# Secure Execution

Lecture 10

# Secure Execution

- ARM TrustZone
  - Memory Attributes
  - Bus Attributes
- Trusted Firmware
- OTP

# Security Extension

ARM TrustZone

# Bibliography

for this section


**Joseph Yiu**, *The Definitive Guide to ARM® Cortex®-M23 and Cortex-M33 Processors*

- Chapter 7 - *TrustZone support in the memory system*
  - Section 7.1 - *Overview*
  - Section 7.2 - *SAU and IDAU*
  - Section 7.5 - *Memory protection controller and peripheral protection controller*

**Raspberry Pi Ltd**, *RP2350 Datasheet*

- Chapter 10 - *Security*
  - Section 10.2 - *Processor Security Features (Arm)*

# Secure Execution Mode

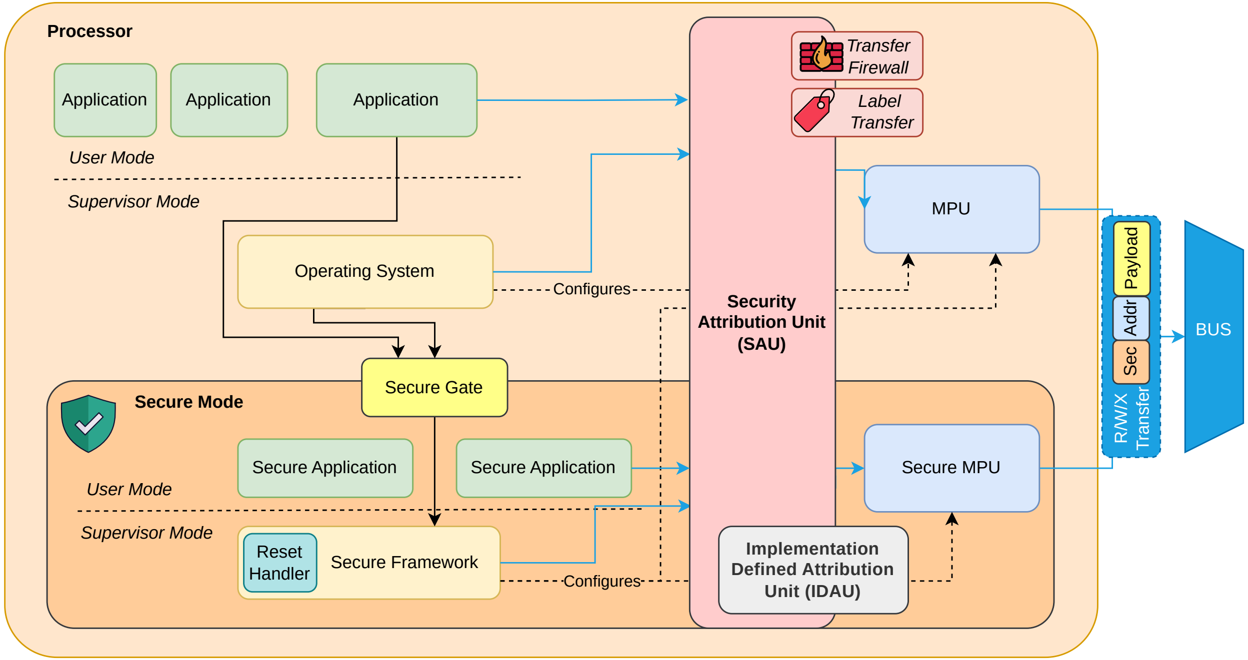
*two execution modes*

- **Secure**
  - unprivileged (user)
  - privileged (supervisor)
- **NonSecure**
  - unprivileged (user)
  - privileged (supervisor)

*memory attributes*

- each bus transfer has an attribute label

*secure gates*



# Memory Attributes

each memory region is labeled with one of the attributes

| Type | Symbol | Description | *Transfer* Attribute |
|---|---|---|---|
| *Secure* | **S** | can be accessed only by code running in **secure mode** | secure |
| *Non Secure Callable* | **NSC** | code running in **non-secure mode** can make function calls into it with some restrictions | non-secure |
| *NonSecure* | **NS** | any code running in **any mode** can access it | non-secure |
| *Exempt* | **E** | any code running in **any mode** can access it (with no execution) | *executing code mode* |

> bus transfers are labeled base upon the execution mode and memory attribute

# Implementation Defined Attribution Unit (IDAU)

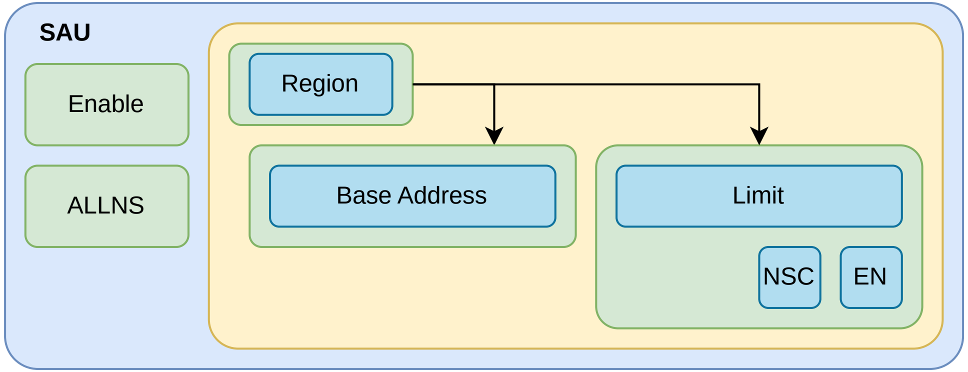
hard wired by the microcontroller's manufacturer

