Towards a Secure Key Generation and Storage Framework on Resource-Constrained Sensor Nodes

Michael Höberl
Alpen-Adria-Universität Klagenfurt
Technikon Forschungs- und Planungsgesellschaft mbH
hoeberl@technikon.com

Ihtesham Haider
Institute of Networked and Embedded Systems
Alpen-Adria-Universität Klagenfurt
ihtesham.haider@aau.at

Bernhard Rinner
Institute of Networked and Embedded Systems
Alpen-Adria-Universität Klagenfurt
bernhard.rinner@aau.at

Abstract

Sensors have an essential role in the currently emerging Internet of Things (IoT) whereby these resource constrained nodes sense the environment, process the sensed data and forward it to a central entity. As some of this data might be sensitive in nature, it is crucial to ensure certain security guarantees on the sensed data such as integrity, confidentiality, and authenticity. Certain cryptographic primitives offer these security properties based on the assumption that each node is capable of generating and storing its secret key securely. Existing solutions for secure key storage are based on secure non-volatile memory. Physically Unclonable Functions (PUFs) are lightweight hardware security primitives that bind a unique key to each device and provide resistance against key compromise attacks as the key exists in form of hardware manufacturing variations, which are difficult to read as compared to memory based alternative solutions. Therefore, in this work we present a PUF-based lightweight, secure key generation and storage framework for resource constrained sensor nodes. Our implementation of a ring oscillator (RO) PUF based key generation scheme shows that the PUF based key generation consumes only a small part of a typical visual sensor node's resources.

Categories and Subject Descriptors

H.3.4 [**Information Systems**]: Information Storage and Retrieval—*Systems and Software*

General Terms

Design, Security

Keywords

sensor nodes, IoT, physically unclonable functions, key generation, key storage

1 Introduction

Sensor networks have shown great promise for enabling a variety of applications including monitoring for safety and security, identifying environmental pollution sources, tracking personal health parameters, and measuring traffic flows. A number of attacks on these sensor networks have been reported in literature, which necessitate protection of sensor nodes, sensed data, and communications in these networks [12]. A lot of research on how to secure sensor nodes and networks is currently underway. However, many of the proposed solutions do not explicitly address the protection of secret keys, which makes the key compromise a serious security vulnerability. In this work, we address key compromise vulnerability by proposing a secure key storage and generation framework for resource-constrained sensor nodes based on Physically Unclonable Functions (PUFs).

In today's age of pervasive computing, sensor networks are increasingly becoming ubiquitous as well. Our motivation for this work are today's emerging ubiquitous sensing paradigms such as Internet of Things (IoT) and Participatory Sensing (PS). These sensing paradigms have the potential to impact every day human life by applications spanning health-care, urban sensing, citizen journalism, and law enforcement etc. The reliability of services offered by these applications, however, depends on the security of sensor nodes, sensed data and communications. Security should therefore be considered for next generation platforms in a systematic way.

The scale and ubiquitousness of these networks present a multitude of challenges in formulating a holistic security solution [12]. First, the attack surface has immensely increased potentially leading to new, unidentified attacks. Second, sensors for these paradigms are envisaged to be heterogeneous, so presenting a generic solution for a wide spectrum of sensors is another challenge. Finally, the security solution must be lightweight for resource-constrained sensors. A key issue while addressing all these security challenges is the secure key storage, which must be lightweight and cost-effective. Physical Unclonable Functions [5] are promising candidates for enabling a secure and lightweight key storage solution.

We propose a lightweight secure key generation and storage framework based on PUFs for resource-constrained distributed sensor nodes. We implement the proposed key generation framework on a SoC, that is compatible with SoCs used in today's sensor nodes.

The reminder of the paper is structured as follows: Section 2 introduces the concept of PUFs and highlights related work. Section 3 discusses the PUF-based key generation framework and its building blocks such as the helper data algorithm and the PUF source. Section 4 describes the design of the framework's building blocks. Section 5 gives an overview of the implementation work done so far. Section 6 concludes this paper and summarizes next steps.

