the LPC55Sxx MCU value additions. The details on how to use the security features are detailed in the MCU user's guide.

Figure 1. Life cycle

Procure

Establish a trusted supply chain so you can ensure the design integrity of all the hardware and software components. To do this, you need to use an MCU and tools that incorporate trusted software and hardware, such as the LPC55Sxx. The security sub-system allows for isolation of secure software IP. The Physically Unclonable Function (PUF) offers the Root Of Trust (ROT) key management and a unique identity.

Develop

During general development, all debug ports are enabled. The ROM enables debug access after part configuration and secure boot and routines complete. The device is not provisioned with keys, so only CRC of the images needs to be used during development.

During secure development SRAM PUF generated ROT keys are tested. Software can be encrypted in Flash and the secure trusted environment can be fully tested. Customers can use public development keys to establish and authenticate communication. Protect the product from inadvertent debug with ROM-based debug authentication.

Manufacture

Key generation and provisioning can be done during this cycle. Because the private and public keys are generated with each power up of the MCU, there is no need to create keys during the manufacturing stage. At this point, the unique identity is recorded and would be allowed to communicate securely with the back-end during the deployment (or target use) of the device.

Deployment

The flexibility in defining debug configuration allows module makers to implement tiered protection approach. The part can be sold by tier 1 customers (secure code developer) to tier 2 customers who develop non-secure code only. In this scenario, tier 1 customer releases the part to always allow non-secure debug, allowing tier-2 customers to further seal the part.

Debug ports are closed as per the application configuration. Debug can be enabled after debug authentication, disabled permanently, or enabled for non-secure debug interfaces.

Only signed images are allowed if secure boot is enabled.

Customer keys are updated as firmware updates public keys and Prince Keys while maintaining RoT Keys.

Secure Firmware can be erased and reprogrammed with boot loader or secure firmware.

Application code firmware is programmed through mechanisms exposed by the API. Separate secure region (independent Prince key) can be used for storing firmware.

Use

IoT device is added to the IoT infrastructure and authenticated by the back-end. Data integrity is checked internally and then communicated securely to back-end systems.

Maintain

The firmware of a trusted thing can be updated with a secure image Over-The-Air (OTA). The API would authenticate the received image and allow and install firmware updates. If a firmware update is revoked, the ROM offers a boot from the SPI NOR flash as a backup.

Customer Return (FA mode)

Keys and firmware are destroyed. Designers use the debug authentication mechanism to set the FA_MODE field in the customer program area or customer ships the de-soldered part to NXP. All keys are erased, the PUF indexes are blocked, and the debug ports are opened.

Decommission

When removing the thing from service, the unique identity of the thing can be removed from the list of trusted things and the firmware can be updated or erased to render the thing non-functional.

1.4 High-level security goals

Today's IoT devices must be designed with advanced security technology to protect against unauthorized access, malicious control, and data theft. The connection of devices to each other and the cloud requires scalable and easy-to-use security solutions. The keys are used to setup secure connections and protect data transmitted by IoT devices.

IoT system developers must determine the level of protection needed to ensure product compliance with applicable security standards and regulations, as well as product user expectations and company risk management policies to prevent local and remote attacks.

For today's embedded designs, there are six essential security goals to consider during the life cycle of a thing. The LPC55Sxx series MCU provides security hardware features, that combined with software, address each of these goals.

Below are the system level requirements for the high-level security goals.

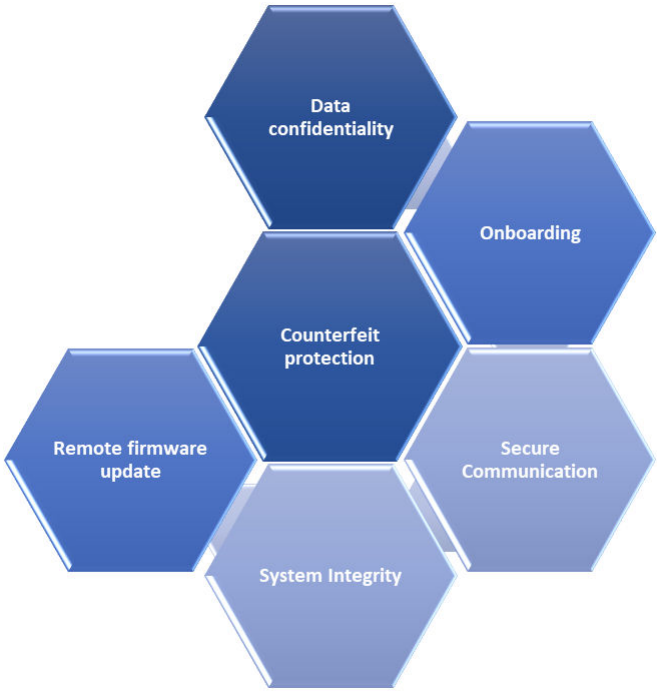

Figure 2. High-level security goals

Counterfeit protection

This is about ensuring that the device is authentic. It starts by establishing a unique identity and supporting authentication that would be difficult for an attacker to reproduce. The RoT Keys are not accessible and cannot be read out if physically (or logically) inspected.

Onboarding

This is about how you bring an IoT device onto your network and commission it into service. The device must enforce privilege levels isolating non-secure firmware from secure firmware. Your device must protect private keys with hardware while protecting shared credentials between the end device and the back-end system.

System integrity

With every power up, the device must protect itself with a secure boot. The MCU must enforce trust with the functionality provided by the MCU. The firmware on the MCU must encrypt sensitive software functions to prevent reverse engineering. The device must support fall back images to sustain minimal functionality.

Secure communication

This is about using asymmetric cryptography to establish a session key, then switch to symmetric cryptography for bulk communications to protect connections and data at rest. The device should maintain audit logs that are kept encrypted in the flash.

Ensure Data confidentiality and integrity

This is about controlling the data flow and protecting data confidentiality and integrity at rest and relying on secure communication. This is about collecting and storing data securely within the SoC.

Update firmware/software when vulnerabilities are discovered

This is about keeping your IoT device doing the job it was designed to do and ensuring critical service availability. Your device should monitor and report unauthorized accesses, limit sensitive functions and services (wireless network connections) and support firmware updates securely. At the end of the product's life, it must be decommissioned to remove all sensitive data and prevent hackers from bringing it to another network.

2 LPC55Sxx Product Overview

2.1 LPC55Sxx MCU platform and general-purpose blocks

The LPC55S69 has one 100-MHz Cortex-M33 core with TrustZone, MPU, FPU, and SIMD and another 100-MHz Cortex-M33. The LPC55S66 only implements a single 100-MHz core. There are two coprocessors on core 0, a DSP accelerator called PowerQuad, and a crypto engine called CASPER. The core platform has a multilayer bus matrix that allows simultaneous execution from both cores and parallel access of the other masters to peripherals and memories. The memory on chip includes up to 640 KB of Flash, up to 320 KB of RAM, and 128 KB of ROM.

Timers include 5 - 32-bit timers, a SCTimer/PWM, a multi-rate timer, a windowed watchdog timer, Real Time Clock (RTC), and a micro timer. Each core has its own systick timer. The LPC55S6x security sub-system includes TrustZone, HW SRAM PUF, Debug Authentication, real-time PRINCE decryption, TRNG, Secure boot, DICE, SHA-2, AES-256, and PFR.

The LPC55S2x is a subset of the core and peripherals of the LPC55S69 with up to 512 KB of flash and 256 KB of RAM. The LPC55S2x security sub-system includes HW SRAM PUF, Debug Authentication, real-time PRINCE decryption, TRNG, Secure boot, DICE,SHA-2, AES-256, and PFR. All of the code in the part is secure when the part is locked. The LPC55S1x is a cost-reduced MCU with one 100-MHz Cortex-M33 core with TrustZone, MPU, FPU, and CASPER crypto engine on the co-processor interface. It also features 256 KB of flash, 96 KB of RAM, and additional peripherals, such as CAN-FD. The LPC55S1x security sub-system includes TrustZone, HW SRAM PUF, Debug Authentication, real-time PRINCE decryption, TRNG, Secure boot, DICE, SHA-2, AES-256, Code Watchdog, and PFR.

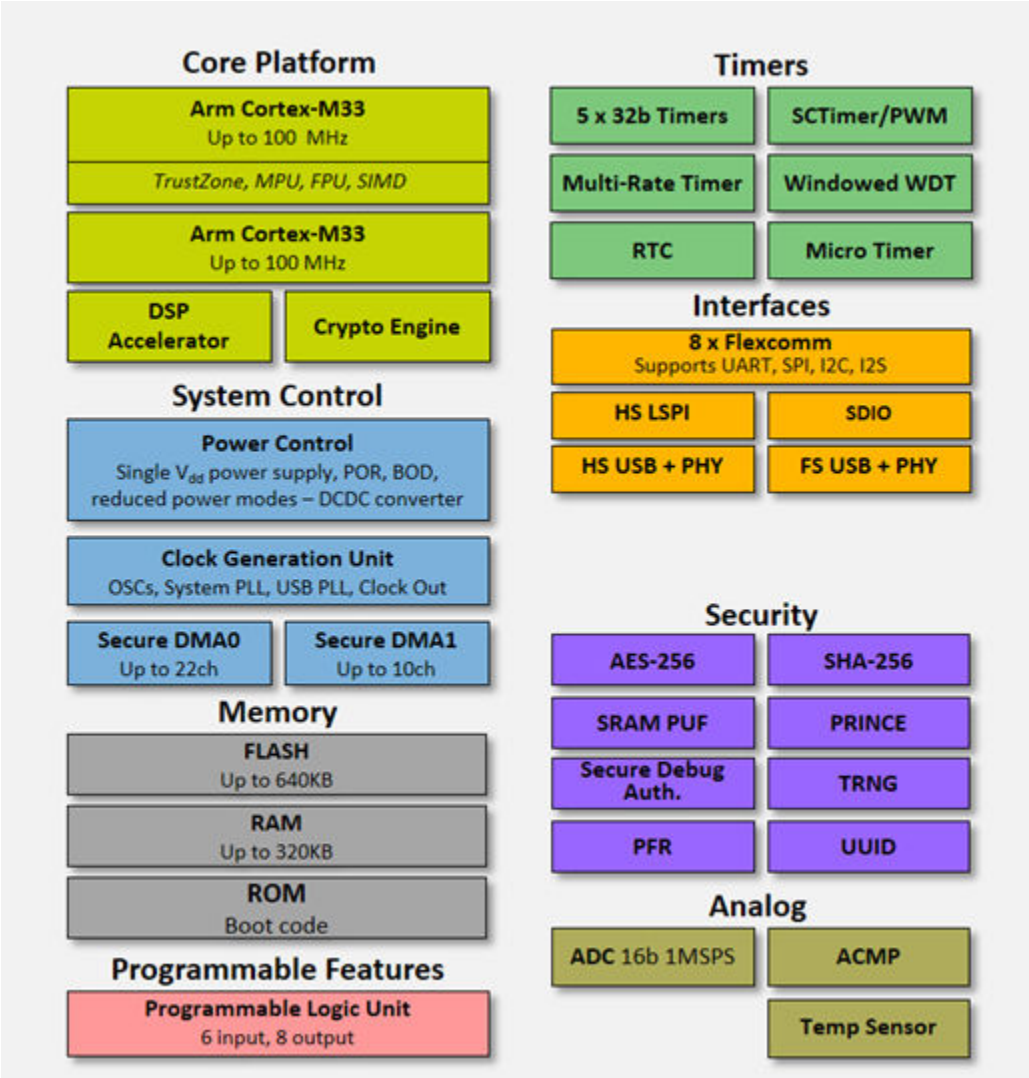

Figure 3. LPC55S6x MCU platform and general-purpose blocks

Communication interfaces include a USB high-speed with on-chip HS PHY, a USB full-speed that can run crystal-less, two SDIO interfaces to support WIFI and SD cards at the same time, 1 high-speed SPI with up to 50-MHz clock rate, and 8 Flexcomms with support of up to 8 SPI, I^2C, UART, or 4 I^2S.

