# AM6x Security Overview

## Nov 2024

## Introduction to Secure Boot

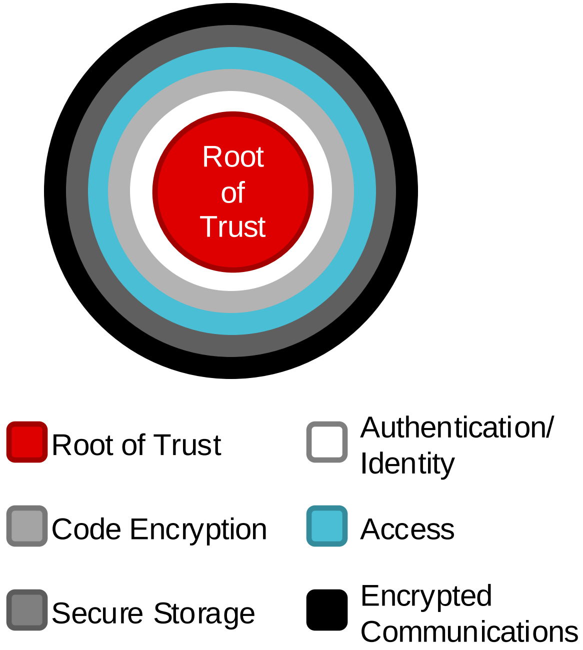
# Agenda

- Why secure boot?
- Setting it up
- Using it
- Summary

# Security starts with trust

- Threat matrix, use cases, applicable standards, etc. will dictate specific product requirements
- A layered approach to security is best to address these requirements
- Many of these requirements will impact hardware and software design
- Hardware Root of Trust will be needed to establish a chain of trust for software
- A Secure Boot process is needed to validate or authenticate the software before allowing execution on the device