RP2350's IDAU setup

| Start Address | End Address | Region | Access |
|---|---|---|---|
| `0x00000000` | `0x000042ff` | Arm boot | Exempt |
| `0x00004300` | `0x00007dff` | USB/RISC-V boot | Non-secure (instruction fetch), Exempt (load/store) |
| `0x00007e00` | `0x00007fff` | BootROM SGs | Secure and Non-secure-Callable |
| `0x10000000` | `0x1fffffff` | XIP | Non-secure |
| `0x20000000` | `0x20081fff` | SRAM | Non-secure |
| `0x40000000` | `0x4fffffff` | APB | Exempt |
| `0x50000000` | `0x5fffffff` | AHB | Exempt |
| `0xd0000000` | `0xdfffffff` | SIO | Exempt |

# Security Attribution Unit (SAU)

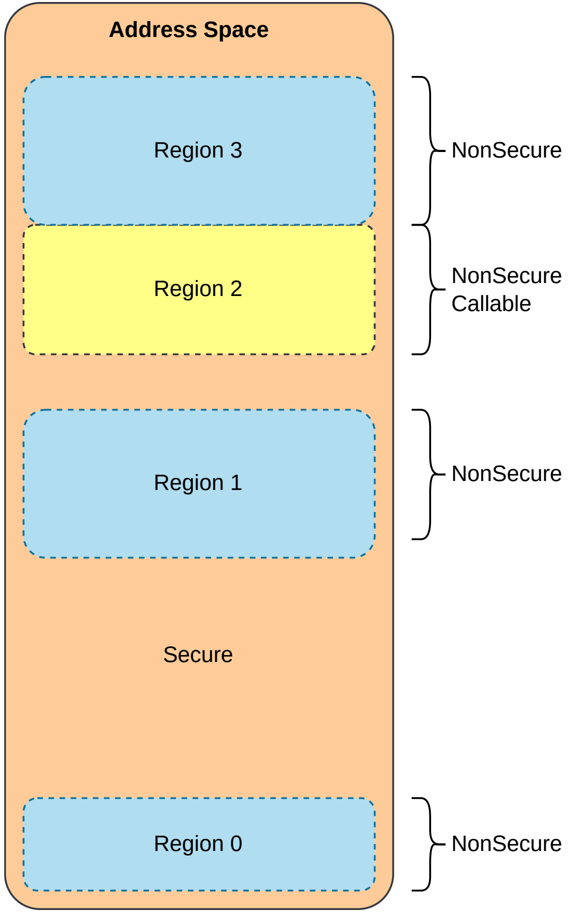
software defined



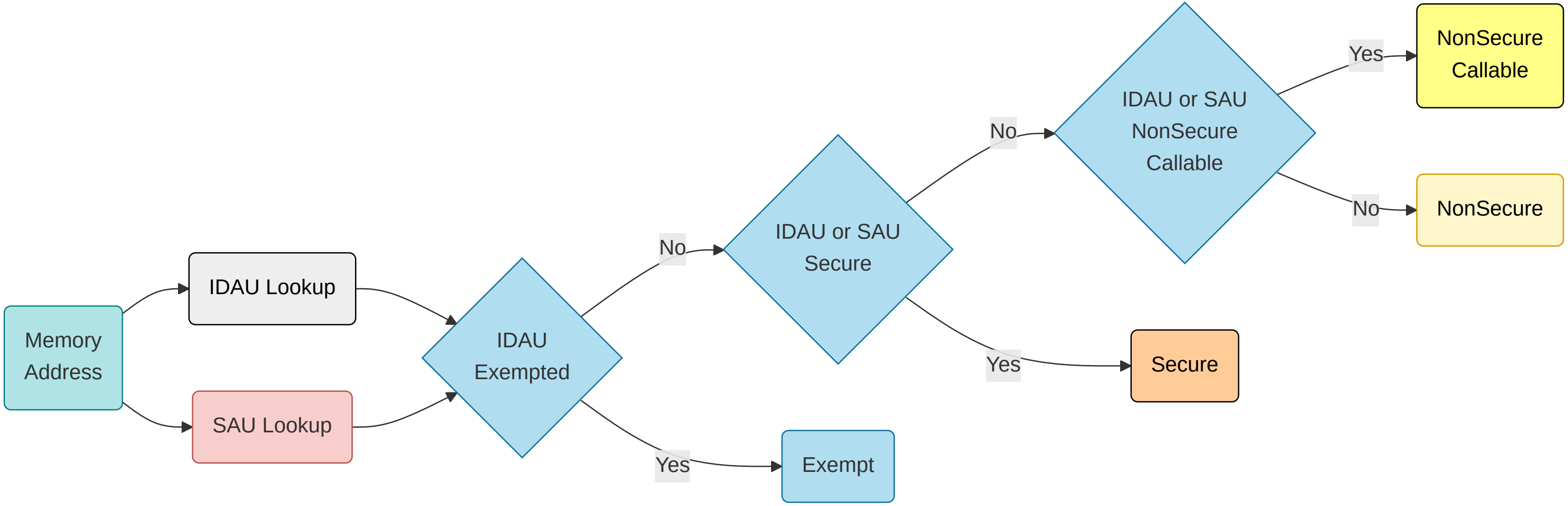

- allows the definition of maximum 8 *memory regions*
- regions cannot overlap
- regions have access permissions (similar to rwx)

