AN12385

Building a Secure System using NXP Secure MCU LPC54S0xx

Rev. 0 — 15 March 2019 Application Note

Contents

1 Introduction.....1
1.1 What is security.....1
1.2 Fundamental security principle.....1
1.3 Threats and general protection principles.....1
1.4 Limitations of security solutions.....3

2 LPC54S0xx Secure Architecture.....4
2.1 AES engine.....4
2.2 SHA.....4
2.3 RNG.....5
2.4 OTP.....5
2.5 PUF(Physical Unclonable Function).....6
2.6 RSA API.....7

3 LPC54S0xx Secure Boot.....7
3.1 The types of Secure boot Image.....8
3.2 Secure boot process.....12
3.3 Boot Keys storage.....14
3.4 Device Identifier Composition Engine(DICE)...14

4 Conclusion.....14

1 Introduction

The chapter describes basic knowledge about security in embedded systems before introducing secure MCU LPC54Sxx.

1.1 What is security

The term *security* in this document means is the protection of computer system from theft or damage to hardware, software or electronic data. It also includes disruption or misdirection of the services. The field is of growing importance due to increasing reliance on computer systems, internet, and wireless networks such as Bluetooth and WiFi, and due to the growth of “smart” devices.

1.2 Fundamental security principle

Confidentiality, integrity, and availability, also known as the CIA triad, is a model designed to guide policies for security. The three key principles should be guaranteed in any kind of secure system. If any one of three is breached it can have serious consequences for the concerned parties.

Confidentiality

Confidentiality is the ability to hide information from unauthorized access. It is perhaps the most obvious aspect of the CIA triad when it comes to security; but correspondingly, it is also the one which is attacked most often. Cryptography and Encryption methods are the examples of an attempt to ensure confidentiality of data transferred from one computer to another.

Integrity

The ability to ensure that data is an accurate and unchanged representation of the original secure information. One type of security attack is to intercept some important data and make changes to it before sending it on to the intended receiver.

Availability

It is important to ensure that the information required is readily accessible to the authorized viewer whenever required. Some types of security attack attempt to deny access to the appropriate user, either for the sake of inconveniencing them, or because there is some secondary effect. For example, by breaking the website for a particular search engine, a rival may become popular.

1.3 Threats and general protection principles

We can describe threats from attack type and location based. Attack type includes physical attacks and logical attacks.

Physical attacks: attackers exploit vulnerabilities in a device either through direct manipulation or by observing its operation.

Logical attacks: attackers only rely on messages sent to the device to cause damage.

location based includes local attacks and remote attacks.

Local attacks: require physical access to the device meaning they are not scalable and the attacker needs more skills. However, it is possible that through a local attack the entire code of a product can be obtained. Analyzing that code may lead to the discovery of device vulnerabilities which may be applied to similar devices through remote attacks.

Remote attacks: can be carried out by sending commands over a network connection. physical presence of the attacker near the target device is not required. Due to their scalability, these attacks are the most dangerous.

Both “physical” and “logical” attacks can be executed locally and remotely. The left side of Figure 1. on page 2 shows the threat spectrum with typical attacks of each category. The categorizations have overlaps. Most remote attacks are indeed logical attacks. However, the rowhammer attack is an example of a remote physical attack. The rowhammer attack attacks the target device by repeatedly changing the contents of a memory location to cause an adjacent, remotely inaccessible, block of memory to change. An attacker can potentially influence the execution of a program that uses that memory.

The right side of Figure 1. on page 2 provides guidelines to the question: what attacks should be protected against? The picture summarizes which attacks to protect against and the relative priorities. However, a trade-off should be made between the risks and cost of protection.

Protection against remote attacks has the highest importance. IoT devices and back-end services need to be protected against remote attack, be it logical or physical. Resistance to local attacks is a different matter though: If an attacker can get local access to the device, protecting against logical attacks has the next priority since it can be again automated and executed by laymen. The local attacks can also take place without the knowledge of the person who enable the actual attack. For example, when malware infected smartphones or USB sticks.

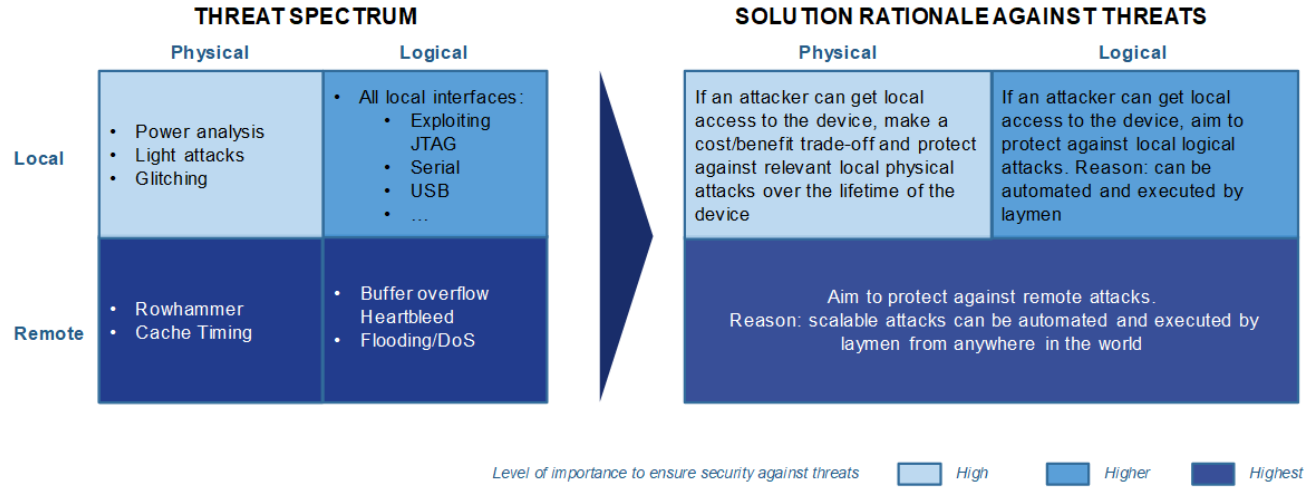

Figure 1. Threat spectrum and solutions

To secure IoT data, four basic requirements must be met: integrity and authenticity must be taken care of, confidentiality and availability when they are required. See Figure 2. on page 3 . These requirements sound straightforward, however, it is hard to achieve all these requirements when attackers are actively targeting the device. Examples are multifold: malware injection leads to loss of integrity; authenticity is breached with counterfeit devices or cloning; confidentiality is breached in examples of hacked toys, and in DDOS attacks the availability is broken.

