Lightweight Anti-counterfeiting Solution for Low-End Commodity Hardware Using Inherent PUFs

André Schaller[1], Tolga Arul[2], Vincent van der Leest[3], and Stefan Katzenbeisser[1]

[1] TU Darmstadt, Security Engineering Group
lastname@seceng.informatik.tu-darmstadt.de
http://www.seceng.de/
[2] CASED, Mornewegstrasse 32, 64293 Darmstadt,
tolga.arul@cased.de
[3] Intrinsic-ID, Eindhoven, The Netherlands
vincent.van.der.leest@intrinsic-id.com
http://www.intrinsic-id.com

Abstract. This paper presents a lightweight anti-counterfeiting solution using intrinsic Physically Unclonable Functions (PUFs), which are already embedded in most commodity hardware platforms. The presented solution is particularly suitable for low-end computing devices without on-board security features. Our anti-counterfeiting approach is based on extracting a unique fingerprint for individual devices exploiting inherent PUF characteristics from the on-chip static random-access memory (SRAM), which in turn allows to bind software to a particular hardware platform. Our solution does not require additional hardware, making it flexible as well as cost efficient. In a first step, we statistically analyze the characteristics of the intrinsic PUF instances found in two device types, both based on a widely used ARM Cortex-M microcontroller. We show that the quality of the PUF characteristics is almost ideal. Subsequently, we propose a security architecture to protect the platform's firmware by using a modified boot loader. In a proof of concept, we embed our solution on a state-of-the-art commodity system-on-a-chip platform equipped with an MCU similar to the ones previously analyzed.

1 Introduction

With the proliferation of mobile computing the influence of low-end devices on our every day communication steadily grows. As their computational power increases, such devices are embedded into many objects we are interacting with on a daily basis. Besides their implementation in smart phones and tablets, low-end devices are employed in Car2X communication [16] where microcontrollers (MCUs) are used in modern cars to improve road safety, traffic efficiency or to act as an infotainment platform [10]. Additional use cases arise with the growing importance of sensor nodes with applications in the fields of health care, military

T. Holz and S. Ioannidis (Eds.): TRUST 2014, LNCS 8564, pp. 83–100, 2014.

and environmental monitoring [18]. Furthermore, objects in our everyday-life (e.g. fridges, televisions, smart-metering, etc.) are increasingly equipped with computing capabilities, which will eventually yield the Internet-Of-Things [4,12].

A high proportion of such low-end devices are not equipped with special on-board security mechanisms such as secure memory or trusted execution environments. Devices without security features range from lightweight MCUs to more complex system-on-a-chip (SoC) platforms. Lightweight devices are designed for microcontroller use (e.g. ARM Cortex-M processor family). They do not implement security features due to hardware constraints and economic considerations. For example, ARM's Cortex-M series does not support ARM TrustZone technology[1]. More heavyweight SoC platforms can be found in smart phones and tablets. Many SoCs omit hardware security mechanisms due to increased costs and to allow for product evolution. Nevertheless, low-end devices are often used to process sensitive data. Thus, they are a worthwhile target of cyber-criminals. Malicious parties are gaining interest in attacking such devices to extract intellectual property.

To overcome this issue, we propose a lightweight anti-counterfeiting solution to bind a given firmware instance to a particular hardware platform providing a strict hardware-software binding for low-end commodity hardware. Our scheme relies on a hardware-based anchor of trust in terms of a PUF instance. It uses intrinsic Physical Unclonable Functions (PUFs) of the on-board SRAM to extract a fingerprint, which is unique for individual devices. The fingerprint is further processed to generate an ephemeral cryptographic key during an early boot stage and to subsequently decrypt the firmware. Our scheme establishes trust in the on-chip-hardware and in the firmware executed on the device by linking both instances. We achieve protection against IP extraction or modification on embedded devices without dedicated security mechanisms.

1.1 Contributions

We first analyze the quality of PUF instances extracted from two widely used low-end devices to demonstrate the feasibility of device-dependent cryptographic keys. The ARM Cortex-M3 is a popular example of a lightweight microcontroller, whilst the TI OMAP 4 exemplifies more complex SoC platforms. After statistically assessing the stability and uniqueness of the derived device fingerprints we propose a lightweight security architecture to bind a given firmware to a specific device. Subsequently, we implement the proposed architecture on a SoC platform and demonstrate the compatibility of our solution with off-the-shelf commodity hardware.

1.2 Structure

In Section 2 we discuss current approaches to Physically Unclonable Functions and in particular SRAM PUFs. The analysis of the PUF characteristics of two

[1] ARM TrustZone security extension is part of most ARM Cortex-A application processors [1].

popular devices is given in Section 3. We describe the attacker model, present the proposed security architecture and explain the course of usage in Section 4. In Section 5 we describe details related to the implementation of the proof-of-concept. Lastly, in Section 6 we conclude our work.

2 Related Work

A Physically Unclonable Function (PUF) is a complex physical structure that generates a value y in response to a stimulus x. The response y depends on the challenge x as well as on the micro- or nanoscale physical structure of the PUF itself. It is assumed that the PUF is unclonable such that it can not be reproduced, not even by the manufacturer. The challenge-response behavior of the physical system is complex enough such that the response to a randomly selected challenge can not be predicted. Furthermore, due to minuscule manufacturing variations during the production process, embedded PUFs can be used to robustly identify a silicon chip.

Silicon-based PUFs include delay-based or memory-based PUFs. For an exhaustive overview of PUFs and details on their taxonomy we refer to [15]. It has been shown that selected static random-access memory (SRAM) shows PUF-like behavior [11]. Further research in this area support the applicability of SRAM as a Physical Unclonable Function [13,17]. Using SRAM as PUFs exploits manufacturing variations, which manifest themselves in a bias of memory cells inside of SRAM modules. During the power-up phase these cells initialize to either the value of zero or one. Most cells show a stable start-up behavior, which in total creates a start-up pattern we will exploit to generate a fingerprint for the device.