$$region\_size = 32 \times N$$

$$base\_address = 32 \times N$$



# Address Attribute Resolution

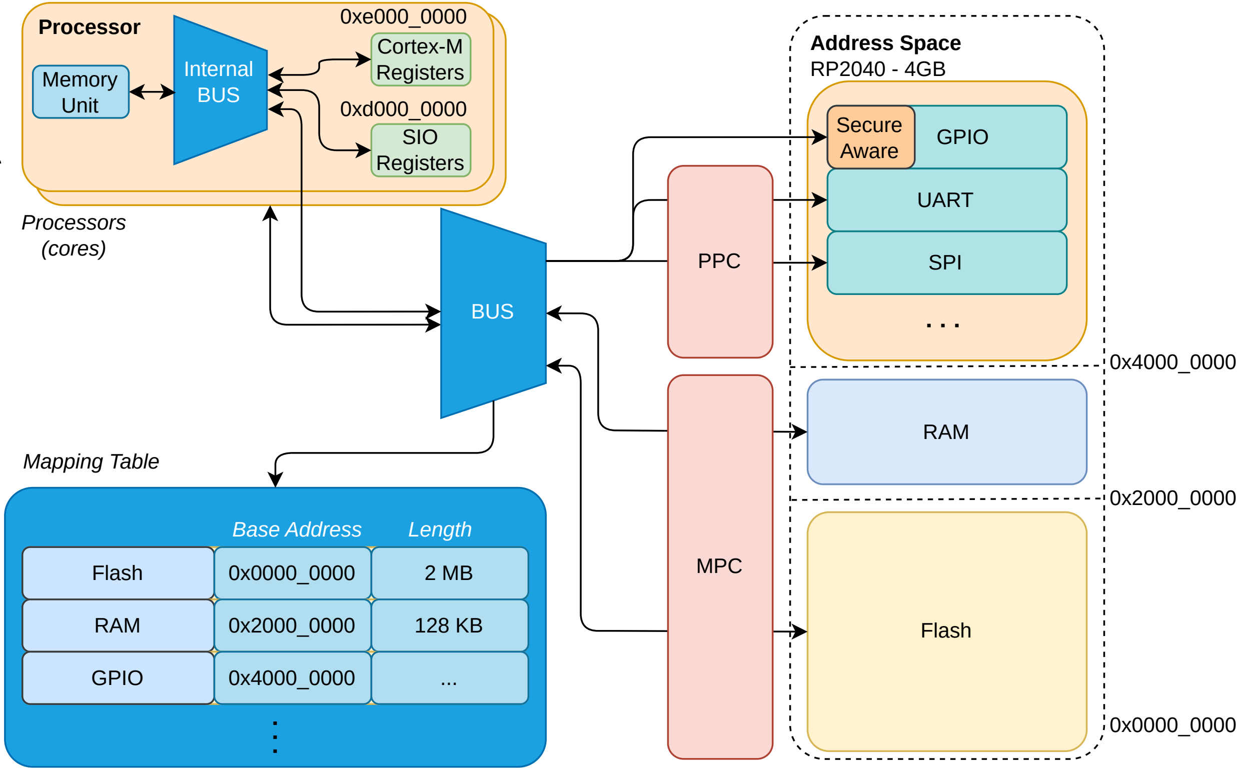



Attributes from IDAU and SAU are merged, using the most restrictive.

# The Bus

secured

1. **Memory Controller** asks for data transfer or instruction fetch
2. **IDAU** and **SAU** determine the access attributes
3. **External Bus Routes** the request
   1. **MPC** for RAM or Flash
   2. Secure Aware Peripheral
   3. **PPC** for Non Secure Aware peripherals



# Memory Protection Controller (MPC)

**optional** - depends on vendor

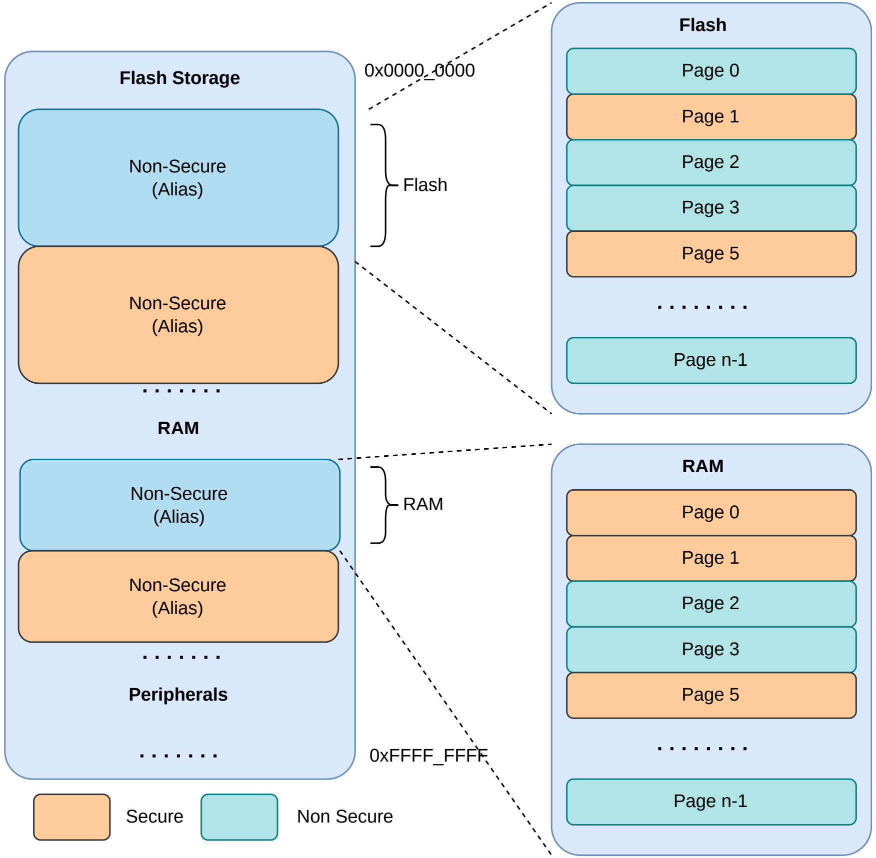
RAM and Flash are aliased - they both appear at two different addresses

- one alias defined (in IDAU and SAU) as **NonSecure**
- one alias defined as **Secure**

RAM and Flash are split in pages

- usually 256 B or 512 B
- each page is defined as **NonSecure** or **Secure**

The two aliases have page holes in them.



