Secure by Design NXP Webinar Series

Securing Embedded Linux Devices: Pitfalls to Avoid

NXP Webinar: October 6, 2020

Presented by:

NXP

SECURE CONNECTIONS FOR A SMARTER WORLD

Agenda

- Security requirements and threat model
- Select NXP processor/board hardware security features
- Available security solutions and options
- Common product security pitfalls
- Security best practices and most frequently used protection steps
- Product software updates and security

Threat Identification

Product Requirements

- Describe product features
- Shall also capture
  - Targeted use cases
  - Operation environment description
  - Compliance
  - Maintenance strategy
  - Desired security measures

Threat Modeling

- Structured process to identify, assess and address threats
- Many different techniques exist including STRIDE, OCTAVE, PASTA

The Process for Attack Simulation and Threat Analysis (PASTA)

Define Objectives	• Identify Business Objectives • Identify Security and Compliance Requirements • Business Impact Analysis
Define Technical Scope	• Capture the Boundaries of the Technical Environment • Capture Infrastructure, Application, Software Dependencies
Application Decomposition	• Identify Use Cases, Define App Entry Points and Trust Levels • Identify Actors, Assets, Services, Roles, Data Sources
Threat Analysis	• Probabilistic Attack Scenarios • Regression Analysis on Security Events • Threat Intelligence Analytics
Vulnerability & Weaknesses Analysis	• Tracking of Existing Vulnerability Reports & Issues Tracking • Scoring (CVSS) • Design Flaw Analysis (Use & Abuse Cases)
Attack Modeling	• Attack Surface Analysis • Attack Tree Development, Attack Library Management • Attack to Vulnerability & Exploit Analysis Using Attack Trees
Risk & Impact Analysis	• Qualify & Quantify Business Impact • Countermeasure Identification and Residual Risk Analysis • ID Risk Mitigation Strategies

Objectives and Technical Scope

- Business impact
  - Regulated market (compliance, certifications)
  - Assess the liability impact and scope

- What do you want to protect?
  - Products' Intellectual Property (your software, algorithms)
  - Processed Data (user information, end customer info, medical records, car user profiles)
  - High importance data (bank info, credit card, authentication keys)
  - Product availability (connected to software function)
  - Product authenticity (device certified by a company)
  - Data integrity (for collected info)

Product Decomposition

- Product block diagram
  - Software Diagram
    - Split into OS and Application
    - Identify used subsystems and APIs
    - Indicate data placements and data flow
  - Hardware Diagram
    - Identify hardware modules
    - List all open interfaces
- Dependencies
  - SBOM and API dependency tree
  - Hardware-software association
- Data
  - Sources (Network, local, sensors)
  - Where is the data stored and how?
- Actors
  - Who is allowed to access the system?
  - What operations are permitted per Actor?
- Execution Flow
  - What is the primary execution flow?
  - What use cases are allowed?
- Current Security architecture
  - Authentication mechanisms
  - What mechanisms are already in use?
  - Diagram trust boundaries
  - What are the entry points to the product?

Identify Threats

- Start to think like an attacker
  - Probabilistic attack scenarios
  - Threat identification based identified security events
  - Where are my vulnerabilities?
  - What are my vulnerabilities?
- Create a list of identified possible attacks
  - Make an assessment of the technical and business impact of the threat

Threats: System

Threat	Examples
Open port exploit	e.g. USB port access, allowing execution of unauthorized software
Protocol exploit	Any open entries into initial boot process can invalidate authentication
Arbitrary code execution	Buffer overflow
Unauthorized access	Using stolen login info
	Weak or no access restrictions
File disclosure	No encryption on high importance data

Threats: Network

Threat	Example
Denial of Service (DoS)	Packet floods,
Spying	Use packet sniffers on traffic in hunt for vital info, e.g.: login credentials
Spoofing	Modifying packets to spoof, e.g. network addresses
Info gathering	Detect network topologies, ports, connected computers

