Bootloader Protection Using Inherent PUFs

03.11.2013 | TU Darmstadt | Security Engineering |
André Schaller & Vincent van der Leest

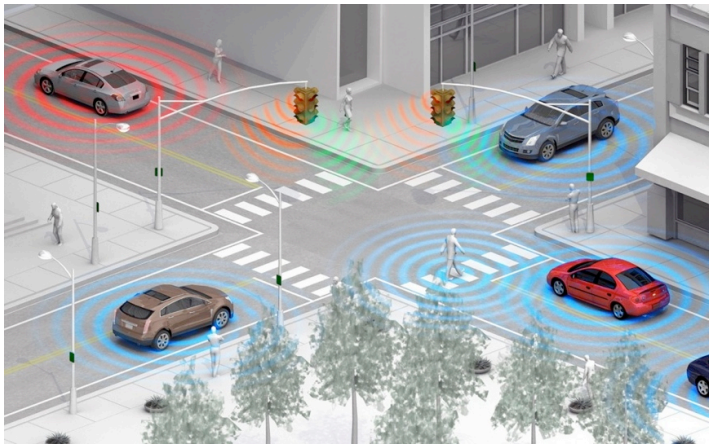
Agenda

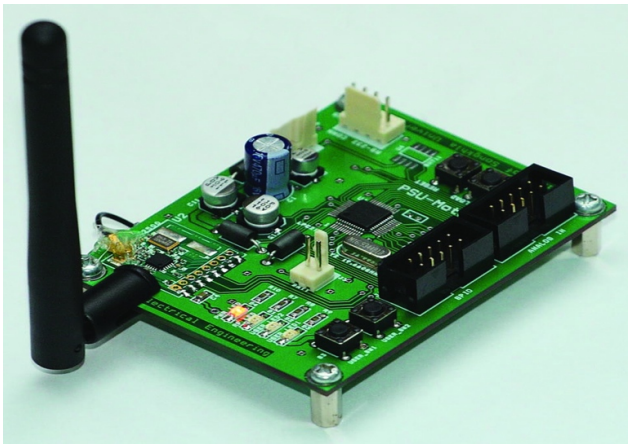
CASED

- Motivation
- Proposed Solution
- Research Objectives
- Demonstration

Motivation

- Rapidly increasing number of lightweight / low-power computing devices
- Mobile embedded devices, Machine-to-Machine (M2M) communications, internet of things, sensor nodes, Car-2-X communication, pervasive / ubiquitous computing, ...





Motivation

CASED

- How to protect these devices from firmware manipulations?

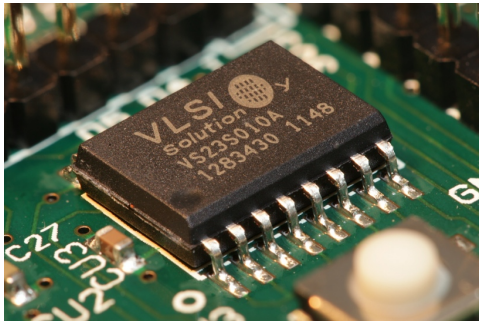
- Software-based approaches (ARM TrustZone, AMD SVM, Intel TXT)
  - Must be supported by underlying hardware platform
  - Approaches often rely on kernelized architectures (secure microkernel)
  - Many microkernel require modification of user space applications

- Hardware-based approaches (TPM chips)
  - Discrete, physical chips ⟶ additional manufacturing costs
  - Additional hardware security chips not possible for low-end devices

Proposed Solution

General Idea

- We do not need extra chips to provide hardware-based security!
- Let‘s exploit hardware which is on-board anyway
- Almost any computing device holds static RAM (SRAM)
- Use start-up values of SRAM to extract fingerprint of corresponding hardware instance [1]
- Use the extracted fingerprint to derive a cryptographic key
- Modified bootloader in turn decrypts firmware using this key
- Actually this binds the firmware to the hardware



[1] - D. Holcomb: „Power-Up SRAM State as in Identifying Fingerprint and Source of True Random Numbers“

Proposed Solution

General Idea

CASED

- Bootloader reads SRAM start-up values & extracts fingerprint
- Fingerprint is used to derive cryptographic key *K*
- *K* is used to decrypt firmware; successful if:
  - a) correct key *K* derived
  - b) correct firmware image present
- Firmware is called

- Security properties:
  - Integrity of firmware
  - Confidentiality of firmware
  - Authenticity of hardware
  - Use of epheremal key ⟶ reduced attack surface

Proposed Solution

Goal & Research Objectives

CASED

- Approach is known for IP protection on FPGAs (with dedicated hardware)

Goal: Port the approach to COTS devices to provide an platform instrinsic hardware-based anchor-of-trust. Motivation to implement it on virtually any commercial computing device.