The analog system includes a 16-channel 16-bit ADC that samples at 1 MSPS, an analog comparator, and a temperature sensor.

Other modules include a programmable logic unit, a buck DC-DC converter, operating voltage from 1.71 to 3.6 V over a temperature range from -40 to 105 °C.

2.2 LPC55Sxx Security Blocks

2.2.1 LPC55Sxx security

The LPC55S6x and LPC55S1x are SoCs with enhanced security features and isolation within the SoC that includes TrustZone. The LPC55S6x and LPC55S1x have a secure and a non-secure resource domain controller called the Trusted Execution Environment (TEE). With the right software configuration, the TEE can create security hardware isolation within the SoC. Also included is a ROM that supports secure boot, authenticated signed and encrypted, and a secure debug interface which requires authentication to unlock the debug feature.

2.2.2 LPC55Sxx Platform Security Architecture

NXP has designed the LPC55Sxx with a comprehensive security foundation supporting trusted execution based on Arm TrustZone M working in conjunction with boot and run-time hardware architecture features. NXP includes hardware that supports software isolation including a Secure Attribution Unit (SAU), secure bus, DMA, and secure GPIO. The LPC55S6x and LPC55S1x security combines the TrustZone with the security hardware. There is security throughout the system for software, CPUs, interconnect, memory, and peripherals. There are trusted peripherals for a fortified security for the entire device life cycle.

This hardware isolation creates a trusted, secure execution environment with divided access to both secure and non-secure resources and an execution environment with restricted access to data stored in memory.

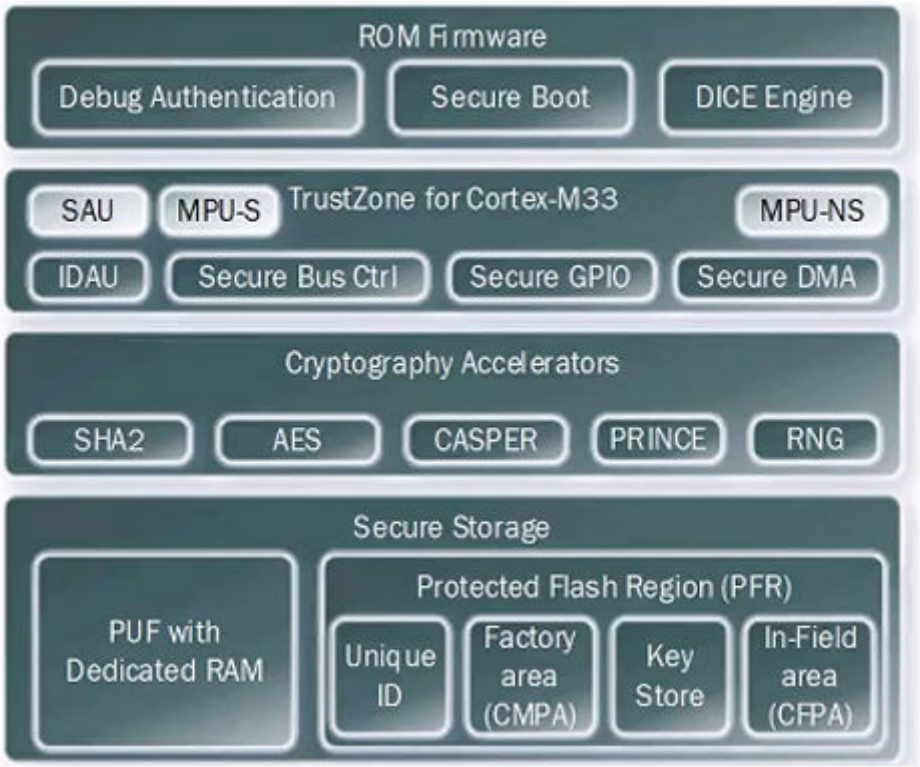

Figure 4. LPC55S6x and LPC55S1x Security sub-systems

2.2.3 LPC55Sxx security features

Figure 4 details all of the blocks which contribute to the security services on the LPC55Sxx. At the bottom, are different hardware capabilities related to the storage and generation of chip specific secrets (UUID, PUF and PFR) This chip provides options for using NXP programmed unique ID, OTP, and cutting edge PUF technology to generate keys. The ROM uses this information, along with cryptographic accelerators for AES, Hashing to apply security services to the secure booting for the chip The ROM supports boot loading from internal flash and support factory floor programming through ISP mechanism over I/O serial interfaces or as a backup bootload from external SPI flash.

Before aligning the security goals and the life cycle, a quick description of the LPC55Sxx SoC hardware security features is provided for the LPC55Sxx product. In the column header in Table 1, 6x(0A) refers to the first silicon and 6x(1B) refers to the second (or production) silicon.

Table 1. LPC55Sxx series security summary table

-			LPC55S			
			6x(0A)	6x(1B)	2x	1x
ROM Supporting	Secure Boot	Signed images	x	x	x	x
		Unchangeable boot firmware	x	x	x	x
		PRINCE encryption	x	x	x	x
		Preconfigured Trustzone	x	x	x	x
	DICE Engine	CDI = HMAC(UDS, RTF)		x	x	x
		RTF = image || NMPA || CMPA	x	x	x	x
		RTF = image || NMPA || CMPA || EPOCH		x	x	x
		RTF = SHA256(image) || NMPA || CMPA||EPOCH				x
	SYSCON registers PSA boot state	BOOT_SEED				x
		HMAC = HMAC (BOOT_SEED, SHA-256(image) || NMPA || CMPA || EPOCH)				x
	Debug Authentication	Supports NXP Debug Authentication Protocol version 1.0 (RSA-2048) and 1.1 (RSA-4096)	x	x	x	x
	Recovery Boot	External SPI Flash recovery boot image		x	x	x
		FLASH_REMAP (dual image)				x
TrustZone for Cortex-M33	Arm's Security System Implementation	Security Attribution Unit (SAU)	x	x		x
		Memory Protection Unit (MPU) Secure	x	x		x
		Memory Protection Unit (MPU) Non-Secure	x	x	x	x
	NXP's Security Sub-system Implementation	Defined Attribution Unit (using IDAU interface)	x	x		x
		Secure Bus Controller	x	x		x
		Secure GPIO Controller	x	x		x
		Secure DMA Controller	x	x		x
		Memory protection checker	x	x		x
		Peripheral Protection checker	x	x		x
Security Subsystem	Cryptography Accelerators	HashCrypt engine: Symmetric AES-256 and Hashing SHA2	x	x	x	x
		AES-ICB (form of CTR) SCA resistant	x	x	x	

Table continues on the next page...

Table 1. LPC55Sxx series security summary table (continued)

		AES-ECB,CBC,CTR				x
		SHA digest context				x
	PRINCE	On-the-fly flash encryption/ decryption engine	x	x	x	x
		3 regions (sub-regions enabling)	x	x	x	x
	CASPER	Cryptographic Accelerator and Signaling Processing Engine with RAM for RSA and ECC	x	x	x	x
	Random Number Generator	RNG				x
		TRNG - not certified yet, some checks were done, will pass most test suites DieHard, NIST SP800-22, FIPS 140-1	x	x	x	x
		RNG health status, entropy accumulation, initial entropy				x
	Code Watchdog	Detects side-channel attacks or execution of unexpected instruction sequences				x
Secure Storage	Physically Unclonable Function (PUF)	SRAM PUF for key generation and identity	x	x	x	x
		Device unique root key (256 bit strength)	x	x	x	x
		Can store key sizes 64 bit to 4096 bit	x	x	x	x
	Protected Flash Region (PFR)	RFC4122 compliant 128-bit UUID per device	x	x	x	x
		PUF Key Store	x	x	x	x
		Activation code	x	x	x	x
		Prince region key codes	x	x	x	x
		FW update key encryption key	x	x	x	x
		Unique Device Secret	x	x	x	x
	Customer Manufacturing Programable Area (CMPA)	Boot Configuration	x	x	x	x
		RoT key table hash	x	x	x	x
		Debug configuration	x	x	x	x
		PRINCE configuration	x	x	x	x
		Flash remap				x
		Customer data (224B)	x	x	x	x

Table continues on the next page...

Table 1. LPC55Sxx series security summary table (continued)

	Customer Field Programable Area (CFPA)	Monotonic counter	x	x	x	x
		Prince IV codes	x	x	x	x
		Customer data (224B)	x	x	x	x

For a complete description on how to use these blocks, see the documentation for the LPC5500 Series Cortex-M33 MCUs at nxp.com.

Figure 4 details all the blocks which contribute to the security services on the LPC55Sxx. The diagram is divided into four sub-groups and then into the hardware blocks that make up those sub-groups. Below are descriptions of these blocks.

2.3 ROM firmware

The internal ROM memory stores the unchangeable boot code, which performs credential handling, decryption, authentication, secure program flow, and security state enforcement. After any reset, the CM33 starts its code execution from the ROM. The bootloader code is executed every time the part is powered on or with every reset. The ROM allows for various boot options and APIs. Depending on the values of the Customer Manufacturing Programmable Area (CMPA) bits, In System Programming (ISP) pin, and the image header type definition, the bootloader decides whether to boot from internal flash or run into ISP mode.