Since not all of the SRAM bytes show a stable behavior in such sense that they are always initialized to a fixed value, the SRAM start-up values include a small amount of unstable bits, so-called noise. Since the goal is to reconstruct a reliable cryptographic key from several noisy measurements, the noise is eliminated by employing a Fuzzy Extractor [9], which extracts the stable part of the PUF response and transforms it to a uniformly distributed value.

3 Finding PUFs in Commodity Hardware

To get a first impression on the feasibility to extract a PUF instance from commodity hardware we evaluated a popular lightweight MCU. We chose ARM's Cortex-M3 as it is a widely distributed low-end processor specially developed for embedded devices. With 212 licenses it is the most popular version among the Cortex processor family. 30% of ARM chips shipped in 2013 were Cortex-M processors [3]. They are integrated into virtually every smart phone.

Secondly, we analyzed commodity hardware from the class of SoC platforms to explore the feasibility to robustly extract a unique fingerprint from more complex devices. Our main interest was to analyze whether active components

adjacent to the SRAM influence the stability of key extraction. We chose the PandaBoard [2] and its successor the PandaBoard ES, based on an OMAP4430 platform, respectively on an OMAP4460. They comprise two ARM Cortex-A9 and two Cortex-M3 processors. We selected the PandaBoard as it is available as a general-purpose (GP) edition, which lacks the support for ARM's TrustZone extension. Furthermore, the OMAP 4 is integrated into many mobile phones and tablets from vendors like Nokia, Motorola, Samsung and more. Thus, the PandaBoard as a general-purpose device reflects the security and multimedia configuration of popular smart phones like the Samsung Galaxy SI and SII, Motorola Droid and Milestone series and several devices from LG.

In the following we briefly describe common characteristics for PUFs and later analyze both devices using these measures.

3.1 Common Characteristics for SRAM PUFs

In general, SRAM PUF instances should show properties that mitigate the prediction of correct start-up values (Hamming weight), enable a robust repeated identification of single devices (Within-class Hamming distance) and lastly generate a unique pattern among a pool of similar devices (Between-class Hamming distance).

The *fractional Hamming Weight* $HW(x)$ of individual measurements from the same device indicates whether the start-up values are biased to either zero or one. This measure gives a first impression on the randomness present in the start-up values. The ideal measure is a Gaussian distribution with a mean value of $HW(x) = 50\%$, representing no bias of the start-up values towards zero or one and thus the same amount of both values.

The *fractional Within-class Hamming distance* gives an indication whether the PUF results for a single device are stable when queried repeatedly. It is a normalized count of bits that differ between two PUF measurements and thus is a rational number between 0 and 1. The robustness of the start-up values is required to reliably identify a given device and subsequently reconstruct the corresponding cryptographic key. An optimal value for the within-class Hamming distance is close to zero. However, all start-up values show a certain amount of noise, which originates from SRAM cells that flip their initialization value across multiple trials.

The *fractional Between-class Hamming distance* test expresses whether the start-up values of different devices for the same challenge are independent. This measure states whether start-up values can be used for identification without enabling adversaries to predict a measurement for a second device on the basis of a given device with known start-up values. The optimal value for between-class Hamming distance is a Gaussian distribution with a mean value of 50%,

which refers to a pair of independent start-up values from two different devices. Devices with an optimal value exhibit a maximum distinguishability regarding their PUF responses given the same challenge.

3.2 Analysis of the Cortex-M

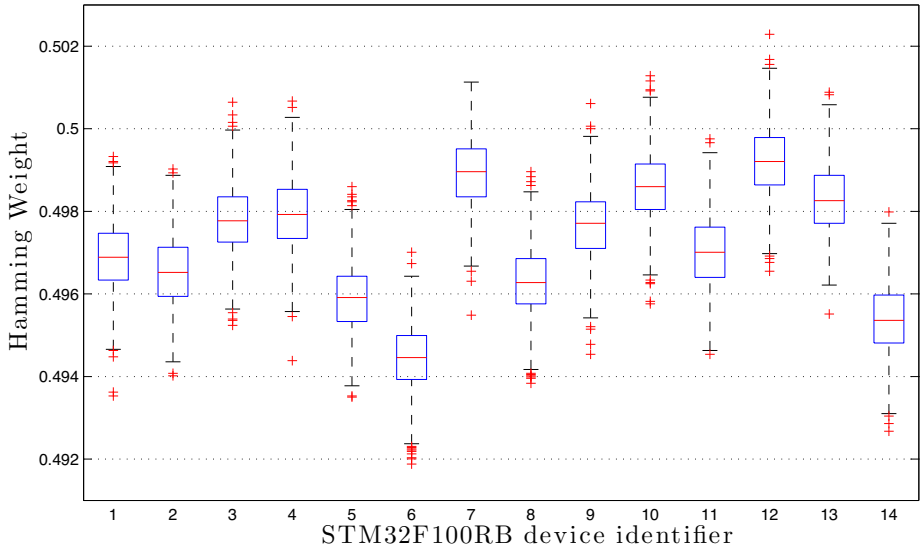
We evaluated the PUF behavior of an STMicroelectronics STM32F100RB development board that integrates an ARM Cortex-M3, 8 KiB on-chip SRAM and 128 KiB flash memory. To read the PUF measurement we modified the start-up code to display the raw start-up values via UART before the RAM gets initialized. The start-up code is essential to every microcontroller as it initializes the hardware as well as the stack and interrupt vectors and calls the main function.

To assess the PUF quality of the on-chip SRAM we tested 14 devices by extracting the 8 KiB SRAM start-up values 1000 times per device[2]. The devices were triggered using a controller board to repetitively turn the devices on, query the intrinsic PUF instance from on-chip SRAM and turn it off. In between these queries a break of 15 seconds was introduced to give the SRAM the chance to discharge. The summarized results in Table 1 show a decent PUF behavior that is suitable to robustly extract a unique fingerprint. Figure 2a shows the bitmap of an example measurement of one device.

