FPGA Intrinsic PUFs and Their Use for IP Protection

Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls

Information and System Security Group
Philips Research Laboratories, Eindhoven, The Netherlands
{Jorge.Guajardo,Sandeep.Kumar,Geert.Jan.Schrijen,Pim.Tuyls}@philips.com

Abstract. In recent years, IP protection of FPGA hardware designs has become a requirement for many IP vendors. In [34], Simpson and Schaumont proposed a fundamentally different approach to IP protection on FPGAs based on the use of Physical Unclonable Functions (PUFs). Their work only assumes the existence of a PUF on the FPGAs without actually proposing a PUF construction. In this paper, we propose new protocols for the IP protection problem on FPGAs and provide the first construction of a PUF intrinsic to current FPGAs based on SRAM memory randomness present on current FPGAs. We analyze SRAM-based PUF statistical properties and investigate the trade offs that can be made when implementing a fuzzy extractor.

1 Introduction

In today's globalized economy, it has become standard business practice to include third party Intellectual Property (IP) into products. This trend has led to the realization that internally developed IP is of strategic importance, for two reasons: (i) it decreases the design cycle by implementing re-use strategies and (ii) it is a source of additional licensing income from external parties. However, licensing IP to external parties forces IP vendors to ensure that they can generate revenues from their developed IP blocks. This is only guaranteed if designs are properly protected against theft, cloning, and gray market overproduction.

1.1 The Problem of IP Protection on Reconfigurable Hardware

SRAM based FPGAs offer a very flexible solution for implementation of valuable designs since they can be reprogrammed in the field. This allows for instance to update current designs with new and improved ones and stands in sharp contrast with implementations on ASICs. FPGA designs are represented as bitstreams and (most commonly) stored in external memory *e.g.* PROM or flash. When the FPGA is powered up, the bitstream is loaded onto the FPGA and the FPGA is configured. During loading, an attacker can easily tap the bitstream and make a copy of it, which he can then use to (illegally) program other FPGAs without paying the required licensing fees to the IP owner. This attack is called a *cloning* attack and it is a serious concern to IP developers nowadays.

Clearly encryption of the bitstream with a key that is specific to a particular FPGA would solve the problem. This observation is due to Kean [21], who also proposes an associated protocol to support IP protection. The protocol is based on bitstream encryption using a key stored in non-volatile memory on the FPGA. By eavesdropping the bus between the external memory and the FPGA the attacker can only obtain an encrypted version of the design. As long as the secret key is securely stored on the FPGA, the attacker can not perform a successful cloning attack. One general problem with this solution is that there is no non-volatile memory on SRAM FPGAs to store a long-term key. In order to solve this problem two main solutions have been proposed: (i) some non-volatile memory such as flash is added to the FPGA and (ii) the FPGA stores a long-term key in a few hundred bits of dedicated RAM backed-up by an externally connected battery. Both solutions come with a price penalty and are therefore not very attractive. The second solution has the additional disadvantage that the battery has only a limited life time and that batteries can get damaged which shortens further their life-time. Both effects have as a consequence that the key and the design are lost after some time, rendering the overall IP block non-functional. Notice that there are certain problems that can not be easily solved via bitstream encryption alone. Simpson and Schaumont [34] have identified two potential problems if the aim of the solution is to secure third party intellectual property and software modules. These are: (i) Intellectual Property (IP) authentication by system (SYS) developers as well as authentication of the hardware platform (where the software IP is running) by the IP providers (IPP) and (ii) protection of the software that is running on the processors configured on the FPGA. We notice that there are other security services which can be envisioned between the different parties involved in the chain, from hardware manufacturer (HWM) to End User. Table 1 summarizes security services that can be required by different parties in the overall IP protection chain. These parties include: the end user, the FPGA customer, the system integrator or designer (SYS), the hardware IP-Provider or core vendor (IPP), the hardware (FPGA) manufacturer (HWM) or vendor, the CAD software vendor, and a Trusted Third Party (TTP). In the remainder of the paper we will only deal with the SYS, IPP, HWM, and TTP. We refer to Kean [21] for a detailed description of the parties involved in the FPGA IP chain.

