



# SRAM PUF for secure firmware updates of IoT devices

*Noemie Beringuier-Boher, Roel Maes, Nicolas Moro, Sander Steeghs-Turchina, Peet van Tooren, Andries Stam*

PHISIC 2025 Workshop, May 20-21, 2025

# Introduction

- This work is part of the Resilient Trust project and is a collaboration between Almende and Synopsys
- EU's Horizon Europe research and innovation program under grant agreement No. 101112282
- Resilient Trust in short:
  - Increase SMEs security
  - 25 partners
  - 4 Use Cases
  - Started in Oct. 2023, until October 2026
- Part of this work was done by Rehan Malak while still at Synopsys



Develops the SRAM PUF IP



Develops Crownstone, a smart power outlet

# Agenda

- Context and motivations
  - Firmware Over-The-Air Updates
  - Typical Secure Boot Flow
- SRAM PUF Overview
  - What is a PUF
  - Synopsys PUF-Software
- Improved Secure Boot Flow (with SRAM PUF)
  - Chosen approach
  - Performances and code size metrics
- Secure Boot Flow for Crownstone IoT devices

# Context and motivations

- Firmware Over-The-Air Updates
- Typical Secure Boot Flow

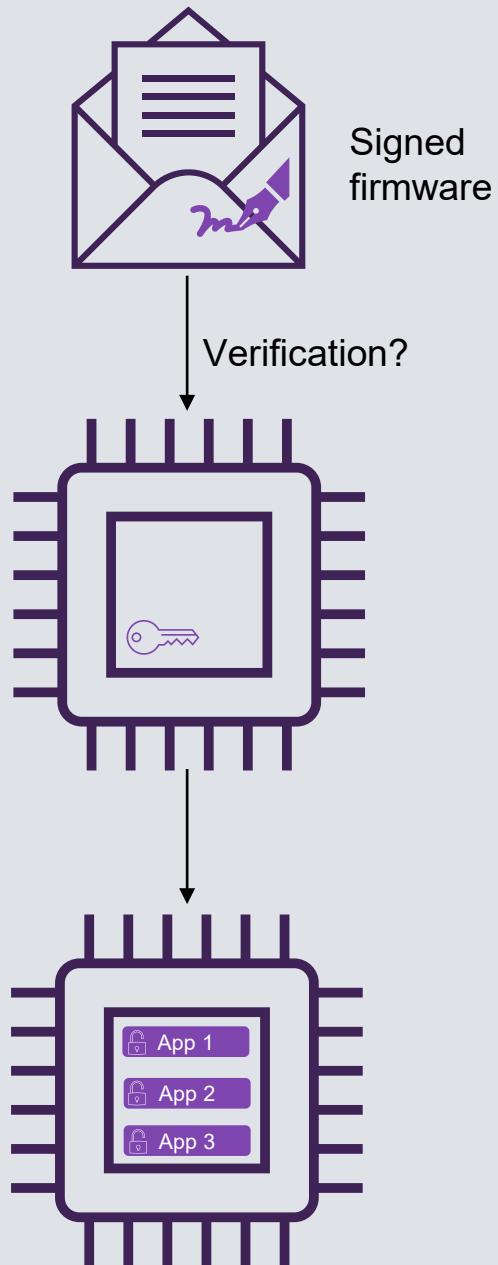
# Context and motivations

- Over-the-air firmware updates
- Network of resource-constrained IoT devices
  - E.g. Arm Cortex-M0+ or Cortex-M4
  - Small quantity of RAM
- Need to protect against:
  - Malicious firmware updates
  - IP theft
  - Device cloning