# Peripheral Protection Unit

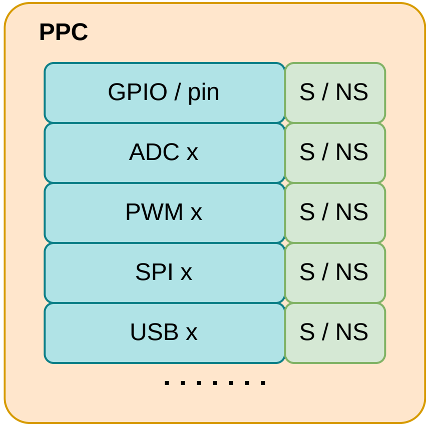
protects peripherals that are not *secure aware*

**optional** - depends on vendor

Each peripheral is marked as **NonSecure** or **Secure**.

- this includes interrupts that are fired

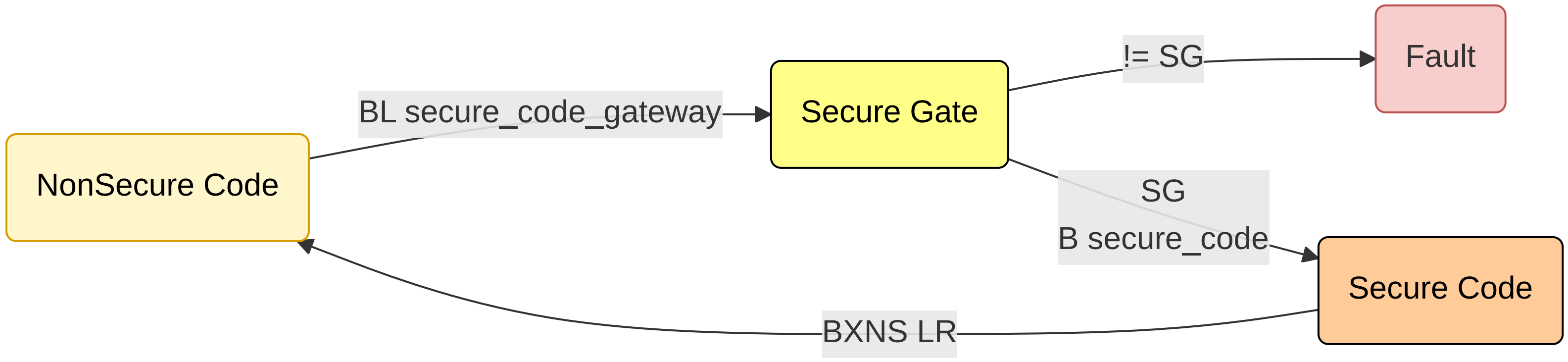
*may be implemented similar to the MPC*



# Switching modes

Calling *Secure API* from Non Secure code

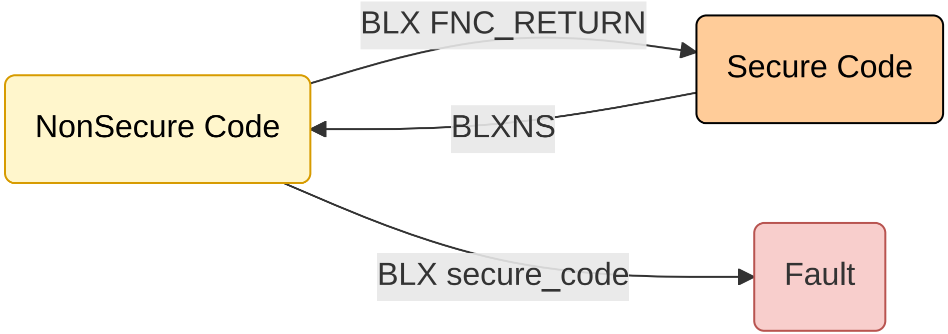
- **Secure** code's compiler defines a *secure gateway entry* point in **NonSecure Callable** memory for every function that can be called from **Non Secure**
- **NonSecure** code calls the *secure gateway entry* point for the API
  - the instruction there has to be `SG`
  - the next instruction is the call to the actual API function
- **Secure** code returns using the `BXNS` instruction



# Switching modes

Calling *NonSecure functions* from Secure code

- **Secure** code calls the **Non Secure** function using `BLXNS`
  - the processor stacks the return address (linked address) and jumps to the function
- **NonSecure** code returns using the `BX FNC_RETURN` instruction
  - `FNC_RETURN` is a value in `LR` when the function starts



# Secure Execution in Rust

unstable feature, use nightly version

Define a function that can be called from **Non Secure** code

```rust
#![feature(cmse_nonsecure_entry)]

#[no_mangle]
#[cmse_nonsecure_entry]
pub extern "C" fn entry_function(v: u32) -> u32 {
    v + 6
}
```

Limitations

- parameters can only be sent via registers, non secure code has no access to the secure stack
- uses C ABI