Threat Rating

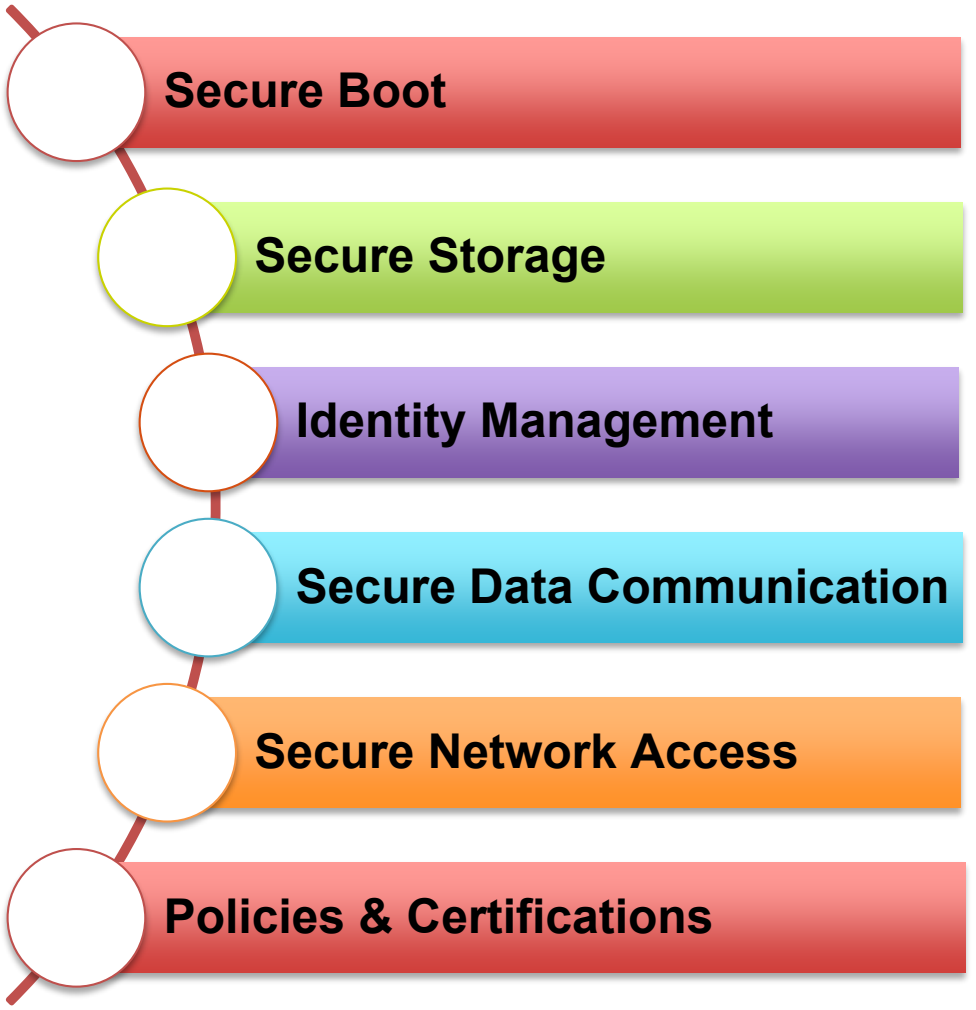
- CVSS scoring for reported CVEs
- DREAD rating for identified threats
  - **D**amage – How bad would an attack be?
  - **R**eproducibility – How easy is it to reproduce the attack?
  - **E**xploitability – How much work is it to launch the attack?
  - **A**ffected users – How many people will be impacted?
  - **D**iscoverability – How easy is it to discover the threat?

Threat	**D**	**R**	**E**	**A**	**D**	**Score**
Denial of Service – Network Case 2	5	3	3	6	7	4.8
Unauthorized access – Case 4	6	5	3	6	7	5.4