→ **Secure boot**



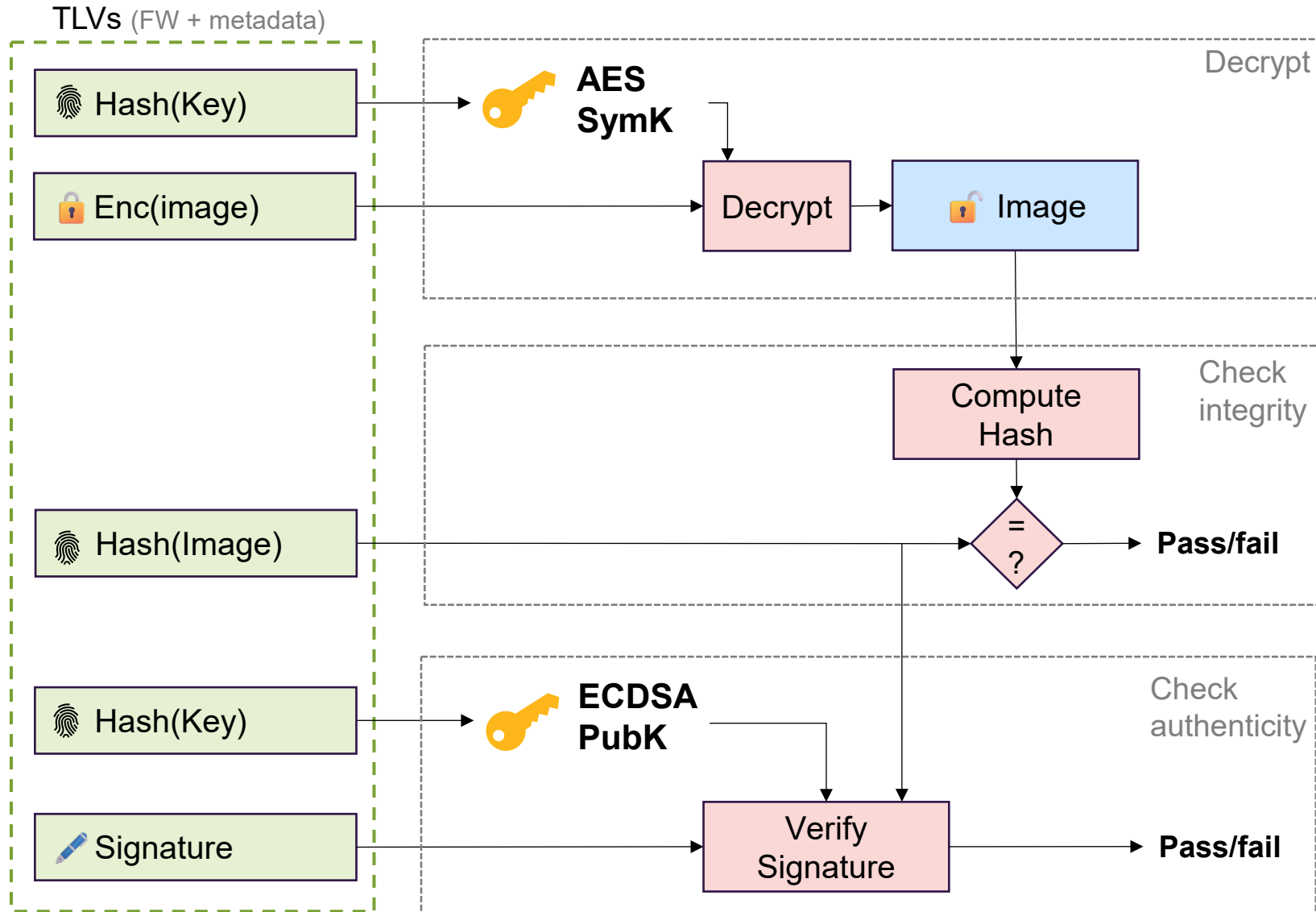
# Secure Boot



- Chain of trust
- Ensures
  - Authenticity (firmware signature)
  - Integrity (firmware hashing)
  - Confidentiality (firmware encryption) of a firmware image
- Critical part of a security architecture
- **MCUboot**
  - Open-source secure bootloader for 32-bits microcontrollers
  - Reference bootloader for the ARM PSA framework
  - Provides support for firmware upgrades
  - Configurable, many options
  - Uses Mbed TLS or Tinycrypt as a crypto library (we chose Mbed TLS for this work)

# Typical Secure Boot Flow

MCUboot flow with encryption and ECDSA signature



1. Decrypt the firmware
2. Verify the hash
3. Verify the signature

- 2 main security assets:
  - AES firmware encryption key
  - ECDSA signature public key

# Main limitations

On a standard MCU system, without a secure enclave



- Devices can easily be cloned  
→ Just dump the Flash, which contains the public key
- Encrypted firmware images can easily be decrypted  
→ The firmware symmetric key is stored in plain in Flash
- Verifying an ECDSA signature **takes a lot of time**  
→ E.g. on a Cortex-M0+ (STM32G070RB), with a basic unencrypted “Hello world” firmware payload

Scenario	Time to boot	Time (at 64 MHz)	Code size
<b>Mbed TLS</b> Standard ECDSA flow	~62M clock cycles	~1s	43 kB

→ **Can we improve this ?**



# Main limitations

On a standard MCU system, without a secure enclave



- Devices can easily be cloned  
→ Just dump the Flash, which contains the public key
- Encrypted firmware images can easily be decrypted  
→ The firmware symmetric key is stored in plain in Flash
- Verifying an ECDSA signature **takes a lot of time**  
→ E.g. on a Cortex-M0+ (STM32G070RB), with a basic unencrypted “Hello world” firmware payload

Scenario	Time to boot	Time (at 64 MHz)	Code size
<b>Mbed TLS</b> Standard ECDSA flow	~62M clock cycles	~1s	43 kB

→ Can we improve this ?

**Spoiler alert: yes**

# SRAM PUF overview

PUF, SRAM PUF and Synopsys PUF-Software

# What is PUF?

## A Silicon Fingerprint

### A Physical Unclonable Function

**A Physical Object**  
which uses its  
**Unclonable Physical Properties**  
to create a  
**Unique Identifier**

### Various Types

- SRAM, ring-oscillator
- Optical, Via, Quantum, Magnetic, ...

```
100001011101100100011101101110110
101000111011011000001011010001111
011001100101100110011110111011011
000010011101101011110001110000101
110110010001110110111011010100011
101101100000101101000111101100110
010110011001111011101101100001001
110110101111000111011000111011001
011001110100000101110000101000111
011001110001110110110001110110010
110011101
```

**Intrinsic Security**

### Benefits of PUF-based IDs/Keys

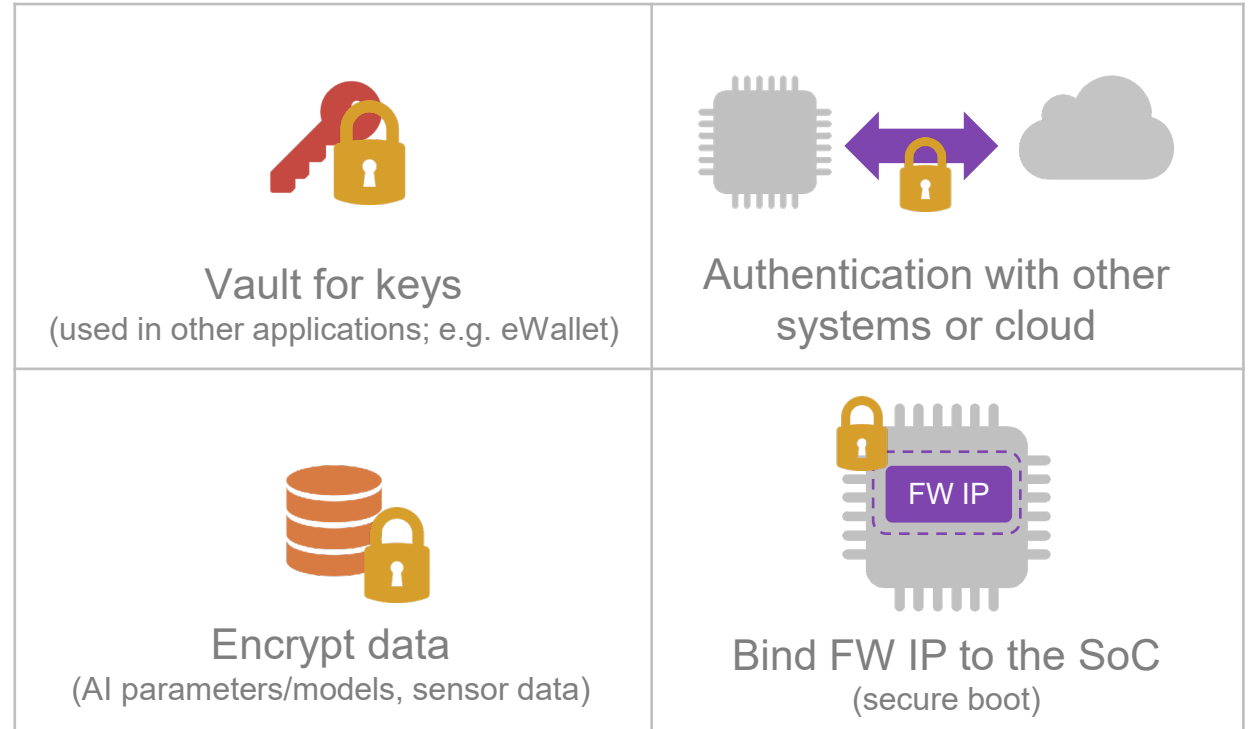
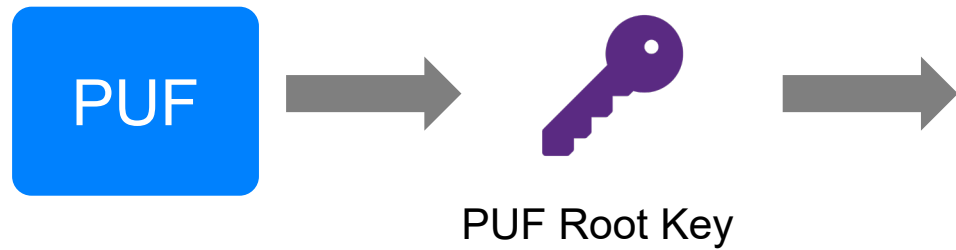
- Device-unique ID and keys
- Key not known outside the SoC
- Unclonable
- Not stored