Layers of Security



Root of Trust

🟥 Root of Trust

□ Authentication/ Identity

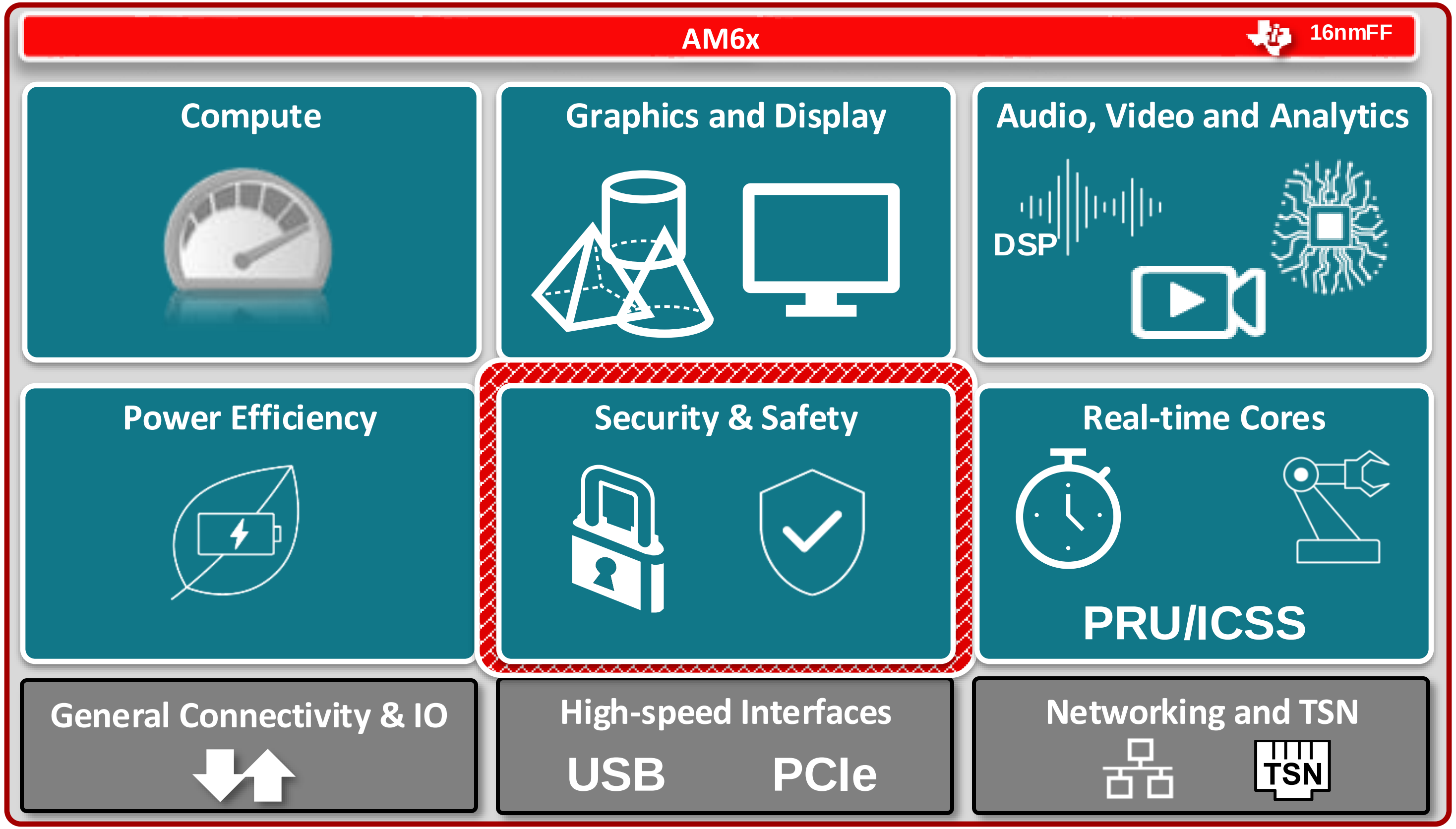
▨ Code Encryption

🟦 Access

▨ Secure Storage

■ Encrypted Communications

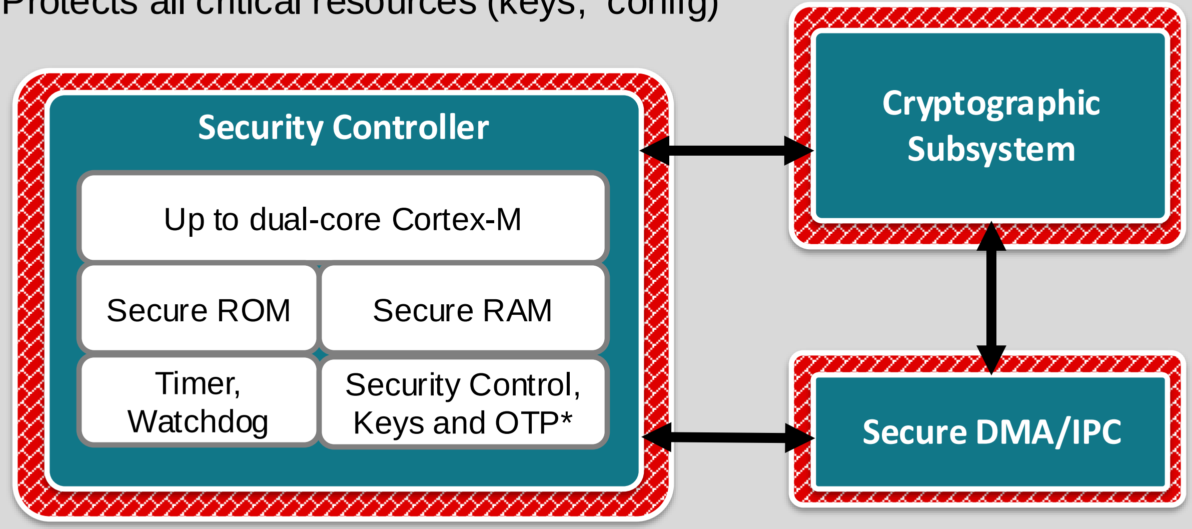
# AM6x Cortex®-A based architecture



# AM6x security architecture

AM6x 16nmFF

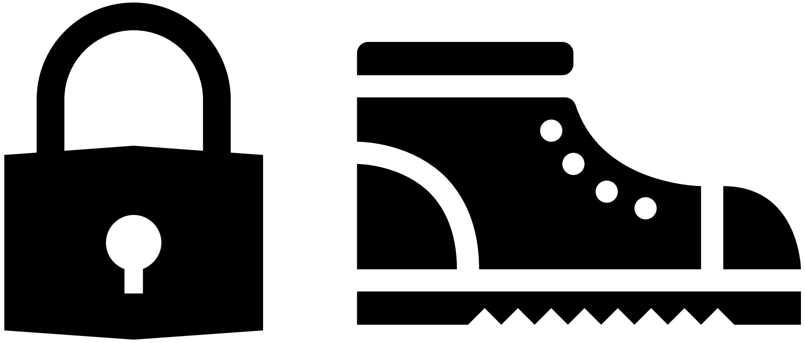
- Central control for security (secure boot, debug, etc.)
- Isolated from the rest of the system by firewalls
- Protects all critical resources (keys, config)



* OTP = One-Time Programmable Memory

# Agenda

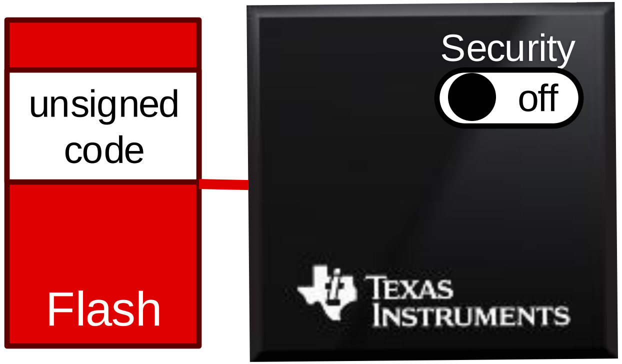
- ✓ Why secure boot?
- Setting it up
- Using it
- Summary



# Non-secure boot

When a device doesn’t have security or has security turned off, the processor simply copies code from memory or a peripheral and executes it

- Good for accelerating software development, especially on new hardware platforms that may have bugs
- Boot source is selected with bootmode pins on the device
- Device will attempt to execute code with minimal checks or validations, non-securely
- Changing the code or boot source is easy for development
    - Simply reflash or send new code via a peripheral

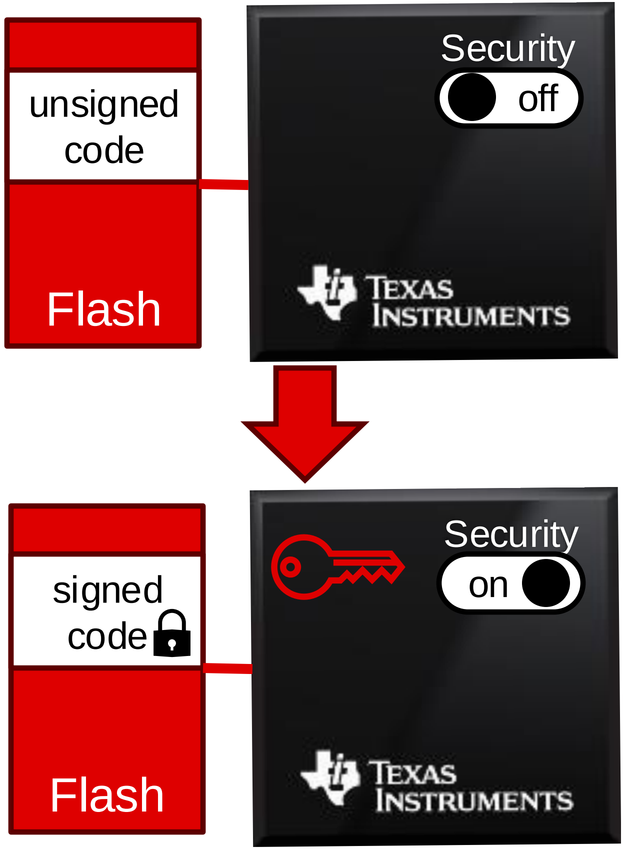


# Secure boot

Secure boot is a hardware based “Root of Trust” to authenticate and protect boot code and data. Customers program their own keys using software/tools supplied by TI.