2.3.1 Protection with secure authenticated boot

The LPC55Sxx allows booting of public-key signed images. The secure boot ROM supports three types of security protected modes to allow a scalable security and manufacturing options for end products that use the MCU. The ROM supports booting from encrypted PRINCE regions. Encrypting images allows strong protection of intellectual property. The ROM supports public keys and image revocation – that is, the method of not allowing new updates to be applied unless they are of a specific version. This is the basis for the roll-back protection. The ROM supports pre-configuration of TrustZone-M settings using the secure image, minimizing the possibility of attacks that interrupt the user firmware flow during TrustZone configuration. The ROM supports secure booting from SPI flash if all other secure flash images fail the authentication. In the ISP mode, the contents of the external SPI flash are copied to the RAM, authenticated, and execute the copied code (if secured).

2.3.2 LPC55Sxx ROM firmware features

The ROM supports secure boot, CRC image integrity checks, and debug authentication. Secure boot checks the integrity of code credentials during the boot process. System integrity is maintained because each boot is checked for authenticity. For firmware updates, the ROM-based authentication supports SPI flash recovery and roll-back protection.

2.3.2.1 LPC55Sxx secure boot

Secure boot prevents execution of unauthorized code. The MCU ROM always executes with every power up and reset. The ROM examines the first user executable image resident in internal flash memory to determine the authenticity of that code. If the code is authentic, then control is transferred to it. This establishes a chain of trusted code from the ROM to the user boot code. This chain can be further extended using boot ROM API.

The method used in this SoC to verify the authenticity of the boot code is to verify RSA signatures over the code. The boot code is signed with RSA private keys. The corresponding RSA public keys used for signature verification are contained in the X.509 certificates that are contained in the signed image.

The boot ROM provides an option to specify the above setting through image header and lock it before control is passed to application code. Since ROM is immutable code, highly secure application could use this option.

The ROM provides a way to perform a recovery boot from an external 1-bit SPI flash device to prevent updates from bricking the MCU and rendering the application useless. You can use this feature in a couple of ways.

1. Recovery media model where the external SPI flash is used to store a factory image in SB2.1 format. When the image on the main flash is corrupted, the ROM will attempt to recover the device by booting the device using the image on the external flash.

2. Staging media model. During OTA, a firmware update capsule in SB2.1 file format can be downloaded to the external SPI flash. Once the image is downloaded and validated, the running firmware can invoke a SPI recovery boot using the ROM API. On re-boot the ROM will execute the newly loaded SB2.1 file to update the main flash.

If secure boot is disabled, the recovery boot function boots a plain text image from the SPI flash, copying the image to SRAM, and jumping to execute the code. If secure boot is enabled a SB2.1 image is loaded from the external SPI flash into internal flash, authenticates the hashed public keys. If authentication passes then the ROM jumps to execute the code.

For detailed description of the steps used for all secured boot operations, see *LPC55Sxx Secure Boot* (document AN12283).

2.3.2.2 LPC55Sxx debug authentication

LPC55Sxx offers a debug authentication protocol as a tool to authenticate the debugger and grant it access to the device. The debug authentication scheme on LPC55Sxx is a certificate-based challenge-response scheme and assures that the debugger in possession of required debug credentials only can successfully authenticate over the debug interface and access restricted parts of the device. This protocol provides a mechanism for a device and its debug interface to authenticate the identity and credentials of the debugger (or user). Access to the right settings can be pre-configured and gets loaded into the debug register upon a successful debug authentication. Until debug authentication process is successfully completed, secure part of the device is not accessible to the debugger. The list below is a generic example of the debug authentication flow.

- Vendor generates RoT key pairs and programs the device with SHA256 hash of RoT public key hashes before shipping.
- Field technician generates his own key pair and provides public key to vendor for authorization.
- Vendor attests the field technician's public key. In the debug credential certificate, vendor assigns the access rights.
- End customer having issues with a locked product takes it to field technician.
- Field technician uses his credentials to authenticate with device and unlocks the product for debugging.

2.3.2.3 Device Identifier Composition Engine (DICE) architecture

The LPC55Sxx MCU boot ROM is a Trusted Platform Module which includes firmware to support the hardware requirements for a Device Identifier Composition Engine (DICE). The specification for the DICE inclusion is provided by the Trusted Computing Group. See www.trustedcomputinggroup.org/wp-content/uploads/Device-Identifier-Composition-Engine-Rev69_Public-Review.pdf. The engine creates an identity value that is derived from a Unique Device Secret (UDS) and the Run Time Fingerprint (RTF), a condensed cryptographic representation of the chip. This derived value is the Compound Device Identifier (CDI). You can use the Compound Device Identifier to attest to the trustworthiness of the booted code.

2.4 Security architecture and TrustZone for ARMv8M Cortex-M33

TrustZone is a technology now available in Cortex M33. TrustZone provides the means to implement separation and access control to isolate trusted software and resources to reduce the attack surface of critical components. The trusted firmware can protect trusted operations and is ideal to store and run the critical security services. The code protects trusted hardware to augment and fortify the trusted software. This includes the modules for hardware assists for cryptographic accelerators, random number generators, and secure storage. Best practices demand that that this code be small, well-reviewed code with provisions of security services.

The LPC55S66x and LPC55S1x have implemented core 0 as a Cortex-M33 with full TEE and TrustZone support enabled. The LPC55S69 has a second Cortex-M33 (core 1) that does not implement the secure environment with TZ. Users of the LPC55S2x MCU do not have the TZ available and must consider the entire MCU to either be secure or non-secure.

Isolation is just the foundation. Security is about layers of protection, adding in further hardware and software to create more layers.

2.4.1 Secure and non-secure memory attribution

Memory can be Secure (S), Non-Secure (NS), or Non-Secure Callable (NSC). These are defined by the Security Attribution Unit (SAU) and the Implementation Defined Attribution Unit (IDAU). Secure data can only be read by S code. Secure code can only be executed by a CPU in S mode. NS data can be accessed by both the secure state and non-secure state CPU. NS code cannot be executed by S code. The NSC is a special region that allows NS code to branch into and execute a Secure Gateway (SG)

opcode. This is the only way for NS code to call an S function. If SG is executed in the NSC region and the CPU is in the NS state, then the CPU moves to the S state. Note that only SG instructions can be legally executed out of the NSC region. The branching to the S function is done in the secure mode. The CPU states can be secure privilege, secure non-privilege, privilege (Handler), or non-privilege (Thread).

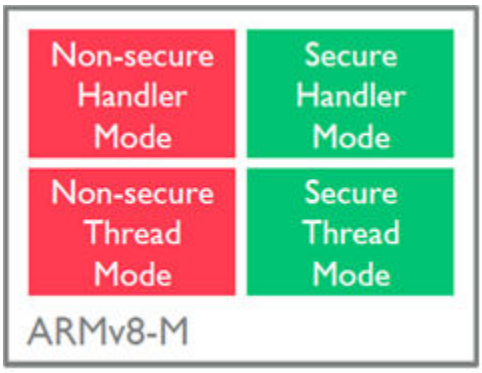

Figure 5. Memory attributes

2.4.2 ARMv8M additional CPU states

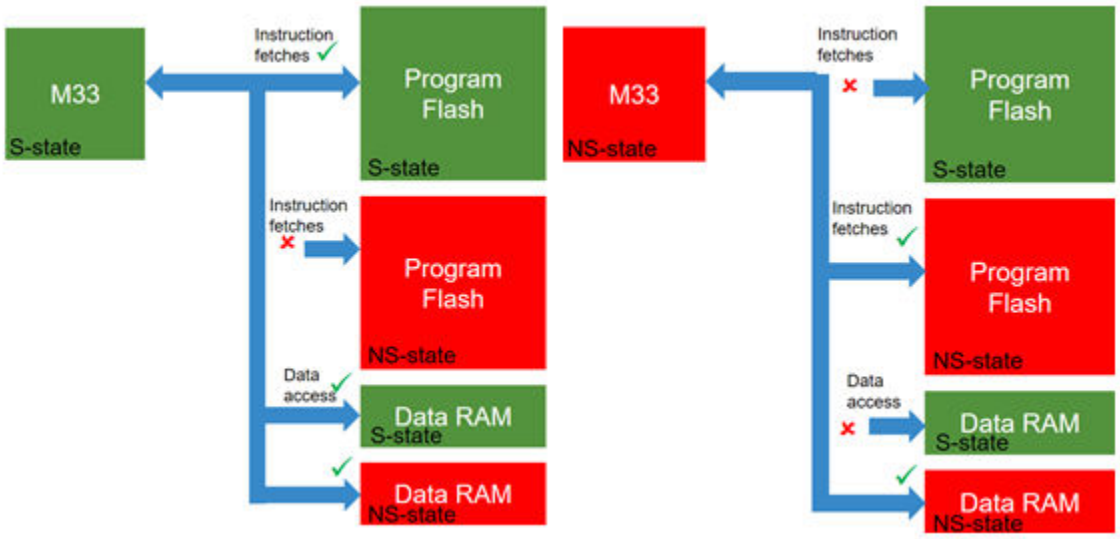

Figure 6. Secure and non-secure CPU states

Secure and non-secure code runs on a single CPU for efficient embedded implementation.

A CPU in a non-secure state can only execute from non-secure program memory. A CPU in a non-secure state can access data from NS memory only.

For the secure, trusted code, there is a new secure stack pointer and stack-limit checking. There are separate Memory Protection Units (MPUs) for S and NS regions and private SysTick timers for each state. The secure side can configure the target domain of interrupts.

2.4.3 Secure MPU, NVIC, SYSTICK and secure Stack Pointer

To support secure state Cortex-M33 architecture core0 extends to include secure MPU, secure NVIC, secure SYSTICK, and secure stack pointer with stack-limit checkers. Each CPU has its own set of these modules to better isolate the secure and non-

secure environment. Core1 does not have a trusted execution environment, therefore is considered a non-secure resource in the microcontroller.

2.4.4 Secure Bus Controller

The LPC55Sxx use a matrix of secure bus controller modules to manage the data flow in the MCU. A combination of a Peripheral Protection Checker (PPC), the Memory Protection Checker (MPC), the Master Security Wrapper (MSW) along with security locking and error log, secure interrupt masking, Hypervisor Interrupt and GPIO masking.