Device Security Layers

- Is the boot process secure?
  - Verification/Encryption
- Is the access to the storage device protected?
  - Physical and logical
- Do we know who accesses the device and what permissions they have?
- Are the data exchanged in a safe way?
- How is network access protected?
  - Protocols, verification mechanisms
- What additional hardware can be connected? Is it limited/controlled?
- Where and how are the certs and keys protected?

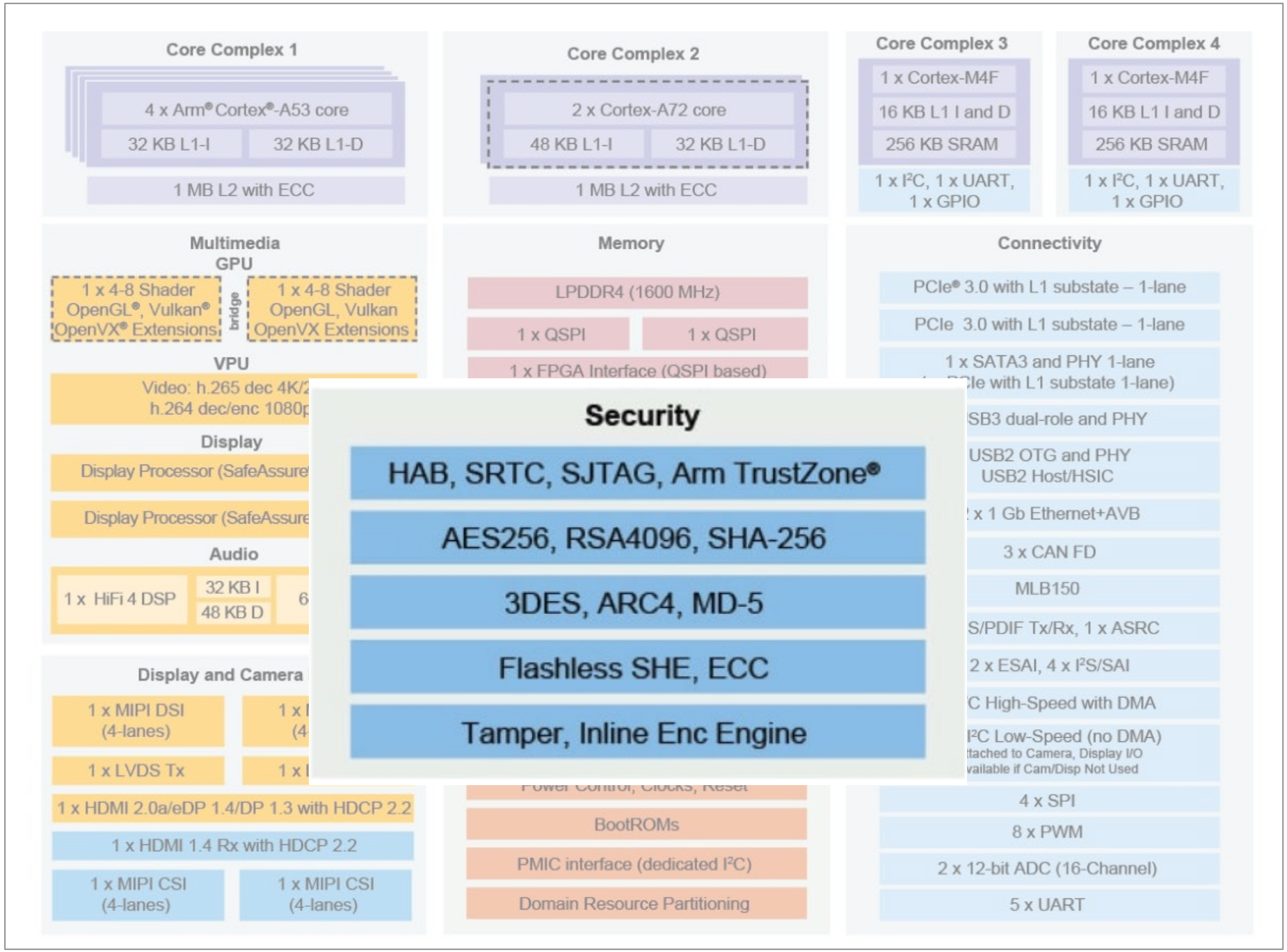
Specific list varies between different products