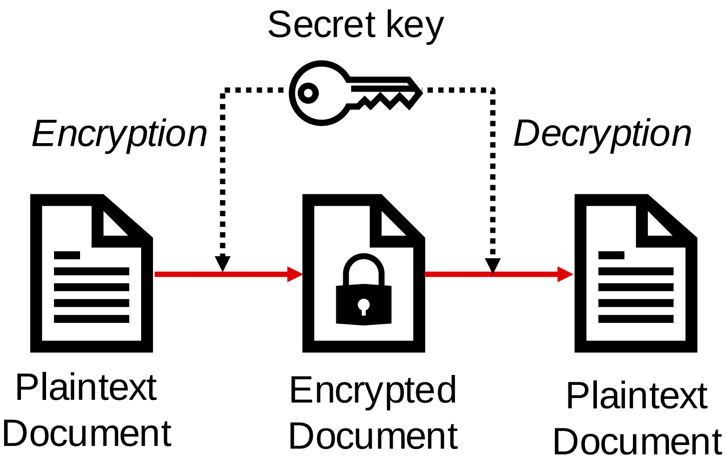
- When security is enabled, the device will only boot code specifically prepared for the device using asymmetric encryption and hashing
- Takeover Protection
  - My device only runs my software (authenticity and integrity)
  - Non-volatile one-time-programmable memory within device is configured so device will only boot “trusted” software. Ensure external flash content is not modified.
  - Overwriting flash or changing the boot source to load new code that is not signed will result in a boot failure
- Chain of Trust can be extended to following boot stages (i.e. OS or Application Image)

Non-secure Boot

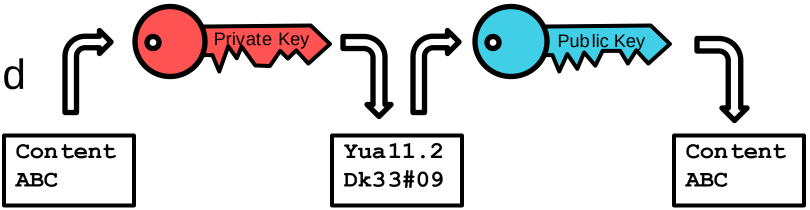


# Encryption basics: symmetric vs asymmetric

- Symmetric encryption
  - *Same key* is used to encrypt and decrypt content
  - Uses algorithms like AES or 3DES
  - Drawback: sender and receiver both need to know and store the secret key
  - Key needs to be securely stored inside device!

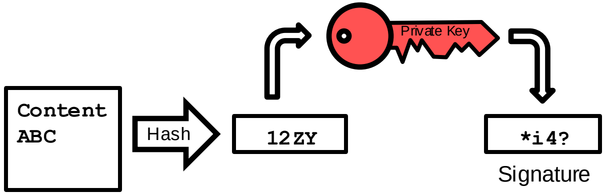


- Asymmetric encryption
  - Pair of keys: Secret private key and public key
  - Often used to **sign messages**. Can be decrypted with the public key, so you know message is from the expected source/sender.
  - Only private key needs to be kept secret

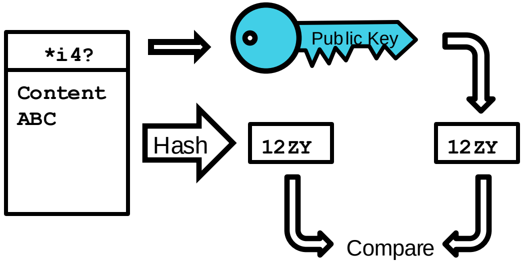


# Hash, signature, certificate

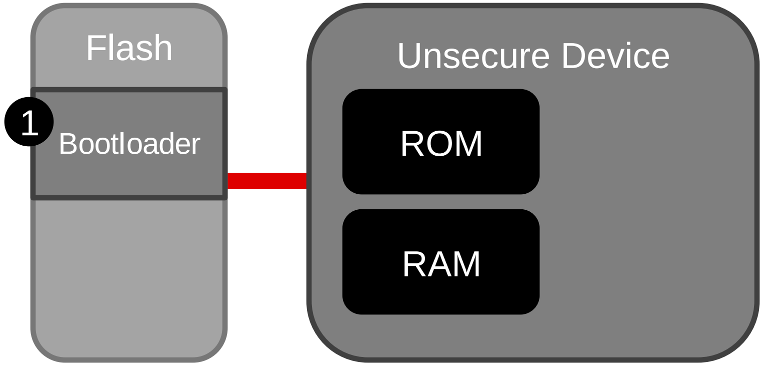
- Hash function: map data of arbitrary size to fixed-size values, e.g. SHA256. Like a checksum but more secure.
  - Cannot get information from hash about content (one way)
- Encrypting a hash with private key creates a signature



- Original content together with signature creates a certificate
- Anyone with the public key can verify:
  - The content hasn't been altered (*integrity*)
  - The content came from a trusted source (*authenticity*)