### Ideal for

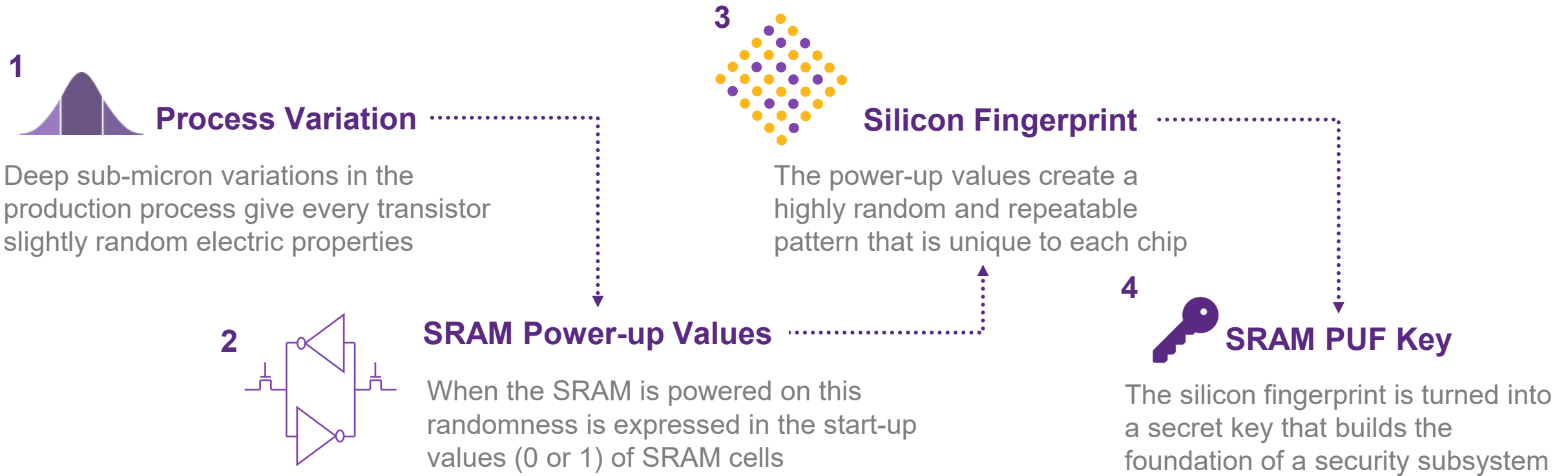
- Tracking
- Identification / authentication
- Anti-counterfeiting
- Protection against reverse-engineering

# PUF: Foundational Security

Keeping Secrets Secret and Data Secure



# SRAM PUF – Keys from Silicon Fingerprint



## SRAM PUF Benefits

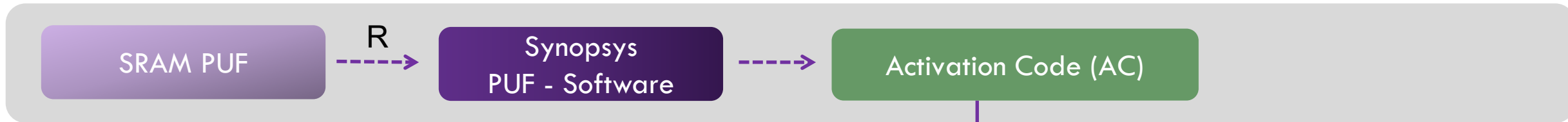
- Device-unique, unclonable fingerprint
- Leverages entropy of mfg. process
- No key material programmed

# Synopsys PUF-Software

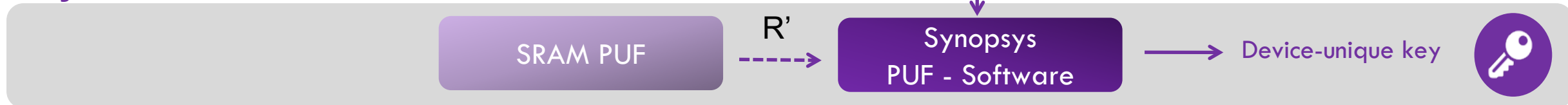
- Software library, uses standard SRAM as a PUF to create a hardware-based trust anchor
- Main features:
  - Device-unique ID
  - Key provisioning
  - **Secure key storage**
  - **Symmetric and asymmetric key cryptography**
  - Random number generation (NIST SP 800-90A/B)

PUF + Crypto Library

## PUF Enrollment – One-Time Process (at device provisioning)



## Key Reconstruction – In the Field



# Wrap keys using Synopsys PUF-Software

- Root keys are re-created from the PUF each time they are needed → **never stored**  
→ stronger protection than traditional key storage in NVM
- We use these PUF root keys to wrap other keys
- Wrapping = **encrypt + MAC + binding to the device**