- Research objectives:
  1. Can we find PUF instances in COTS microprocessors?
  2. Can we extract a fingerprint to derive a key?
  3. Implementation feasible for lightweight devices with small memory (32 - 128 kByte on-chip memory)?

PUF Instances

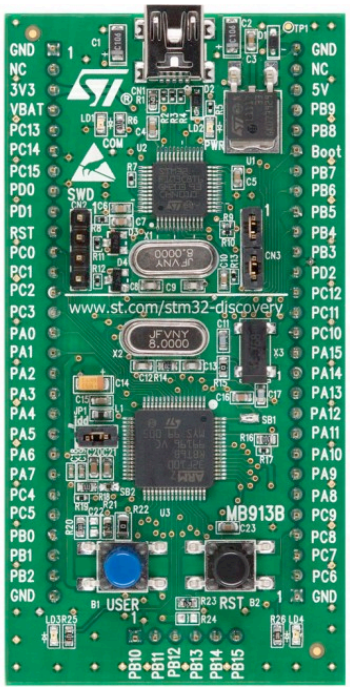
Can we find PUF instances in COTS microprocessors - I

CASED

- Which wide-spread microprocessors to choose?

- ARM Cortex A/M family: widely-used 32-bit embedded MCU for lightweight (M) and powerful (A) devices
  - Cortex-M*x*: optimized for cost and power sensitive
  - Cortex-A*x*: high-performance applications processors

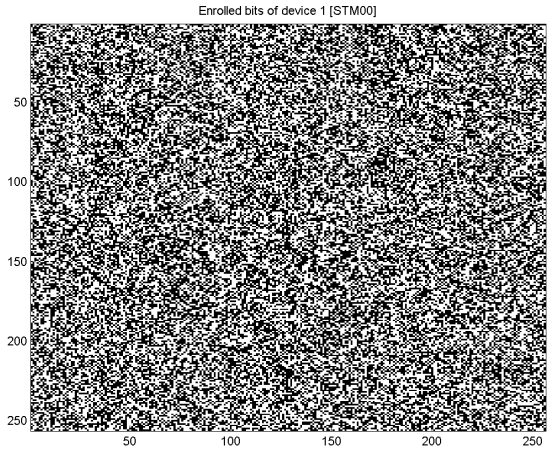
- Firstly, focus on simple Cortex-Mx MCUs:
  - Cortex-M3 (STM32 F100 Value Line)
  - 24 MHz, 128 KB Flash, **8 KB RAM**
  - Easy access to SRAM to get a first impression