These requirements can be mapped into the four principles for IoT solutions:

- Prevent attacks to IoT devices
- Able to recover after device being compromised
- Reduce attractiveness of compromised devices

- Secure end-to-end communication

When designing a device the key principle is that, you want to prevent the device from an attack. However, keep in mind that there is no absolute security and each device can eventually be hacked. Higher security level simply increases the effort to a hacker requires to hack the device. A device should be designed with the notion that it might be hacked somewhere during its lifetime, and therefore you should be able to recover after it is compromised. This requires a root-of-trust that you can always get back to, even after the device is compromised.

To assess the required level of security, assess the cost for a hacker to hack the device compared to the potential value that the hacker gets on success. As long as the former outweighs the later, the device is unlikely to be a prime target for hackers and thus reduced the attractiveness of the device to get attacked. Unfortunately, the business case for the attacker is not always obvious for the developer of the device.

Based on above principles for IoT solutions, the following requirements for IoT devices are derived:

- Secure system lifecycle: System should be able to go securely through its different life stages, including power-up/boot phase, debug, OTA updates of firmware/software and decommissioning.
- Crypto and Key protection: A device with a private key can authenticate and communicate with other devices protected by encryption. Storing this private key securely to prevent theft or extraction of the key is vital. Crypto and key protection for instance enable a secured channel for onboarding and encrypted communication to the cloud and encrypted storage of sensitive data on the memory of the chip.
- Resource isolation(HW and SW): Resource isolation allows for tighter control over the access and manipulation of data, processes, and peripherals. Arm MCU with TrustZone is a good example for resource isolation.
- Run time integrity and attestation: As IoT devices are often connected to a cloud and they do not have a human-machine interface, It is important that the runtime integrity of a device can be remotely assessed and attested.

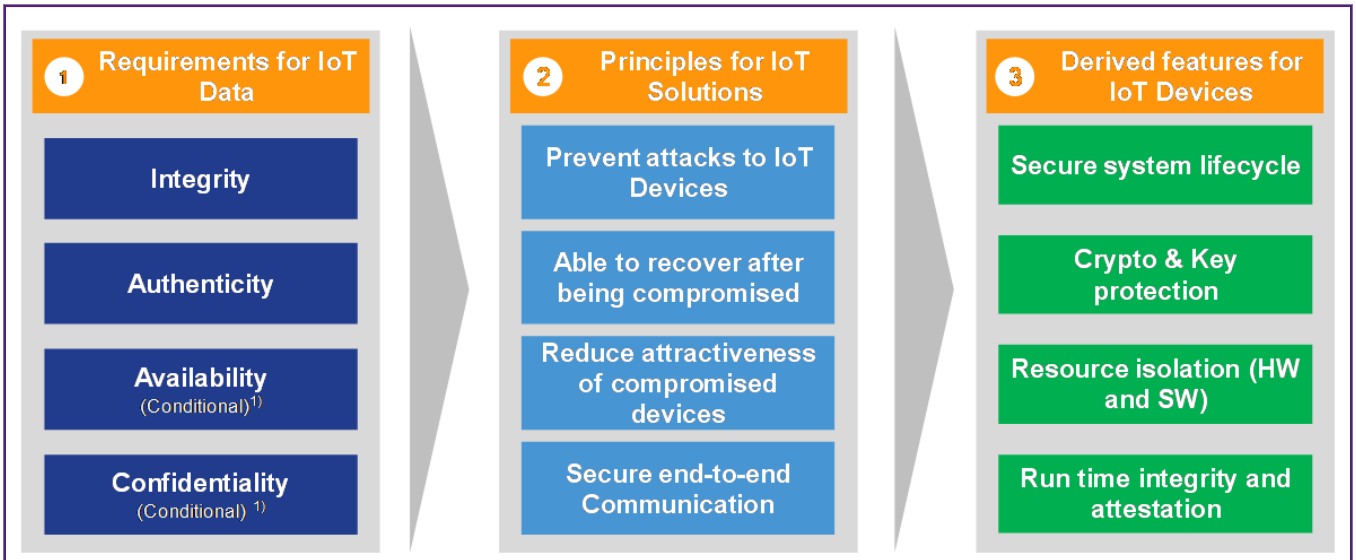

Figure 2. IoT device security features derived for IoT data requirements

1.4 Limitations of security solutions

All security solutions are designed to defend against only a subset of the possible attacks that they may experience. Defending against all possible attacks is an impossible task; if it is “valuable” for somebody, always there is someone willing to spend time and money to break security system using all possible methods. A design must therefore decide which assets it want to protect, and which of the possible attacks it want to protect the assets against. This is perhaps the most critical part of the design process. If a successful attack will take too long or cost too much, the defense is a success.

2 LPC54S0xx Secure Architecture

The chapter describes the architecture of secure MCU LPC54S0xx. The security system on LPC54S0xx has four hardware blocks and ROM code to implement the security features of the device. The hardware consists of an AES engine, an SHA engine, a random number generator, and a key storage block that supports the keys stored in OTP or keys from an SRAM-based PUF(Physically Unclonable Function). Figure 3. on page 4 shows an overview of the LPC54S0xx security system. All components of the system can be accessed by the processor or the DMA engine to encrypt or decrypt data and for hashing. The ROM is responsible for secure boot in addition to provide support for various security functions.