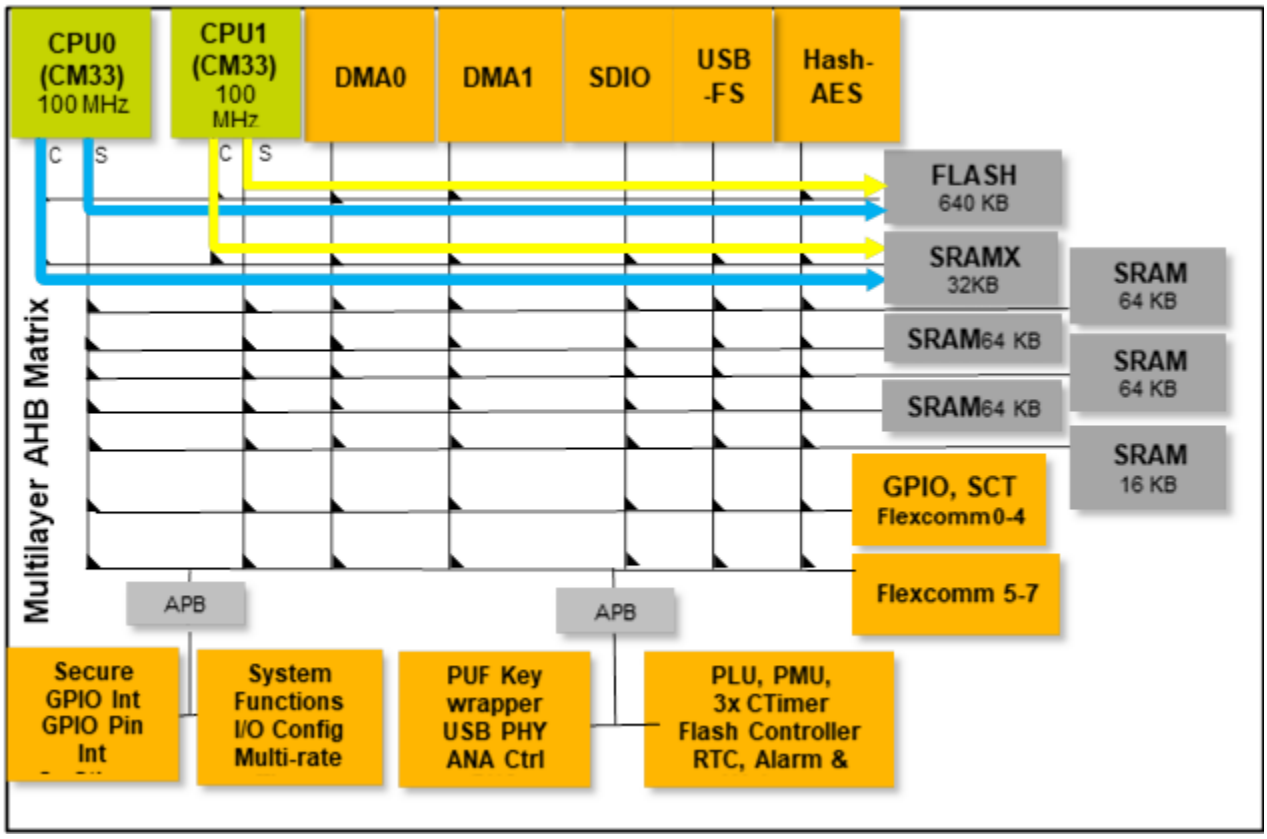

Figure 7. Secure Bus Matrix for LPC55Sxx

The Bus matrix between bus masters like the M33 core or DMA engines are wrapped with the Master Security Wrapper (MSW) and have security side band signals used for tamper detection. There is a PPC for each bus slave port. The MPCs are used for memories and bus bridges.

2.4.5 Security defined by address

Unique to the LPC55Sxx is the NXP IDAU (Implementation specific Device Attribution Unit) implementation of ARM TrustZone for core0 involves using address bit 28 to divide the address space into potential secure and non-secure regions. Address bit 28 is not decoded in memory access hardware, so each physical location appears in two places. Other hardware determines which kinds of accesses (including non-secure callable) are allowed for any address.

Table 2. TrustZone and System General Mapping

Start Address	End Address	TrustZone, CORE0	CPU Bus	CM-33 usage (both CPUs)

Table continues on the next page...

Table 2. TrustZone and System General Mapping (continued)

0x0000 0000	0x1FFF FFFF	Non-Secure	Code	Flash Memory, Boot ROM, SRAM X.
0x2000 0000	0x2FFF FFFF	Non-Secure	Data	SRAM A, SRAM B, SRAM C, SRAM D, SRAM E.
0x3000 0000	0x3FFF FFFF	Secure	Data	Same as above
0x4000 0000	0x4FFF FFFF	Non-Secure	Data	AHB and APB Peripherals
0x5000 0000	0x5FFF FFFF	Secure	Data	Same as above

2.4.6 Attribution Units

TrustZone for ARMv8-M implementation consists of the Security Attribution unit (SAU) and Implementation Defined Attribution Unit (IDAU). Device Attribution Unit (DAU) connects to CPU0 via IDAU interface as shown in Figure 8. Combination of SAU and IDAU assign a specific security attribute (S, NS, or NSC) to a specific address from the CPU0. Access from CPU0, dependent on its security status and the resultant security attribute set by the IDAU and SAU, is then compared by the secure AHB Controller to a specific checker which marks various access policies for memory and peripherals Figure 9. All addresses are either secure or non-secure. The Security Attribution Unit (SAU) inside of the ARMV8M works in conjunction with the MPUs. There are 8 SAU regions supported by the LPC55S6x and LPC55S1x.

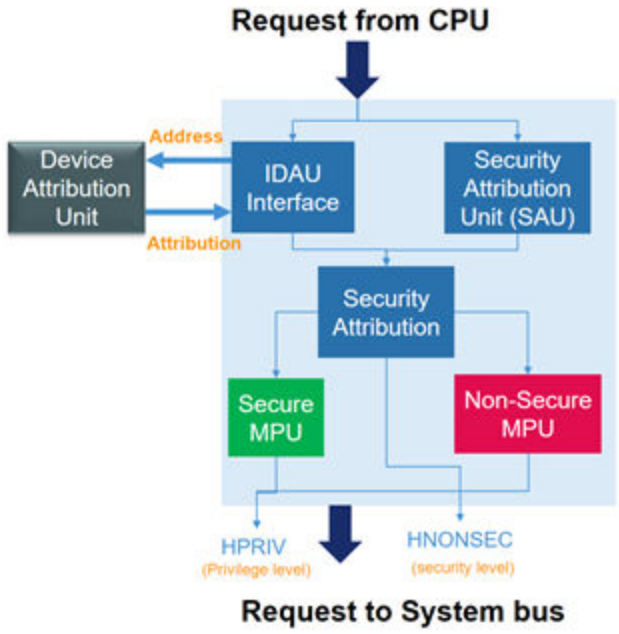

Figure 8. Security Attribution Units

NXP incorporated an Implementation specific Device Attribution Unit (IDAU) allowing a secure OS to be completely decoupled from the application.

2.4.7 LPC55Sxx IDAU and Security Attribution Unit Operation

The IDAU is a simple design using address bit 28 to allow aliasing of the memories in two locations. If address bit 28 is = 0 the memory is Non-Secure. If address bit 28 = 1 the memory is Secure.

The SAU allows 8 memory regions and allow the user to override the IDAU's fixed Map, to define the non-secure regions. By default, all memory is set to secure. At least one ASU descriptor should be used to make IDAU effective. If either IDAU or SAU marks a region, then that region is secure. NSC area can be defined in NS region of the IDAU.

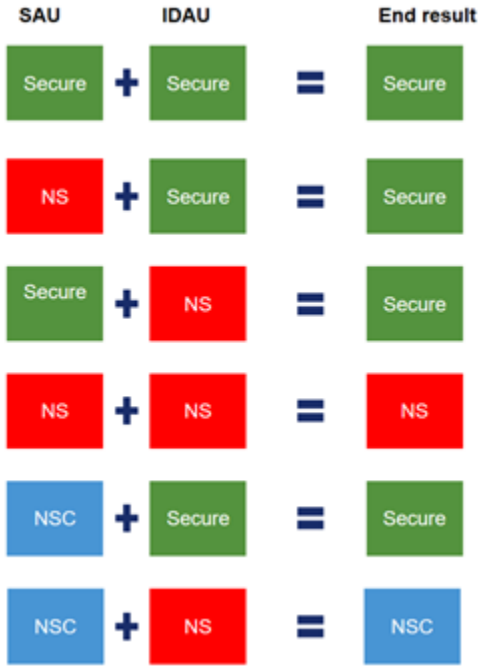

Figure 9. Security Attribution Logic

2.4.8 Memory Protection Checkers (MPC)

The MPC is used with all memory devices, on-chip Flash and SRAM. Memory blocks have one checker setting per 'sector' where typically, memory is divided into 32 sectors. For example, a 128KB memory would have a granularity of 4 kB per sector.

All rules are set in the Secure Control register bank. A user must have the highest level "Secure Privileged" to set rules. The Privilege level is ignored if left in default states. By default, only the security level is checked.

2.4.9 Master Security Wrapper (MSW)

The MSW wraps three types of bus masters, TrustZone aware Cortex M33 with security extension, simple masters such as DMA, SDHC, USB-FS and Hash-Crypt and smart masters such as micro-CM33 and EZH.

2.4.10 Security Locking

The secure bus controller allows locking of the following configurations:

- All PPC & MPC checkers settings
- All master security level (MSW) settings
- SAU Settings
- Secure MPU settings
- Secure Vector offset address (S_VTOR)
- Non-secure MPU settings
- Non-secure Vector offset address (NS_VTOR)
- Core1-CM33 MPU settings

- Core1-CM33 Vector offset address (VTOR)

The boot ROM provides the user an option to specify above setting through image header and lock it before control is passed to application code. Since ROM is immutable code, highly secure application could use this option.

2.4.11 Security Error Log

When a Memory Protection Checker (MPC) detects a security violation a secure violation interrupt is raised if a violation occurs on data or instruction access core0. Exception detail can be recorded readable flags in the Secure AHB Controller Module and Secure Control registers. The following is available to be logged:

- Security violation address for each AHB slave port/layer.
- Status bit indicating on which AHB slave port/layer violation has occurred.
- Master security level, privilege level and access type (code/data).

Core0 will switch to secure mode to handle the exception.

2.4.12 Hypervisor Interrupt

The LPC Hardware implements a feature whereby Non-Secure access of the Secure AHB Controller Module. Access to the secure AHB controller module is restricted to the Secure-Privilege level. AHB controller will raise a secure violation interrupt which can be configured to be secure and thus allow Non-Secure code to raise a Secure interrupt. This is used as the call to the Hypervisor.