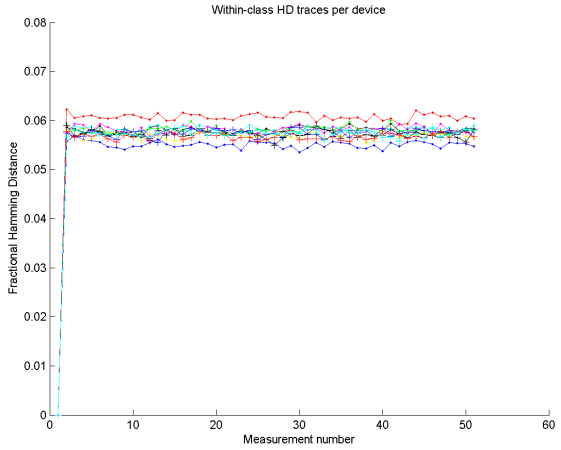


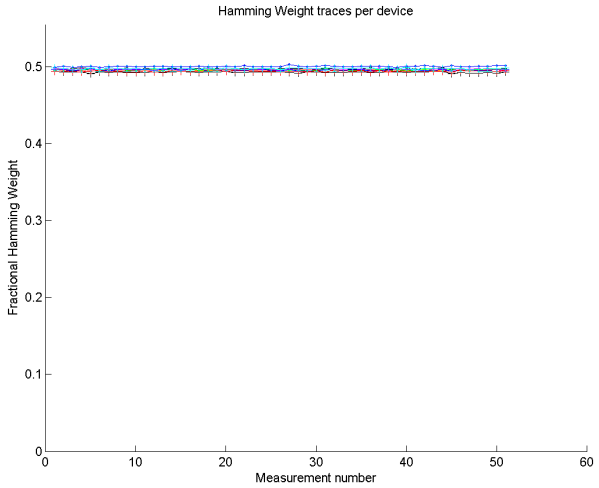
PUF Instances

Can we find PUF instances in COTS microprocessors - I

- Modified startup files ⟶ access to SRAM start-up values
- Statistical analysis revealed **useful PUF characteristics**
  - max. within-class Hamming distance (HD) = 6.23% (ideal HD: close to 0%)
  - min. Hamming Weight (HW) = 46.28 % (ideal HW: GD around 50%)
  - min. between-class Hamming distance (HD) = 49.08 % (ideal between-class HD: GD 50%)







PUF Instances

Can we find PUF instances in COTS microprocessors - II

CASED

- Move on to more complex hardware ...
- PandaBoard (ES): ARM-based OMAP44xx System-on-a-Chip (SoC) platform
  - 2x Cortex-A9, 2x Cortex-M3
  - 1 GB external DDR memory
  - Several on-chip-memory (OCM) instances
  - Onboard 10/100 Ethernet
  - 802.11 b/g/n WiFi
  - 1080p Full-HD video encoding/decoding
  - SD/MMC card cage
  - ...

- Full-grown modern smartphone / tablet platform

PUF Instances

Can we find PUF instances in COTS microprocessors - II

CASED

- Object of interest: on-chip-memory instances:
  - OCM Save-and-Restore (SAR) ROM (4 KB)
  - OCM Save-and-Restore (SAR) RAM (8KB)
  - OCM Level 3 RAM (56 KB)
- Access to start-up values by modifying the bootloader (u-boot)
- Staged boot process

1st stage bootloader

- hard-coded in ROM
- can not be access / disabled
- probably initializes processors

→

2nd stage bootloader (SPL)

- tiny part of u-boot
- sets up stack, configures external memory and other components
- must fit in OCM L3 RAM

→

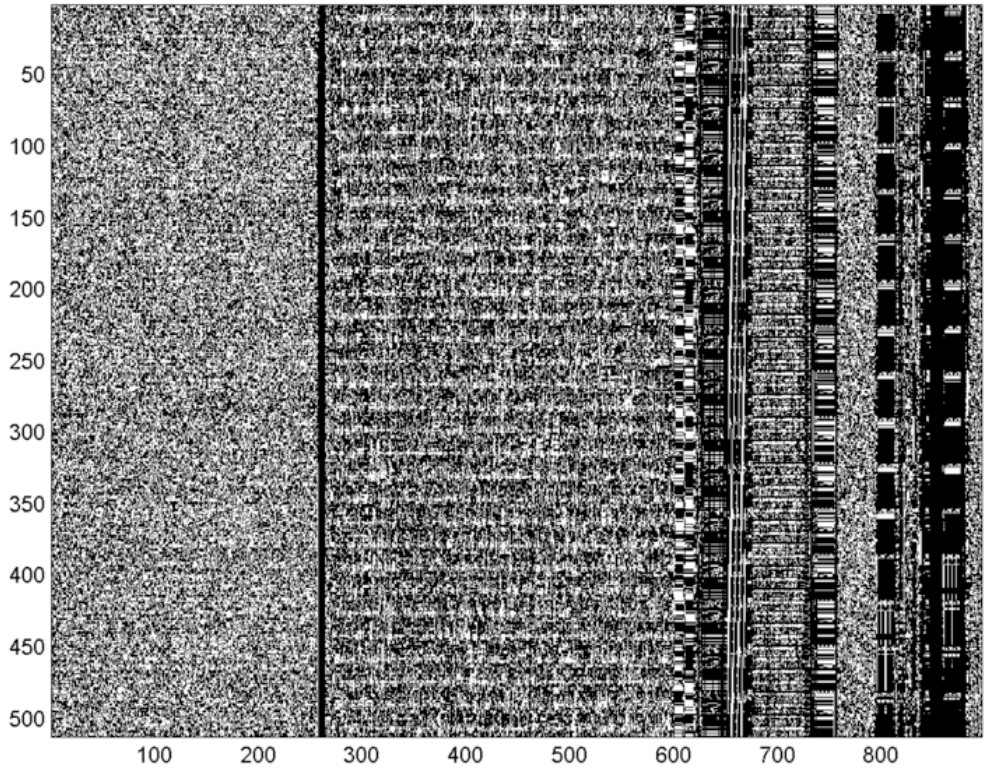
3rd stage bootloader (u-boot.img)