Hardware Selection

Processor Requirements

- Secure boot support
- Trusted Execution Environment
- Secure memory
- Secure key storage
- Key revocation
- Tamper detect
- Hardware acceleration for cryptographic
- Secure debug options
- Easy access to code signing tools and detailed documentation!

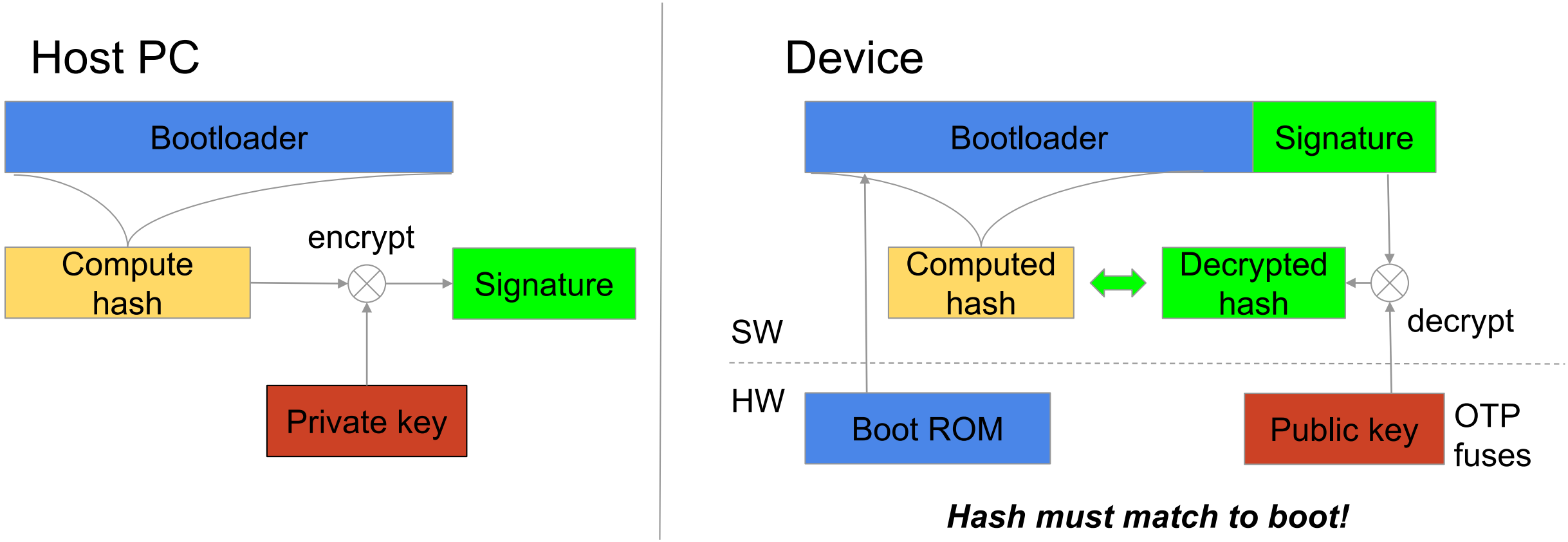

Core Complex 1
4 x Arm® Cortex®-A53 core
32 KB L1-I
32 KB L1-D
1 MB L2 with ECC
Core Complex 2
2 x Cortex-A72 core
48 KB L1-I
32 KB L1-D
1 MB L2 with ECC
Core Complex 3
1 x Cortex-M4F
16 KB L1 I and D
256 KB SRAM
1 x I²C, 1 x UART, 1 x GPIO
Core Complex 4
1 x Cortex-M4F
16 KB L1 I and D
256 KB SRAM
1 x I²C, 1 x UART, 1 x GPIO
Multimedia
GPU
1 x 4-8 Shader OpenGL®, Vulkan® OpenVX® Extensions
bridge
1 x 4-8 Shader OpenGL, Vulkan OpenVX Extensions
VPU
Video: h.265 dec 4K/
h.264 dec/enc 1080p
Display
Display Processor (SafeAssure
Display Processor (SafeAssure
Audio
1 x HiFi 4 DSP
32 KB I
48 KB D
Display and Camera
1 x MIPI DSI (4-lanes)
1 x LVDS Tx
1 x HDMI 2.0a/eDP 1.4/DP 1.3 with HDCP 2.2
1 x HDMI 1.4 Rx with HDCP 2.2
1 x MIPI CSI (4-lanes)
1 x MIPI CSI (4-lanes)
Memory
LPDDR4 (1600 MHz)
1 x QSPI
1 x QSPI
1 x FPGA Interface (QSPI based)
Power Control, Clocks, Reset
BootROMs
PMIC interface (dedicated I²C)
Domain Resource Partitioning
Connectivity
PCIe® 3.0 with L1 substate – 1-lane
PCIe 3.0 with L1 substate – 1-lane
1 x SATA3 and PHY 1-lane
le with L1 substate 1-lane)
SB3 dual-role and PHY
USB2 OTG and PHY
USB2 Host/HSIC
x 1 Gb Ethernet+AVB
3 x CAN FD
MLB150
S/PDIF Tx/Rx, 1 x ASRC
2 x ESAI, 4 x I²S/SAI
C High-Speed with DMA
I²C Low-Speed (no DMA)
ttached to Camera, Display I/O
vailable if Cam/Disp Not Used
4 x SPI
8 x PWM
2 x 12-bit ADC (16-Channel)
5 x UART
Security
HAB, SRTC, SJTAG, Arm TrustZone®
AES256, RSA4096, SHA-256
3DES, ARC4, MD-5
Flashless SHE, ECC
Tamper, Inline Enc Engine

Software Integrity

Secure Boot and Chain of Trust

- Provides
  - Authentication (unauthorized images not allowed to run)
  - Integrity (authorized images can not be ‘tampered’ with)
- Digital signatures for authentication
  - Private key -> used for signing
  - Public key -> used to verify

Secure Boot Flow

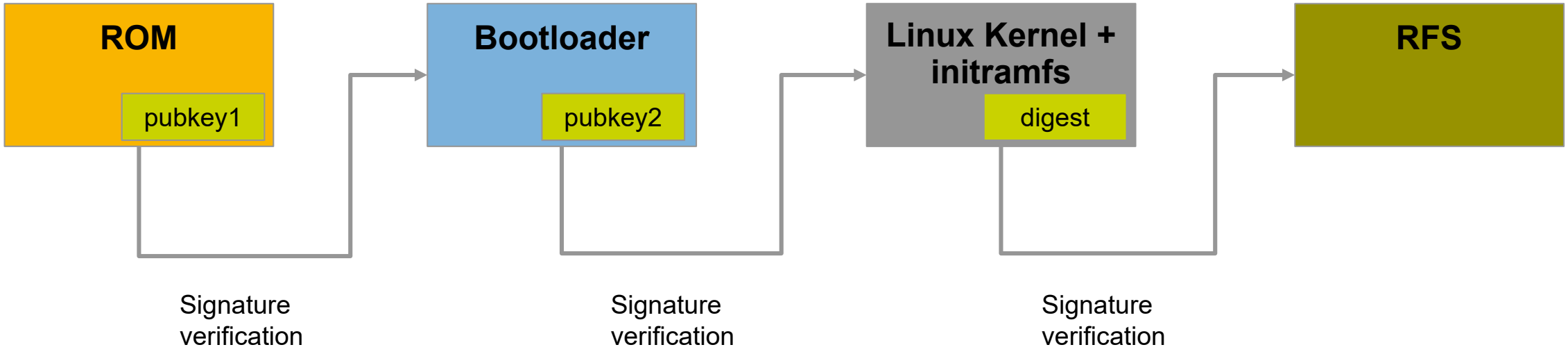


Secure Boot Steps on i.MX (Overview)

- Create private/public key pairs
- Burn the public key hash to OTP
- Enable secure boot option in U-Boot config
- Sign bootloader using code signing tools provided by NXP
- Test and boot using signed image
- Close configuration (irreversible step)
  - Manufacturing tool images need to be signed