2 Related Work

A PUF is a function embodied in a physical object that maps an input to an output. This mapping is based on the physical structure of the object, whereas this structure is influenced by uncontrollable variations during the manufacturing process and is believed to be unique for each instance. Thus, two PUF instances have never the same mapping, as the chip manufacturer is not able to control these process variations. As the response is based on the physical structure of the integrated circuit (IC), two responses from the same instance differ also slightly from each other. A commonly used measure for these variations is the so-called Hamming distance, which is defined as the number of bits in which the two responses differ. In the PUF context the intra-Hamming distance is used to quantify the error rate, wheras the inter-Hamming distance is a measure on how different two responses from two PUF instances are. Another prominent feature of a PUF response is its Hamming weight, which is defined as the sum of the bitstring and is thus a measure of the ratio between ones and zeros.

In [6] a comprehensive overview of different PUF types is given. Two well-known types are the SRAM and the ring oscillator (RO) PUF. The former is based on the random start-up values of SRAM cells and is therefore a member of the memory based PUFs. The latter is based on the delay within a ring oscillator and is thus a member of the delay based PUFs. An additional classifier between PUFs is their available challenge space which defines how many challenges are available to generate independent responses. The so-called strong PUF has a large challenge space and is often used for authentication proposes, whereas the weak PUF has a small challenge space and is often used for key generation. Note, that the terms strong and weak do not reflect the security of this PUF type in any kind.

Many cryptographic systems do not specify how the secret key has to be stored in a system in order to circumvent a possible key compromise. The underlying assumption is that a system is capable of storing the sensitive key material in a secure way for instance as in ZigBee [1]. Existing solutions such as [4], [11], [9] and TrustEYE [10] rely on TPM based secure non-volatile memory, which makes the solution costly, resource consuming, and vulnerable to physical attacks. Authors of [14] proposed PUFs for secret key generation for ARM Trustzone as an alternative to a secure memory. However, the solution uses an external SRAM as PUF source and thus this solution is not embedded anymore.

In this work we propose a PUF based secure key generation and storage as a framework for extremely resource-constrained distributed sensor nodes. In comparison to secure NVM, PUFs offer better physical security and enable more tightly integrated solutions. Moreover, PUF based key generation is extremely resource efficient and fast. As proof of concept, we also implemented a PUF based key generation mechanism to secure a visual sensor node.

3 Secure Key Generation & Storage Framework

Using a PUF for key generation on a sensor node has the main advantage that the generated key is unique on each node. This is based on the intrinsic variations of the hardware structure which are extracted and used as a basis for the generated key. Thus, a PUF-based key generation framework is not only able to generate and store a key in a secure way, but also to bind the key to the node. Compared to a secure on- or off-chip memory a PUF based solution is much cheaper as the helper data can be stored in an insecure non-volatile memory.

The framework consists of two main building blocks, where the choice of the first one has a direct influence on the design of the second one. The first building block is the deployed PUF instance which has to be chosen according to the given system. The parameters of the PUFs such as intra-Hamming distance and Hamming weight of the response must be considered when designing the second block, called the helper data algorithm. The aim of the helper data algorithm is to ensure that the generated key fulfills its cryptographic features and that the system is able to regenerate the key from the noisy PUF response.

The key on the sensor node can be used to implement a range of security functions for instance integrity, authenticity, and confidentiality of sensed data. Using the key, the design of the sensor node could be protected by ensuring access control and secure boot etc.

3.1 PUF Source

The main building block in the key generation framework is the PUF instance, which has to be chosen at first in order to design the remaining blocks. As most of today's deployed sensor nodes are composed of a processing unit (e.g. SoC) and several peripherals an additional SRAM might be available on the platform as well. If this SRAM is not initialized during the start-up of the system, it can be used as a PUF source. The on-chip memory of the processing system is also based on an SRAM, however, it is most of the time initialized during start-up and cannot be used as a PUF source. Another well-known PUF type is the ring oscillator PUF, which uses frequency variations of ring oscillators to generate a binary response. This PUF type has the advantage that it can be implemented on an ASIC as well as on an FPGA and the response's length can be scaled according to the system's requirements by varying the number of oscillators.

3.2 Helper Data Algorithm

A PUF response does not fulfill the requirements imposed in a cryptographic key such as reproducibility, unfirom distribution of bits and full entropy, therefore post-processing measures have to be taken into account. The reproducibility of a PUF response is ensured by applying error-correcting codes, which have to be designed based on the PUF's error

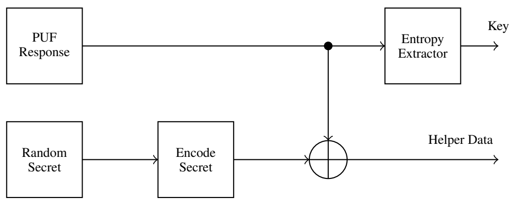

Figure 1. Generating a PUF-based key and helper data (enrollment phase).