With a key create a keycode to securely store it



With a key code retrieve a stored key

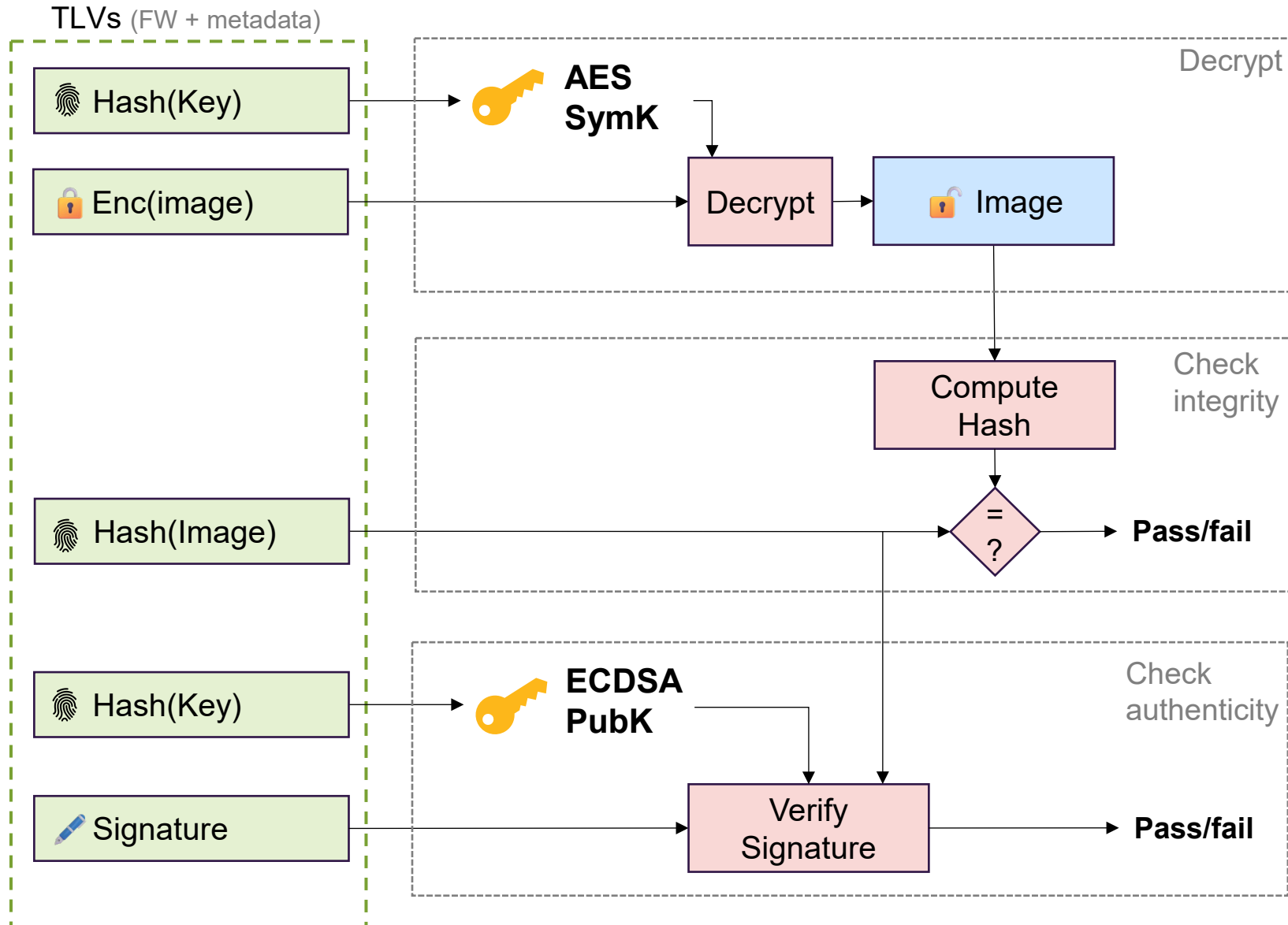
# Improved Secure Boot Flow

Use SRAM PUF to improve the standard secure boot flow



# Typical Secure Boot Flow

MCUboot flow with encryption and ECDSA signature



1. Decrypt the firmware
2. Verify the hash
3. Verify the signature

- 2 main security assets:
  - AES firmware encryption key
  - ECDSA signature public key

# General ideas

## 1. Protect the firmware encryption key

Wrap the symmetric key using PUF-SW (easy)

- The network owner generates a symmetric key
- They distribute it to each device during provisioning
- Each device wraps the key with its own PUF

## 2. Prevent cloning

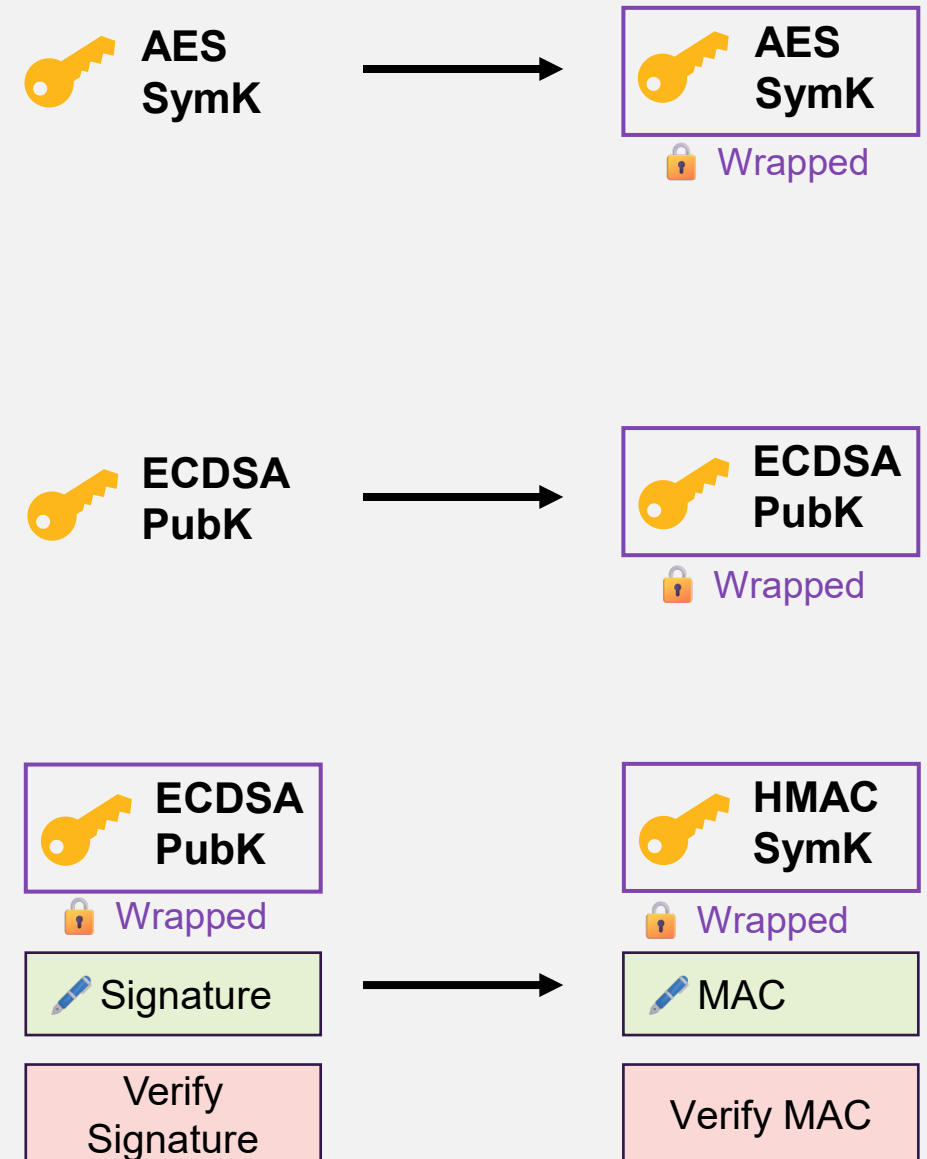
Wrap the public key using PUF-SW (easy)