Chain of Trust

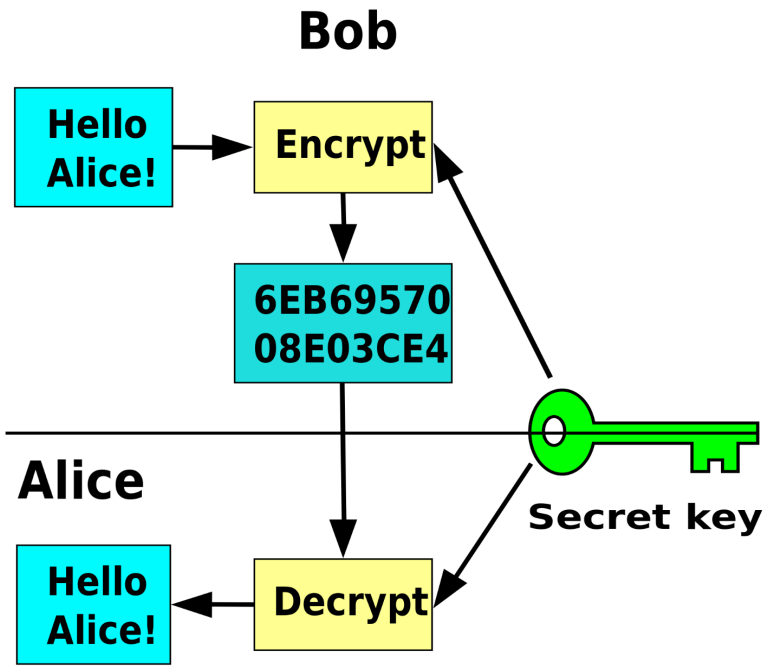
- Extend the authentication to all layers in the software stack