ARMv8-M Supervisor call is banked and can therefore exist in Secure mode and a separate supervisor handler exists for Non-Secure. Using the SVC (Supervisor Call opcode) does not allow non-secure code to call the hypervisor since the level reached would be Non-Secure Privileged, whereas Secure-Privileged is needed.

2.5 Cryptography Accelerators on LPC55Sxx

To address the challenges of Asymmetric cryptography NXP has developed CASPER, a Cryptographic Accelerator and Signaling Processing Engine with RAM-sharing. It is a hardware accelerator engine for certain asymmetric cryptographic algorithms, such as, Elliptic Curve Cryptography (ECC). CASPER sits on the Cortex-M33 co-processor buses.

Other cryptography accelerators include Hash-Crypt engine (AES and SHA), the PRINCE on-the-fly flash encryption/decryption engine, and the TRNG.

Figure 10. AES and SHA Cryptographic Accelerators

2.5.1 SHA & AES Engine

The SHA block can support SHA1 and SHA2-256 algorithms. The AES block supports symmetric cryptographic using AES- ECB, AES- CBC and AES-CTR modes with side channel protection for key wrapping. The modules can be uses with direct writes from the bus or DMA writes.

2.5.1.1 AES engine

The LPC55Sxx devices provide an on-chip hardware AES encryption and decryption engine for data encryption or decryption, data integrity, and proof of origin. Data can be encrypted or decrypted by the AES engine using a key from the PUF or a software supplied key. The AES engine supports 128 bit, 192 bit, or 256 bit key in Electronic Code Book (ECB) mode, Cipher Block Chaining (CBC) mode, or Counter (CTR) mode. The AES engine supports 128-bit key in ICB (Indexed Code Book) mode, that offers increased protection against side-channel attacks.

2.5.1.2 ICB-AES

Whereas AES as a block cipher is an Electronic Code Book, a special hybrid cipher mode is available, called ICB. ICB is a form of CTR cipher mode, but its purpose is to be Side Channel Analysis resistant. It uses a method which is a counter-measure to various Side Channel attacks such as SPA/DPA/DPX (power analysis) and emanation analysis. ICB is slower than normal AES ECB and CTR mode, as a consequence of being SCA resistant. This mode can be used for extra-secure on-chip storage for sensitive information.

2.5.1.3 SHA

The LPC55Sxx devices provide on-chip Hash support to perform SHA-1 and SHA-2 with 256-bit digest (SHA-256). Hashing is a way to reduce arbitrarily large messages or code images to a relatively small fixed size “unique” number called a digest. The SHA-1 Hash produces a 160 bit digest (five words), and the SHA-256 hash produces a 256 bit digest (eight words).

Hashing is used for four primary purposes:

- Core of a digital signature model, including certificates, for example and for secure update.
- Support a challenge/response or to validate a message when used with a hash-based Message Authentication Code (HMAC).
- In a secure boot model, which verifies code integrity.
- Verify external memory has not been compromised.

2.5.2 TRNG

The TRNG module is a hardware accelerator module that generate 256-bit entropy. The purpose of the module is to generate high quality, cryptographically secure, random data. Random number generators are used for data masking, cryptographic, modeling and simulation application which employ keys that must be generated in a random fashion.

LPC55S6x embeds a hardware IP (combined with appropriate software and the writing of a stochastic model) that can be used to generate true random numbers with high levels of quality (FIPS140-2, AIS31, P2/PTG.3).

Using less sophisticated software, target for the RNG feature should be lowered to AIS31, P2/PTG.2 or even AIS31, P1/PTG.1. The difference being the way entropy sources are activated and monitored.

2.5.3 Hash-based Message Authentication Code (HMAC)

An HMAC can be achieved on LPC55Sxx using a pre-shared key and hashing using the SHA256 engine. It is a way of verifying a message (encrypted or not) and can also be used for challenge or response. Both sides must have a pre-shared key that is just a shared secret value and not an encryption key.

HMACs are significantly faster than signatures, but work only with pre-shared keys, which must not be leaked or lost (unlike a public key). The HMAC key can be shared dynamically using trust models like Diffie-Hellman or maybe a board-unique key shared by two devices.

2.5.4 PRINCE real-time encryption/decryption

LPC55Sxx devices offer support for real-time encryption and decryption for on-chip flash using the PRINCE encryption algorithm. Compared to AES, PRINCE is fast because it can decrypt and encrypt without adding extra latency. PRINCE operates as data is read or written, without the need to first store data in RAM and then encrypt or decrypt to another space. It operates on a block size of 64 bits with a 128-bit key.

This functionality is useful for asset protection, such as securing application code, securing stored keys, and enabling secure flash update.

2.5.5 CASPER - Cryptographic Accelerators

CASPER Crypto co-processor is provided to enable hardware acceleration for various functions required for certain asymmetric cryptographic algorithms, such as, Elliptic Curve Cryptography (ECC).

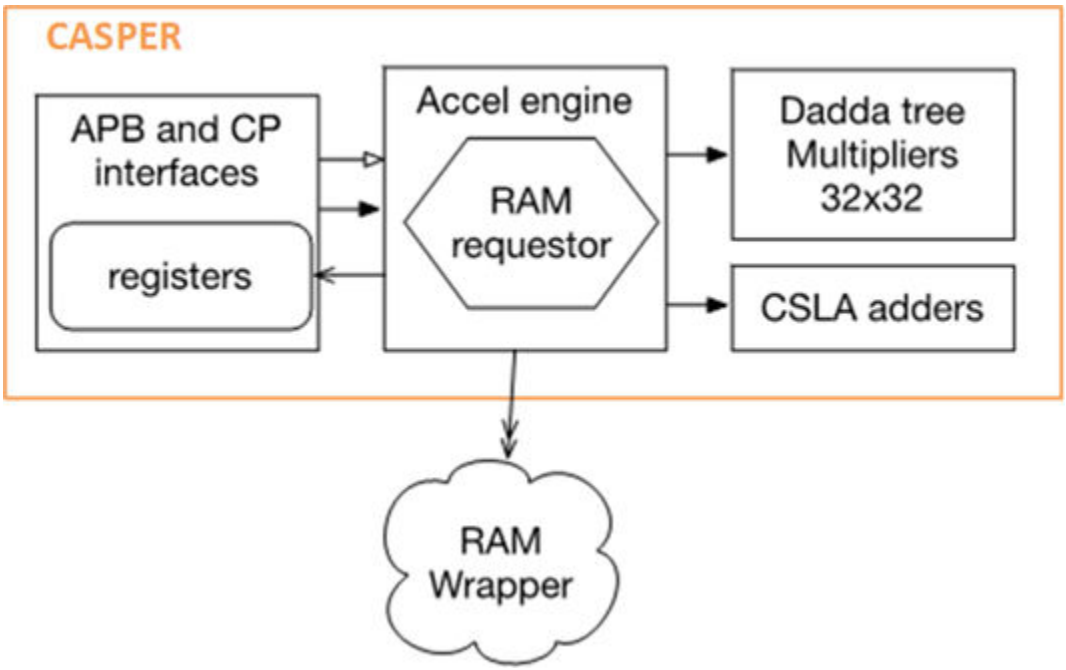

Figure 11. CASPER

- Co-proc interface to ARMv8-M Cortex-M33
  - 64b data-bus
  - Offloads the CPU: One write to kickoff accelerator and possibility to kick off a sequence
- Fast shared RAM access
  - 2x 32b RAMs allocated for CASPER RAM interface allowing 64bit parallel access
  - RAM is shared with System Memory
- A bank of Adder and registers to allow MAC (multiply and accumulate)
- Two 32x32 Multipliers
- Side channel protection
  - Using Random masking

NXP developed a SW crypto library that maps standard bignum library functions to CASPER hardware capabilities.

2.6 LPC55Sxx Secure Storage PUF, PFR and UUID

The LPC55Sxx secure storage features can be used to implement the counterfeit protection security goal.

The hardware capabilities related to the storage and generation of chip specific secrets are included in the following blocks

- Universally Unique Identifier (UUID)
- Physically Unclonable Function (PUF)
- Customer Manufacturer Programmable Area (CMPA)

This chip provides options for using NXP programmed unique ID, PFR, or PUF to generate keys. The ROM uses this information, along with cryptographic accelerators to apply security services to the secure booting, debugging and PRINCE configurations for the chip.

2.6.1 Protected Flash Region

The protected Flash Region (PFR) consists of customer programmable areas and the factory programed areas. The Customer Manufacturing Programmable Area (CMPA) is used to store the boot configuration, the RoT key table hash, the debug authentication configuration and the PRINCE configuration. The PRF regions can be programmed with the ROM API using BLHost commands. BLHOST commands are executed by a host device like a PC or host MCU to interact with the ROM API in ISP mode. Please refer to Boot Rom chapter in the Users Manual for details on the commands. Figure 12 is an excerpt of the status and control bits stored in the CMPA.

The Customer Field Programmable Area (CFPA) contains version control counters, RoT (RKTH) Key revocation, and PRINCE region IV codes. The CFPA Ping and Pong pages are used for selecting the CFPA page with the highest version number. There are 17 revocations ID's possible. The NXP Manufacturing Programmed Area (NMPA) is where the user can find the UUID (universal unique identification ID) which can be used to uniquely identify the device.