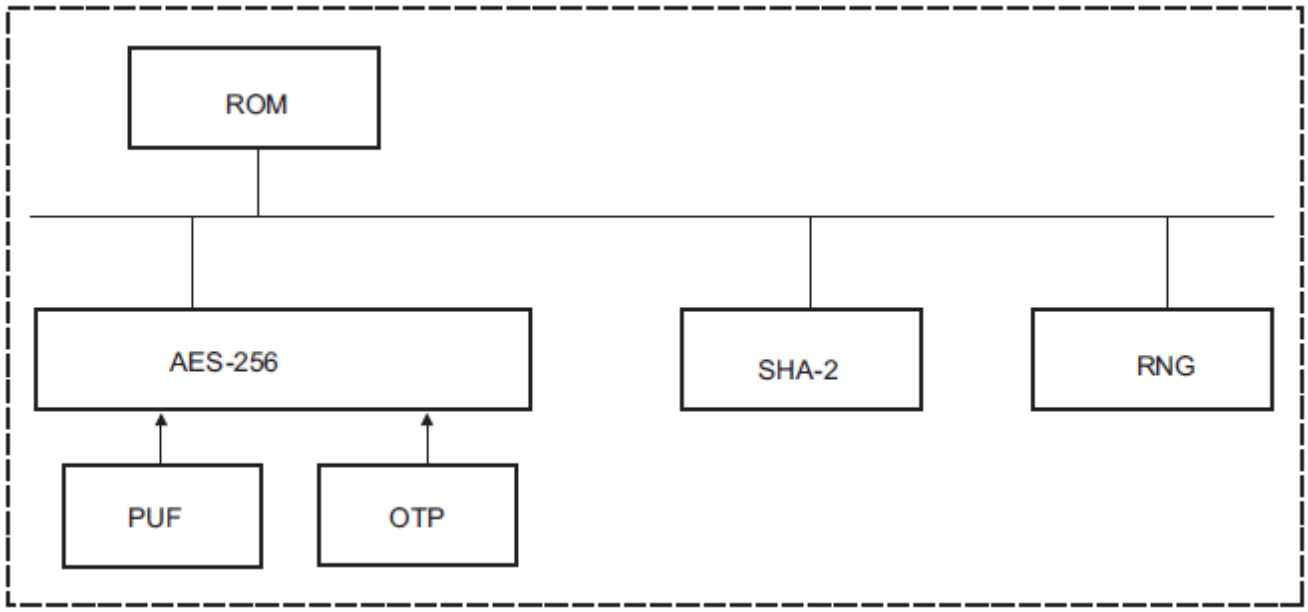

Figure 3. LPC54S0xx security system

2.1 AES engine

The LPC54S0xx devices provide an on-chip hardware AES encryption and decryption engine to protect the image content. It also accelerates processing for data encryption or decryption, data integrity, and proof of origin. Data can be encrypted or decrypted by the AES engine using the encrypted key in the OTP or in PUF or software supplied key.

The features of AES engine are:

- Encryption and decryption of data.
- Secure access to AES key(OTP or PUF) that cannot be read by runtime software.
- AES engine supports 128, 192 or 256-bit key in the several modes: ECB, CBC, CFB, OFB, CTR, GCM.
- The AES engine is compliant with the FIPS Publication 197, AES.
- Data is processed in little endian mode.
- DMA transfers supported through the DMA controller.

2.2 SHA

LPC54S0xx 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 message or code images to a relatively small fixed size “unique” number called a digest. The SHA-1 hash produces a 160-bit digest, and the SHA-256 hash produces a 256-bit digest. Use of SHA-256 is recommended.

For the SHA hardware:

- Even a small change to the input message causes a major change in the digest output. Therefore, for the given input message/image there is only one digest.
- There is no predictable way to modify one input to result in a specific digest. A message cannot be added, inserted, or modified to get the same hash in any direct way.

These two properties make it useful for verifying that a message is valid, whether corrupted intentionally or unintentionally.

2.3 RNG

LPC54S0xx devices provide on-chip entropy of the generated random numbers. The RNG generates a 32-bit random number that cannot be reasonably predicted. Random number generators are used for cryptographic, modeling, and simulation application, which employ keys that must be generated in a random fashion.

The features of RNG:

- Random Number Generator accessible through API call.
- FIPS 140-1/2, NIST, DieHard compliant.

128-bit and 256-bit entropy is supported. The quality of randomness(entropy) of the numbers generated by the Random Number Generator relies on the initial states of the internal logic. If a 128-bit or 256-bit random number is required, it is not recommended to concatenate several words of 32 bits to form the number. To constitute one 128/256 bit number, a 32-bit random number is read, then the next 32 numbers are read but not used. The next 32-bit number is read and used and so on. Thus, 32 32-bit random numbers are skipped between two 32-bit numbers that are used. A typical entropy consumer is a pseudo random-number generator(PRNG) that can be implemented to achieve both true randomness and cryptographic-strength random numbers using the RNG output as its entropy seed.

2.4 OTP

The OTP memory contains four memory banks of 128 bits each. The first memory bank(OTP Bank 0) is reserved. The other three OTP banks are programmable. In LPC54S0xx devices, OTP banks 1 and 2 are available for storing the AES keys. OTP bank 3 is used for customer programmable device configuration data.

The features of OTP:

- The OTP memory stores user settings in bank 3:
  - ISP and boot source modes
  - Secure boot
  - SPIFI boot delay
  - Customer definable bits
- Root of Trust(RoT) hash digest for secure authenticated boot(OTP Bank 1, 2)
- Scrambled 128-bit AES key for secure encrypted boot(OTP Bank 2)
- USB Vendor and Product IDs(OTP Bank 2).
- Boot ROM API support for programming the OTP memory provided

2.5 PUF(Physical Unclonable Function)

PUF is a new topic, so let us discuss it in detail. In case of SRAM PUF this is determined by the power-up values of the SRAM cells. Analytically, each SRAM cell has two stable states that represent a 1 or a 0. When a cell is powered up, the resulting state is unpredictable, but it turns out that uncontrollable deep-submicron manufacturing variations between the transistors in the cell give every cell a preference to come up as a 0 or a 1. For a block of cells this results in a random pattern, like a silicon fingerprint, that is unique per IC and unclonable. However, some of the cells that are closely balanced can be unstable during SRAM power cycle and generate inverted bit values on the initial pattern of zeros and ones(cell flipping). The number of bits that are inverted divided by the total number of bits in the pattern is defined as the SRAM PUF noise. Due to its noisy behavior, an SRAM PUF response cannot be used directly as a key. Post-processing of the PUF response is needed. This can be done by key extraction algorithms based on error-correction functions and randomness extractors. The key extractor must be able to compensate for the noise of the PUF and derive the same cryptographic key each time it is queried. See Figure 4. on page 6 we can get a hardware-based device-unique key.