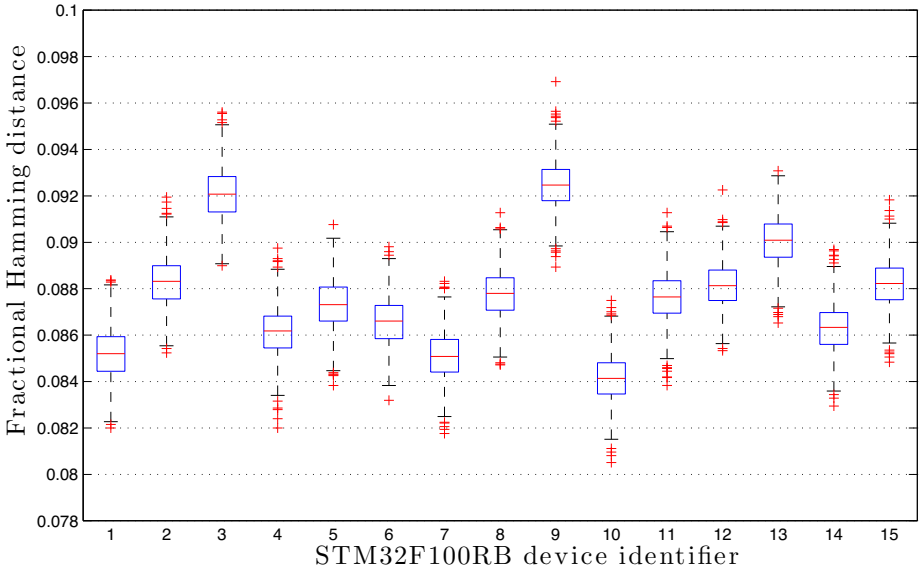
The SRAM start-up values have a worst-case *Hamming Weight* of $HW(x) = 49.18\%$. This value is close to the ideal of 50%. The STM32F100RB start-up values contain almost the same proportion of zeros and ones as depicted in Figure 1a. The worst-case bias is negligible. The number for the *Within-class Hamming distance* in the worst-case is 9.61%, see Figure 1b. The existing proportion of noise can easily be correct by standard error correction algorithms. The minimum *Between-class Hamming distance* is 46.48%, see Figure 1c. The number shows that there is some correlation between the measurements of different devices. This has a negative impact on the size of SRAM bytes needed to reconstruct the device-dependent key. However, this measure is much higher then the noise retrieved for the individual devices. Hence, the start-up values can be used to uniquely identify devices if they are pre-processed by error correction algorithms.

The boxplots shown below can be interpreted as follows. The red line across the central region of each box marks the data median. The blue bottom/top indicates the $25^{th}/75^{th}$ percentile for the data set. The height of the box corresponds to the inter quartile range (IQR) of the data set. The ends of the whiskers mark the lowest and highest values of the data set that are within 1.5 times the IQR of the box edges. The plus signs represent single values that are outside the range of the whiskers.

[2] Due to the fragile nature of the test setup, some measurements produced on-chip SRAM values of incorrect length. We removed these obvious measurement errors from the data set. In total 27 out of 14.000 measurements have been removed.

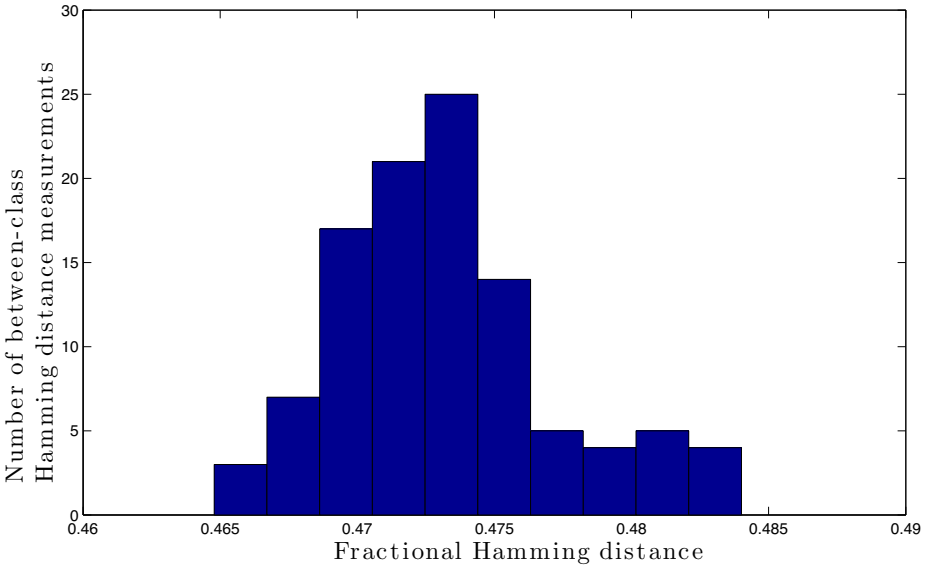


(a) Fractional Hamming Weight



(b) Within-class Hamming distance

Fig. 1. Detailed PUF characteristics for STM32F100RB devices: (a) Fractional Hamming Weights of SRAM start-up values. (b) Within-class fractional Hamming distance of SRAM start-up values. (c) Between-class fractional Hamming distance histogram.