- The network owner generates a public/private key pair
- They distribute the public key to each device during provisioning
- Each device wraps the key with its own PUF

## 3. Improve performance (medium)

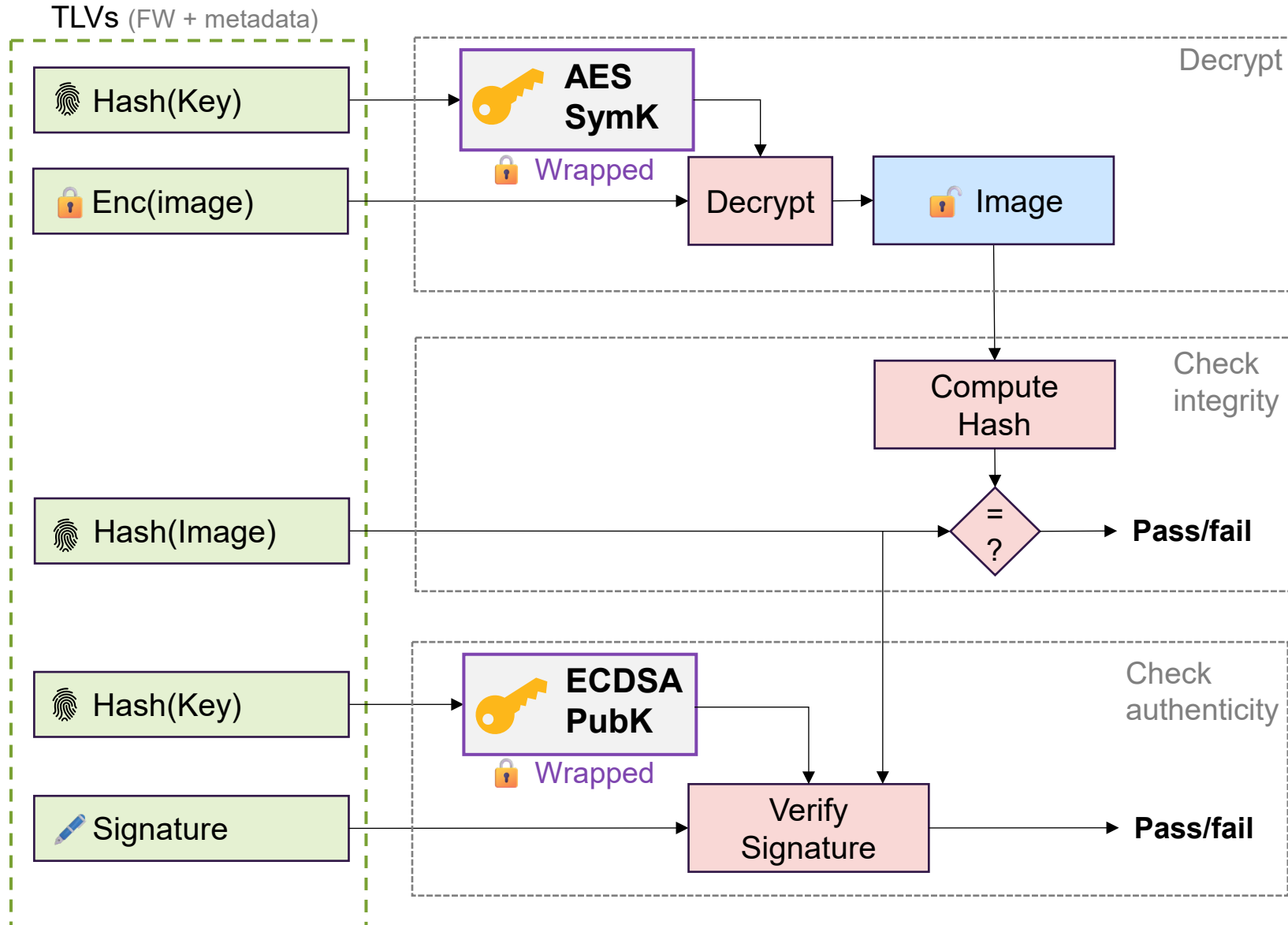
MAC = symmetric signature

Use an **HMAC** in place of the **ECDSA** signature  
(and provide a solution to securely store the HMAC key)



# Improved secure boot flow

Encryption, ECDSA signature, keys wrapped with PUF-SW



First, the two easy steps

Keys wrapped with PUF-SW at provisioning phase:

- Encryption symmetric key
- Signature public key

## Pros

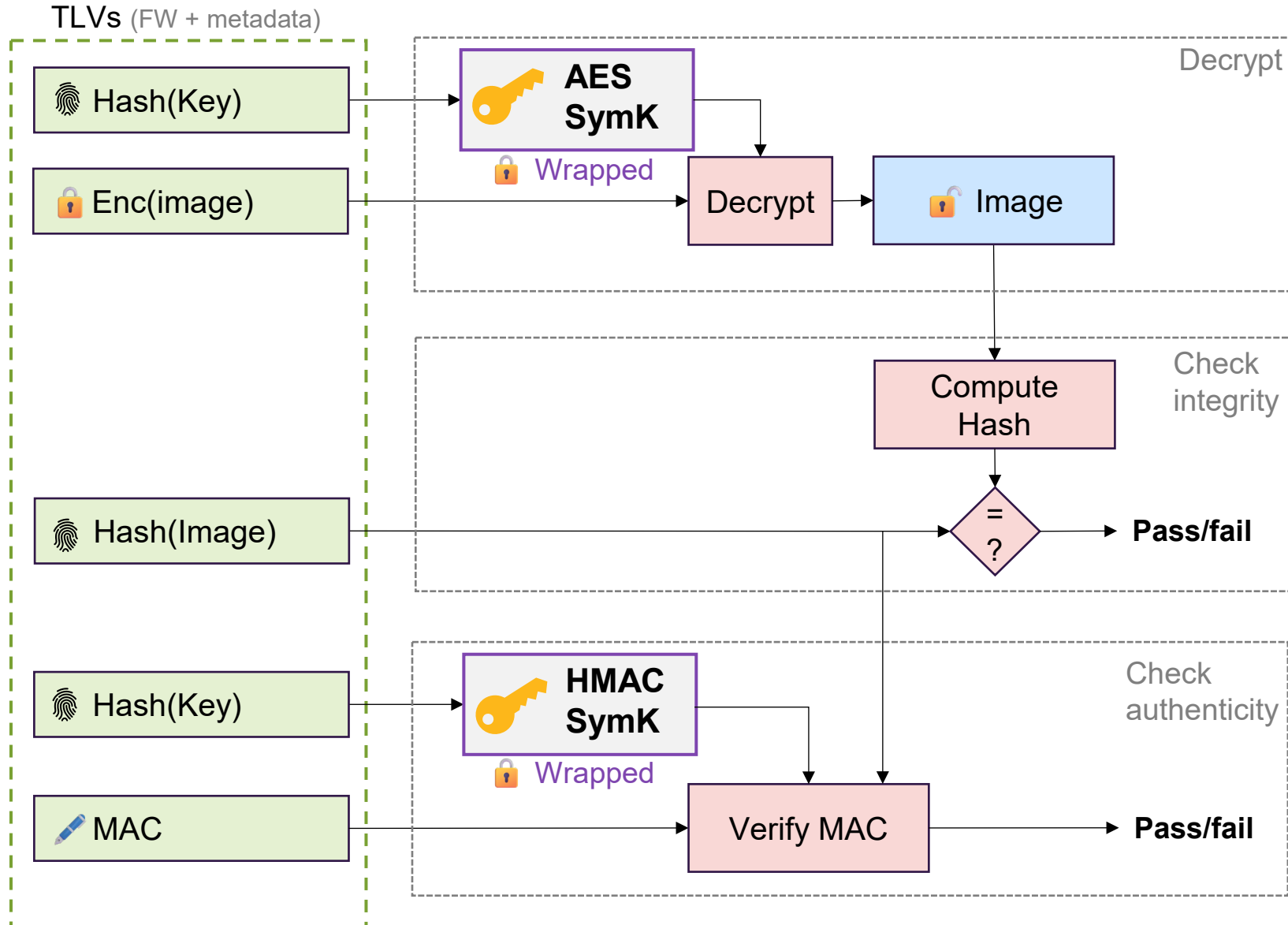
- Simple
- Protects against cloning
- Protects the two keys

## Cons

- Slow boot sequence

# Improved secure boot flow

## Naïve approach with HMAC



## Pros

- Faster
- Simple
- Protects against cloning
- No need for asymmetric crypto (smaller code size)

## Cons

- Common key for all devices
- If the key is compromised, attackers can forge valid firmware images

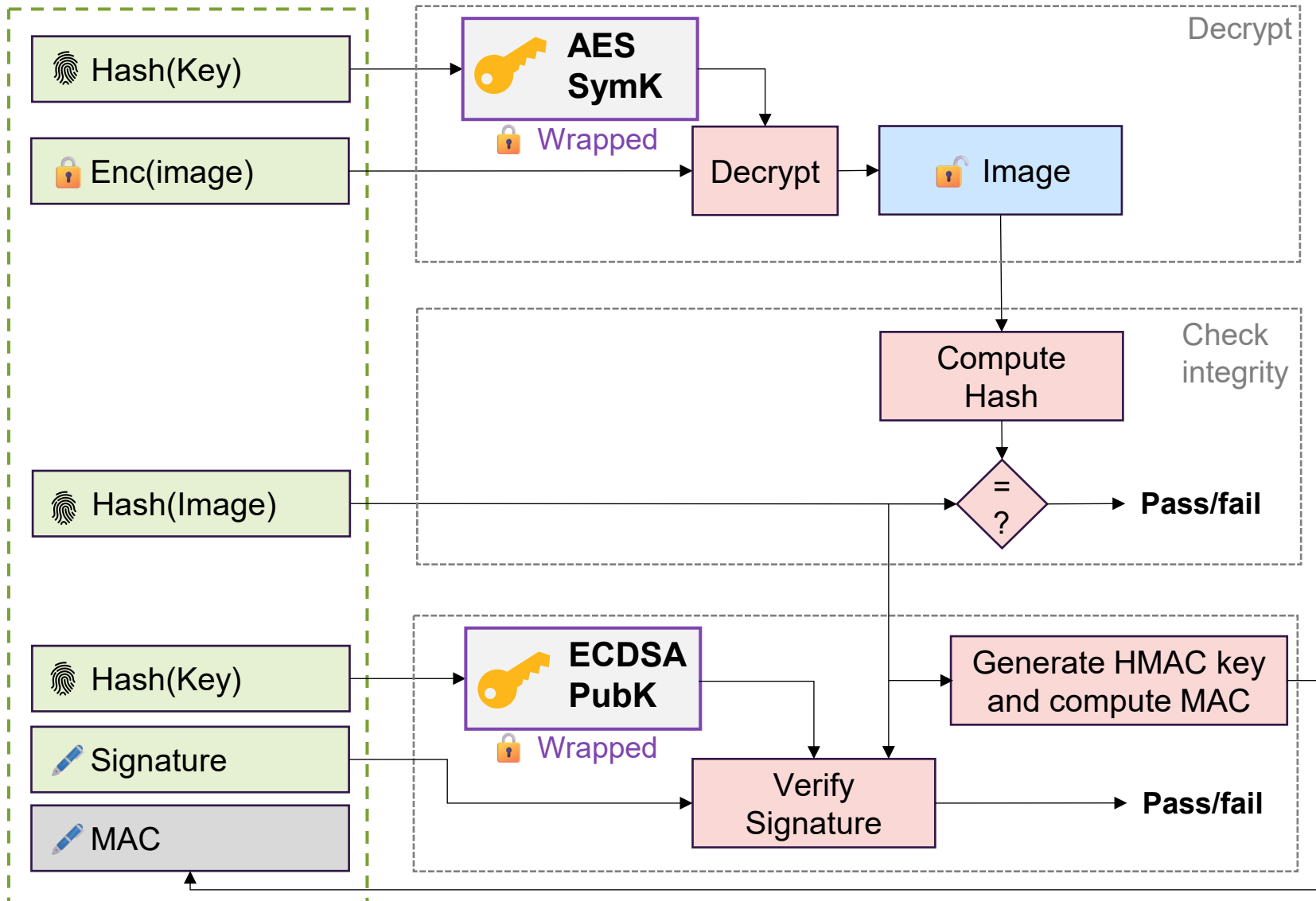
## Overall

Too big of a risk

# Improved secure boot flow

Hybrid approach, ECDSA + HMAC

At first boot sequence (verification)



- ECDSA at first verification
- Generate a device-unique MAC key (and wrap it)
- Compute a MAC and store it next to the FW image
- Use the MAC for subsequent boot sequences