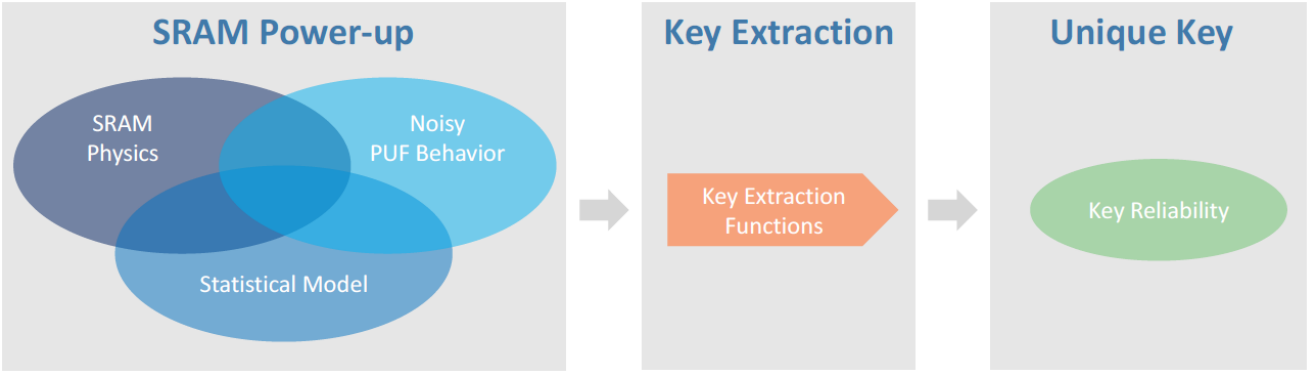

Figure 4. Unique key generator

PUF controller not only generates unique key, but also provides secure key storage without storing the keys. This is done by using the digital fingerprint of a device derived from SRAM. Instead of storing the key, a Key code is generated which in combination with the digital fingerprint is used to reconstruct keys to be used by the user application. Figure 5. on page 7 Shows the top level block diagram of the secure key storage system. The key management in the block diagram also provides a dedicate read-only key output interface to crypto block, and other blocks cannot read it.

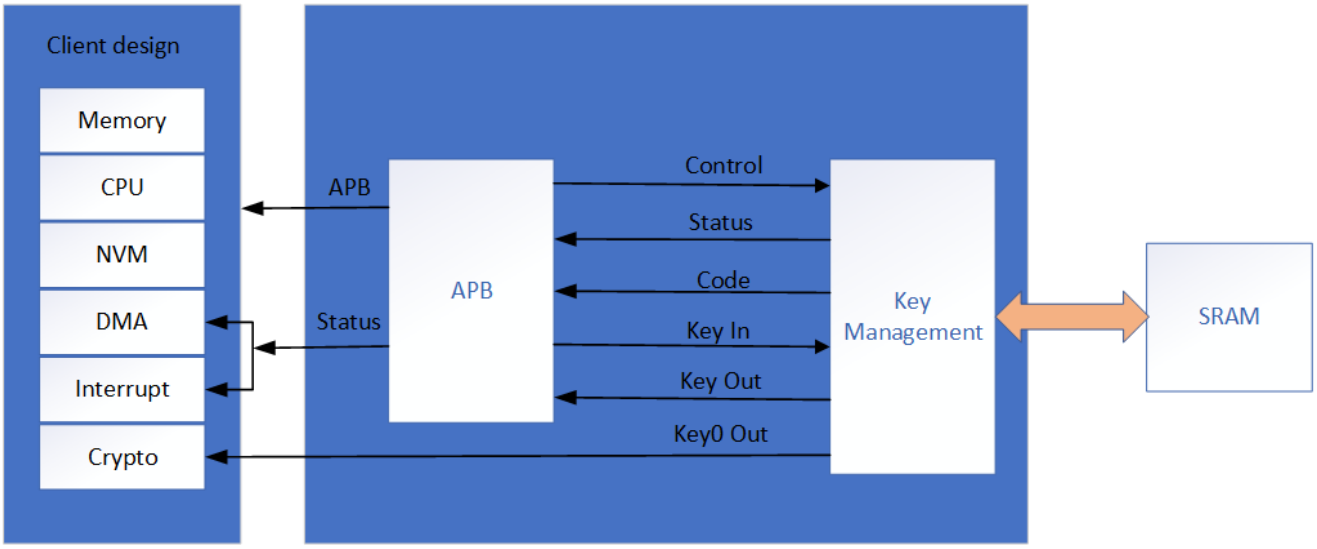

Figure 5. PUF block diagram

2.6 RSA API

There is no RSA engine in LPC54S0xx devices, but boot ROM provides APIs to perform RSA2048 signature checking routines in application code. RSASSA-PKCS1-v1_5-SIGN with SHA-256 hash digest is used for signature verification function.

Function	Offset	Description
rsa2k_decode_e3	0x00	RSA (2048-bit exponent = 3) decode API Parameter 1: Pointer to secure_param structure Return: 0 => = Success 1 0 => SHA256 comparison failed -1 0 => Not a secure part (Security is not supported by the part) Note: The calling application must ensure that there is at least 1200 Bytes of free space left in the stack. If the amount of stack space left is less, then calling this API might cause a stack overflow.
get_secure_firmware_version	0x04	Get the version number of the security firmware. API takes no parameters. In the return value bit 31 to bit 16 are reserved and will be 0. Bit 15 to bit 8 will contain the major number. Bit 7 to bit 0 will have minor number.

Figure 6. RSA2048 signature checking API

3 LPC54S0xx Secure Boot

64 KB ROM memory with firmware(known as boot code) is in LPC54Sxx device. The boot code must always run when the device is powered-on or is hardware-reset. The boot code in LPC54Sxx device is secure boot forming the Root of Trust （RoT）which can secure the boot process by preventing the loading of non-authentic application code. LPC54S0xx has no internal flash for code and data storage, therefore, images must be stored elsewhere to download on reset or the CPU can execute from an external

memory(XIP). **The secure boot supports loading of images into on-chip RAM from external non-volatile memory devices connected to SPI, SPIFI and EMC interfaces, but the secure boot verifies the image and then decide whether running it.** If the code is authentic, the control is transferred to it. A chain of trusted code is established from ROM to the user code. Secure boot uses following cryptography algorithms:

- SHA256 is used for hash function.
- NXP defined 'Image key certificate' is used for public key certificates. To verify that the public key in the certificate belongs to the OEM.
- The public key of the certificate authority or the Root of Trust Key is validated using SHA256 hash check against the OTP contents.
- RSASSA-PKCS1-v1_5-SIGN with SHA-256 hash digest is used for signature verification function. Using RSA keys with 2048-bit public key modulus and 32-bit public key exponent.
- AES algorithm is GCM mode is used for encrypted boot.
  - A 128-bit AES key is used when OTP is used for key store.
  - A 256-bit AES key is used when PUF is used for key store.

3.1 The types of Secure boot Image

To authenticate secure boot code before running, the plain user image is signed and/or encrypted. The secure boot supports following types of secure boot images:

- **Signed image**: RSA-2048 signed images.
- **Encrypted image**: AES-GCM encrypted and authenticated images.
- **Signed encrypted image**: Plain image is encrypted first then signed. This image is recommended to use.
- **Encrypted signed image**: Plain image is signed first and then whole image including signature is encrypted.