(c) Between-class Hamming distance

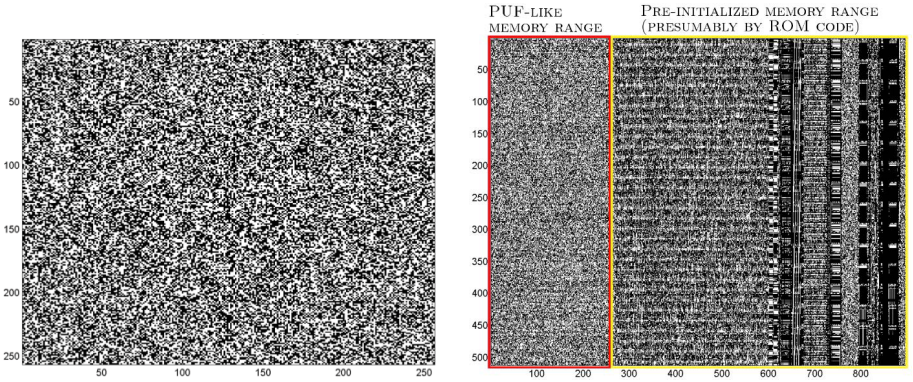
Fig. 1. (*Continued.*)

3.3 Analysis of the OMAP

The PandaBoard's OMAP 4 SoC contains two Cortex-M3 processors and thus PUF characteristics are expected, which are similar to those of the Cortex-M3. However, an extensive analysis is necessary because of highly-integrated active components (multimedia hardware accelerator, programmable DSP, integrated graphics processor), which could interfere with the SRAM start-up values. The SRAM is implemented as several instances of on-chip memory (OCM) featuring (i) OCM Save-and-Restore ROM (4 KiB) (ii) OCM Save-and-Restore RAM (8KiB) and (iii) Level-3 RAM (56 KiB).

Analysis of the OCM modules revealed that only a specific part exhibit PUF-like behavior. In particular, the Level-3 on-chip RAM (L3 OCM RAM) can be partially used to extract a fingerprint. The memory is shared among different sub-modules including the Cortex-M3 subsystem. Figure 2b shows the bitmap of the L3 OCM RAM from a PandaBoard shortly after the device gets out of reset.

The bitmap indicates that there are repeating structures in the middle and high address regions. As no other hardware is initialized at this early phase of the boot process we assume that these patterns represent structures used by the on-board ROM code shipped with every PandaBoard. The repeating structures could be caused by the ROM code's API interfaces. Furthermore, we assume that the L3 OCM RAM is used in a similar way as a stack as only higher addresses



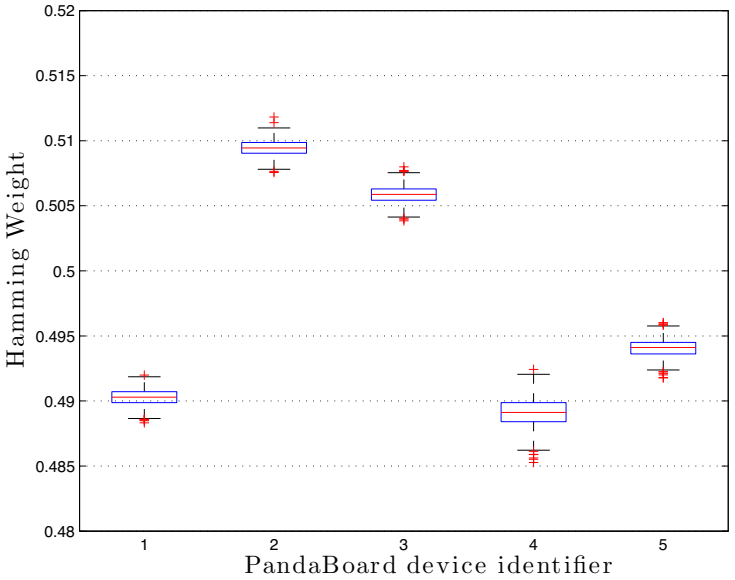
(a) STM32F100B (b) PandaBoard

Fig. 2. (a) Bitmap of a measurement for STM32F100RB a device. (b) Bitmap of a PUF measurement of a PandaBoard L3 OCM RAM (56 KiB). The red area (12 KiB) is used for fingerprint extraction. The yellow area contains initialized values and does not show PUF characteristics.

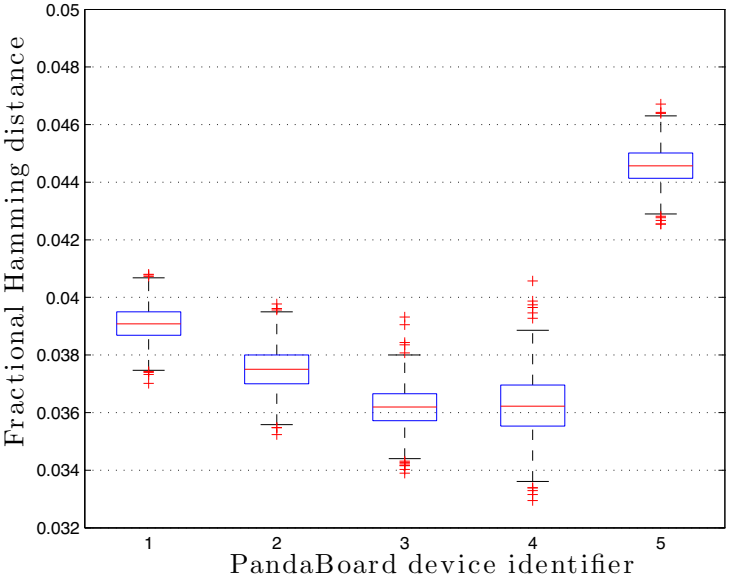
exhibit such patterns. However, in the area of the first 12 KiB (`0x40300000 - 0x40303000`) an apparently random distribution of zeros and ones can be seen. This memory range refers to the part of the L3 OCM RAM we selected for further analysis regarding its PUF behavior.

We performed measurements on a set of 5 PandaBoard instances. The test set included two versions of the platform – an early version of the PandaBoard equipped with an OMAP4430 and an advanced version, PandaBoard ES, based on an OMAP4460. We were using the same experimental setup as for the STM32-F100RB to conduct 1000 measurements per board.