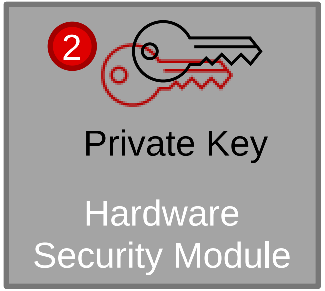


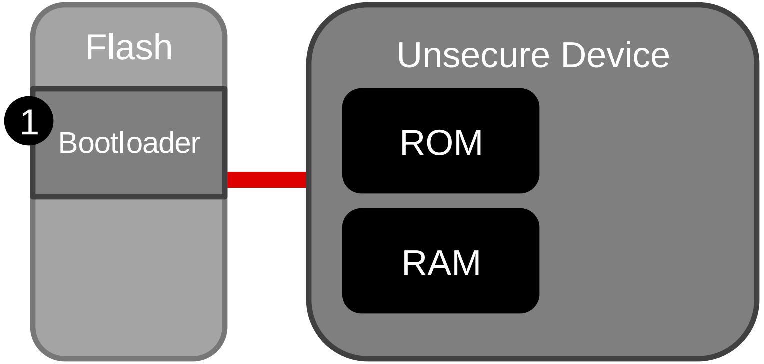
# Steps required for secure boot



1 Write software on unsecured device

# Steps required for secure boot

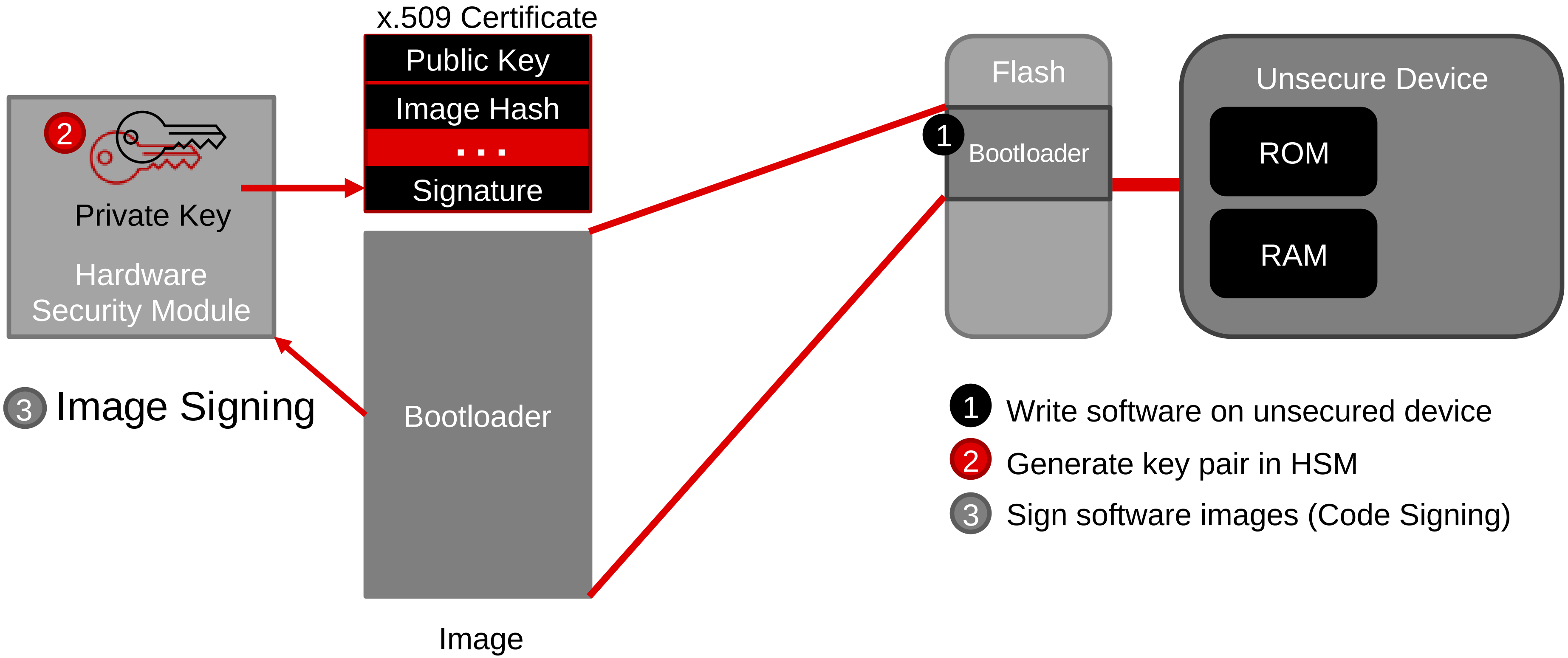




Note: This is ***NOT*** the HSM in the TI SoC

1. Write software on unsecured device
2. Generate key pair in HSM