All secure boot images have image header which provides various parameters to the secure boot to initialize the boot interface, load address and authenticate/decrypt the image.

3.1.1 Signed image

An RSA-2048 signed image contains the following,Figure 7. on page 9:

- Image with image header.
- Image Key Certificate.
- RSA-2048 signature of the entire image.

In this architecture, the method used to verify the authenticity of the user image is to verify RSA signature over the entire user boot image. The user image is signed with RSA private keys. The corresponding RSA public key used for signature verification is contained in the "Image Key Certificate" that is contained within the signed user image. The image is signed using the RSASSA-PKCS1-v1_5-SIGN algorithm with SHA-256 hash digest.

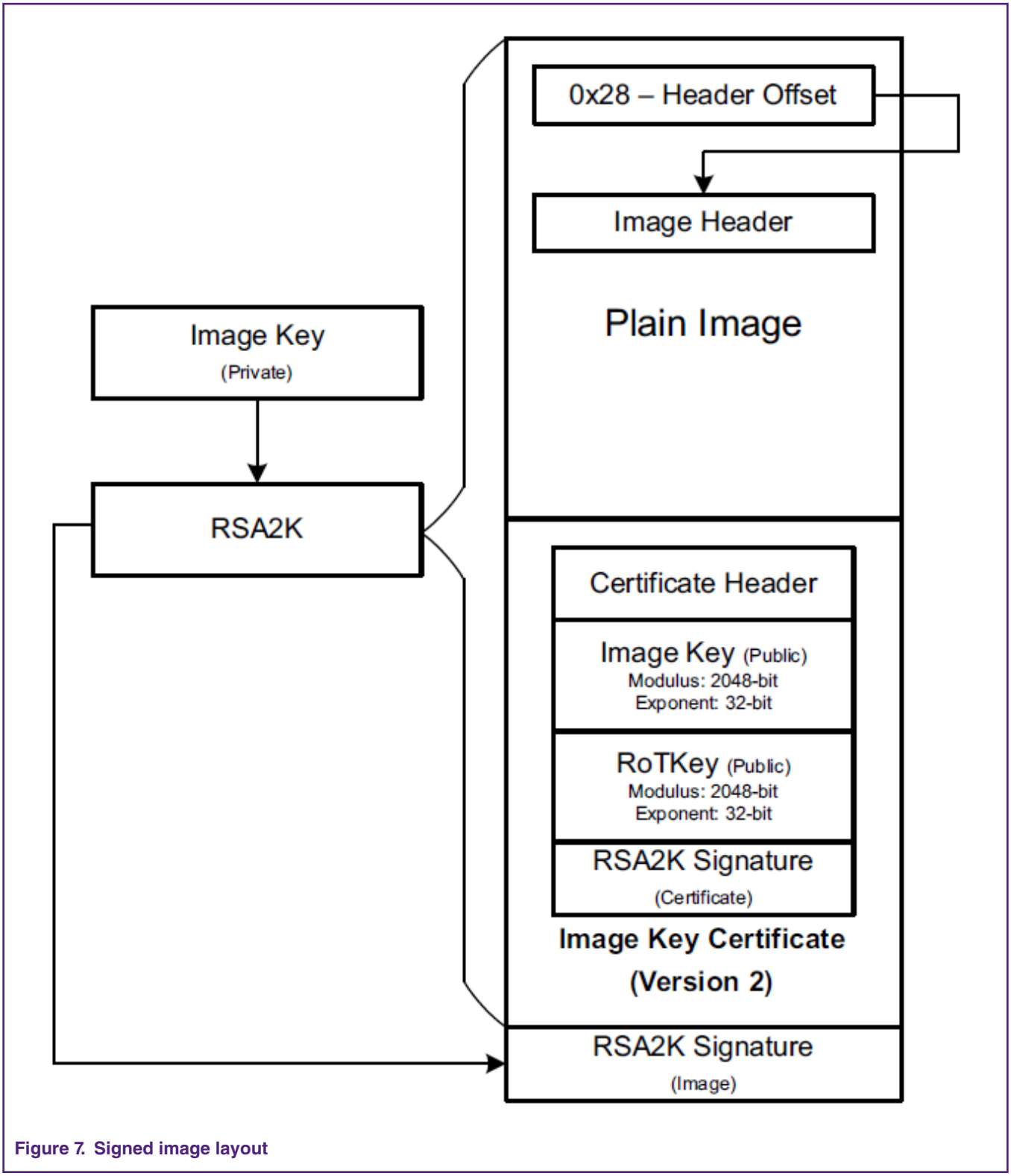

Figure 7. Signed image layout

What is image key certificate in the signed image? In cryptography, a public key certificate, also known as a digital certificate, is an electronic document used to prove the ownership of a public key. A certificate consists of a public key and other data, such as the name of the person and the certificate expiration data, and is signed using the private key of the person or body, known as Certificate Authority(CA), that issued the certificate. If the signature is valid, and the software examining the certificate trusts the

CA, then it can use the public key for further cryptography operations. The secure boot defines a simplified certificate format called **Image Key Certificate**Figure 8. on page 10

Signature = RSASign(RoTKPrivate, SHA256(Certificate_Header || Image_KeyPublic || RoTKPublic))

The RoT Key pair generated by device manufacturer is used for verifying the image key certificate, the private portion of which must be kept securely. The 256-bit SHA2 digest of the public portion of the key is programmed into OTP fuses during manufacturing process.

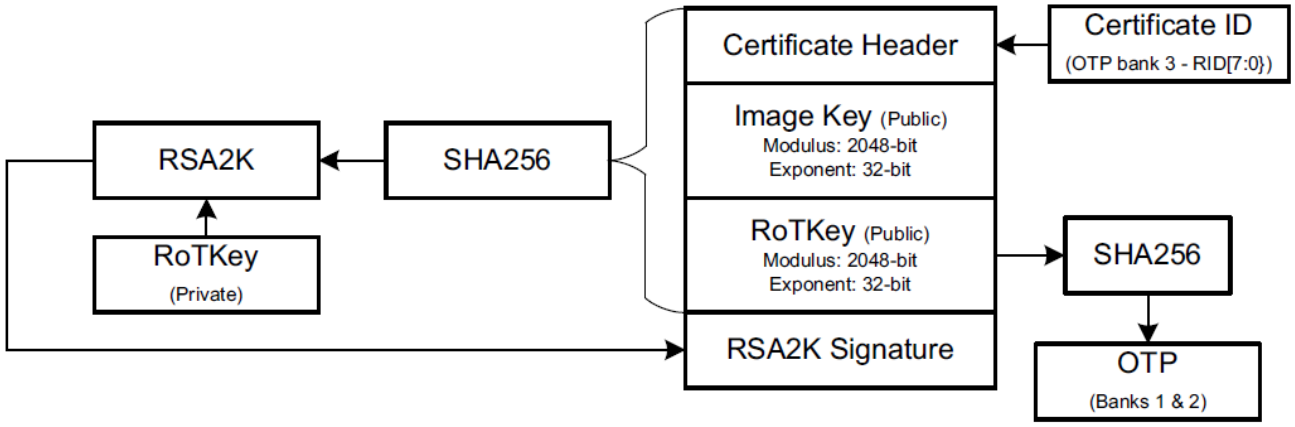

Figure 8. Image key certificate

3.1.2 Encrypted image

The encrypted image contains the following, Figure 9. on page 11

- Encrypted image.
- Image header of the encrypted image(header appended to top of encrypted image).