Table 1. Security Services in the IP Protection Chain

	Security Service	Description
S1	Hardware IP authentication	A hardware design runs only on a specific hardware device, hence it can not be cloned.
S2	Hardware platform authentication	The hardware platform (FPGA) allows only authentic designs to run on it.
S3	Complete design confidentiality	The intended design recipient (this could be the system integrator, the end user, etc.) has only access to the design as a black box (input/output behavior). No other party (in addition to the design developer) knows anything about the hardware IP.
S4	Secure hardware IP updating	Given that there is already an authentic design running on the FPGA, the IP provider would like to update it and at a minimum keep all the security guarantees that the previous design kept.
S5	Design traceability	Given an IP block, the designer can trace back who the intended recipient of the design was.
S6	User privacy	A design should not be linkable to the identity of the end-user

1.2 Our Contributions

In this paper, we will focus on providing services S1, S2 and S3 from Table 1. In particular, we propose new and improved protocols for IP protection on FPGAs. We show that the protocols of [34], while secure (i.e. we do not present any attacks against them), can be considerably simplified. We describe simplifications in terms of communication complexity, assumptions, and number of encryptions performed. We believe that one reason for this is the fact that the assumptions made on the primitives used in [34] were not clearly stated. To this end, we provide a review of the primitives and of the encryption schemes that can be used in such protocols. We then clearly state the assumptions made about these primitives and base the security analysis of our newly proposed protocols on them. A second contribution of the paper is the introduction of protocols which provide privacy from the TTP. In other words, previous protocols allow the TTP to have access to the IP block exchanged between the IPP and the SYS. In practice, this might not be desirable from the IPP's point of view. Thus, we introduce a protocol that allows for this at the cost of introducing a public-key (PK) based operation. The cost is minimal and it does not affect the resource requirements of the FPGA implementation when compared to the work in [34]. This is achieved by performing the PK operation during the *online* phase of the protocol. A third contribution of the paper regards the implementation of an actual Physical Unclonable Function (PUF) on an FPGA which is *intrinsic* to the FPGA. Notice that this means that the PUF is already present on the FPGA and thus, it requires no modifications to the actual hardware. As far as we are aware, this is the first time that such a PUF is reported in the literature. Notice that the work of [34] only *assumes* the existence of such PUF on an FPGA and models its behavior via an AES module. Finally, we show some of the trade-offs that can be made when implementing a fuzzy extractor [11, 26].

Organization. Section 2 provides an overview of PUFs, security assumptions and their properties. In addition, we survey symmetric-key schemes that provide both privacy and authentication. In Sects. 3 and 4, we use these constructions to simplify the protocols proposed in [34]. We also introduce a protocol that provides total privacy, even from the TTP. Section 5 introduces intrinsic PUFs and a construction based on the properties of SRAM blocks present on FPGAs. In addition, we analyze SRAM-based PUFs randomness and statistical properties. We end in Sect. 6 analyzing possible fuzzy extractor implementation options.

2 Preliminaries

2.1 Physical Unclonable Functions

Physical Unclonable Functions consist of inherently unclonable physical systems. They inherit their unclonability from the fact that they consist of many random components that are present in the manufacturing process and can not be controlled. When a stimulus is applied to the system, it reacts with a response.

Such a pair of a stimulus C and a response R is called a *challenge-response* pair (CRP). In particular, a PUF is considered as a function that maps challenges to responses. The following assumptions are made on the PUF:

1. It is assumed that a response R_i (to a challenge C_i) gives only a negligible amount of information on another response R_j (to a different challenge C_j) with $i \neq j$.
2. Without having the corresponding PUF at hand, it is impossible to come up with the response R_i corresponding to a challenge C_i, except with negligible probability.
3. Finally, it is assumed that PUFs are tamper evident. This implies that when an attacker tries to investigate the PUF to obtain detailed information of its structure, the PUF is destroyed. In other words, the PUF's challenge-response behavior is changed substantially.

We distinguish between two different situations. First, we assume that there is a large number of challenge response pairs (C_i, R_i), $i = 1, \ldots, N$ available for the PUF; i.e. a strong PUF has so many CRPs such that an attack (performed during a limited amount of time) based on exhaustively measuring the CRPs only has a negligible probability of success and, in particular, $1/N \approx 2^{-k}$ for large $k \approx 100$ [28, 35]. We refer to this case as strong PUFs. If the number of different CRPs N is rather small, we refer to it as a weak PUF. Due to noise, PUFs are observed over a noisy measurement channel *i.e.* when a PUF is challenged with C_i a response R_i' which is a noisy version of R_i is obtained. Examples of PUFs include optical PUFs [28, 29], silicon PUFs [14] and coating PUFs [38]. Although coating PUFs are very cheap to produce they still need a small additional manufacturing step. In this paper we introduce the notion of an *Intrinsic* PUF (IPUF), *i.e.* a PUF that is inherently present in a device due to its manufacturing process and no additional hardware has to be added for embedding the PUF. We will give an example in Sect. 5.