The SRAM start-up values have a worst-case *Hamming Weight* of $HW(x) = 48.53\%$, which is close to the ideal value of 50%. The measurement contains almost the same amount of zeros and ones, see Figure 3a, with a negligible bias towards 0. The OMAP SoC performs identical to the STM32F100 regarding this characteristic. The numbers for the *Within-class Hamming distance* are depicted in Figure 3b. They show a maximum within-class Hamming distance of 4.67%. This value is well below a bit error rate of 15%. In literature, an average bit error rate of 15% can be regarded as a reference value for SRAM PUF noise [13,14]. Compared to the STM32F100RB, the OMAP SoC exhibits less noise which leads to a decreased false rejection rate during the key reconstruction process. The minimum *Between-class Hamming distance* is 49.66%, see Figure 3c. Compared to the STM32F100RB the results for the OMAP 4 are even better, guaranteeing to provide a unique fingerprint for individual devices among a pool of similar platforms. However, it should be noted that the values for this measures are not as statistical significant as one might like. This is because we could only gather values from five distinct devices as it was not possible for us to get sufficient devices to provide as statistical significant results as desired for cost reasons. Table 1 shows the summarized results.

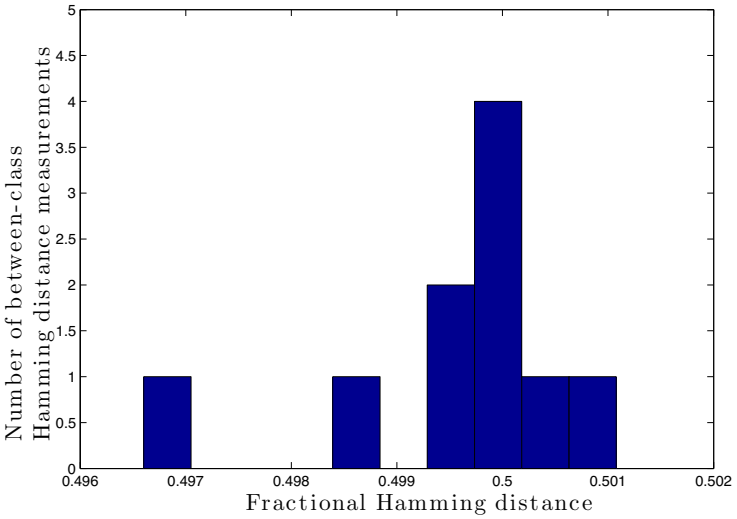


(a) Fractional Hamming Weight



(b) Within-class Hamming distance

Fig. 3. Detailed PUF characteristics for PandaBoards devices: (a) Fractional Hamming Weights of SRAM start-up values. (b) Within-class fractional Hamming distance of SRAM start-up values. (c) Between-class fractional Hamming distance histogram.



(c) Between-class Hamming distance

Fig. 3. (*Continued.*)

Table 1. PUF characteristics of the STM32F100RB and the OMAP 4 SoC

Characteristic	STM32	OMAP 4
Maximum within-class Hamming distance	9.61%	4.67%
Minimum between-class Hamming distance	46.48%	49.66%
Minimum Hamming weight	49.19%	48.53%
Maximum Hamming weight	50.23%	51.18%

4 PUF-Based Anti-counterfeiting Architecture

In this section we first introduce the capabilities of the attacker. Second, we introduce our proposed anti-counterfeiting architecture with respect to the attacker model. Finally, we show in detail how PUF enrollment and reconstruction works.

4.1 Attacker Model

A malicious party has several motives to attack deployed low-end devices. The attacker might want to extract intellectual property (IP) stored on the device in the form of software or secrets. After successful IP extraction the attacker would be able to use the IP on counterfeit devices or even sell self-made counterfeit solutions for less money than the original product. In another scenario the attacker might want to circumvent the vendor's licensing model by manipulating the firmware. To evade licensing restrictions the attacker could try either to modify the firmware to unlock features reserved for higher valued product versions or he attempts to downgrade to a previous firmware version, to exploit

design flaws and consequently escalate privileges on the system. Another motivation to alter the firmware is to capture valuable user data like passwords, credentials and usage data. Furthermore, the attacker might be able to actively alter output data, as for example in the case of smart metering, to report fake consumption data.

For the attacker to achieve one of the mentioned goals we consider him to have the following abilities. The attacker has physical access to the device due to the device's ubiquitous availability or because the attacker possesses the device as a legitimate user. The attacker can read out the contents of the external memory (DDR or flash memory) as it is highly exposed to external accesses. Thus, we imply that the attacker can read out and change the firmware that is stored on external memory. Additionally, the attacker is able to inspect and modify on-chip memory values with software of his choice after the boot process.

Besides these capabilities we assure that the attacker can not perform one of the following actions. The attacker is not able to change the code of the boot loader. We assume that the boot loader is stored in a masked read-only memory (ROM), which is under control of the manufacturer. Furthermore, we consider an attacker to not be able to replace the ROM chip with a second one of his choice, containing boot code under his control. Especially in the case of system-on-a-chip platforms on-chip memory is highly integrated and a replacement of a memory module is beyond the means of the average-skilled attacker. Lastly, the attacker is not able to read out the start-up values of the on-chip SRAM during start-up. The start-up values are protected by the boot loader and are erased shortly after the device gets out of reset. As soon as the device is powered the boot loader reads the start-up values, immediately overwrites them and erases any their instances before the firmware is called. The boot loader is assumed to be trusted and cannot be replaced. Hence, the first possibility for the attacker to execute code of his choice is after the boot loader finished execution.

We are aware of the fact that a physical attacker in possession of sufficient resources in terms of time and money can circumvent virtually any security mechanism. Nevertheless, if the attacker would succeed to extract the SRAM start-up values, i.e. the cryptographic key, he would only able to attack this individual device and has to perform the same attack for any other device.