Data Protection

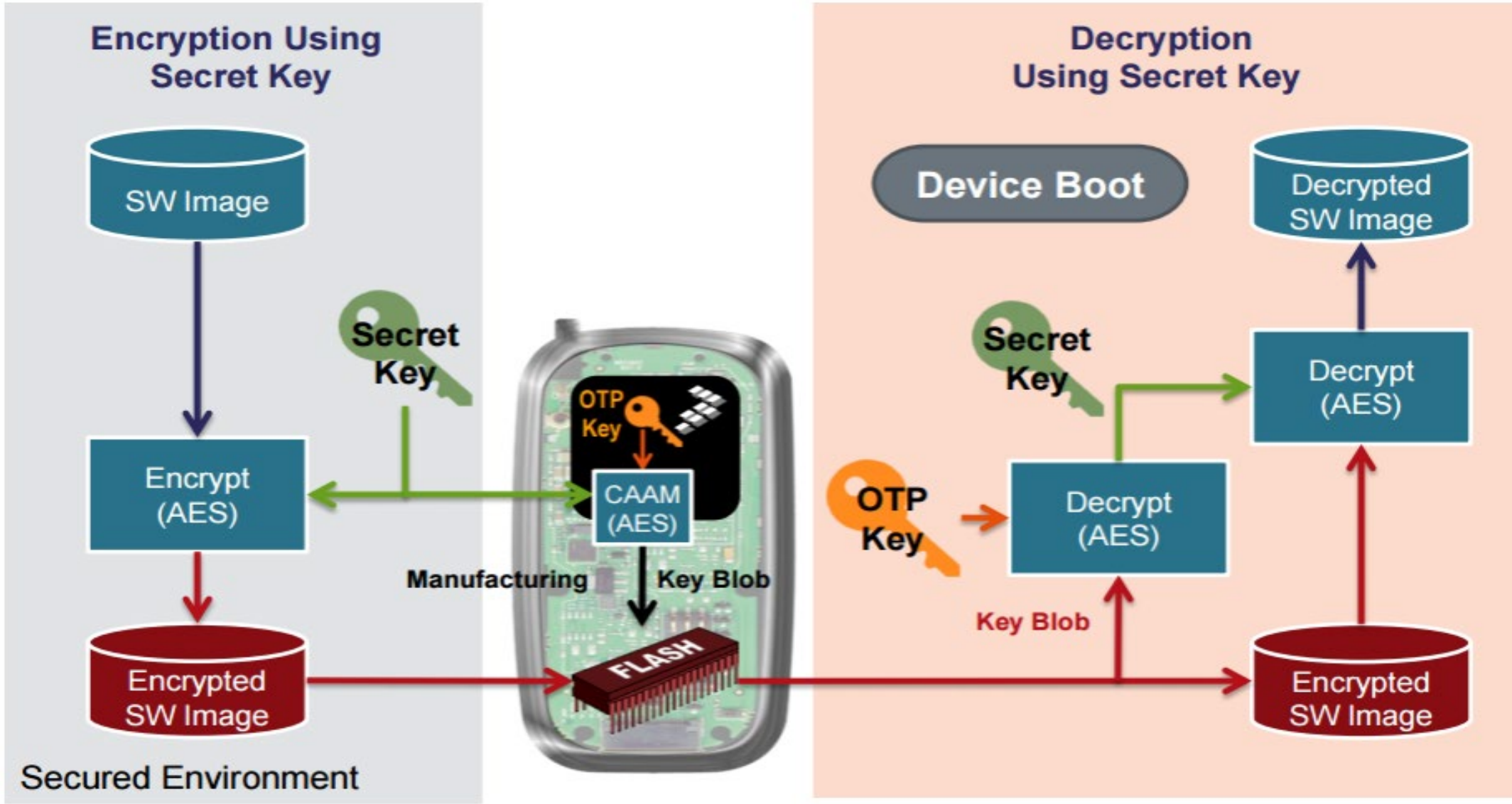
Secure Boot + Encryption

- Uses symmetric key cryptography
  - Same key used for encryption and decryption
- Provides
  - Confidentiality
  - IP Protection (Anti-cloning)
    - Key needs to be unique per device
- Identify what to protect
  - Bootloader, kernel, RFS, select applications?
  - Affects firmware update design



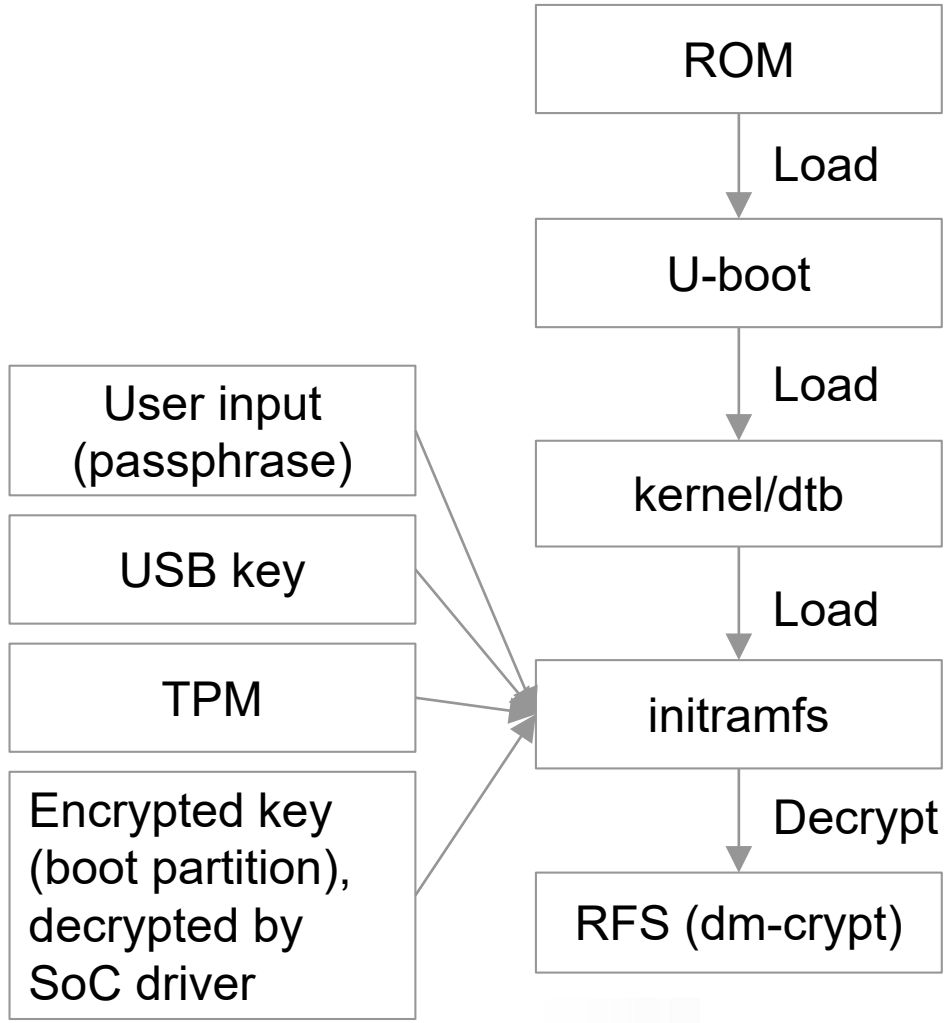
How do we encrypt the system?