2.2 Fuzzy Extractor and Helper Data Algorithm

In [38] it was explained that PUFs can be used to store a secret key in a secure way. Since, PUF responses are noisy as explained above and the responses are not fully random, a Fuzzy Extractor or Helper Data Algorithm is needed to extract one (or more) secure keys from the PUF responses. For the precise definition of a Fuzzy Extractor and Helper Data algorithm we refer to [11, 26]. Informally, we need to implement two basic primitives: (i) *Information Reconciliation* or error correction and (ii) *Privacy Amplification* or randomness extraction. In order to implement those two primitives, helper data W are generated during the *enrollment phase*. Later during the *key reconstruction* phase, the key is reconstructed based on a noisy measurement R_i' and the helper data W. During the enrollment phase (carried out in a trusted environment), a probabilistic procedure called Gen is run. It takes as input a PUF response R and produces as output a key K and helper data W: $(K, W) \leftarrow \mathsf{Gen}(R)$. During the key reconstruction

phase a procedure called Rep is run. It takes as input a noisy response R' and helper data W and reconstructs the key K (if R' originates from the same source as R) *i.e.* $K \leftarrow \mathsf{Rep}(R', W)$. In order to implement the procedures Gen and Rep we need an error correction code $\mathcal{C}$ and a set $\mathcal{H}$ of universal hash functions [9]. The parameters[1] $[n, k, d]$ of the code $\mathcal{C}$ are determined by the length of the responses R and the number of errors t that have to be corrected. The distance d of the code is chosen such that t errors can be corrected. During the enrollment phase a response R is obtained and a random code word $C_S \leftarrow \mathcal{C}$ is chosen from $\mathcal{C}$. Then, a first helper data vector equal to $W_1 = C_S \oplus R$ is generated. Furthermore, a hash function h_i is chosen at random from $\mathcal{H}$ and the key K is defined as $K \leftarrow h_i(R)$. The helper data $W_2 = i$. Summarizing the procedure Gen is defined as follows, $(K, W_1, W_2) \leftarrow \mathsf{Gen}(R)$. Finally, during the key reconstruction phase R' is obtained. During the procedure Rep the following steps are carried out: (1) *Information Reconciliation*: Using the helper data W_1, $W_1 \oplus R'$ is computed. Then the decoding algorithm of $\mathcal{C}$ is used to obtain C_S. From C_S, R is reconstructed as $R = W_1 \oplus C_S$; and (2) *Privacy amplification*: The helper data W_2 is used to choose the correct hash function $h_i \in \mathcal{H}$ and to reconstruct the key as follows: $K = h_i(R)$.

2.3 On Authenticated Encryption

There has been considerable work in the crypto community on authenticated encryption. In other words, how to obtain privacy and integrity at the same time in the symmetric-key setting. Our aim in this section is to summarize known results and to caution against combining primitives without any formal analysis. In later sections, we will use these results to justify the security of the schemes that we propose or to notice potential vulnerabilities of the proposed schemes. Throughout the paper we will refer to *encrypting* [6], denoted $\mathtt{Enc}_K(\cdot)$, meaning an encryption scheme providing semantic security under chosen plaintext attacks[2] [15, 12], commonly written IND-CPA. Finally, we write $\mathtt{MAC}_K(\cdot)$, to indicate a message authenticating code (MAC) computed with the secret-key K providing integrity of plain texts (see [5]). Next, we recall different constructions considered in the literature and their conclusions.

Bellare and Namprempre [5] analyze three *generic* composition paradigms to provide privacy and authentication via symmetric-key encryption schemes. We emphasize that their analysis is for generic composition, meaning that they make black-box use of symmetric encryption and MAC schemes. Three composition methods are considered: (i) Encrypt-and-MAC := $\mathtt{Enc}_{K_{enc}}(M)||\mathtt{MAC}_{K_{MAC}}(M)$, (ii) MAC-then-encrypt:= $\mathtt{Enc}_{K_{enc}}(M||\mathtt{MAC}_{K_{MAC}}(M))$, and (iii) Encrypt-then-

[1] Given a $[n, k, d]$-code $\mathcal{C}$ over $\mathbb{F}_q$ its words are n-tuples of $\mathbb{F}_q$ elements. The code has minimum distance d, it can correct up to $\lfloor (d-1)/2 \rfloor$ errors, and it has cardinality q^k; i.e. it can encode up to q^k possible messages.