# Secure Execution in Rust

unstable feature, use nightly version

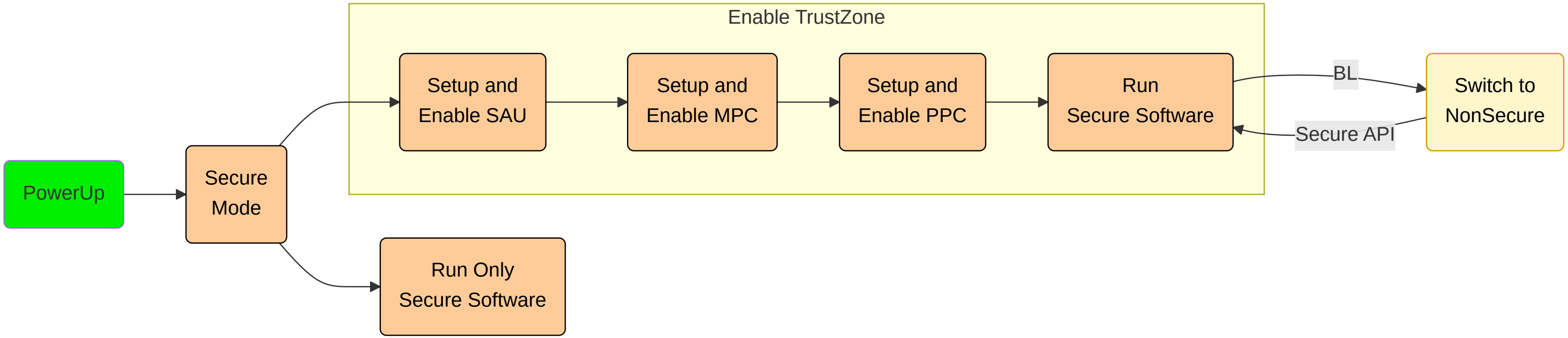
Call a **Non Secure** function

```
#![feature(abi_c_cmse_nonsecure_call)]
#![no_std]

unsafe extern "C-cmse-nonsecure-call" non_secure_function(u8, u16, u32) -> f32;

fn run() {
  unsafe { non_secure_function(1, 100, 300) };
}
```

Limitations

- parameters should only be sent via registers, secure code should not access the non-secure stack
- uses C ABI

# Boot




The processor starts in **Secure** mode

If it enables *SAU* it can switch to **NonSecure** mode

# Trusted Firmware

# Bibliography

for this section


**Raspberry Pi Ltd**, *RP2350 Datasheet*

- Chapter 10 - *Security*
  - Section 10.1 - *Overview (Arm)*
- Chapter 13 - *OTP*

**ARM**, *Trusted Firmware-M Documentation*

- *Introduction*
- *Getting Started*
- *Security*

# Trusted Firmware-M

what it does

- Secure Boot
- Secure Update
- Secure API

Requires

- ARM microcontrollers that provide TrustZone
- Examples
  - STM32L5, STM32U5
  - RP2350

# Secure Firmware / Bootloader

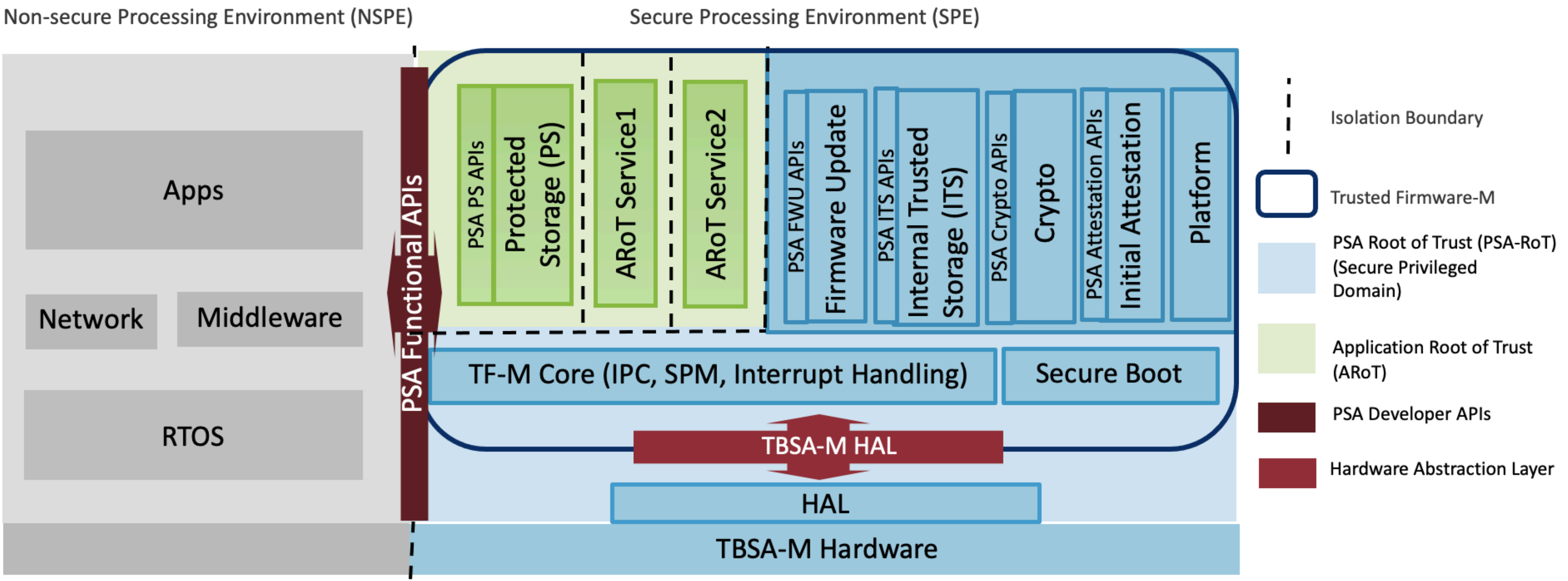
provided by the vendor

Depends on the MCU

- implements the TF-M standard (Trusted Firmware - Cortex M)
- certification levels 1 - 3
  - **Level 1**: Software-based isolation; foundational crypto, attestation, and secure boot.
  - **Level 2**: Adds protection against non-invasive attacks.
  - **Level 3**: Adds protection against side-channel and invasive attacks; often requires hardware features like tamper detection and secure key storage.