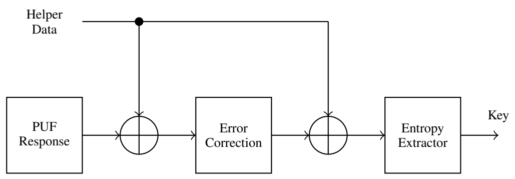

Figure 2. Reconstruction a PUF-based key using helper data (reconstruction phase).

rate. The error rate is often quantified, as already defined, by calculating the intra-Hamming distance that indicates the number of digits in which two bit strings differ.
In general a helper data algorithm consists of two stages. The first stage is often referred to as information reconciliation phase which deals with the noise. The second stage or privacy amplification deals with the non-uniform distribution of the responses. Both stages need so-called helper data that is generated during the enrollment phase. This phase must be executed in a secure and trusted environment. Later, the reconstruction phase uses the helper data as an input to reproduce along with a noisy PUF measurement the key in an insecure environment.

3.3 PUF-Based Secure Key Generation

The goal of a secure key generation framework is twofold. First, a key has to be generated with a pre-defined entropy (e.g. 64 bit or 128 bit) and stored in a secure way. Second, the key has to be reconstructed at any given time with a low failure rate (e.g. $P_{\text{fail}} \leq 10^{-6}$). Figure 1 depicts the enrollment phase which generates a key and the helper data that can be stored in a non-secure NVM such as an SD card.

Figure 2 depicts the reconstruction phase that aims to regenerate the key by reading out the PUF again, loading the helper data and using the error correcting code to correct the noisy bits within the response to be able to generate the key.

4 Design

The design of the key generation framework is crucial in order to fulfill the required parameters such as failure rate and entropy of the generated key. The failure rate of the system is mainly influenced by the deployed PUF and the corresponding error-correcting code within the helper data algorithm. A very noisy PUF needs a more sophisticated and complex error-correcting code than a less noisy PUF in order to cope with all possible bit errors. Thus, a sophisticated error-correcting code has to be used, which also increases the hardware utilization of the framework. Therefore, a low bit error rate is targeted when choosing the PUF to enable the usage of a less complex error-correcting code.

4.1 Architecture

The RO PUF generates response bits by comparing the counter value of two ring oscillators. An additional reference counter is used to stop the two RO counters at the same time to ensure a fair comparison of the oscillators. In total the design contains $b \times a$ oscillators, which are organized in so-called batches. A batch consists of a ring oscillators and an address decoder to choose the desired RO within each batch and to forward its output to the corresponding counter. In order to make the solution as lightweight as possible only two global RO counters and one reference counter have been implemented for the entire design.

4.2 Finding RO Pairs

A comparison of two ROs results in one response bit and as there are $n!$ possible combination to combine n ROs a well thought-through strategy has to be applied to find RO pairs. Such a strategy aims to find RO pairs which (i) generates responses with a low intra-Hamming distance by only using pairs with a high frequency mismatch and (ii) uses a RO just once to ensure independent bits.

The most natural strategy is to compare always neighboring oscillators. This might result in a high error rate, if neighboring oscillators have a very similar frequency. In [13] an algorithm is presented, referred to as Sequential Pairing Algorithm (SPA), which aims to find oscillators with a predefined frequency mismatch f_{th}. This is done by measuring each oscillator multiple times at the highest and lowest temperature for which the system is designed and choosing from these measurements RO pairs with a frequency mismatch $> f_{\text{th}}$. The generated link-list contains the RO pairs and has to be stored afterwards with the helper data to ensure that these pairs are always used when the PUF is evaluated.