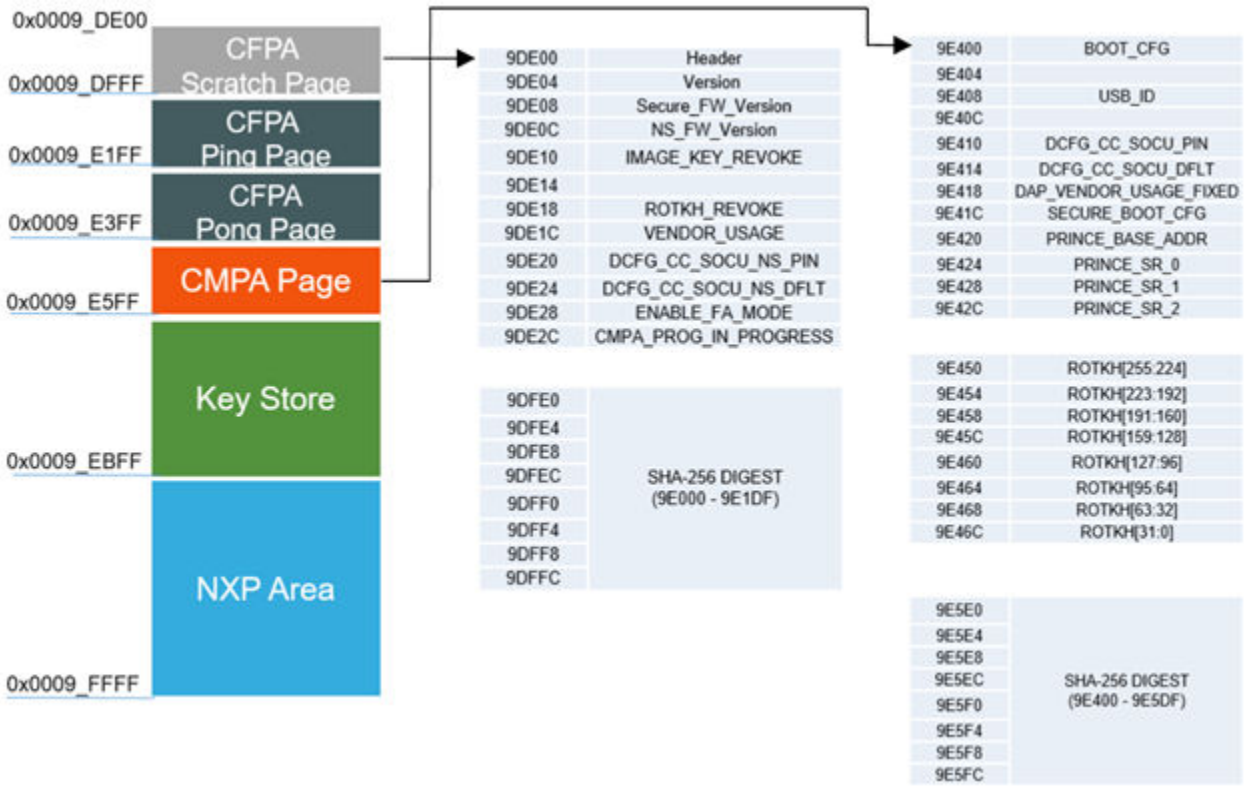

Figure 12. PFR – Protected Flash Region

2.6.2 UUID Unique ID

LPC55Sxx store 128-bit IETF RFC4122 compliant non-sequential, cryptographically distributed IDs with a 128-bit UUID per device. The hardware of the LPC55Sxx provides support to the unique identity protects against the possibility of counterfeit devices or clones which could inject untrusted data into the network among other things.

2.6.3 PUF Features and Usage

This family of microcontrollers include a Physically Unclonable Function (PUF) controller based on SRAM that enables the secure generation of a unique device fingerprint and device-unique cryptographic keys. The SRAM PUF mechanism is tightly integrated into the LPC55S00 family enabling keys from the PUF to be directly used by the device's internal Advanced Encryption Standard (AES)-256 encryption engine. The unique and unclonable keys provide significant security benefits over other means of key injection or storage. The PUF keys reduce the threat of "break once repeat everywhere" attacks by basing the foundation of the security on device-unique unclonable keys.

The SRAM PUF hardware constructs a root key used as a Key Encryption Key (KEK) to protect other user keys. The PUF generates a device-unique 256-bit KEK using the digital fingerprint of a device derived from uninitialized SRAM and error correction data called the Activation Code (AC). The AC is a 1192-byte long data chunk generated during an “enrollment” process which takes place during the provisioning/personalization of the device. Each time PUF *Enroll* is performed, a different AC is generated resulting in a different digital fingerprint as well. The AC must be stored in the Customer Field Programmable Area (CFPA) to allow subsequent booting to reproduce the PUF key. The encrypted keys (key codes) can then be stored in the MCU.

The keys may be symmetric secret keys or asymmetric public/private key pairs. For example, the application software may use the key generation interfaces to create an elliptic curve public key pair for use by the TLS protocol. This enables the strong storage of per-device keys that may be used to remotely authenticate a device.

A system may have additional secrets that need to be configured into the device. Such secrets may use a PUF generated key to locally encrypt and store this information. This ‘wrapped’ information may be securely stored in internal flash memory since it is encrypted.

The LPC55Sxx PUF hardware routes the KEK to the AES and PRINCE encryption engines to support the decryption and authentication of internal flash during device boot. On each boot the ROM uses the KEK to authenticate and decrypt the firmware. In these operations, neither the device key or the firmware is available unencrypted external to the device.

The PUF KEK, index 0, never leaves the device. The index zero key is only routed via an internal path to the AES and PRINCE blocks, while indexes 1-14 can be used to perform encryption and decryption of internal flash memory and during the boot process.

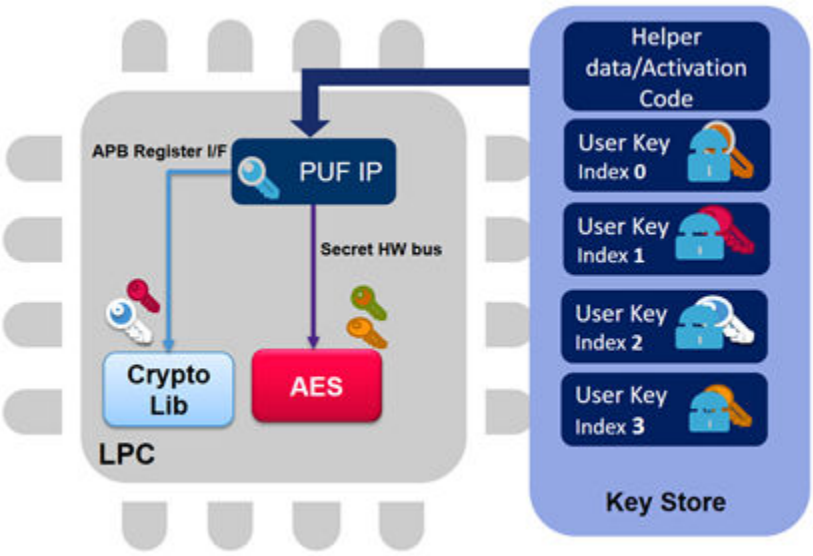

Figure 13. PUF Key Storage

2.6.4 Provisioning of PUF KEK

A product is personalized by injecting and running an “enrollment image” into a device using BLHOST commands. This code is transient and only runs once. *LPC55Sxx Usage of the PUF and Hash Crypt to AES Coding* (document AN12324) describes how to securely generate, store, and retrieve user keys using the root key by providing a step-by-step guide through the provisioning process.

2.7 Secure Peripherals

2.7.1 GPIO Masking

On LPC55Sxx, all digital pins states are readable through GPIO controller registers. Information can leak through pins connected to peripheral configured as secure peripheral. For example, I2C pins connected to a secure element. Secure software sensitive

to such leakage should mask the corresponding pins using SEC_GPIO_MASK0/1/2/3 registers in the secure AHB controller Module. Example code in the SDK demonstrates use of the secure AHB controller to secure the GPIO pins.

2.7.2 Secure Interrupt Masking

LPC55S69 has two CPUs. Core0 has TrustZone and core1 does not have TrustZone. On LPC55S69 both CPUs have access to all interrupts. To safeguard secure application, interrupt masking feature is implemented on LPC55S6x. Any interrupt to core1 can be masked out by programming the mask register in the secure AHB Controller module. Core0 has internal programmability to configure any interrupt as secure interrupt, making it visible to NVIC_S only and mask it from NVIC_NS.

2.7.3 Secure Peripherals

There are two other peripherals considered secure, the Secure DMA and the secure GPIO. There are two DMA controllers that can be configured as secure and the other as non-secure. It is suggested to use the DMA controller with 8 channels as the secure DMA. All 32 Port0 pins have Secure GPIO as selectable pin-mux function. The functionality is the same as the standard GPIO controller.

3 Uses of security blocks

The LPC55Sxx series of MCUs has the security features to counter local and remote attack types. During the design of your IoT device you will be faced with question about what feature to use to accomplish your goal. Provided in the following tables is a summary of the security goals versus which block on the LPC55Sxx MCU is utilized, which block to use for the typical security algorithms and what each block is intended to protect.

3.1 Align the Security Goals to the Security blocks

Table 3 aligns the Security Goals to the Security blocks that are available on LPC55Sxx.

Table 3. LPC55S00 Security Blocks to address Security Goals

Security Goal	Security Blocks on LPC55Sxx
Counterfeit Protection	Unique ID Chip unique Root Key (PUF) PRINCE – Encrypted flash
Secure Trust provisioning to the device	During manufacturing –Secure boot using encrypted flash image and generating PUF keys.
Secure Communication	Secure trusted software using hardware accelerators and communication protocols to establish and maintain authenticate communication. Hardware includes: CASPER - asymmetric cryptography accelerator PUF public keys through AES AES 256 Engine SHA Hashing Engine (SHA1 & SHA2) TRNG
Data Confidentiality	Maintain authenticated communication transmit encrypted data from and to the IoT device. PUF (key store), AES-256 Engine, CASPER

Table continues on the next page...

Table 3. LPC55S00 Security Blocks to address Security Goals (continued)

Security Goal	Security Blocks on LPC55Sxx
Firmware Update	Software API to provide OTA updates once secure boot accomplished by the ROM API with Secure Binary (SB) file processing. PRINCE encrypt and decrypt to protect secure firmware.
System Integrity	Secure Boot, PRINCE, Run-time protection with TrustZone for ARMV8M, Pole/anti-pole checks
Onboarding	Flexible provisions with PUF keys, RoT Public Key Hash in customer configuration region for trust enablement.

3.2 Align the security algorithms to the accelerator blocks

Table 4 aligns the security algorithms to the accelerator blocks that are available.

Table 4. Functions with the algorithms and the available MCU accelerator

Function	Algorithms	Accelerator
Symmetric crypto	AES (ECB, CBC, CTR)	**HashCrypt** 128, 192 & 256 bit keys
Symmetric crypto	PRINCE (CTR)	**PRINCE**
Asymmetric crypto	RSA, ECC	**CASPER**
Hash	SHA1, SHA2-256	**HashCrypt**
MAC	HMAC, CMAC	**HashCrypt + SW**
Signature	HASH (SHA256) + Asymmetric crypto (RSA, ECDSA)	**HashCrypt + CASPER**

Software must be developed to drive the hardware. The LPC55Sxx SDK is available to help you get started securing your products. In Table 3 we align the different Security blocks to ways software would use these blocks. Please refer to the MCUXpresso SDK available for the LPC55Sxx for drivers, demos, RTOS, and examples of software/hardware implementations for the security blocks.