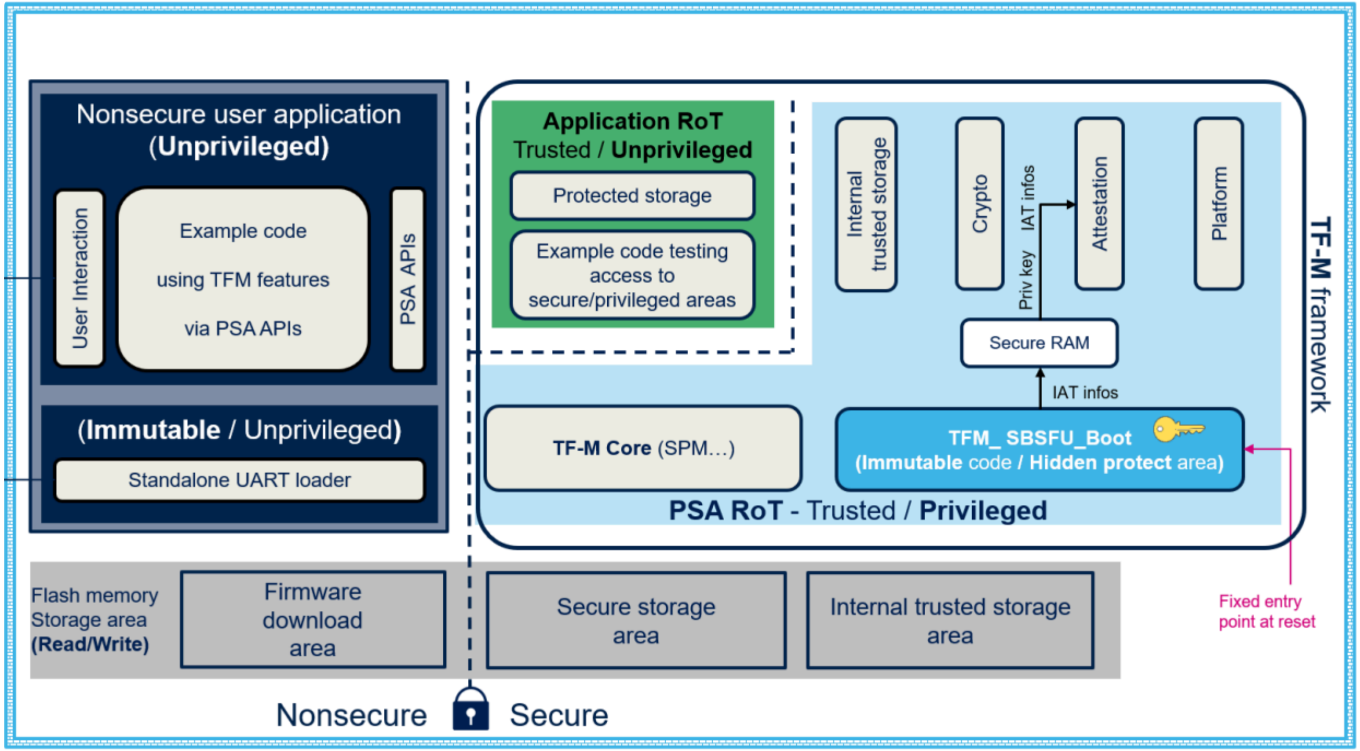
# ARM TF-M Reference Implementation

open source



# STM32 Implementation

Provided in SDK



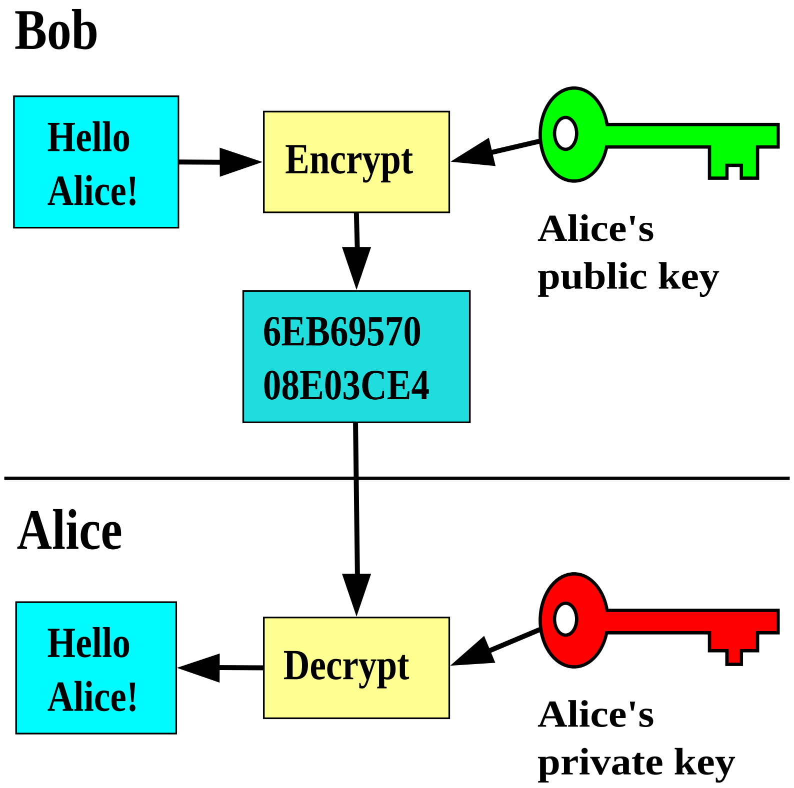
# RP2350

- provides a ROM bootloader:
  - Secure Boot
  - Secure Update
  - Try-before-you-by
  - A/B partitioning
  - Rollback

# Public Key Infrastructure

*key pair*

- private key 🔐
- public key 🔓
- algorithms
  - Rivest–Shamir–Adleman (*RSA*)
  - Elliptic Curves (*ECS*)