# Steps required for secure boot

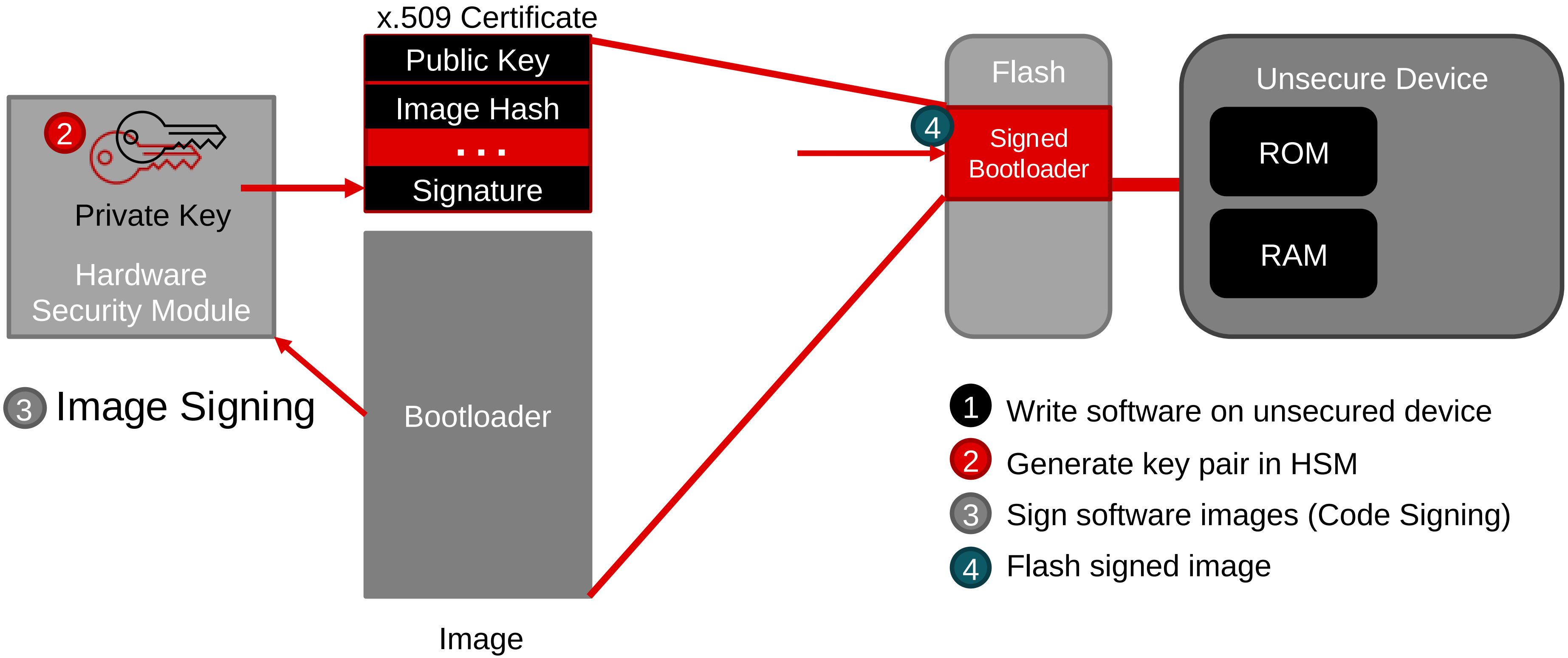


1 Write software on unsecured device

2 Generate key pair in HSM

3 Sign software images (Code Signing)

# Steps required for secure boot



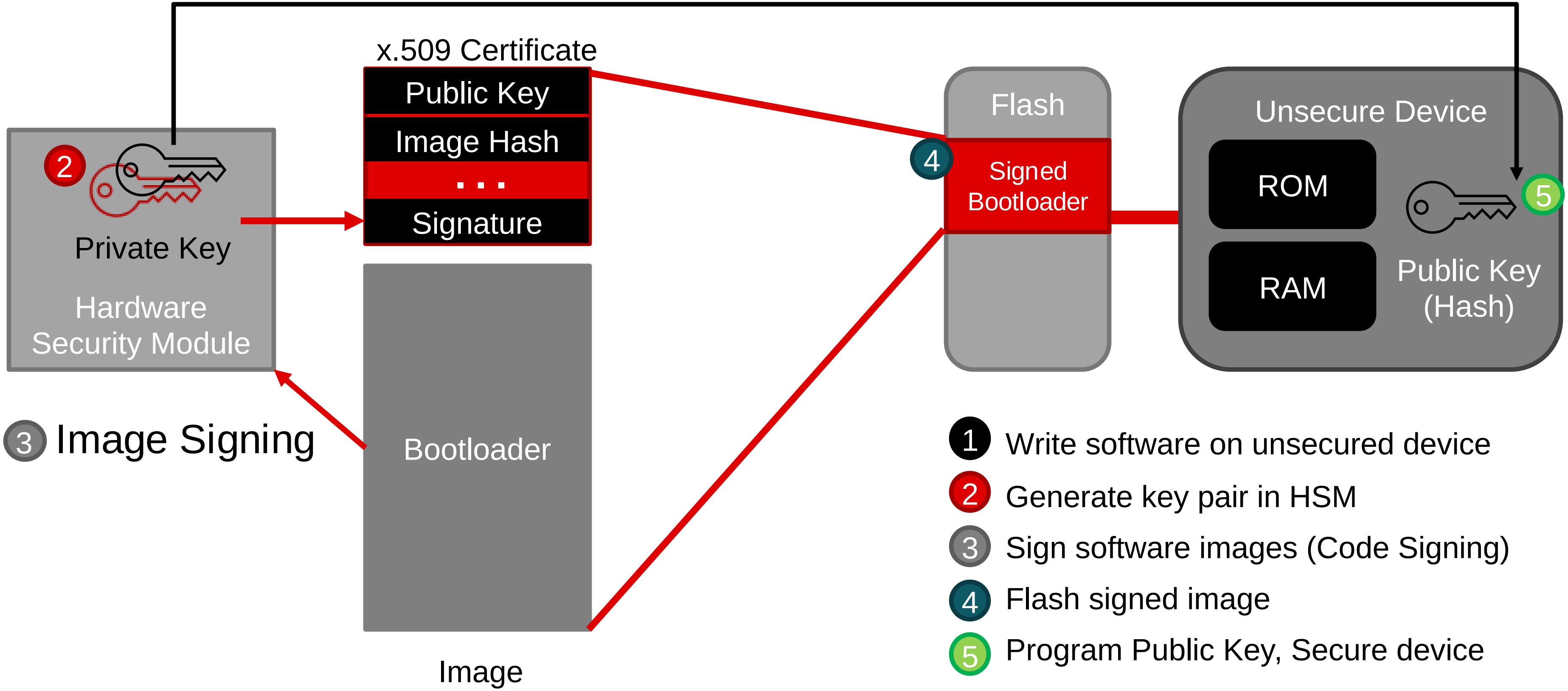
1 Write software on unsecured device

2 Generate key pair in HSM

3 Sign software images (Code Signing)

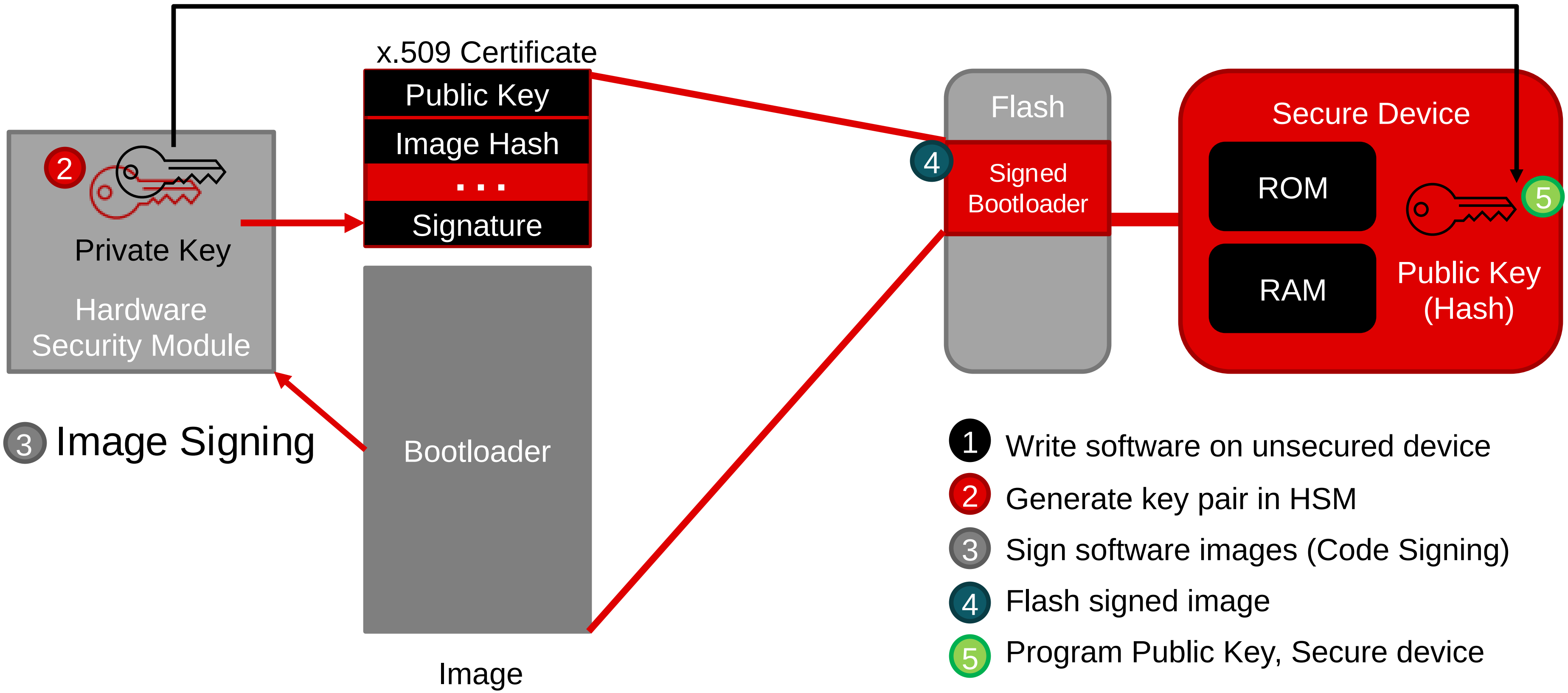
4 Flash signed image

# Steps required for secure boot



1. Write software on unsecured device
2. Generate key pair in HSM
3. Sign software images (Code Signing)
4. Flash signed image
5. Program Public Key, Secure device

# Steps required for secure boot



1 Write software on unsecured device

2 Generate key pair in HSM

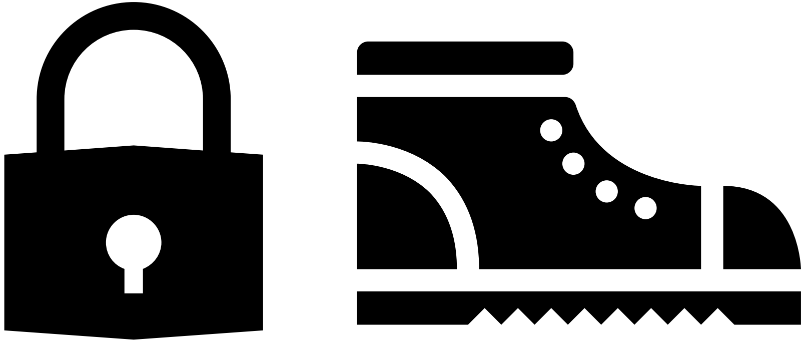
3 Sign software images (Code Signing)