3.3 What LPC55Sxx blocks protect

Table 5. What these blocks are used to protect

Security Blocks on LPC55Sxx	Protection Domain	Examples
SHA Engine (SHA1, SHA2), AES 256 Engine TRNG, CASPER	Communication	Cryptography, Signature validation
SRAM PUF, AES-256 Engine, CASPER	Data	Secrets, keys, personal information
PRINCE (Encrypted flash)	Firmware	IP theft, reverse engineering
UUID (RFC4122), Chip unique Root Key(PUF)	Operational integrity	Maintaining service and revenue

Table continues on the next page...

Table 5. What these blocks are used to protect (continued)

Pole/anti-pole checks, SRAM PUF,Secure boot, Secure boot	Anti-tamper	Physical attacks, keys not stored but generated.
Secure boot ROM, PFR (For RoTKH),	ROT	Secure boot ROM, PFR (For RoTKH)

4 LPC55Sxx features address security goals

In this section we dive deeper into how the security features of the LPC55Sxx can be used to address the needs of each of the identified goals. Within each sub-section we discuss one of the six security goals establishing the place in the life-cycle that the security goal is important then describe the LPC55Sxx security feature that you could use to support that goal. The first time a security feature is referenced is where the detail will be. In subsequent security goal sub-sections the feature covered in previous sections may be suggested. They were:

1. Counterfeit protection.
2. Onboarding.
3. System integrity.
4. Secure communication.
5. Ensuring data confidentiality and integrity.
6. Updating firmware/software when vulnerabilities are discovered.

4.1 Counterfeit protection

The LPC55Sxx secure storage features can be used to implement the counterfeit protection security goal. The SoC has a Unique ID, a chip unique Root Key, a secure boot, and flash encryption/decryption on the fly.

This protection is needed in all stages of the product life cycle, from manufacturing to deploy to use to maintain to decommissioning.

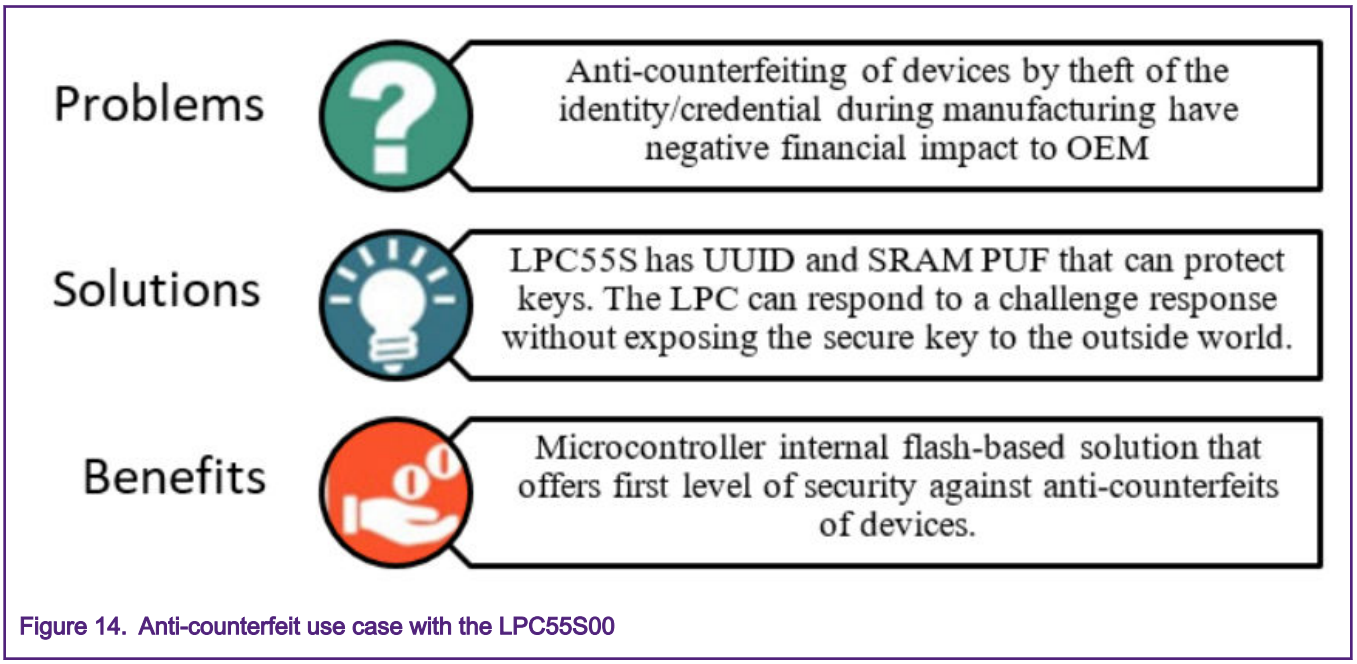

Figure 14. Anti-counterfeit use case with the LPC55S00

Table 6. Required actions for anti-counterfeit of LPC55Sxx

#	Action	Can be performed by
1	Creation of root and public/private key pair via SRAM PUF.	• OEM. • In the field after power-up.
2	Private key never stores, nor leaves the device.	—
3	Public key signed and stored as a certificate.	• OEM. • In the field after power-up.
4	Respond to a challenge response of an external source.	Any IC/device that can communicate with the LPC cloud provider and/or OEM.

4.1.1 Establish a unique identity and authentication that is difficult to reproduce by an attacker

The basis for the RoT is the LPC55Sxx SRAM PUF. Each thing gets their own unique identity based on unclonable hardware silicon footprint.

4.2 Onboarding

There is a normal process of placing a Thing into use in its target environment, that means to bring a new device onboard the trusted environment. For example, if you had 1000 light bulbs that you needed to install in your office, you would need to onboard the new bulbs to the lighting control system. A bulb, controlled by the LPC55Sxx, would have a unique identity and encrypted private and public keys. You could commission the bulb to only work in a secure environment.

This protection is needed in the product after the development completes, from manufacturing to deploy to use to maintain to decommissioning.

4.2.1 Digital signatures

A digital signature combines public/private keys such as RSA or ECC with SHA Hashing. The advantage of the signature model is that the public key and the signature can be public. Therefore, the signature can be stored in a non-secure location along with the image and then verified using a copy of the public key.

4.2.2 Enforcing privilege levels

LPC55Sxx allows flexible provisions with PUF keys. It has PUF-based PKI, RoT Public Key Hash in customer configuration region for trust enablement.

4.2.3 Protecting shared credentials between end device and back-end system

The boot security as well as the asymmetric cryptography of the LPC55Sxx protects the shared credentials between the end device and the back-end so that the device identity is extended to the back-end based services which are linked to the device.

4.2.4 Protecting symmetric and private keys with hardware

PUF keys are generated, not readable since they are not stored in a fuse or non-volatile memory space.

The LPC55Sxx access via the debug interface can be set to authenticate debug or to completely lock debug out. The ROM boot can securely check the customer configuration space and the image integrity prior to running the embedded code stored in the on-chip flash.

4.3 System integrity

This protection is needed in every stage of the product life cycle, from procure to develop, to manufacturing, to deploy, to use, to maintain, to decommissioning.

4.3.1 Protecting physical and logical conditions from intrusions

The LPC55Sxx includes basic tamper resistance using two complementary bits in different registers for all secure selections. The SRAM PUF generates keys on power up. The keys are not present when the device is powered off or visible upon visual inspection. There is a secure bus between the PUF and the AES and PRINCE engines for credential injection mechanism.

4.3.2 Enforcing trust with functionality provided by MCU

The LPC55S6x and LPC55S1x TrustZone and secure hardware insures the integrity of the system, trust in its operation, and it is maintained over the lifetime of the device.

4.3.3 Encrypting sensitive software functions to prevent reverse engineering

The image in the flash can be encrypted using PRINCE. During the execution, the code is decrypted on the fly.

4.4 Secure communication

This protection is needed in the stages of the product life cycle from deploy to use to maintain to decommissioning. The Thing would establish communications with an asymmetric cypher then once secure communications have been established exchange data or updates using the faster symmetric encryption.

4.4.1 Maintain audit logs that are kept encrypted

Customer IP should log all communication attempts, successful or not.

4.5 Data confidentiality and integrity

This protection is needed in the stages of the product life cycle from deploy to use to maintain to decommissioning.

This security goal is achieved with the implementation of the secure communication goal. This way the data identified by the security policy is kept confidential while it is handled by the device. In addition, local control of secure peripherals can be maintained.

4.6 Secure Remote Firmware Update

Whenever changes are needed for the functionality of the device, whether to address security concerns or enhancing product features, the secure remote firmware update must be done. A secure boot is the first step in this process.

The LPC55Sxx allows booting of signed images. The secure boot ROM supports three types of security protected modes to allow a scalable security and manufacturing options for end products that use the LPC55Sxx. The ROM supports booting from encrypted Prince regions. Encrypting images allows protection of intellectual property. The ROM supports public keys and image revocation – that is, the method of not allowing new updates to be applied unless they are of a specific version. This is the basis for roll back protection. The ROM supports pre-configuration of TrustZone-M settings. Also, as previously discussed, DICE is enabled by the ROM, as well as redundant image support (fall back image).

The LPC55S6xx secure boot:

- Uses RSASSA-PKCS1-v1_5 signature of SHA256 digest as cryptographic signature verification.
- Supports RSA-2048 bit public keys (2048 bit modulus, 32-bit exponent).
- Supports RSA-4096 bit public keys (4096 bit modulus, 32-bit exponent).
- Uses x509 certificate format to validate image public keys.

- Supports up to four revocable Root of Trust (RoT) or Certificate Authority keys, Root of Trust establishment by storing the SHA-256 hash digest of the hashes of four RoT public keys in protected flash region (PFR).
- Supports anti-rollback feature using image key revocation and supports up to 16 image key certificates revocations using Serial Number field in x509 certificate.
- Supports image fall back option if flash image is not secure. Image restoration in the ISP mode uses an external SPI NOR flash to bootstrap the MCU.

LPC55Sxx offers two stages of flash programing. First stage is development and deployment. At this stage, JTAG or SWD ports are used for flash erase and programing. However, once device is deployed this mode of flash programing is disabled. Thereafter, only alternative to program flash is over the air via secure boot ROM.