- main part of u-boot
- provides rich functionality to manipulate and configure hardware
- eventually calls OS kernel

PUF Instances

Can we find PUF instances in COTS microprocessors - II

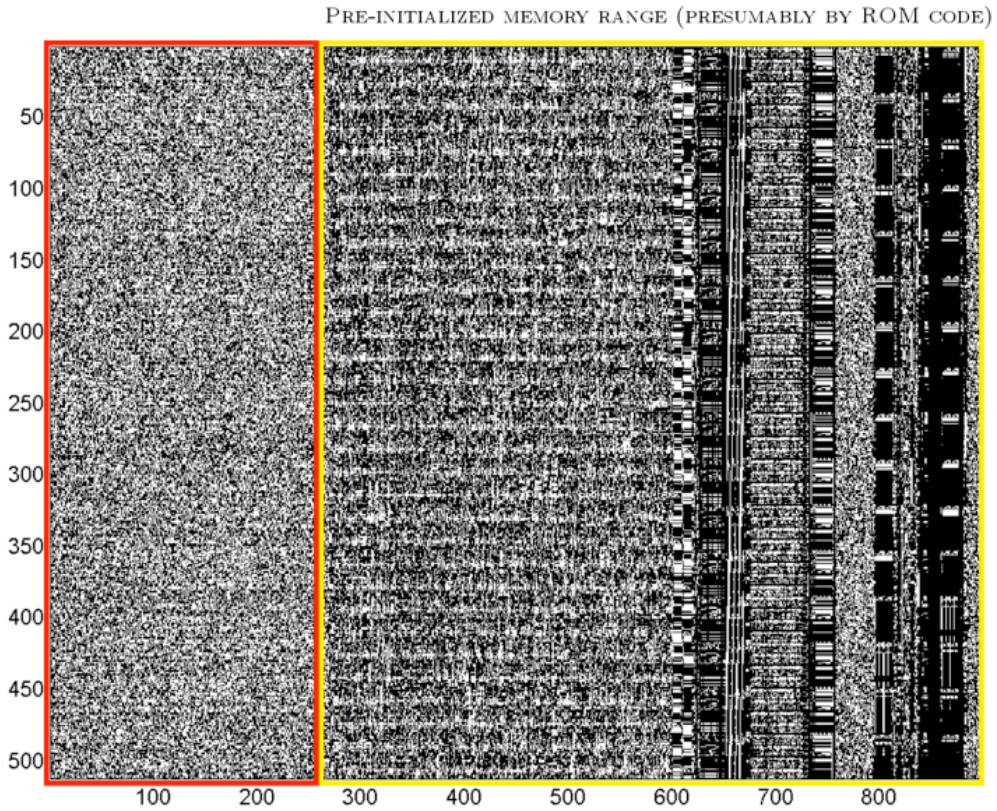
- Successfully extracted start-up values with modified 2nd stage bootloader (SPL)
- However, bitmapping these values revealed this picture ...



PUF Instances

Can we find PUF instances in COTS microprocessors - II

- Successfully extracted start-up values with modified 2nd stage bootloader (SPL)
- However, bitmapping these values revealed this picture ...