[2] There are stronger versions of security, such as semantic security under chosen ciphertext attacks (IND-CCA), however, common modes of operation (e.g. CBC) only provide IND-CPA.

MAC:= $D||\texttt{MAC}_{K_{MAC}}(D)$, where $D = \texttt{Enc}_{K_{enc}}(M)$. It is proved in [5] that under generic composition the Encrypt-and-MAC scheme fails to preserve privacy, while providing integrity. Furthermore, this is true for *any* deterministic MAC such as [4, 3, 24]. The other two constructions preserve privacy under CPAs and provide integrity of plaintexts. We refer to [5] (see also [23]) for the details but notice that the third construction is the one that provides the strongest security guarantees. In [1], An and Bellare study whether adding redundancy to a message and then encrypting it (i.e., $\texttt{Enc}_K(M||\tau)$ where $\tau = h(M)$, h some function of M), provides both privacy and authenticity. They show that the privacy of the encryption-with-redundancy is inherited from the original encryption scheme $\texttt{Enc}_K(\cdot)$. However, integrity depends on whether the function h is public or keyed with a secret key. In particular, for redundancy computed via public functions known to the adversary (e.g. via a keyless hash function like SHA-1), the resulting scheme does not provide integrity. On the other hand, if the redundancy function is computed incorporating a secret key, then the resulting scheme provides integrity. We notice that this is probably the reason why in [34], the integrity information is encrypted with a second key[3]. Finally, a number of schemes have been explicitly developed to provide authentication and privacy in the symmetric-key setting (see for example [39, 19, 31]).

3 Offline HW/SW Authentication for FPGAs

In the remainder of this paper, we will denote an IP block by *SW* and use this terminology interchangeably. In [34], Sympson and Schaumont describe a protocol which provides hardware IP authentication (S1) and hardware platform authentication (S2). For completeness, the protocol is shown in Fig. 1. In Fig. 1, we have written $\texttt{Enc}(\cdot)$ to mean the symmetric encryption of the argument. Although, no assumption is mentioned in [34], we assume that $\texttt{Enc}(\cdot)$ is IND-CPA secure. The protocol in [34] assumes that the hardware manufacturer implements a security module on the FPGA. This security module includes a PUF and an AES decryption module, which allows to decrypt encrypted configuration files and/or other software IP blocks. However, in [34] there is no discussion about fuzzy extractors, which are required to deal with noise and extract randomness from a PUF. The protocol assumes secure and authenticated channels between all parties involved in the protocol during the enrollment and online phases. During the offline phase an unauthenticated public channel is assumed. Notice that the public channel allows the TTP to have access to *SW* since it is only encrypted with a PUF response, which is stored in the TTP database. We ask the following questions:

1. Can we simplify the protocol of [34] and still attain the same security guarantees? In particular, the protocol of [34] does not take advantage of the assumptions made on the primitives, which leads to unnecessarily complicated protocols. For example, is it possible to come up with a similar pro-

[3] Reference [34] uses a public hash function for integrity.

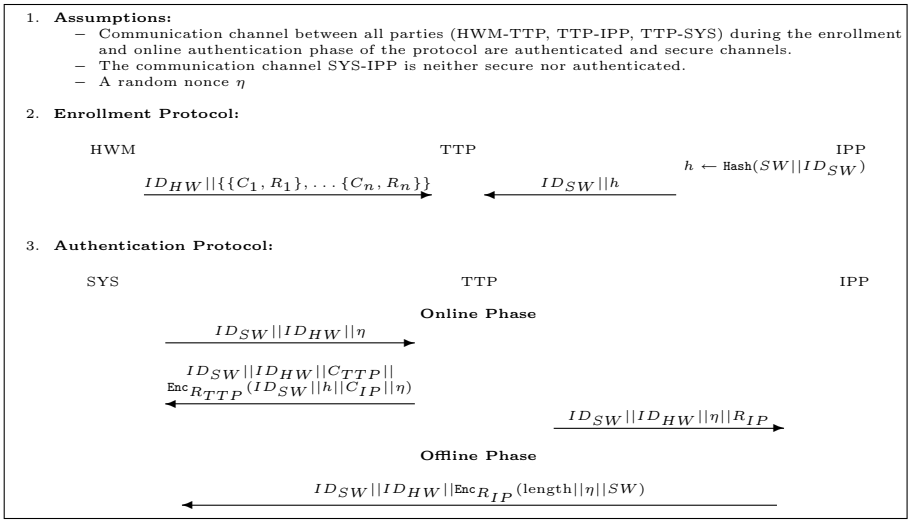

Fig. 1. Offline HW/SW authentication for FPGAs according to [34]

tocol, which does not require secure channels during the online phase of the protocol?