4 Flash signed image

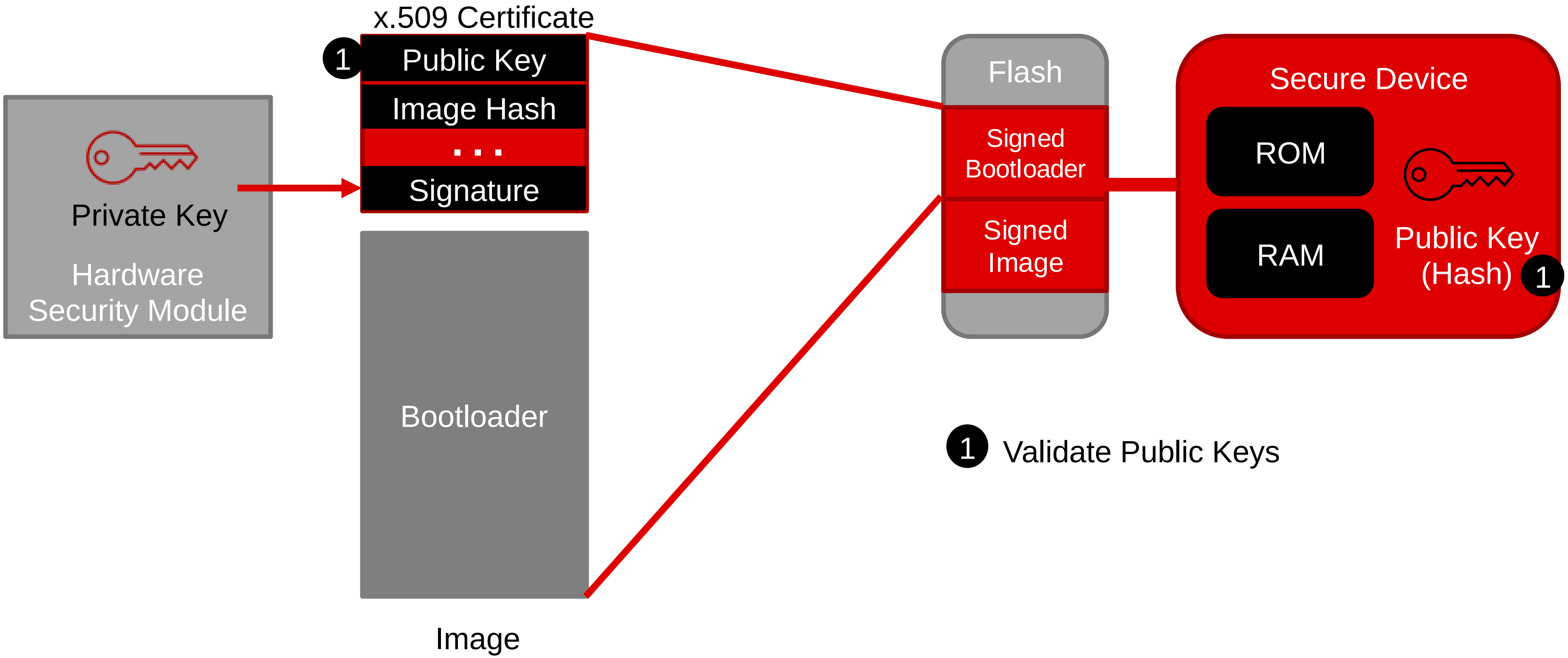
5 Program Public Key, Secure device

# Agenda

- ✓ Why secure boot?
- ✓ Setting it up
- Using it
- Summary

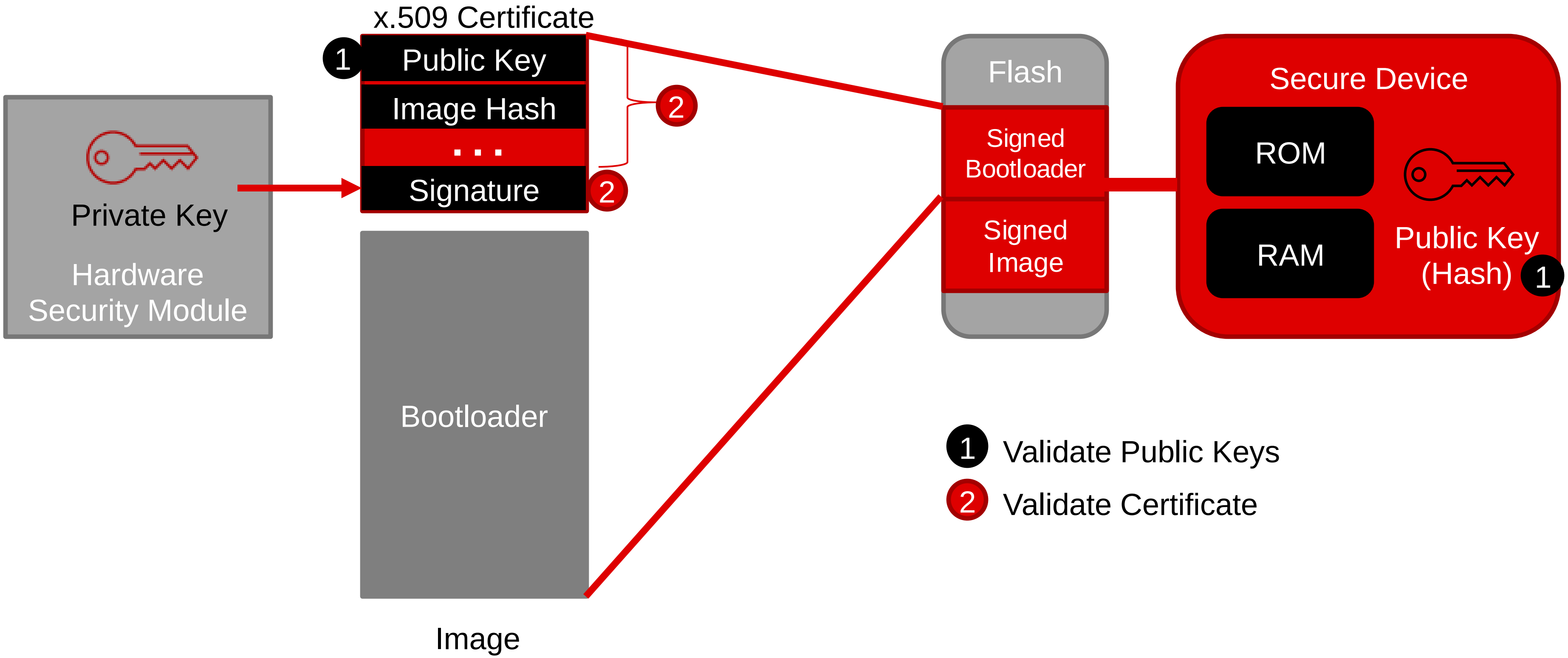


# Use secure boot



1 Validate Public Keys

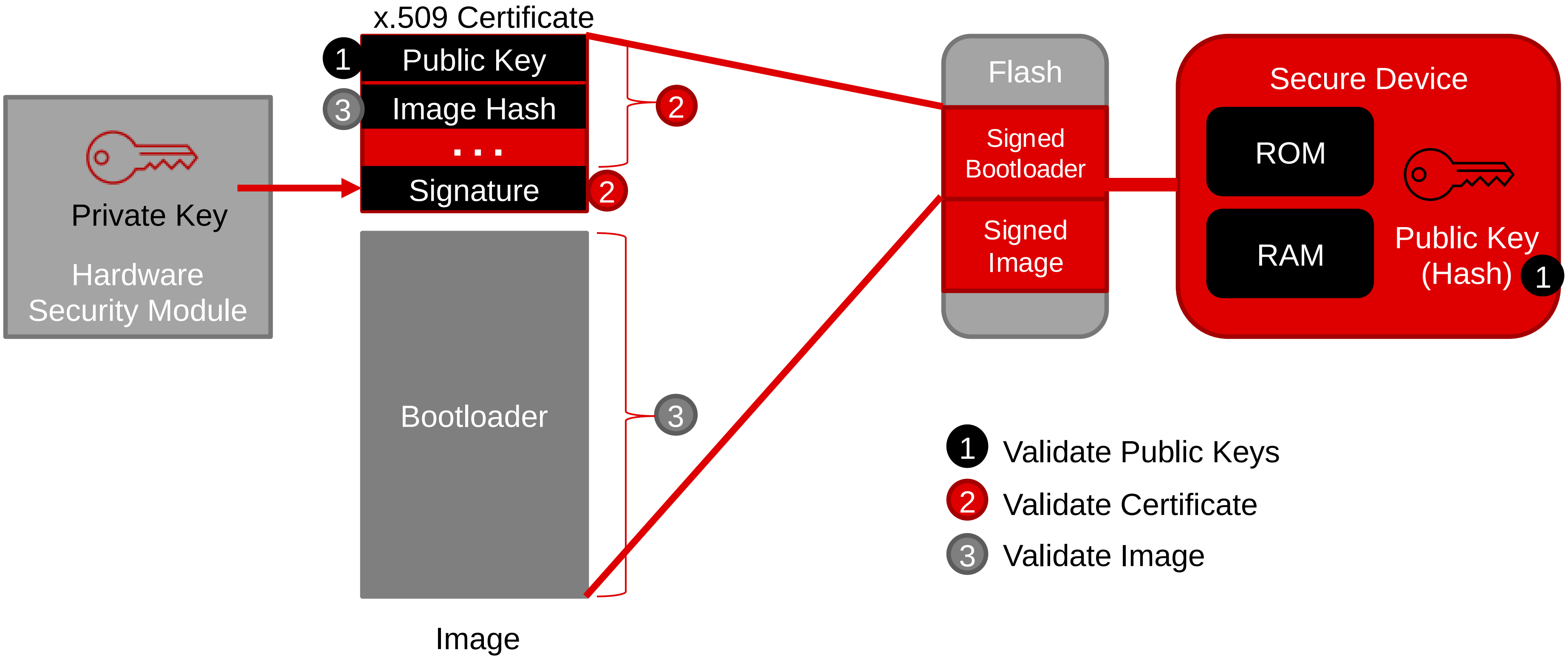
# Use secure boot



1 Validate Public Keys

2 Validate Certificate

# Use secure boot



1 Validate Public Keys

2 Validate Certificate

3 Validate Image

# Agenda

- ✓ Why secure boot?
- ✓ Setting it up
- ✓ Using it
- Summary

# Summary

- Secure boot is required to know that software is authentic and valid
- Leverages Root of Trust (RoT) to establish a chain of trust for software
- Included in standards and regulations
- Private 🔑 management is paramount
- Available today!

## Secure boot steps

1. Write software on unsecured device
2. Generate key pair in HSM
3. Sign software images (Code Signing)
4. Flash signed image
5. Program Public Key, Secure device

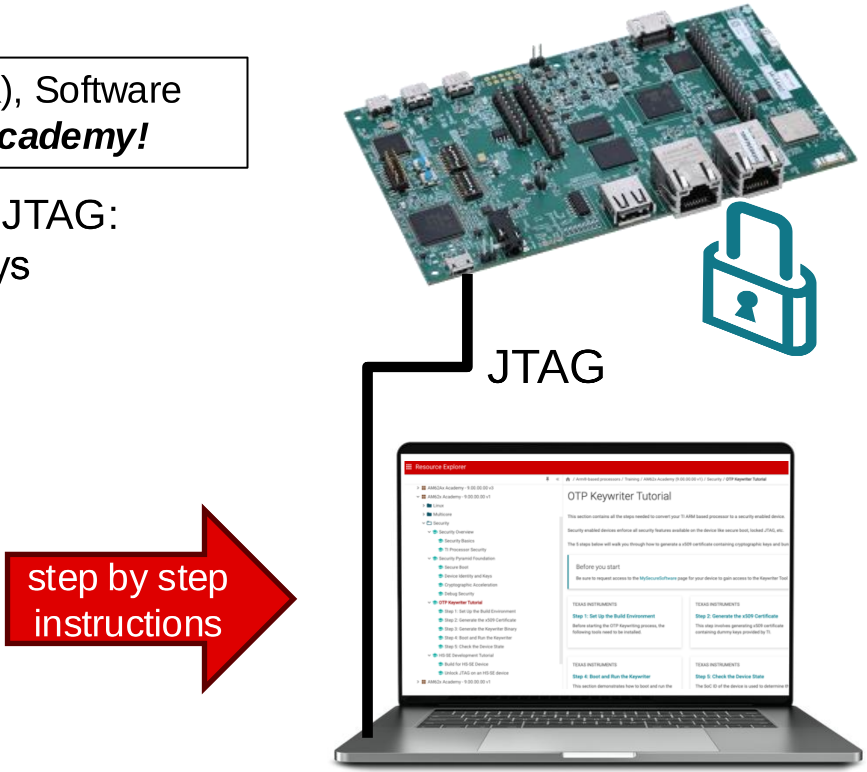
# Security getting started

Start using security today with a current Starter Kit (SK), Software Development Kits (SDKs), and tools with ***Security Academy!***

Learn hands-on how to use secure boot and JTAG:

1. Sign software with TI ”shared” private keys
2. Program “known” public keys to a device
3. Verify secure boot
4. Unlock JTAG for debug

| Device | Academy |
|---|---|
| AM62x | Link |
| AM62Ax | Link |
| AM64x | Link |
| AM67x | Link |



step by step instructions

JTAG

/ Arm®-based processors / Training / AM62x Academy (9.00.00.00 v1) / Security / **OTP Keywriter Tutorial**

- AM62Ax Academy - 9.00.00.00 v3
- AM62x Academy - 9.00.00.00 v1
  - Linux
  - Multicore