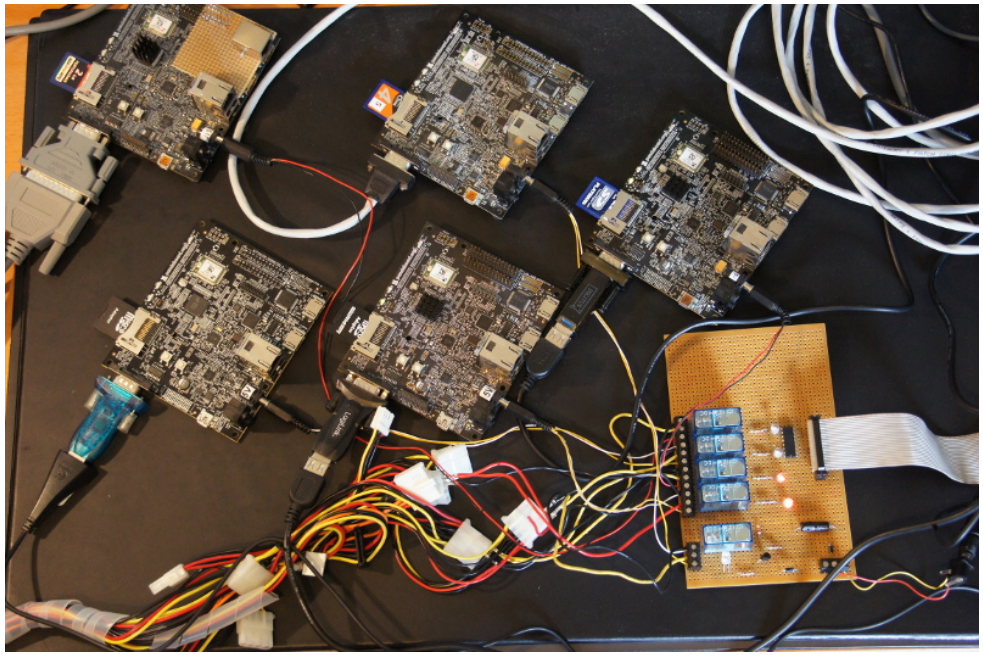


PUF Instances

Can we find PUF instances in COTS microprocessors - II

CASED

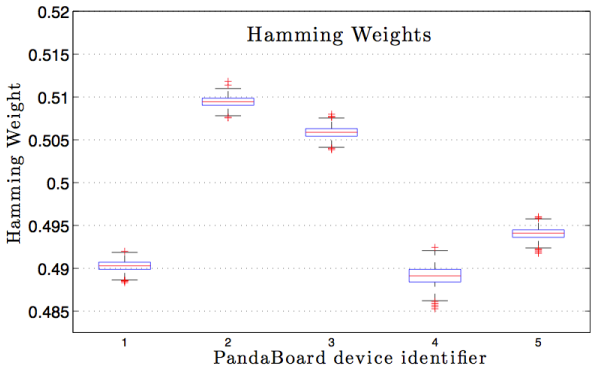
- Statistical analysis of first 13 KB OCM L3 RAM (5 Pandaboards, 1000 trials)
  - max. within-class Hamming distance = 4.67% (ideal: close to 0%)
  - min. Hamming Weight = 48.53 % (ideal: 50%)
  - min. between-class Hamming distance = 49.66 % (ideal: 50%)