2. Can we design a protocol with similar security guarantees and which does not allow the TTP to know the software SW? In other words, can we provide complete privacy of the SW (even the TTP has no access to SW)? Notice that the protocol in [34] does not provide this type of privacy guarantee since the TTP knows R_{IP} and the SYS-IPP channel is public.
3. Is a protocol with four messages required or can we simplify it? In other words, can we reduce the communication complexity of the protocol in [34].
4. In Sect. 2.3 we saw how in general $\mathtt{Enc}_K(M||\tau)$, where $\tau = h(M)$ and h a public function, does not provide integrity. Similarly, Encrypt-and-MAC provides integrity but violates privacy. As a result, [34] provide the following construction $\mathtt{Enc}_{K_1}(h(M))||\mathtt{Enc}_{K_2}(M)$. This requires two decryptions and one hash computation. Is it possible to simplify the protocol, so that only *one* encryption and one MAC are required ?

In the next section, we provide answers to these questions in a constructive manner. In particular, we design simplified protocols which (in some cases) do not allow the TTP to learn any information about the IP block. In addition, our protocols require only one encryption and one MAC as opposed to two encryptions and one MAC (hash) operation as in [34].

4 New HW/SW Authentication Protocols for FPGAs

In this section, we introduce two new protocols and analyze them. First, we propose a protocol that provides partial privacy (only the TTP is able to learn

the IP block) and integrity. Then, we introduce a protocol which provides total privacy, in the sense that not even the TTP has access to the IP block originating from the IP provider. Notice that in our protocols we write C_i to denote the PUF challenge *and* the corresponding helper data required to reconstruct the PUF response R_i from a noisy version R_i'. Finally, we assume, as implicitly done in [34], that the circuit used to obtain CRPs during the enrollment protocol is destroyed (e.g. by blowing fuses) after enrollment and that subsequently, given a challenge C_i the corresponding response R_i' is *only* available internally to the decryption circuit in the FPGA. Without, this assumption, anyone could access R_i, and the protocols proposed (including those in [34]) would be completely broken. We begin by describing how the combination of bitstream encryption and a key extracted from a PUF works in practice. It consists of the following steps: (i) loading the encrypted bitstream, (ii) challenging the PUF with a challenge C_i, (iii) measuring the PUF response R_i', (iv) retrieving helper data W_1, W_2 from memory, (v) using a fuzzy extractor to extract the key $K \leftarrow \mathtt{Rep}(R_i', W_1, W_2)$, (vi) decrypting the bitstream, and finally (vii) configuring the FPGA.

New IP Protection Protocols. For the sake of simplicity we assume that the length information is already contained[4] in the IP block denoted by SW.

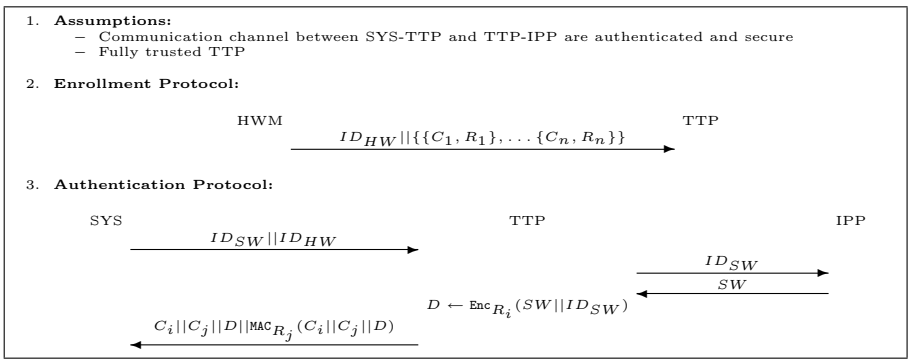

Fig. 2. New IP Protection Authentication Protocol

ANALYSIS. Notice that the TTP is fully trusted in this model. Thus, it is allowed for the TTP to have access to the SW. Confidentiality of the SW follows immediately from the assumptions on the PUF. Authentication during the running of the protocol follows from the fact that we have an authenticated channel between TTP and SYS. However, after running of the protocol, $C_i||C_j||D||\mathtt{MAC}_{R_j}(C_i||C_j||D)$, where $D = \mathtt{Enc}_{R_i}(SW||ID_{SW})$ are stored in insecure non-volatile memory. In this case, privacy follows from the inability of an attacker to generate R_i corresponding to the challenge C_i and integrity of SW from $\mathtt{MAC}_{R_j}(C_i||C_j||\mathtt{Enc}_{R_i}(SW||ID_{SW}))$ and the inability of the attacker to gen-