The encrypted image should be created by encrypting an image using AES-GCM algorithm. An image header is appended to the top of the AES encrypted image. The image header contains image initialization vector(Image_IV), header initialization vector(Header_IV), authentication tag of the encrypted image (Auth_tag), and GMAC of the image header(Header_tag). The 'Additional Authentication Data(AAD) 'is 0 during the encryption process.

[Image GMAC tag, Encrypted_image] = AES-GCM(key = OTP_key, IV = Image_IV, Plain_text = Plain_image, AAD = 0)

[Header GMAC tag, 0] = AES-GCM(key = OTP_key, IV = Header_IV, Plain_text = 0, AAD = Encrypted Image Header)

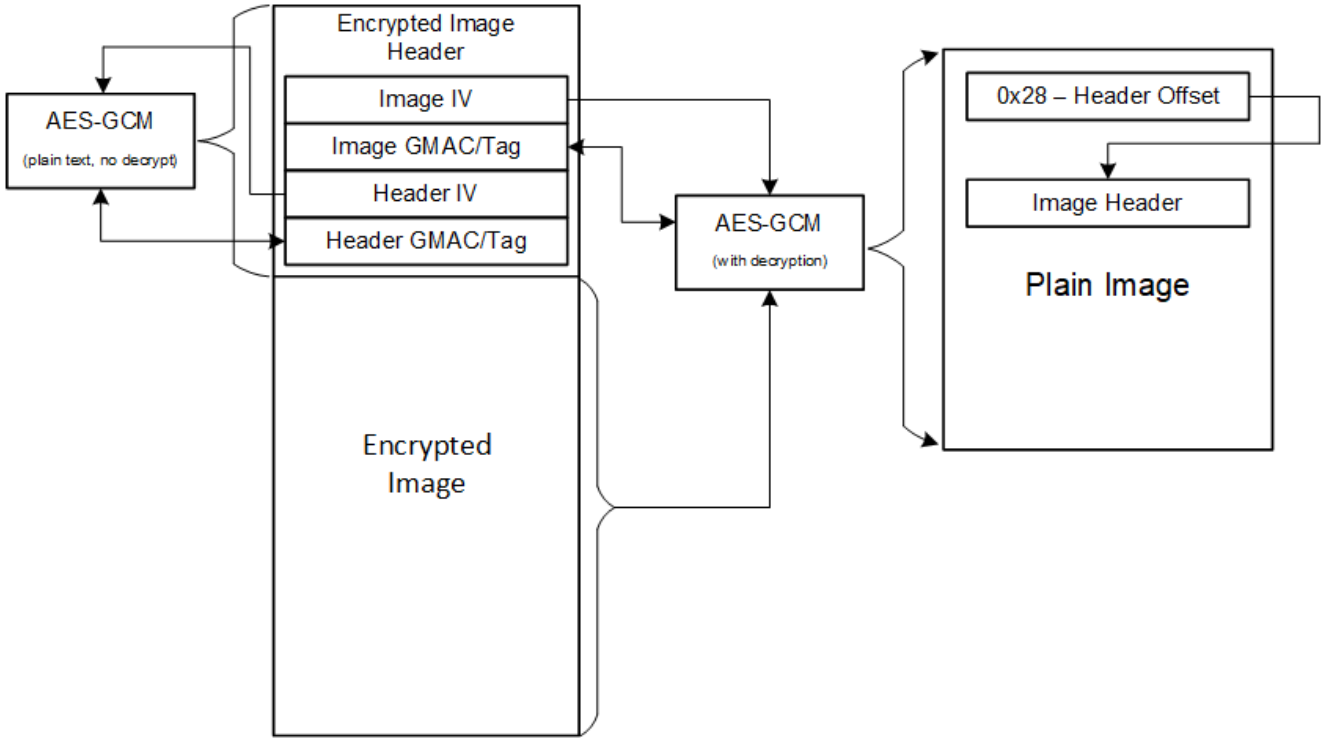

Figure 9. Encrypted image layout

3.1.3 Signed encrypted image

Figure 10. on page 11 shows the signed encrypted image layout. Plain image is encrypted then signed. To know about how to encrypt image, see Encrypted image. To know about how to sign image, see Signed image.