- hashing function
  - SHA 256



# Digital Signatures

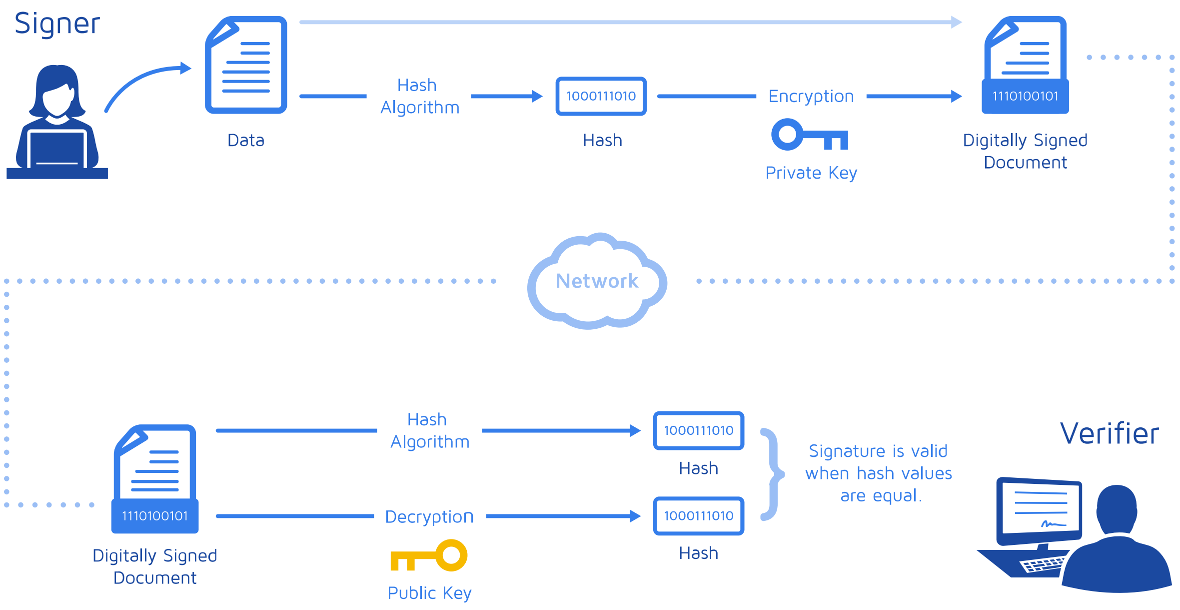
needs a *key pair* (RSA or ECS) and a *hashing algorithm*

Signing

1. data is **hashed**
2. the **hash** is **encrypted** using the **private key**
3. the **encrypted hash** is added to the data

Verifying

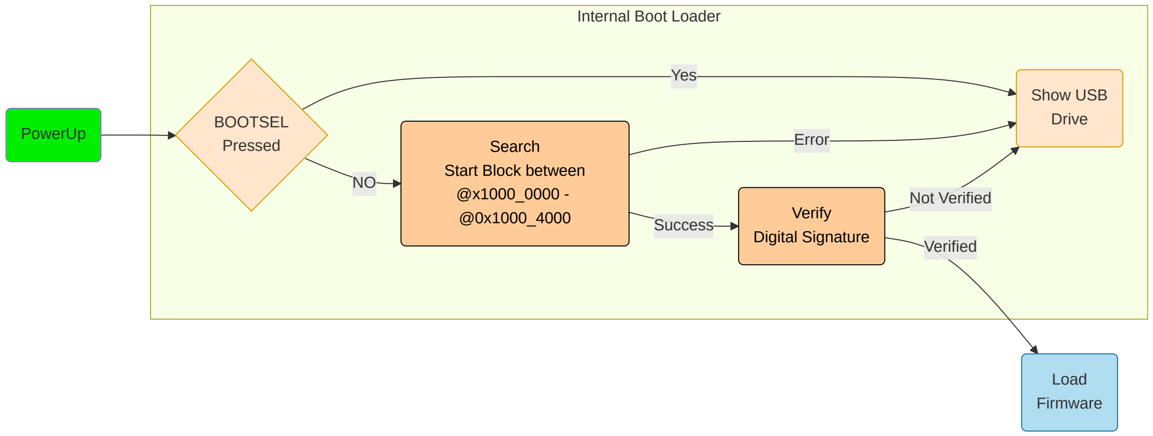
1. data is **hashed**
2. the **encrypted hash** is decrypted using the **public key**