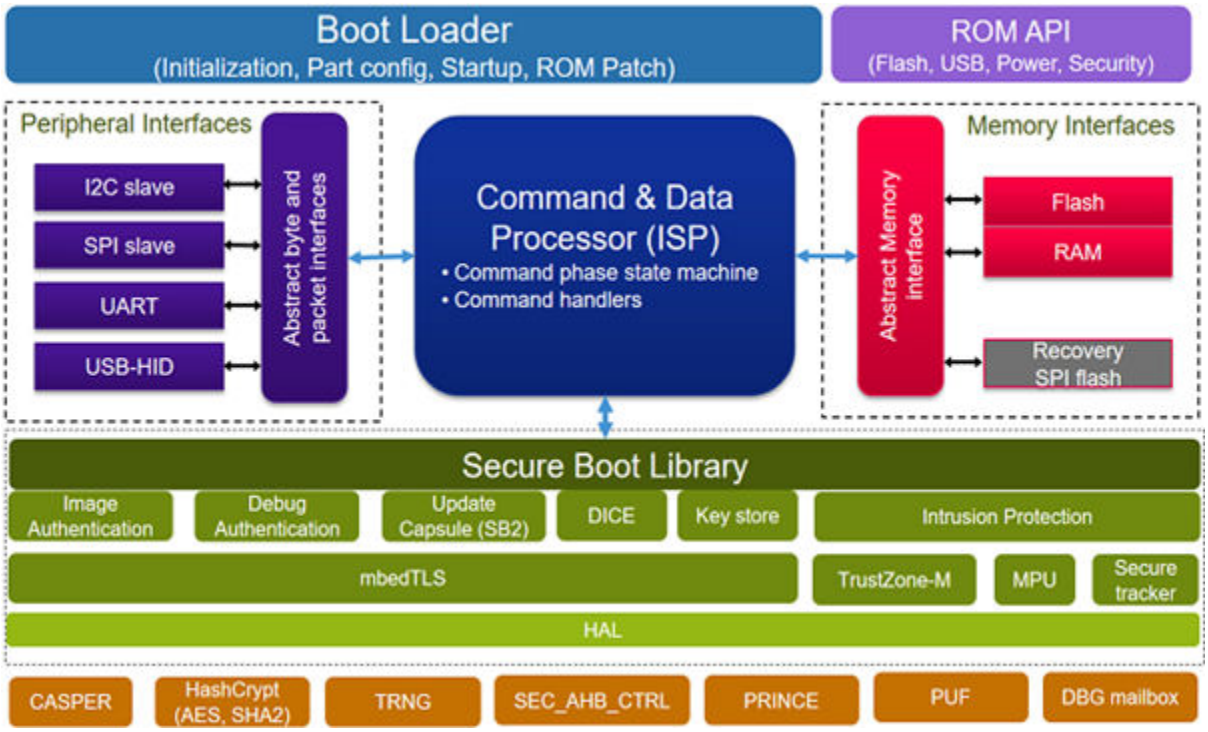

Figure 15. LPC55Sxx Boot ROM Block Diagram

Image validation is a two-step process. The first step is the validation of the certificate embedded in the image. This contains the image public key which is used in the second step to validate the entire image signature, including the certificate, to allow customers to add additional PKI structure. The ROM validates and extracts the Image public key from x509 certificate embedded in the image using SHA-256 with RSA signing for image authentication. RSA key size is determined 'RSA4K' field in SECURE_BOT_CFG control word. Default is 2048-bit (public key modulus size). If flag RSA4K is set, then 4096-bit keys are enforced.

Inside the x509 certificate, there is a field (Serial Number) that must be compared with the keys from the PUF and PFR. If serial number (which can be trusted because it is signed) does not align with the PFR, the image validation process will fail. If serial number is equal the firmware can allow the firmware update. The serial number can be a later version (higher by 1 version only) but it cannot be an earlier version. It can jump forward but cannot roll back.

5 Conclusion

5.1 Establish a trusted supply chain

NXP provides you will a comprehensive set of tools to take you through each of the life cycle stages included the MCUXpresso IDE, Config and Clock tools and Elf2SB GUI. NXP provides a robust set of software drivers, example projects, applications notes,

and training to assist you. There are demo projects in the SDK that demonstrate the drivers for each of the secure hardware in the LPC55Sxx as well as demonstrations of secure communication with the AWS server.

NXP has partners that can perform penetration tests, certify Crypto and TRNG hardware, can provide device and user management in the cloud, bulk programming and cloud data management.

5.2 Go Create

With the LPC55Sxx, software, tools and methods you can create Things that support secure transactions over the lifecycle of the device. With the advent of MCUs like the LPC55Sxx the security requirements for thing to cloud can be realized.

The LPC55Sx MCU series leverages the Arm Cortex-M33 technology, combining significant product architecture enhancements and greater integration over previous generations; offering power consumption improvements using the accelerators and advanced security features including PUF based ROT and provisioning, real-time execution from encrypted images and asset protection with Arm TrustZone-M (TZ-M).

As detailed in this application note the security goals are addressed with the functionality integrated into the LPC55Sxx. NXP provides this SoC to let you to create that low cost, low power IOT device that can be trusted for the entire life of the product.

6 Resources

Here are the resources needed to further investigate the essential secure features and implementation on the LPC55Sxx MCUs:

1. NXP.com LPC5500 MCU Series

   www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/lpc-cortex-m-mcus/lpc5500-series-cortex-m33-mcus:LPC5500_SERIES

2. LPC55S6x MCU Family Fact Sheet

   www.nxp.com/docs/en/fact-sheet/LPC55S6XFS.PDF

3. *LPC55Sxx Secure Boot* (document AN12283)

   MCUXpresso Software Development Kit (SDK)

4. Intrinsic ID White Papers on IoT Security and PUF

   www.intrinsic-id.com/resources/white-papers/

5. Hardware evaluation kit

   LPCXPRESSO55S69 DEVELOPMENT BOARD (LPC55S69-EVK)

   www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc5500-cortex-m33/lpcxpresso55s69-development-board:LPC55S69-EVK

6. Code Signing Tool

   elf2USB GUI - tool documentation is in the LPC55S69 SDK, in the *.../LPCXpresso55S69/middleware/mcu-boot/doc* folder.

7. LPC5500 MCU Series: Securing the edge with worlds first ARM Cortex M33 MCU

   www.nxp.com/video/:NXP-LPC5500-VIDEO

8. IoT Solutions -Arm Special Edition by Lawrence C. Miller

9. *LPC55Sxx Usage of the PUF and Hash Crypt to AES Coding* (document AN12324) - this application note describes how to securely generate, store, and retrieve user keys using the root key.

10. Advance your IoT Security Leveraging Hardware Protected Keys on Microcontrollers

    www.nxp.com/design/training/advance-your-iot-security-leveraging-hardware-protected-keys-on-microcontrollers:TIP-ADVANCE-YOUR-IOT-SECURITY

11. ARM+NXP Webinars on LPC5500 - https://pages.arm.com/webinar-recording-achieving-secure-execution-environments-typ.html?aliId=eyJpIjoiOVVaTjEwY2tuRXNSQ3czUSIsInQiOiJIeVwveVQrbjNkUFRtandKK1NkYktKQT09In0%3D

12. LPC55xx Secure GPIO and usage - www.nxp.com/doc/AN12326

13. Element14 Securing IoT Sensors with NXP LPC5500 MCUs and Trustzone Block - https://event.on24.com/eventRegistration/

14. DICE - Trusted Computing Group Specification - www.trustedcomputinggroup.org/wp-content/uploads/Device-Identifier-Composition-Engine-Rev69_Public-Review.pdf. Microsoft developed a set of DICE package code, if you want to know more about its usage, you can check here: www.microsoft.com/en-us/research/project/dice-device-identifier-composition-engine/

7 Revision history

Table 7 summarizes the changes done to this document since the initial release.

Table 7. Revision history

Revision number	Date	Substantive changes
0	02/2019	Initial release
1	05/2020	Included security information on LPC55S2x and LPC55S1x MCUs. Add new table of security features. Included additional resources.

A Glossary

AC—Activation Code for PUF.

AES—Advanced Encryption Standard- The AES engine supports 128 bit, 192 bit, or 256 bit keys for encryption and decryption operations.

API—Application Processor Interface

CASPER—Cryptographic Accelerator and Signaling Processing Engine with RAM co-processor engine

CRC—Cyclical Redundancy Check

CRYPT—Cryptography

CBC—Cypher Block Chaining

CDI - Compound Device Identifier - an identity value that is derived from a Unique Device Secret and the identity of the first mutable code used to identify the software running on a system that was used to generate this data.

CFPA—Custom Field Programmable Area

CMPA—Customer Manufacturing/Factory Programmable Area

CTR—Counter encryption method

DICE - Device Identifier Composition Engine (DICE) - the DICE is immutable and creates the CDI

DMA—Direct Memory Access, one DMA can be made secue

DSP—Digital Signal processor

ECB—Electric Code book

ECC—Elliptic Curve Cryptograph

FIPS— Federal Information Processing Standard, Publication 197

HASH—Algorithm that is a summary of the image used to authenticate a file

HMAC—HMAC(K,x) - HMAC-SHA256 algorithm applied to octet string x using key K.

ICB - I Code book

IoT—Internet of Things

IP—Intellectual property

IV—Initialization Vector – used with AES- CBC mode

NMPA - NXP Manufacturing Programmable Area

NS—Non-secure

NVIC—Nested Vectored Interrupt Controller. There is one NVIC for each CPU.

M2M—Machine to Machine, usually a communication

MCU—Microcontroller Unit

MITM—Man in the Middle. An attack that attempts to get credentials from IoT

MPC—Memory Protection Checker – ROM and each RAM bank have assolicated MPC. The Flash has one MPC

Onboarding—A process of provisioning a client with credentials for accessing a network resource and assigning appropriate permissions.

OTA—Over The Air, referring to updating firmware from a remote network

PFR—Protected Flash Region

PKI—Private Key Interface

PRINCE—Auto Encrypt/Decryption block for on-chip flash using “Prince” method

PUF—Physically Unclonable Function

ROM—Read Only Memory

RSA—Public key cryptography encryption algorithm

RTF - Run Time Fingerprint

RoT—Root of Trust

RoTKH—Root of Trust Key Hash

S—Secure

SAU—Secure Attribution Unit—NXP security block on the LPC55Sxx

SHA—An Encryption method, SHA2 and SHA256 are other methods

SoC—Silicon on Chip, refers the the Microcontroller or microprocessor

TEE—Trusted Execution Environment

Thing—A device that joins other devices in the Internet of things such as Home Automation, Fitness Wearables, Smart Meters, Building access, Sensors

TPM - Trusted Platform Module

TRNG—True Random Number Generator

TZ-M—TrustZone for Cortex M based on ARM V8M specification

UDS - Unique Device Secret - the UDS is known only to the manufacturer and the DICE, and is used in the creation of the CDI by the DICE

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 05/2020

Document identifier: AN12278

arm