## Pros

- Still fast
- Protects against cloning
- Device-unique MAC key

## Cons

- Still requires asymmetric crypto

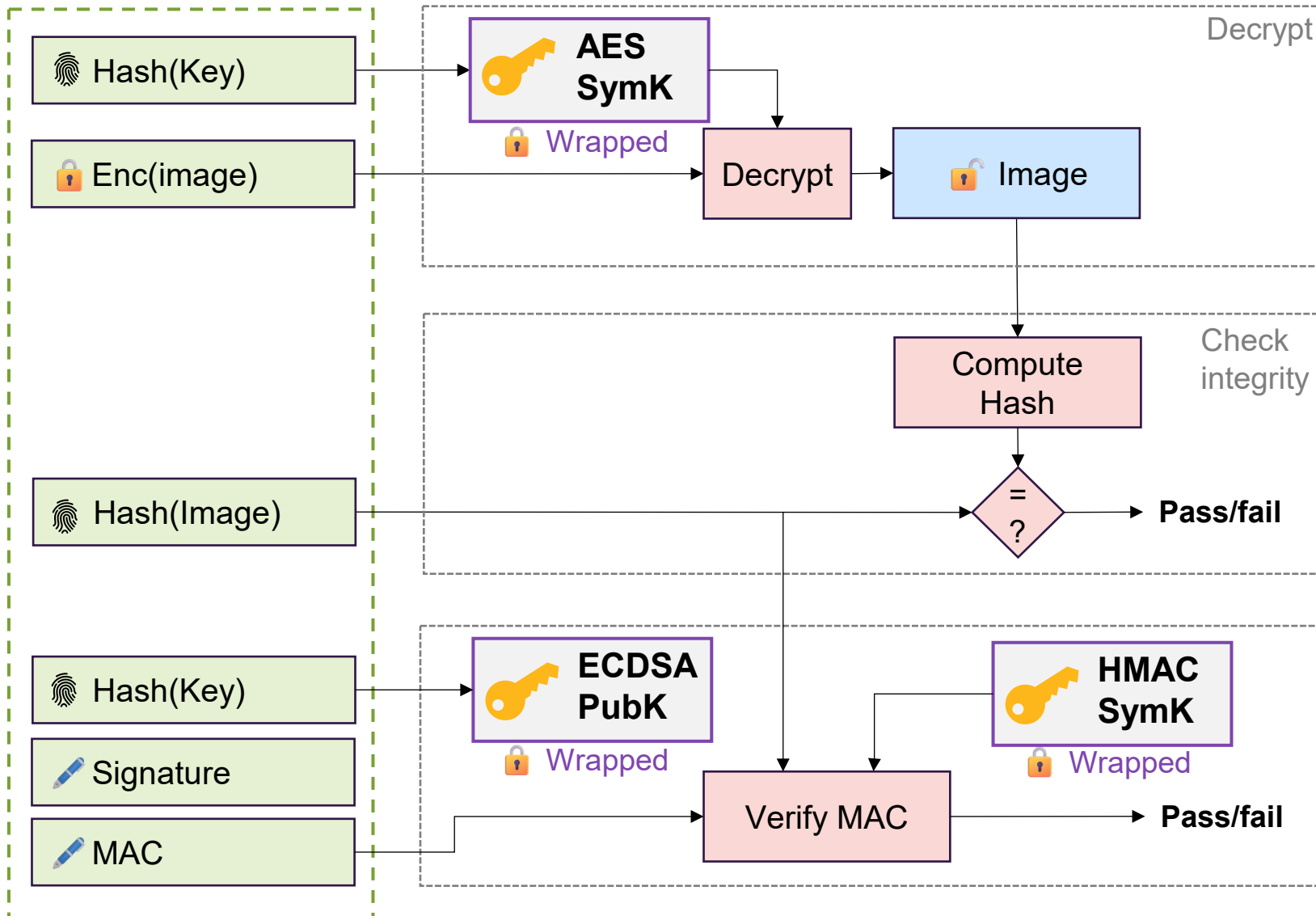
## Overall

We get the best of both worlds

# Improved secure boot flow

Hybrid approach, ECDSA + HMAC

After first boot sequence



- ECDSA at first verification
- Generate a device-unique MAC key (and wrap it)
- Compute a MAC and store it next to the FW image
- Use the MAC for subsequent boot sequences

## Pros

- Still fast
- Protects against cloning
- Device-unique MAC key

## Cons

- Still requires asymmetric crypto

## Overall

We get the best of both worlds

# Efficient secure boot flow with SRAM PUF

## Performance figures and code size

- On a Cortex-M0+:  
STM32G070RB, with a basic unencrypted “Hello world” firmware payload

Scenario	Time to boot	Time (at 64 MHz)	Code size
<b>Mbed TLS</b> Standard ECDSA flow	~62M clock cycles	~1s	43 kB
<b>Synopsys PUF-SW</b> Same flow, faster crypto lib	~15M clock cycles	~250ms	42 kB
<b>Synopsys PUF-SW HMAC</b> Hybrid HMAC flow, after first verification	~600k clock cycles	~10ms	20 kB (HMAC-only) 42 kB (hybrid)

- This solution has also been implemented on other STM32 devices (Cortex-M0 and Cortex-M33)

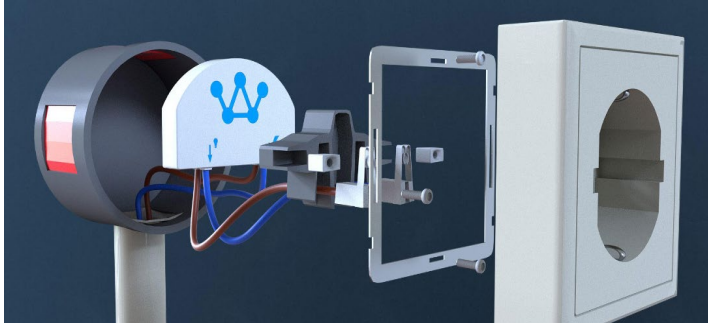
# Secure Boot Flow for Crownstone IoT devices