i.MX CAAM Assisted Encrypted Boot Flow



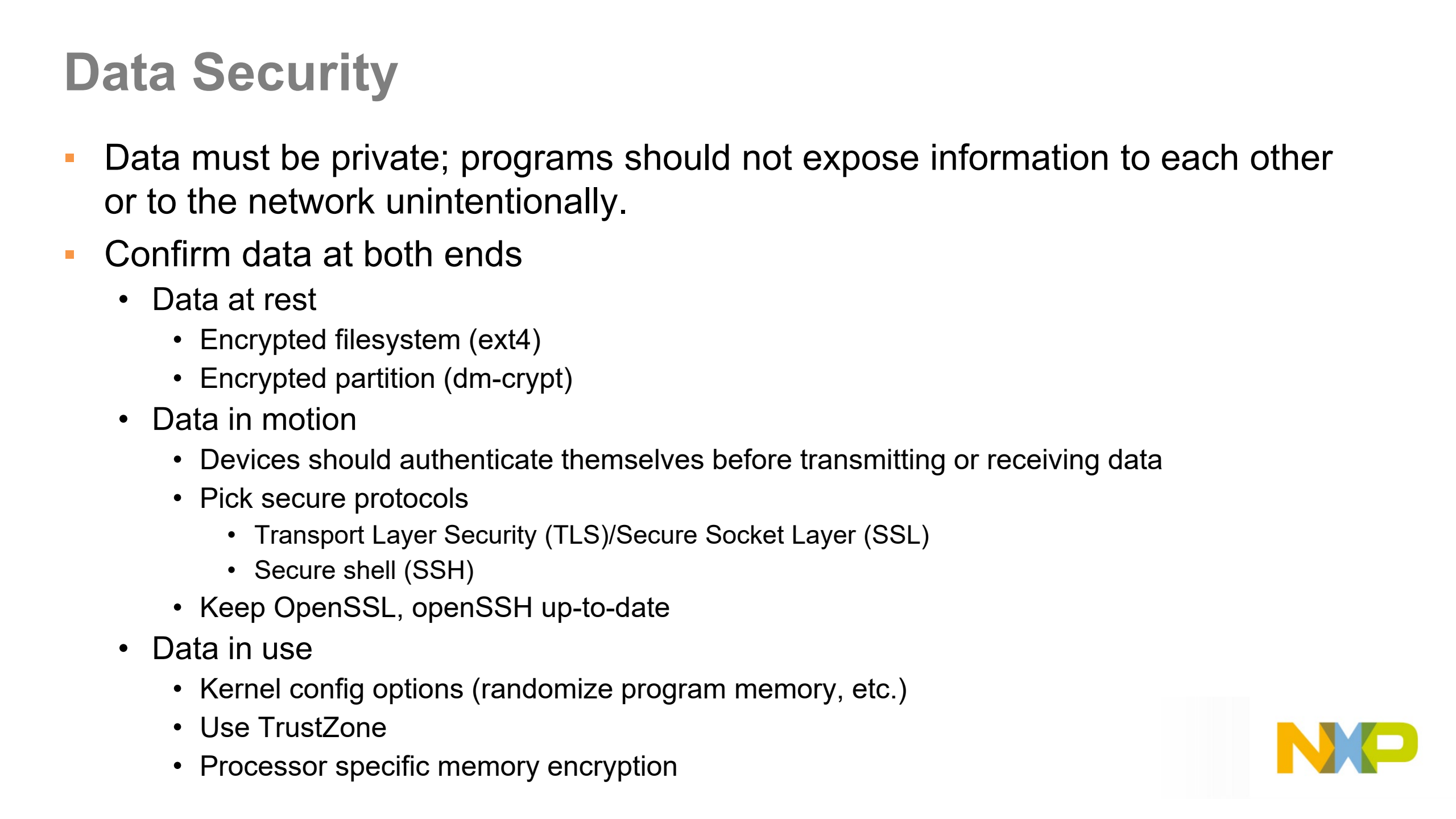
Encryption with dm-crypt

- Block level
- Option for RFS encryption or partitions
  - Key stored outside RFS
- Supported on all major distros (Debian, Ubuntu) and Android
- Easy setup
- Key management on embedded system tricky
  - Needs a unique hardware ID/key



Data Security

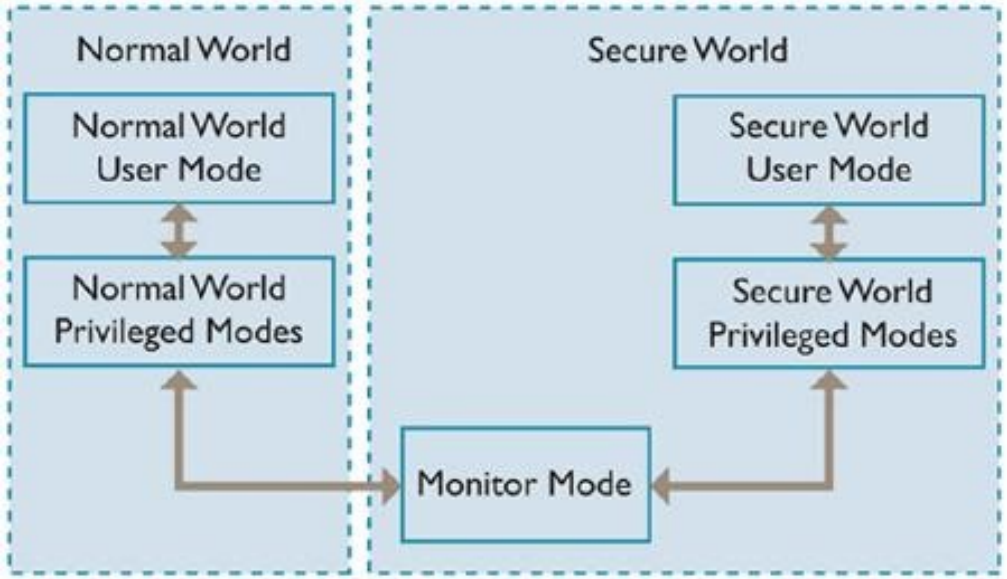
- Data must be private; programs should not expose information to each other or to the network unintentionally.
- Confirm data at both ends
  - Data at rest
    - Encrypted filesystem (ext4)
    - Encrypted partition (dm-crypt)
  - Data in motion
    - Devices should authenticate themselves before transmitting or receiving data
    - Pick secure protocols
      - Transport Layer Security (TLS)/Secure Socket Layer (SSL)
      - Secure shell (SSH)
    - Keep OpenSSL, openSSH up-to-date
  - Data in use
    - Kernel config options (randomize program memory, etc.)
    - Use TrustZone
    - Processor specific memory encryption