4.3 Helper Data Algorithm

The used error correcting code within a HDA has to be designed carefully based on the PUF's error rate as well as on the available entropy within the response. In particular the bit error probability p_{b} is used, thus the probability that a bit does not flip is $1 - p_{\text{b}}$. We assume that all bits are independent of each other, therefore it is possible to calculate the probability that a string of n bits has more than t errors with Equation 1.

$$P_{\text{fail}} = \sum_{\text{i=t+1}}^{n} \binom{n}{i} p_{\text{b}}^{i}(1-p_{\text{b}})^{n-i} = 1 - \sum_{\text{i=0}}^{t} \binom{n}{i} p_{\text{b}}^{i}(1-p_{\text{b}})^{n-i} \tag{1}$$

As already stated, a helper data algorithm is a two step approach, where the first stage generates the key but also the helper data. The second stage uses the helper data along with a noisy response to regenerate the response used during the enrollment phase. There are two architectures known for this purpose namely the Code-Offset and the Syndrom construction [3].

The code-offset construction's helper data generator $Gen(r)$ takes a random input word m, encodes it to a codeword c and forms the helper data by $p \leftarrow r \oplus c$. The reconstruction procedure $Rep(r', p)$ is implemented as follows. First, the noisy codeword c' is obtained by $r' \oplus p$. Second, the decoding algorithm of the error correcting code generates

c by decoding the noisy codeword. Finally, the response r is computed by an additional XOR between of the corrected codeword and the helper data.

The Syndrom construction generates the helper data by calculating a matrix product of the response and a parity check matrix. More precisely, the helper data generator $Gen(r)$ is implemented as $p \leftarrow r * H^T$. The reconstruction algorithm $Rep(r', p)$ is implemented as follows. First, the syndrome is determined by $s = p - r' * H^T$ and with the help of the underlying decoding algorithm of the error correction code the error vector e is computed. Second, with the error vector e it is possible to compute the response by $r \leftarrow r' - e$.

In a PUF framework it is common to use block codes as an error correcting code [3], which are defined by four parameters. First, the message length of a binary linear code C is denoted with n and the corresponding codeword length with k. Second, the minimum Hamming distance d_{min} corresponds to the minimum distance of two valid codewords of C. Third, a $[n,k,d_{\min}]$-code is a binary code C of length n, a message length of k, and a minimum distance of $d_{\min}$. Thus, a linear code with a given minimum distance d is able to correct up to $t = \lfloor (d_{\min} - 1)/2 \rfloor$ errors.

The HDA takes a response as input and compresses it to a key k by using a hash function. The number of response bits needed for a key with a certain length is based on the entropy of the PUF's response. When deploying the already in Section 4.2 discussed SPA strategy every ring oscillator is just used once to generate a response, therefore the bits can be assumed to be independent and thus every bit has an entropy of 1. Therefore, if a key with full entropy and a length of 128 bits is demanded from a RO PUF the response length must be at least N. Based on this result an error correcting code can be chosen. First, the amount of source bits have to be calculated, which yields in this case to 128. Second, it has to be assumed that all bits within the response are independent and therefore the probability P_{fail}, that a response r with n bits has more than t errors, is given by Equation 1. Third, the bit error probability p_{b} has to be known. If the experiments resulted in an error probability of p_b it is a good practice to add a minor margin to be on the safe side. Furthermore, a failure during the decoding step should happen with the probability $P_{\text{fail}} \leq 10^{-6}$. Now it is possible to take certain error correcting codes into account and evaluate them.

4.4 Securing the Framework

Using a PUF as a security anchor in a system is quite new approach and thus there are frequently new attacks reported on this technology. Most of these attacks are aiming at manipulating the public available helper data to gain information on the response. Storing the helper data in a secure memory would undermine the entire PUF principle, thus it is necessary to apply other countermeasures. In [3] an approach is presented, that detects helper data manipulation. During the enrollment phase an additional Hash value $z = Hash(p,k)$ is stored. During the reconstruction z' is calculated and if $z \neq z'$ the key generation is aborted. If an additional link-list for choosing the RO pairs has to be stored as well, it has to be included to the Hash as well.

Another potential attack target is the error correction code if used in a code-offset construction as pointed out in [7]. The authors propose a technique called codeword-masking, which uses the linearity of the used code, as a countermeasure.

Table 1. Intra-Hamming distance and Hamming weight of the implemented RO PUF based on two different evaluation strategies.

	Neighboring ROs	SPA
Intra-Hamming Distance	7%	1.4%
Hamming Weight	49.3%	51.2%

5 Implementation

This section discusses initial results from our prototype implementation of the RO-PUF based key generation core and a visual sensor node. We used a MicroZed board as a platform for implementation. Some prominent features of this platform are (i) the Zynq7010 SoC, which consists of a programmable logic equivalent to $\approx 430K$ ASIC gates, (ii) a dual core ARM Cortex-A9 MPCores and (iii) 1 GB DDR3 SDRAM. For detailed specifications, readers are referred to [2].