4.2 General Architecture

The proposed anti-counterfeiting solution is designed for implementation on a variety of commodity hardware without on-board security facilities ranging from lightweight devices to more complex SoC platforms. Our solution requires hardware components, which are already present in virtually any computing device. In particular, we require the devices to be equipped with a masked ROM to hold the modified boot loader, the processor containing the MCU itself, on-board RAM (which is the source of the PUF instance) and external memory to store the encrypted firmware and so-called Helper Data. Helper Data is needed

to reliably reconstruct the key using the Fuzzy Extractor; its use will be explained in Section 4. Furthermore, we assume that the manufacturer can modify the boot loader to implement the key extraction and decryption functionality.

The structures of lightweight MCUs and more complex SoCs differ and require different approaches with respect to the architecture of our proposed solution. The general architecture of lightweight devices comprise the boot loader and the firmware. The entire boot loader needs to be stored irreversibly in masked ROM as it protects the SRAM start-up values and implements the functionality to derive the cryptographic key. The key is used to decrypt the firmware that is stored in external memory. In the case of more complex SoCs, usually a multi-staged boot loader is used consisting of a smaller part (1^{st}-stage boot loader) that fits into on-chip SRAM and a larger part (2^{nd}-stage boot loader) stored on external memory. Such platforms operate with a rich embedded operating system instead of a more basic firmware. Here, the 1^{st}-stage boot loader will be modified to perform key extraction and decryption routines and must be stored in masked ROM. The derived key will be used to decrypt the 2^{nd}-stage boot loader instead of the firmware in the MCU scenario. Figure 4 illustrates the two cases.

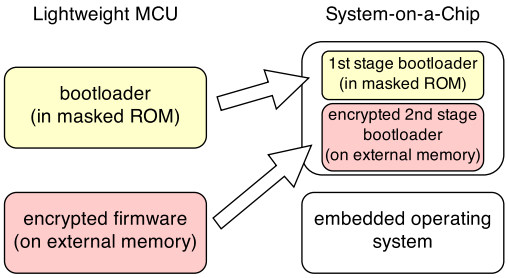


Fig. 4. Architecture of lightweight devices and more complex SoCs

The following paragraph explains the architecture for the SoC case in more detail as we used an SoC for our prototype implementation. In this scenario the 1^{st}-stage bootloader is wired programmed in a masked ROM and is executed as the first binary after the device start-up[3]. It queries the SRAM PUF, deriving the device-dependent key K. The key exists in on-chip memory only for the period of the following two steps. K is used to decrypt the 2^{nd}-stage boot loader, stored on non-volatile memory (e.g. flash memory). After successful decryption the 2^{nd}-stage boot loader derives a second key K' by hashing the concatenation of K and a salt value N: $K' = H(K|N)$. Key K' is subsequently used to decrypt the compressed kernel file that also resides in non-volatile memory (NVM). The second key K' is derived to impede the reconstruction of the initial key K in case an attacker captures K'. If the attacker captured K' then a new version of the

[3] More precisely, on our implementation board the first code executed is vendor-specific initialization code, which cannot be disabled and leads to a pre-initialized part of the L3 OCM RAM.

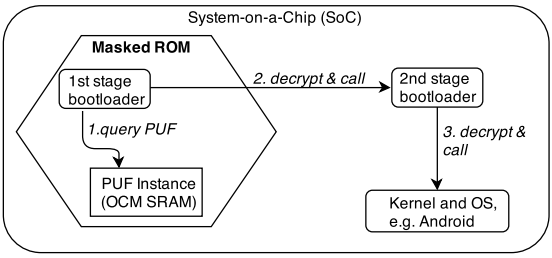


Fig. 5. Schematic view on the general architecture for SoC platforms

2^{nd}-stage boot loader including a new salt must be deployed which will generate a different K'_{new}. Furthermore, a new version of the firmware, encrypted under a new key K'_{new} must be distributed to regain a secure state. This design assures that only a 2^{nd}-stage boot loader can be executed that was encrypted by the correct device-depended cryptographic key K. Since the second key K' is derived from K also only such firmware can be properly loaded and executed, which was encrypted by the correct key as well. Thus, the operating system will only boot properly, if the correct combination of hardware and software is in place. The overall architecture is depicted in Figure 5.

4.3 Process of Usage

The usage of the proposed anti-counterfeiting architecture involves the enrollment of the used PUF (performed by the manufacturer) and the reconstruction phase (conducted every time the user boots the device).

Enrollment. The enrollment process is carried out by the manufacturer and is performed once for each device. It serves two main purposes: the derivation of key K and the generation of Helper Data W. The cryptographic key K is derived from a randomly chosen secret S by hashing it with a hash function H: $K = H(S)$. The secret S is predefined by the manufacturer and must be unique for every device. Since the start-up values always contain a certain amount of noise the raw start-up values need to be further processed to derive a stable output. This is done by applying a Fuzzy Extractor algorithm, to generate so-called Helper Data W. The Helper Data is constructed by XORing the output of the concatenated Fuzzy Extractor with the SRAM reference measurement R (the raw SRAM start-up values).

Helper Data will be used later during the reconstruction phase to reconstruct the secret S and to derive the key K given a noisy SRAM measurement R'. W is stored in external memory as it does not leak information about S. To protect the helper data from tampering several methods can be applied, such as the approach described by Boyen [6]. The enrollment process is shown in the upper part of Figure 6.