- Crownstone IoT device
- Lightweight Secure Boot Flow

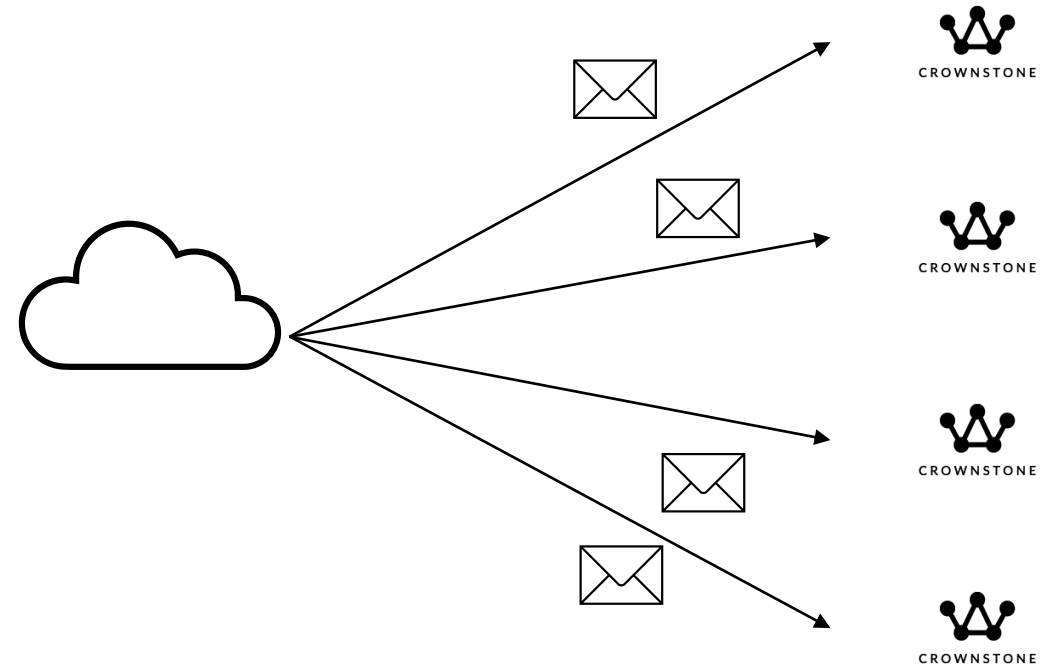


# Crownstone IoT device

Smart power outlet

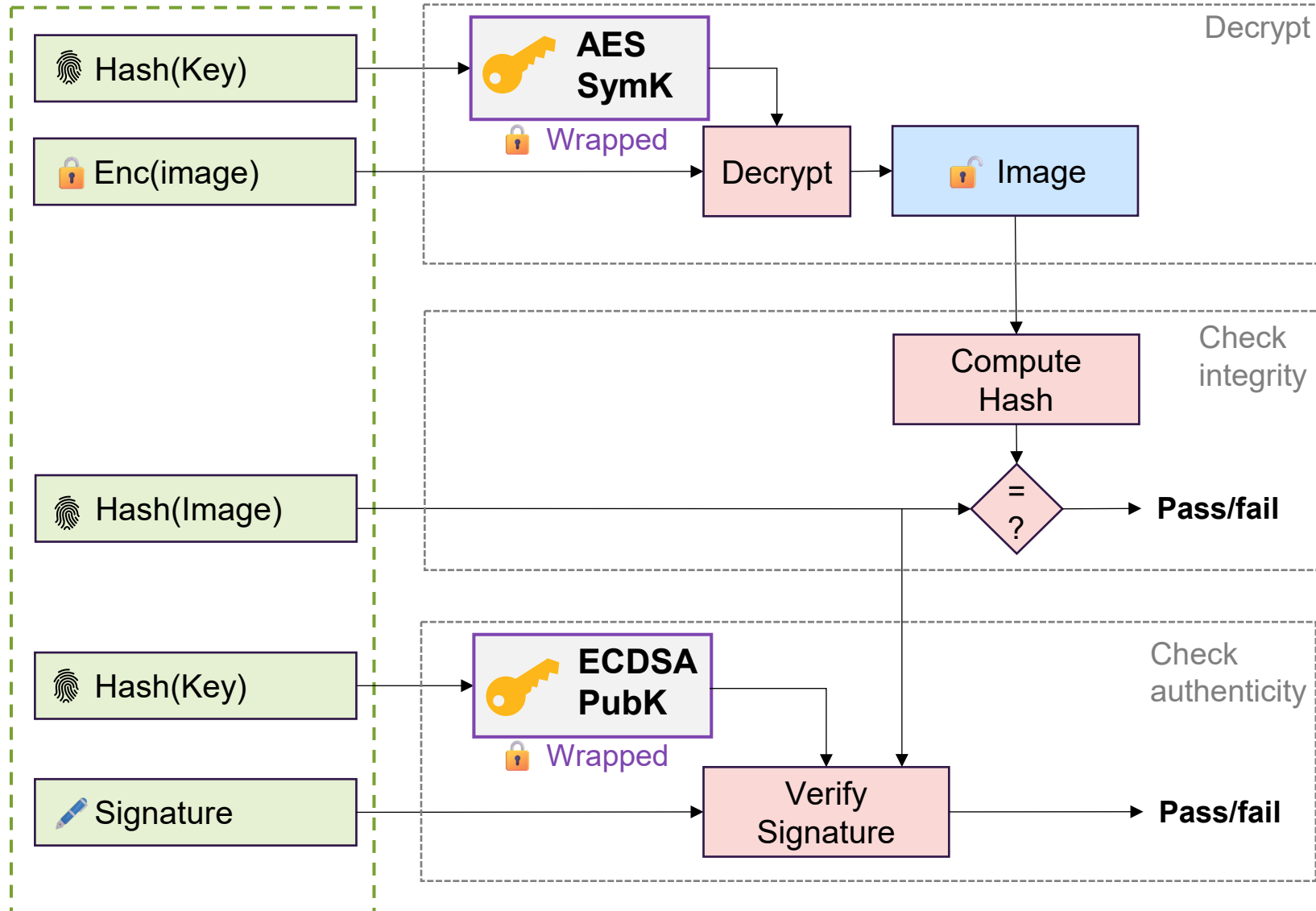


- Nordic nRF52832 SoC
  - 2.4 GHz transceiver
  - Arm Cortex-M4 (32-bit, 64 MHz)
  - 512 kB flash/64 kB RAM
- Use cases:
  - Indoor localization
  - Power Outlet Management
- Initially developed for private use (home), now moving to large office space



# Secure Boot Flow for Crownstone

Encryption, ECDSA signature, keys wrapped with PUF-SW



Keys wrapped with PUF-SW at provisioning phase:

- Encryption symmetric key
- Signature public key

Devices always on

- boot-time perf not an issue
- no HMAC-hybrid approach

# Conclusion

# Conclusion

- **Yes**, we can improve the typical IoT secure boot flow with SRAM PUF
- Using SRAM PUF in an IoT secure boot flow **brings benefits**:
  - Binds cryptographic key material to a device
  - Secure key storage without a secure vault
  - Faster boot time by using symmetric crypto
  - Protection against firmware cloning
- Next steps in Resilient Trust:
  - Continue the integration of PUF-SW on the Crownstone device
  - Have a first integration working by the end of the year

**SYNOPSYS®**

Thank you