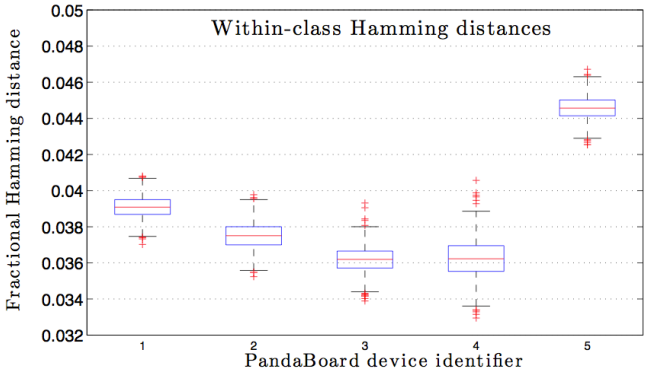


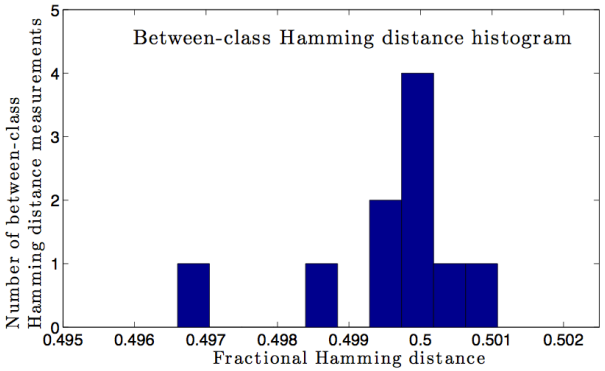
PUF Instances

Can we find PUF instances in COTS microprocessors - II

- Statistical analysis of first 13 KB OCM L3 RAM (5 Pandaboards, 1000 trials)
  - max. within-class Hamming distance = 4.67% (ideal: close to 0%)
  - min. Hamming Weight = 48.53 % (ideal: 50%)
  - min. between-class Hamming distance = 49.66 % (ideal: 50%)
- Even better results as for the STM32!





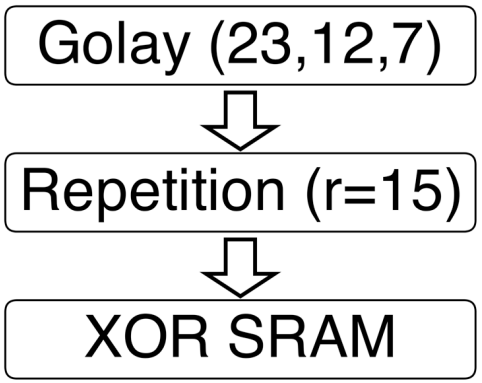


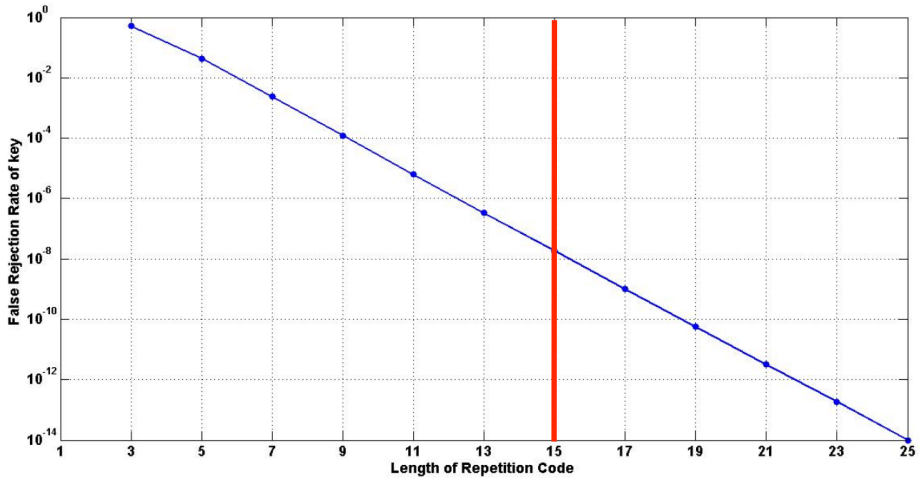
Fingerprint Extraction

Can we extract a fingerprint to derive a stable key?

CASED

- To derive a cryptographic key, we need a 100% stable fingerprint
- Within-class Hamming distance = noise = 5%
- Can be handled by Fuzzy Extractor algorithm (Code Offset Method)
- Combination of 2 linear codes: Golay(24, 12, 7) + repetition code (r = 15)
- False Positive Rate of almost 10^{-8}





Implementation

... feasible for lightweight devices?

CASED

- Actually Pandaboard is not a lightweight device
- However, memory requirements are the same - actually they are even stronger:
- Memory of lightweight / low-power devices usually between 32 - 128 KB
- Here: only 13 KB!
- 2 phases of application scenario:

Enrollment (at manufacturer site):

1. Define device-specific key K
2. Read reference SRAM measurement M
3. Create Helper Data W

   $W = \text{FuzzyExtractor}(K) \oplus M$
4. Encrypt 3rd stage bootloader B with K:

   $B_{enc} = \text{Enc}_{K}(B)$
5. Store W and B_{enc} on MMC

Reconstruction (at costumer site):

1. Read SRAM measurement M'
2. Read Helper Data W from MMC
3. Reconstruct key: $K = \text{Fuzzy Extractor}(M' \oplus W)$
4. Decrypt 3rd stage bootloader B_{enc} with K:
   $B = \text{Dec}_{K}(B_{enc})$
5. Call B

Questions / Discussion

CASED

Demonstration

Questions / Discussion

CASED