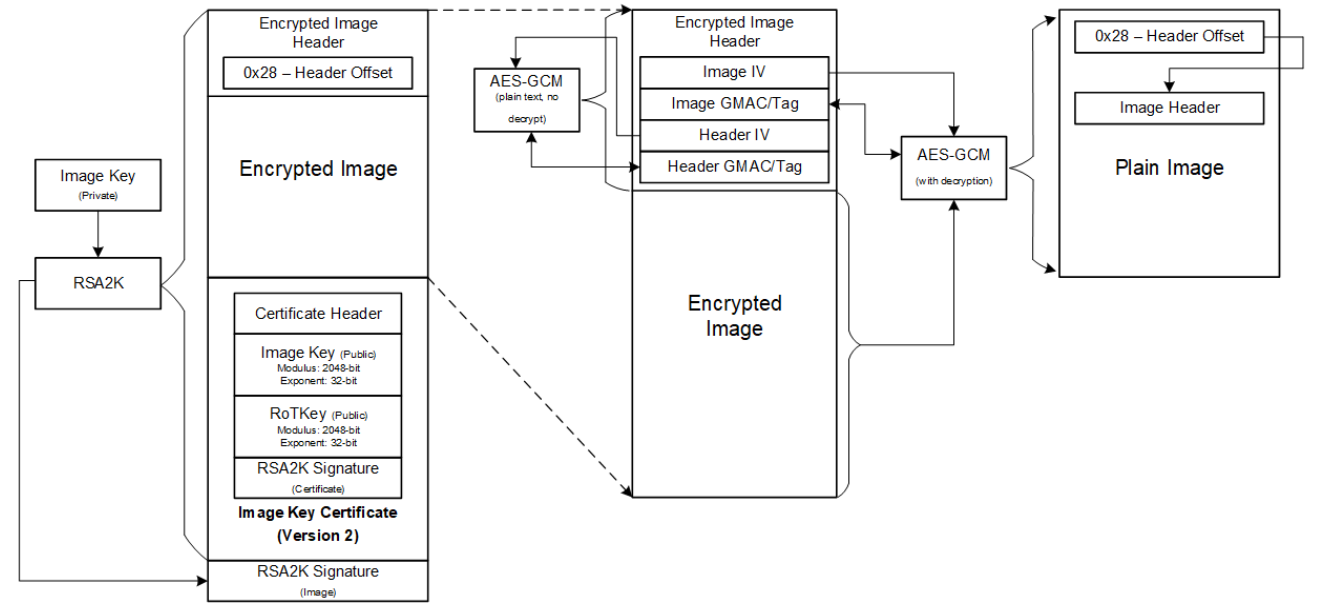

Figure 10. Signed encrypted image layout

3.1.4 Encrypted signed image

Figure 11. on page 12 shows the encrypted signed image layout. Plain image is signed first and then whole image including signature is encrypted. About how to encrypt image, see Encrypted image. About how to sign image, see Signed image.

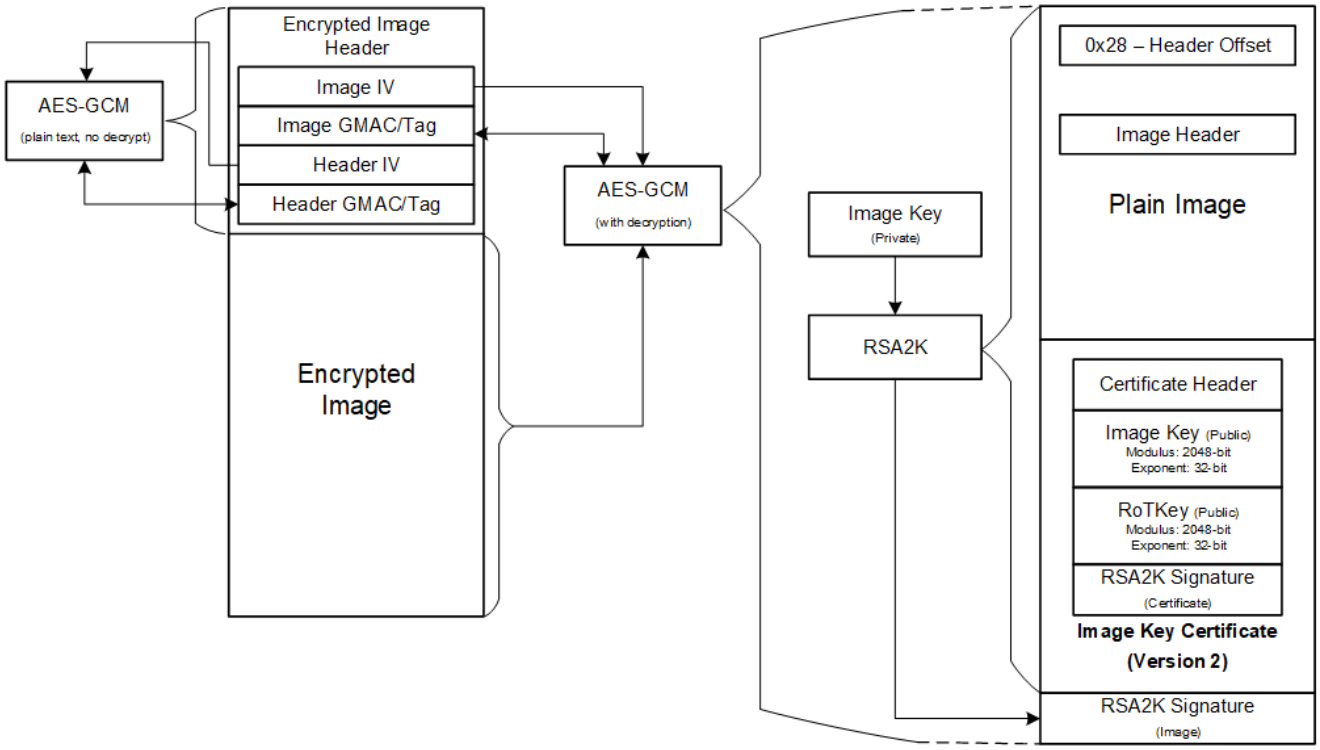

Figure 11. Encrypted signed image layout

3.2 Secure boot process

The secure boot is executed every time the device is powered-on or reset, Figure 12. on page 13 Shows the top-level boot process. User code can boot from external flash(QSPI flash, SPI flash and EMC flash), but it need to be copied to internal RAM to run after authentication or/and encryption are successful.