5.1 RO PUF

The RO PUF is based on a number of symmetric ring oscillators. The ring oscillators have to be symmetric in terms of their routing to prevent a biased response. The Xilinx toolchain offers for this purpose so called Hard Macros, which assign the same routing to each RO instance. As $2n$ ROs are needed to form a response with n bits, we decided to implement a 3-stage ring oscillator which fits into one Slice in order to minimize the hardware utilization. Two ROs are chosen via selecting their corresponding batch and their address within the batch. As soon as the measurement circuit is triggered, both ROs start to oscillate, their counter counts the rising edges of the oscillation and a reference counter, which uses the board's clock as an input, starts to count as well. When the reference counter reaches a predefined threshold it stops both ROs' counters and their result is forwarded to a register in order to read it on the ASIC. On the ASIC the comparison of both counter values yields to a response bit. This is done for all ROs in order to generate a response with n bits. The entire control overhead of the circuit such as addressing or resetting is done via dedicated registers on the ASIC to make the system as flexible as possible during the prototyping stage. Note that the final system should not handle the sensitive counter values separately and perform the comparison directly on the programmable logic.

The implemented RO PUF was used to perform a first assessment of its parameters as intra-Hamming distance and Hamming weight. The results of this analysis are summarized in Table 1. It can be seen that the SPA algorithm decreases the error rate from 7% to 1.4%. Note that at this stage only measurements at room-temperature are available and therefore the SPA algorithm uses only measurements obtained at room temperature. In order to assesses the uniquness of the PUF response and thus the resulting key, the PUF instance was implemented on 9 different Xilinx boards and the inter-Hamming distance between these responses'

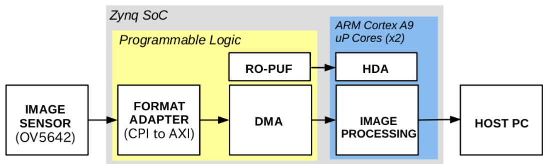

Figure 3. High level architecture of ProSecCo visual sensor node.

was caluclated. This evaluation resulted in an average inter distance of $\approx 49\%$, which indicates that each PUF instance generates a unique response.

5.2 HDA Building Block

Based on the PUF's intra-Hamming distance presented in the last section a tailored HDA can be designed and implemented. This HDA has to cope with an error-rate of $\approx 2\%$. As the bit error probability p_b and the response's entropy is known the error-correcting code can be parametrized by applying the concept presented in Section 4.3. We considered a Repetition, BCH and Reed Muller code with the parameters presented in Table 2.

Table 2. Three different error correcting codes for a bit error-rate of 2%**.**

Code (n,k,d)	P_{fail}	# of source bits
RM (16,5,8)	$2.401331 * 10^{-3}$	410
Rep. (9,1,9)	$3.770032 * 10^{-7}$	1152
BCH (31,6,15)	$1.338228 * 10^{-7}$	662

Based on the results presented in Table 2 the BCH(31,6,15) code was chosen and implemented as it (i) provides a failure rate $\leq 10^{-6}$ and (ii) the number of source bits is by a factor of ≈ 2 smaller than for the Repetition code. The disadvantage of the BCH code's more complex decoding algorithm can be neglected as only half of the ROs are needed in comparison with the HDA based on a repetition code. The BCH code was implemented in the code-offset construction to generate the key and the helper data.

5.3 Visual Sensor Node

To demonstrate the comparison of resource consumption between a typical sensor node and PUF based key generation core, we used the ProSecCo visual sensor node, which uses a *Zynq*7010 SoC as a development platform [8]. Readout for the Omnivision's 5642 image sensor utilizes reprogrammable fabric of the SoC, whereas image processing functions are implemented using the ARM Cortex-A9 cores. We have implemented a 42×32 based PUF circuit on the FPGA part of the SoC, which generates a response with 672 bits, and the associated HDA on the ARM processor core. Table 3 shows a comparison of FPGA logic area utilization between the PUF core and the two cores of image sensor readout implemented on FPGA, i.e., Format Adapter and DMA as illustrated in Figure 3. The results show that PUF based key generation consumes negligible resources as compared to the total resources of the considered sensor.