  - Security
    - Security Overview
      - Security Basics
      - TI Processor Security
    - Security Pyramid Foundation
      - Secure Boot
      - Device Identity and Keys
      - Cryptographic Acceleration
      - Debug Security
    - OTP Keywriter Tutorial
      - Step 1: Set Up the Build Environment
      - Step 2: Generate the x509 Certificate
      - Step 3: Generate the Keywriter Binary
      - Step 4: Boot and Run the Keywriter
      - Step 5: Check the Device State
    - HS-SE Development Tutorial
      - Build for HS-SE Device
      - Unlock JTAG on an HS-SE device
- AM62x Academy - 9.00.00.00 v1

OTP Keywriter Tutorial

This section contains all the steps needed to convert your TI ARM based processor to a security enabled device.

Security enabled devices enforce all security features available on the device like secure boot, locked JTAG, etc.

The 5 steps below will walk you through how to generate a x509 certificate containing cryptographic keys and bur

Before you start

Be sure to request access to the MySecureSoftware page for your device to gain access to the Keywriter Tool

TEXAS INSTRUMENTS
Step 1: Set Up the Build Environment
Before starting the OTP Keywriting process, the following tools need to be installed.

TEXAS INSTRUMENTS
Step 2: Generate the x509 Certificate
This step involves generating x509 certificate containing dummy keys provided by TI.

TEXAS INSTRUMENTS
Step 4: Boot and Run the Keywriter
This section demonstrates how to boot and run the

TEXAS INSTRUMENTS
Step 5: Check the Device State
The SoC ID of the device is used to determine th

***Note: This process is very similar for all AM6x family members…***

# Thank You!