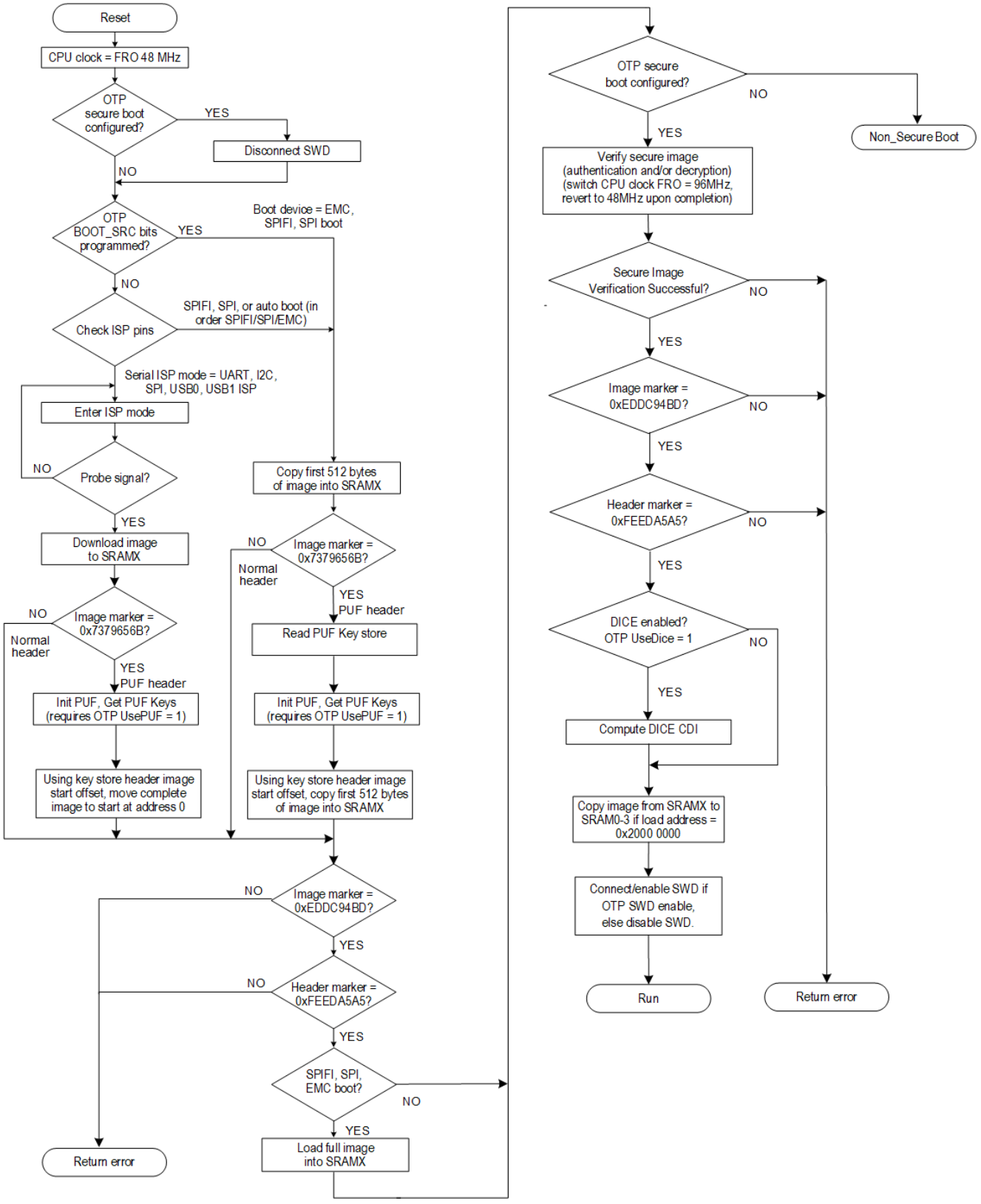

Figure 12. Secure boot process flow chart

3.3 Boot Keys storage

A cryptographic key is a string of bits used by a cryptographic algorithm to transform plain text into cipher text or vice versa. This key remains private and ensures secure communication. Two kinds of keys need to be stored: SHA256 of RoTKPublic and AES Key. The SHA256 of RoTKPublic is SHA-256 digest of RoT Public Key for authenticating signed image. AES Key is for decrypting encrypted image, and is scrambled if storing in OTP Bank.

Table 1. Boot Keys storage on page 14, we list the keys storage with corresponding images.

Table 1. Boot Keys storage

Key storage	Signed image	Encrypted image	Signed encrypted image		Encrypted signed image	
	SHA256 of RoTKPublic	AES Key	SHA256 of RoTKPublic	AES Key	SHA256 of RoTKPublic	AES Key
In OTP (OTP_USE_PUF = 0)	OTP Bank 1: Low 128-bit; OTP Bank 2: Upper 128-bit.	OTP Bank 2: 128-bit; scrambled AES Key[1]	OTP Bank 1: Lower 128-bit[2]	OTP Bank 2: 128-bit scrambled AES Key[3]	OTP Bank 1: Lower 128-bit[2]	OTP Bank 2: 128-bit scrambled AES Key[4]
In PUF (OTP_USE_PUF = 1)	OTP Bank 1: Lower 128-bit; OTP Bank 2: Upper 128-bit[5] .	PUF Key Store: 256-bit AES Key	OTP Bank 1: Lower 128-bit; OTP Bank 2: Upper 128-bit[5] .	PUF Key store: 256-bit AES key	OTP Bank 1: Lower 128-bit; OTP Bank 2: Upper 128-bit[5] .	PUF Key store: 256-bit AES key

1. Software cannot read The scrambled AES key.
2. Only the lower 128-bits of SHA256 hash are compared against OTP bank 1 contents to validate RoTKPublic.
3. Software cannot read The scrambled AES key
4. Software cannot read The scrambled AES key
5. Only the lower 128-bits of SHA256 hash are compared against OTP bank 1 contents to validate RoTKPublic when OTP bank 2 is used for following conditions, otherwise, 256-bits of SHA256 hash are compared with OTP bank 1 and 2. - OTP bank 2 is used for storing customer-specific USB-IF certified vendor identifier(VID) and product identifier(PID). - USB IDs are used by boot ROM during USB DFU boot and ISP operations. - Custom IDs are only needed if USB DFU boot is used in end-products. - OTP bank 2 word 0, should contain USB ID Marker value of 0x43555342.

3.4 Device Identifier Composition Engine(DICE)

LPC54S0xx secure boot supports Device Identifier Composition Engine(DICE) specification specified by Trusted Computing Group, see https://trustedcomputinggroup.org/wp-content/uploads/Implicit-Identity-Based-Device-Attestation_PublicReview.pdf.

DICE provides a means for each LPC54S0xx device to cryptographically generate a 32 byte unique ID called a Compound Device Identifier(CDI). DICE computes CDI after authentication of user image and before transferring the control to user image. DICE uses Unique Device Secret(UDS) and SHA-256 digest of user image to calculate CDI. CDI = HMAC(UDSKey, SHA2(SBL_IMG)), SBL_IMG = L0_IMAG without L0_Signature UDS is from PUF or unique identifier(UUID). For PUF-based UDS, UDS is index 15 key retrieved using key code from key store(obtained during provisioning/manufacturing) and ROM disables decoding of index 15 keys in PUF after CDI calculation. For UUID-based UDS, UDS is calculated from the UUID and OTP bank 3(register 2/3 locked after CDI calculation). On exit of secure boot and entry to the user application, the 32 byte CDI is saved at the base of SRAM0.

4 Conclusion

LPC54S0xx is a secure MCU integrating many secure peripherals, such as AES engine, SHA, RNG, PUF, DICE, etc. Secure boot is resident in ROM of LPC54S0xx to ensure the application code is authentic. All of these advanced secure features make LPC54S0xx a good fit for building a secure system.

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: 15 March 2019
Document identifier: AN12385

arm