[4] This is also a realistic assumption as bit stream configuration files for current FPGAs already have length information embedded in them.

erate R_j from C_j. This protocol has the drawback that all communications go through the TTP. In particular, every SYS has to contact the TTP to obtain the desired IP block, which could prove to be a system bottleneck. One can solve this by simply having the TTP forward pairs $\{C_i, R_i\}, \{C_j, R_j\}$ to IPP and having IPP, in turn, send $C_i||C_j||D||\texttt{MAC}_{R_j}(C_i||C_j||D)$, where $D = \texttt{Enc}_{R_i}(SW||ID_{SW})$ directly to the SYS. In this case, we do not assume an authenticated or secure channel between the IPP-SYS. The privacy of the SW follows simply from having SW encrypted with R_i and integrity from checking $\texttt{MAC}_{R_j}(C_i||C_j||D)$. Notice that the pairs $\{C_i, R_i\}, \{C_j, R_j\}$ are only available to the TTP and to authentic IPPs in touch with the TTP, by assumption.

New IP Protection Protocols Providing SYS-IPP Confidentiality. In this section, we answer positively the question of whether it is possible to develop protocols with similar properties to the previous ones but without having the TTP have access to the SW. In the following, we do not assume any of the channels to be secure. However, we make the following assumptions: (1) the channels TTP-SYS, TTP-IPP, SYS-IPP are authentic (e.g. man-in-the-middle attacks are not possible), (2) it is possible to obtain the public-key of IPP (in an authenticated way) and use it for sending encrypted data to it, and (3) the TTP is "honest-but-curious". In other words, the TTP follows the protocol in an honest manner but tries to find out as much information as possible (i.e. he wants access to SW). The resulting protocol is shown in Fig. 3.

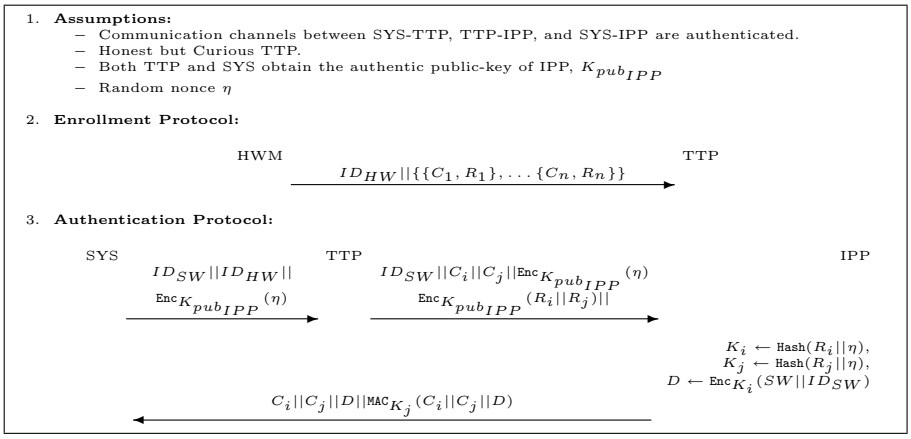

Fig. 3. IP Protection Authentication Protocol with SYS-IPP Confidentiality

ANALYSIS. We assume that the SYS and TTP have obtained the IPP's authentic public key and that they have established authenticated channels (SYS-TTP, TTP-IPP, IPP-SYS). Privacy and authenticity of SW follows from the Encrypt-then-Authenticate scheme, the inability of an attacker to derive R_i, R_j corresponding to C_i, C_j, and the fact that the keys used to encrypt and authenticate depend on R_i, R_j and the nonce η which is only known to the SYS and IPP. No-

tice that the TTP is *not* allowed to tamper with $\texttt{Enc}_{K_{pub_{IPP}}}(\eta)$ (e.g. substitute it) since we are in the honest-but-curious setting. Thus, the protocol provides privacy with respect to the TTP as well. Notice that the cost of the protocol on the SYS side is now one decryption, one MAC, and two additional hash function computations. The hash function computations do not require additional hardware resources if performed via an AES-based hash as in [34].