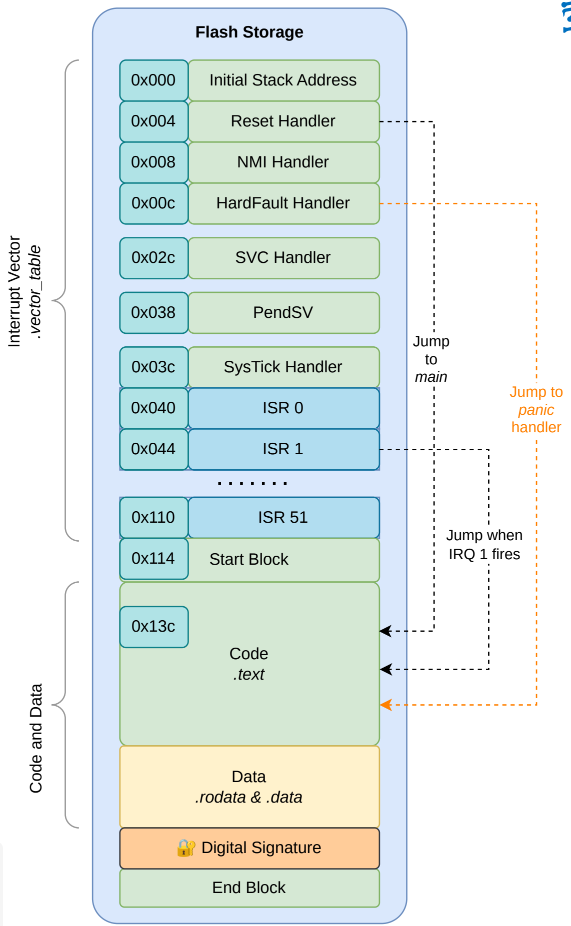


# Signed Firmware

The firmware contains a digital signature

- `.vector_table`
- `.start_block` and `.end_block`
- `.text` and `.data`




* drawing is not at scale, code and data are significantly greater than the interrupt vector

> RP2350 has a bootloader that knows how to securely boot, other chips need custom secure firmware

# OTP

one time programmable

# OTP

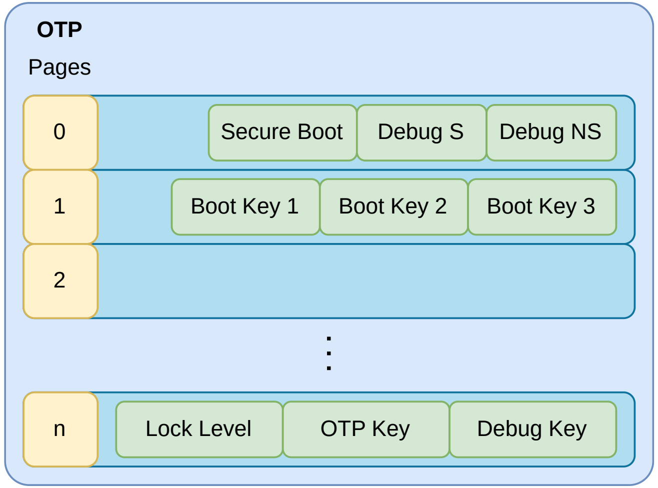
flash memory that can be programmed only once

- Usually has three lock levels
  - Read/Write - works as normal flash
  - Read Only - works as ROM
  - Inaccessible - cannot be accessed
- The lock is not reversible
- Different vendors have different naming for these levels

# Information in OTP

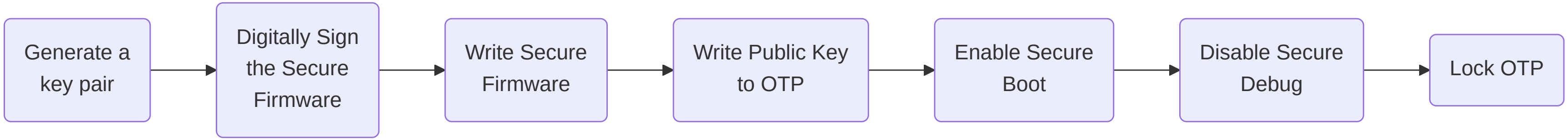
Stores information that:

- should not be modifiable
- should not be read *from the outside* using a debugger or using **Non Secure** software that reads and sends the information
- Secure Boot Enabled
- Debug in **Secure** mode Enabled
- Debug in **NonSecure** mode Enabled
- Bootloader's public keys
- Bootloader's public keys
- OTP's Pages Lock Level
- OTP's (read) key
- Debug key
- Secure Access Permissions



# Provisioning Devices

how to securely provision a new device for production




- Generate a different key pair for every device
  - store the private key securely
- Disabling debugging in secure mode will prevent any debugger from reading the OTP with the stored key
- Locking the OTP will prevent any writes to the key
- Enabling Secure Boot will prevent any unsigned Secure Firmware update
- **NonSecure** debug is still available, but it cannot replace the **Secure** Firmware
- Flashing **NonSecure** firmware is still possible

# Decommissioning Devices

- Add the capability to the **Secure** firmware to increase the Lock Level to OTP
  - this will render OTP unusable
  - the system will not boot anymore as it cannot read the public keys
- Some OTP memories allow reversing locks:
  - they erase all the OTP
  - they erase the whole Flash

> This prevents reading the keys and secure firmware as secure debug becomes available

# Use Cases

- POS devices
  - payment software should not be tempered with
- Smart Cards
  - keys should never be read from the device
  - software in these devices should not be tampered with
  - JavaCard (applets are uploadable)

# Conclusion

- ARM TrustZone
  - Memory Attributes
  - Bus Attributes
- Trusted Firmware
- OTP