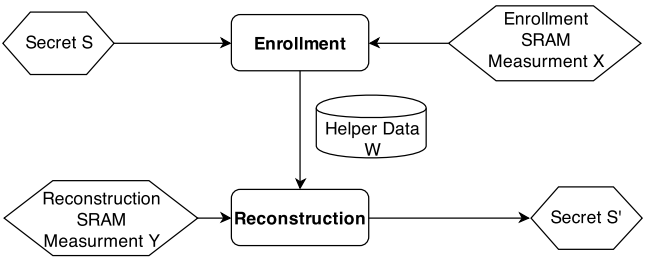


Fig. 6. Illustration of the enrollment and reconstruction of the Fuzzy Extractor

Reconstruction. The reconstruction process is performed at user's side and is executed every time the user boots the corresponding device. After the device gets out of reset, the 1^{st}-stage boot loader reads and stores the noisy SRAM start-up values R' and immediately overwrites the start-up values to make them inaccessible to a physical attacker. In a second step the boot loader reads the Helper Data W from external memory. The Fuzzy Extractor XORs R' with W and decodes the output using the concatenated decoders to construct the secret S'. The generated secret S' will only be equal to the correct secret S if the Helper Data corresponds to the respective device having SRAM start-up values R' that are similar to the enrollment measurement: $d(R, R') < \epsilon$. Next, the key K is derived by hashing secret S', which in turn is used to decrypt the firmware or the 2^{nd}-stage boot loader, depending on the scenario. The schematic composition of the reconstruction processes is depicted in the lower part of Figure 6.

5 Proof of Concept

We implemented the proposed anti-counterfeiting architecture on a SoC platform. We chose an SoC platform for the following reasons. Primarily, we wanted to prove the feasibility to robustly extract a unique fingerprint also from more complex platforms. Furthermore, the boot process is more complicated compared to lightweight devices. Our intention was to show that the proposed solution can be implemented into existing boot loaders. Lastly, the size of the memory available on the SoC for implementing the Fuzzy Extractor logic is comparable to the STM32F100 and similar low-end devices as shown in Section 3. Hence, the requirements regarding memory footprint are equal to those of low-end devices. A successful implementation for the SoC proves the feasibility to implement the architecture for lightweight devices as well.

We used u-boot [8,7], one of the most widely deployed boot loaders. It integrates a 1^{st}-stage boot loader (Memory Locator — `MLO`), which is small enough to fit into on-chip memory and a larger 2^{nd}-stage boot loader (`u-boot.img`). The `MLO` performs minor hardware initialization as well as the setup of external DDR memory. Afterwards it calls `u-boot.img` that is copied to DDR memory. It initializes further hardware components and eventually calls the operating system kernel.

Enrollment. According to Section 4, the Helper Data W is derived from a randomly chosen secret S and a reference measurement R using a Fuzzy Extractor during the enrollment phase. The Fuzzy Extractor design is based on the construction presented by Bösch et. al [5] and will be explained in more detail in Section 5.1. We also adapted the size for the secret S (22 Byte) from Bösch's design. During the enrollment also the key K is generated by hashing the secret S with the SHA-1 hash function to a 128 bit key such that it can be used as input for the AES cryptosystem in the next step. Having a secret that is larger then the generated key ensures sufficient entropy. K is used to encrypt the `u-boot.img` using the AES-128 block cipher. The second key K' is used to encrypt the kernel image file (`uImage`). The generated Helper Data is 675 bytes in total. Eventually, the Helper Data, as well as the encrypted files are stored in non-volatile memory (e.g. the flash card).

Reconstruction. The main part of the reconstruction logic is implemented in the `MLO`, being one of the first pieces of code to be executed. As described in Section 4 the MLO extracts the on-chip memory chip's fingerprint R' and processes it using a Fuzzy Extractor to derive the device-dependent key K. The Fuzzy extractor requires 675 bytes of L3 OCM RAM SRAM start-up values as well as W from the external memory to reconstruct K. Subsequently, K is used to decrypt `u-boot.img`. In particular, one after another 16 bytes of the decrypted `u-boot.img` are read in on-chip memory, get decrypted and are written back to external memory. Subsequently `u-boot.img` is called and in case of successful decryption it is executed. If the false key K was generated a fault handler routine is called, displays a warning message and cancels the boot process. A detailed scheme of the reconstruction process is depicted in Figure 7.

5.1 Fuzzy Extractor Design

To reproduce the secret key S from various noisy measurements error-correction is required. Following the suggestions of [5] we decided to implement a concatenated code comprising of two linear codes – a Golay code and a repetition code – to reconstruct the 22 byte secret. In particular, we are using a binary Golay-(23,12,7) code in combination with a repetition code with 15 repetitions. The false rejection rate (FRR) of the concatenated code $P_e(\text{total})$ can be calculated by equation (3). It is derived from the FRR probabilities of the linear repetition code – equation (1) – and the Golay code – equation (2).

$$P_e(\text{Repetition}) = \sum_{i=\lceil s/2 \rceil}^{s} \epsilon^i (1-\epsilon)^{s-i} \binom{s}{i} \tag{1}$$

$$P_e(\text{Golay}) = \sum_{i=4}^{23} P_e(\text{Repetition})^i (1 - P_e(\text{Repetition}))^{23-i} \binom{23}{i} \tag{2}$$