5 FPGA Intrinsic PUFs

The key component of the previously discussed protocols is the existence of a PUF. Before introducing our new construction, we review previous PUF constructions. Pappu et al. [28, 29] introduced the idea of Physical One-Way Function (POWF). They use a bubble-filled transparent epoxy wafer and shine a laser beam through it (at precise angles defined by the challenge) leading to a response interference pattern. However, this kind of analog PUF is hard to use in the field because of the difficulty to have a tamper resistant measuring device. Gassend et al. [13] define a Controlled Physical Random Function (CPUF) which is a PUF that can only be accessed via an algorithm that is physically bound to the PUF in an inseparable way. This control algorithm can be used to measure the PUF but also to protect a "weak" PUF from external attacks by making sure that any tampering with the control logic also destroys the PUF. Based on this idea, Gassend et al. introduce silicon Physical Random Functions (SPUF) [14] which use manufacturing process variations in integrated circuits (ICs) with identical masks to uniquely characterize each IC. The statistical delay variations of transistors and wires in the IC were used to create a parameterized self oscillating circuit to measure frequency which characterizes each IC. However, silicon PUFs are very sensitive to environmental variations like temperature and voltage. Therefore Lim et al. [25] introduce the concept of *arbiter based* PUF which uses a differential structure - two identical delay paths - and an arbiter to distinguish the difference in the delay between the paths. In [38], Tuyls et al. present a coating PUF in which an IC is covered with a protective matrix coating, doped with random dielectric particles at random locations. The IC also has a top metal layer with an array of sensors to measure the local capacitance of the coating matrix that is used to characterize the IC. The measurement circuit is integrated in the IC, making it a controlled PUF. Su et al. present in [37] a custom built circuit array of cross-coupled NOR gate latches to uniquely identify an IC. Here, small transistor threshold voltage V_t differences that are caused due to process variations lead to a mismatch in the latch to store a 1 or a 0. The disadvantage of most of these approaches is the use of custom built circuits or the modification of the IC manufacturing process to generate a reliable PUF. We approach the problem by identifying an *Intrinsic* PUF which we define as a PUF generating circuit already present in the device and that requires no modification to satisfy the security goals. We show that SRAM memories, which are widely available in almost every computing device including modern FPGAs, can be used as an Intrinsic PUF.

5.1 PUFs Based on SRAM Memories

A CMOS SRAM cell is a six transistor (6T) device [2] as shown in Fig. 4 formed of two cross-coupled inverters (load transistors PL, PR, NL and NR) and two access transistors (AXL and AXR) connecting to the data bit-lines (BLC and BL) based on the word-line signal (WL). Previous research on process variations in SRAM has been aimed at increasing the static-noise margin (SNM), defined as the minimum DC noise voltage to flip the cell state. SNM is one of the major concerns in SRAM design to guarantee the stability of the SRAM under intrinsic parameter fluctuations. In [7], the authors show that microscopic variations in the dopant atoms in the channel region of the MOSFET induce differences in the threshold voltage V_t of the transistors of an SRAM cell. The transistors forming the cross-coupled inverters (PR,PL, NR and NL) are constructed particularly weak to allow driving them easily to 0 or 1 during a write process. Hence, these transistors are extremely vulnerable to atomic level intrinsic fluctuations which are outside the control of the manufacturing process and independent of the transistor location on the chip. In [10], the authors also discuss other device characteristic variations caused by intrinsic parameter fluctuations in a CMOS SRAM cell. In practice, SRAM cells are constructed with proper width/length ratios between the different transistors [32] such that these fluctuations do not affect the reading and writing process under normal operation. However, during power-up, the cross-coupled inverters of a SRAM cell are not subject to any externally exerted signal. Therefore, any minor voltage difference that shows up on the transistors due to intrinsic parameter variations will tend toward a 0 or a 1 caused by the amplifying effect of each inverter acting on the output of the other inverter. Hence with high probability an SRAM cell will start in the same state upon power-up. On the other hand (as shown next), different SRAM cells will behave randomly and independently from each other. We consider as a challenge a range of memory locations within a SRAM memory block. For example, we show in Sect. 6 that to derive a 128-bit secret we require about 4600 SRAM memory bits (under extreme conditions). The response are the start-up values at these locations. If the memory block used is about 512 kbits, we can expect to have close to 110 CRPs. As previously discussed, we assume a security module that allows reading of the SRAM start-up values only by the manufacturer during the enrollment process. Upon successful enrollment a fuse is blown such that the response to a challenge is only available internally inside the FPGA. Notice also that SRAM-based PUFs produce a binary string as result of a measurement, in contrast, to other PUFs reported in the literature, which have to go through a quantization process before obtaining a bit string from the measurement. This results in a reduction in the complexity of the measurement circuit.