Table 3. Utilization of the implemented RO-PUF solution on the FPGA.

	Registers	Look Up Tables
Available	$35,200$	$17,600$
PUF Utilization	11%	22%
VSN Utilization	24.2%	40.7%

6 Conclusion

In this paper we presented a PUF-based key generation and storage framework and preliminary results of a visual sensor node that features this framework. The framework generates a cryptographic 128 bit key and is able to regenerate the key on-the-fly. Thus, only helper data has to be stored and as the helper data does not contain any information about the key itself, an insecure NVM is feasible as storage. The presented framework is also resistent to known attacks on a RO PUF based key generation scheme by applying the corresponding countermeasures.

Before this framework is deployable on a sensor node in general, further steps are necessary. A detailed characterization of the implemented PUF instance is mandatory, since up to now only measurments at room temperature have been contucted. However, the PUF's error rate depends on the temperature and also on the applied voltage, therefore an evaluation of the error rate under different environment and operating conditions is part of our future work. Whereas different environment conditions will not influence the error rate as the applied sequential pairing algorithm is designed to find RO pairs, which result in a stable bit over the entire temperature range. Addtional to variations of the supply voltage, the error rate is also influenced by aging effects, which have to be analyzed and addressed. Therefore, for the final design we also have to consider the usage of concatenated codes instead of a single code for the error correction part of the HDA in order to achieve a negliable failure rate. When comparing the hardware utilization of the VSN system to the key generation framework it can be seen that the key generation framework uses considerably less silicon area than a basic VSN system. Thus it can be argued that the additional security feature comes with a low cost in terms of hardware utilization, but enhances the security of the system significantly. A detailed comparison of consumed resources by PUF-based key generation and VSN including logic area, time and energy is work in progress. We also plan to implement the key generation framework on a variety of resource constrained sensors.

Acknowledgments

This research has been funded by the Austrian Research Promotion Agency (FFG) under grant number 842432. Michael Höberl was supported by the FP7 research project MATTHEW under grant number 610436.

7 References

[1] Z. Alliance. Zigbee specification, 2006.
[2] AVNET. Microzed board, 2015. `http://zedboard.org/product/microzed`, [Online; accessed 04-November-2015].

[3] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede. Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(99):889–902, 2014.

[4] A. Dua, N. Bulusu, W.-C. Feng, and W. Hu. Towards trustworthy participatory sensing. In *Proceedings of the 4th USENIX conference on Hot topics in security*, pages 8–8. USENIX Association, 2009.

[5] R. Maes. Puf-based key generation. In *Physically Unclonable Functions*, pages 143–168. Springer, 2013.

[6] R. Maes and I. Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. In *Towards Hardware-Intrinsic Security*, pages 3–37. Springer, 2010.

[7] D. Merli. *Attacking and Protecting Ring Oscillator Physical Unclonable Functions and Code-Offset Fuzzy Extractors*. PhD thesis, München, Technische Universität München, 2014.

[8] ProSecCo. Smart camera prototype. `http://trusteye.aau.at/prosecco/`, [Online; accessed 16-December-2015].

[9] S. Saroiu and A. Wolman. I am a sensor, and i approve this message. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, pages 37–42. ACM, 2010.

[10] T. Winkler, A. Erdelyi, and B. Rinner. TrustEYE.M4: Protecting the Sensor - not the Camera. In *Proceedings of the IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, Seoul, Korea, 2014.

[11] T. Winkler and B. Rinner. Securing Embedded Smart Cameras with Trusted Computing. *EURASIP Journal on Wireless Communications and Networking*, 2011:1–20, 2011.

[12] T. Winkler and B. Rinner. Security and Privacy Protection in Visual Sensor Networks: A Survey. *ACM Comput. Surv.*, 47(1):2:1–2:42, May 2014.

[13] C.-E. Yin and G. Qu. LISA: Maximizing RO PUF's secret extraction. In *Proceedings of the International Symposium on Hardware-Oriented Security and Trust*, pages 100–105, June 2010.

[14] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng. Providing Root of Trust for ARM TrustZone using On-Chip SRAM. Cryptology ePrint Archive, Report 2014/464, 2014. `http://eprint.iacr.org/`.