Trusted Execution Environment (TEE)

- Why is TEE needed?
  - It provides an isolated environment to ensure code/data integrity and confidentiality
  - involves a tiny security oriented operating system TEE OS
  - trusted applications
- Several implementations available
  - OP-TEE, Trusty, QSEE, etc.
- Runs alongside a Rich OS (Linux distro, Android)
  - Provides services to apps on Rich OS
- Isolated environment
  - Code/data confidentiality & integrity
- Policy in software, enforced by hardware
  - Arm TrustZone (memory/IO protection based on state)

TEE architecture on Arm



(Image credit: ARM)

Hardening

Security Hardening Overview

- Reduces attack surface
- Enabling security measures to reduce risk of being compromised
- Implemented at various levels
  - Hardware
  - Bootloader
  - Kernel
  - User space
  - Installation environment

Hardware

- Disable JTAG or enable secure JTAG
- Disable access to serial console
- Enable tamper protection
- Enable secure memory options
  - DDR encryption

Reducing Attack Surface

- Disable default passwords, root logins
- Disable weak protocols, cryptographic algorithms and/or key lengths
- Prune Certificate Authority (CA) list to few trusted
- Minimize packages, features in a package to essential only
- Review security config of any software that opens an external port
  - Eg: Disable password based ssh, run on non standard port, etc.
  - Eg: Disable querying time on ntpd, etc.
- Disable any unused ports and services
- Allow only approved hardware
  - Disable automount of USBs based on VID, etc.
- Remove development and/or manufacturing keys from production units

Enable Security Measures

- Firewalls
- Kernel config options (randomize program memory, etc.)
- User and/or process permission review (Access Control List)
  - Discretionary Access Control (e.g.: file / user permissions)
  - Mandatory Access Control (e.g.: SELinux, AppArmor)

Security Best Practices

- Pick LTS releases when possible
- Pick secure protocols (TLS, SSH)
- Limit access to signing keys
- Define a security test plan
  - Negative test cases: Rejects unsigned image, rejects expired certificate, etc.
- Run static analysis tools
- Perform a security audit of the device
  - Pen testing

Secure Upgrades

OTA

- Integral part for keeping the device secure overtime
- Considerations
  - Automatic fallback if firmware upgrade fails
  - Power-failure safe
  - Upgrade strategies
    - A/B partitioning scheme, image based update
    - Delta image upgrades
- Security of OTA is essential

OTA Security

- Server authentication
- Secure OTA image transfer to devices
- Signature verification of signed image bundle
- Image decryption with locally stored keys
- Individual image signature verification
  - Verified as part of Boot / Chain of Trust
- Disallow version downgrades
  - Prevents older software with vulnerabilities from being exploited (e.g.: jailbreak)

On-going Security

Security Needs to Be On-going

- Vulnerability monitoring
  - NXP Vigiles (NVD, Debian, bulletins, mailing lists), review and patch applicable software
    - Integrated with NXP Yocto BSPs
    - Available for Buildroot and other build systems
- Remediation
  - Defined vulnerability management process: Defined periodic security releases
  - NXP BSP Maintenance services: Offload the burden of maintaining OS security
- Monitoring for breach
  - Audit logs of unauthorized access
  - Key revocation
- Vulnerability disclosure program
  - Notify end users

Common Product Security Pitfalls

Security Pitfalls

- Not gathering security requirements upfront
- Failing to address security in design stages to meet end-product requirements and certifications
- Reinventing the wheel instead of using vetted open source solutions
- Incorporating third-party software with unknown vulnerability exposure
- Not adequately hardening devices
- Failing to secure devices at manufacturing
- Running ineffective security maintenance and update processes
- Not having a defined security incident response strategy

Upcoming Webinars

In-depth Dive

November 5	**Software Integrity and Data Confidentiality:** Establishing Secure Boot and Chain of Trust on i.MX Processors
December 10	**Trusted Execution Environment:** Getting Started with OP-TEE on i.MX Processors
January 21	**Linux Kernel Security:** Overview of Security Features and Hardening
February 25	**Security Hardening:** Protecting Your Embedded Linux Device from the Risk of Being Compromised
April 6	**Designing OTA Updates:** An Integral Part of a Secure System

For More Information and to Become More Secure

Timesys is an embedded linux security expert, please contact us to discuss your project sales@timesys.com

Use this link to go to the Services for Securing your device

Thank You!

SECURE CONNECTIONS
FOR A SMARTER WORLD