$$P_e(\text{total}) = 1 - (1 - P_e(\text{Golay}))^g \tag{3}$$

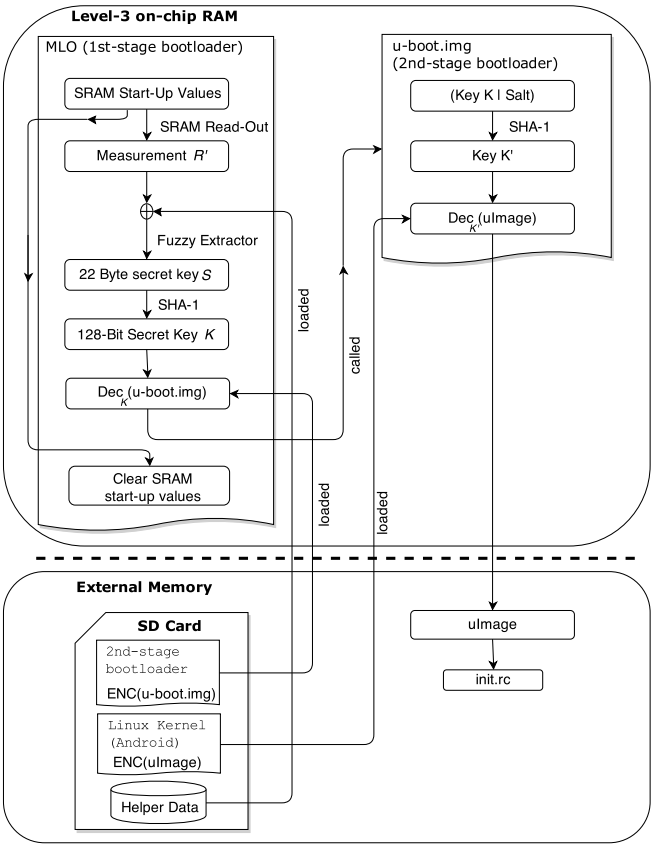


Fig. 7. Scheme of the reconstruction process of the ephemeral key inside the anti-counterfeiting architecture

Here, s is the number of repetitions, ϵ is the average bit error rate (BER) and g is the number of Golay code words needed (g = secret length/12). With this construction we can achieve a false rejection rate of 10^{-8} given an average BER of 15% as commonly used in literature [5]. The false rejection rate of the OMAP devices will be even lower since the measured BER is well below the reference value of 15% used in the calculations. Thus, the Fuzzy Extractor Design is suitable to reliably reconstruct a cryptographic key given several noisy SRAM PUF measurements for individual devices. The implemented Fuzzy Extractor requires 675 bytes of SRAM data to reconstruct a 22 Byte secret. The implementation requires only 0,05% of the available on-chip SRAM memory.

6 Conclusion

In this paper we proposed an anti-counterfeiting architecture and implementation for low-end devices by using intrinsic Physically Unclonable Functions found

in commodity hardware. Our approach does not require additional hardware and can be implemented the on-chip memory as a PUF and modifying the boot loader to extract a device-specific cryptographic key to decrypt the firmware with the device-specific key. We showed that low-end devices – including lightweight devices as well as System-on-a-Chip platforms – contain PUF instances that can be used to robustly identify the device. The analysis of the extracted on-board PUF instance showed almost optimal characteristics. Thus, our proposed solution is suitable to strengthen the security of low-end devices without on-board security mechanisms, keeping the costs at a minimum while significantly raising the efforts for attackers trying to extract or modify the firmware stored on such devices.

References

1. ARM TrustZone, `http://www.arm.com/products/processors/technologies/trustzone/index.php` (last accessed on January 17, 2014)
2. PandaBoard Platform, `http://pandaboard.org/content/platform` (last accessed on January 17, 2014)
3. ARM. ARM Holdings PLC Reports Results For The Fourth Quarter and Full Year (2013), `http://www.arm.com/about/newsroom/arm-holdings-plc-reports-results-for-the-fourth-quarter-and-full-year-2013.php` (last accessed on March 7, 2014)
4. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A Survey. Computer Networks: The International Journal of Computer and Telecommunications Networking, 2787–2805 (2010)
5. Bösch, C., Guajardo, J., Sadeghi, A.-R., Shokrollahi, J., Tuyls, P.: Efficient Helper Data Key Extractor on FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 181–197. Springer, Heidelberg (2008)
6. Boyen, X.: Reusable Cryptographic Fuzzy Extractors. In: ACM Conference on Computer and Communications Security, pp. 82–91 (2004)
7. Denk, W.: Das U-Boot – the Universal Boot Loader, `http://www.denx.de/wiki/U-Boot` (last accessed on July 9, 2013)
8. Ding, X., Liao, Y., Fu, J., Huang, H., Liu, W.: Analysis of Bootloader and Transplantation of U-Boot Based on S5PC100 Processor. In: Proceedings of the 2011 Third International Conference on Intelligent Human-Machine Systems and Cybernetics - Volume 01, IHMSC 2011, pp. 61–64 (2011)
9. Dodis, Y., Reyzin, L., Smith, A.: Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 523–540. Springer, Heidelberg (2004)
10. Feiri, M., Petit, J., Kargl, F.: Efficient and Secure Storage of Private Keys for Pseudonymous Vehicular Communication. In: Proceedings of the 2013 ACM Workshop on Security, Privacy & Dependability for Cyber Vehicles, CyCAR 2013, pp. 9–18 (2013)
11. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA Intrinsic PUFs and Their Use for IP Protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
12. Kortuem, G., Kawsar, F., Sundramoorthy, V., Fitton, D.: Smart Objects As Building Blocks for the Internet of Things. IEEE Internet Computing, 44–51 (2010)

13. Maes, R., Tuyls, P., Verbauwhede, I.: Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 332–347. Springer, Heidelberg (2009)
14. Maes, R., Tuyls, P., Verbauwhede, I.: Soft Decision Helper Data Algorithm for SRAM PUFs. In: Proceedings of the 2009 IEEE International Conference on Symposium on Information Theory - Volume 3, ISIT 2009, Coex, Seoul, Korea, pp. 2101–2105 (2009)
15. Maes, R., Verbauwhede, I.: Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In: Towards Hardware-Intrinsic Security, Information Security and Cryptography, pp. 3–37 (2010)
16. Papadimitratos, P., De La Fortelle, A., Evenssen, K., Brignolo, R., Cosenza, S.: Vehicular Communication Systems: Enabling Technologies, Applications, and Future Outlook on Intelligent Transportation. IEEE Communications Magazine, 84–95 (2009)
17. Schrijen, G.J., van der Leest, V.: Comparative analysis of SRAM memories used as PUF primitives. In: DATE, pp. 1319–1324 (2012)
18. Yick, J., Mukherjee, B., Ghosal, D.: Computer Networks: The International Journal of Computer and Telecommunications Networking. Comput. Netw., 2292–2330 (2008)