FPGA SRAM PUF. Most of the advanced FPGA that are in use today belong to the category of volatile SRAM FPGAs. The biggest manufacturers of these FPGAs, Altera and Xilinx, also provide extra built-in SRAM memory blocks that can be used by the designer to store data. For our proof of concept, we use such an FPGA with dedicated RAM blocks.

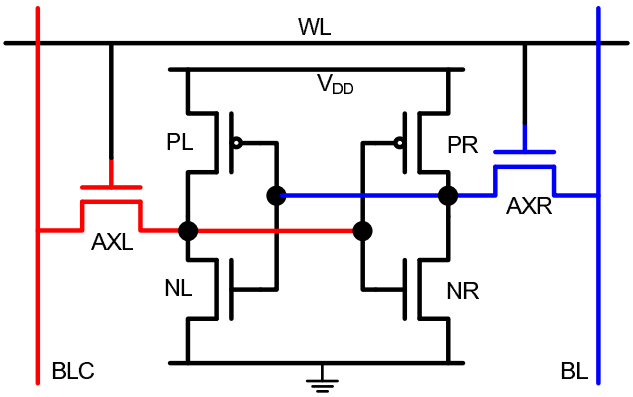

Fig. 4. Six transistor SRAM cell

5.2 Statistical Analysis of SRAM PUFs

In order to be useful as a PUF, SRAM startup values should have good statistical properties with respect to robustness over time, robustness to temperature variations, aging robustness, and identification performance. These criteria are described in the remainder of this section.

Robustness over Time. The Hamming distance between bit strings from repeated measurements of the same SRAM block (intra-class measurements) should be small enough, such that errors between enrollment and authentication measurements can be corrected by an error correcting code admitting efficient decoding. The main criteria here is to check the stability of the startup values over a series of intra-class measurements done over a two week period. Figure 5 shows the fractional Hamming distance between a first measurement and repeated measurements of the same SRAM block that were carried over approximately two days. The experiment was done with four different RAM blocks, located in two different FPGAs. The measurements show that less than 4% of the startup bit values change over time.

Robustness to Temperature Variations. The Hamming distance between bit strings measured in the same SRAM block (intra-class) at different environmental temperatures should be small (for the same reason as mentioned above). Stability tests of SRAM startup values at different temperatures are currently being performed. Preliminary data indicates that measurements at temperatures ranging from $-20°$C to $80°$C result in bit strings with maximum fractional Hamming distances of 12% when compared to a reference measurement performed at $20°$C.

Aging Robustness. Intra-class Hamming distances of the SRAM startup values should remain small, even when other data has been written into the memory before the FPGA was restarted. In particular, it is important that the startup values are unaffected by aging and the use of the SRAM blocks to store data. SRAM memory retention has been previously considered in [16, 17, 36] from a

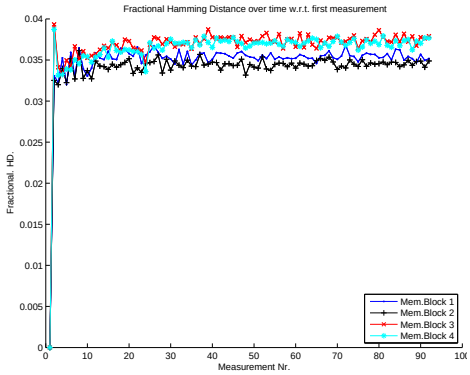

Fig. 5. SRAM startup values time test.

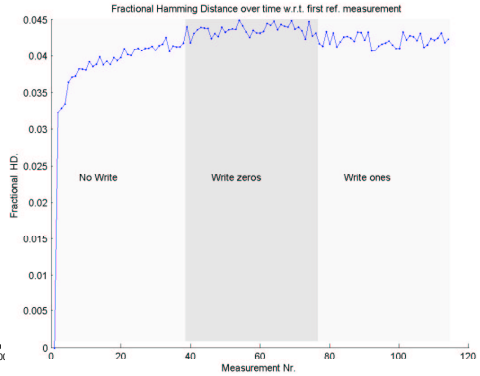

Fig. 6. SRAM startup values aging test.

security point of view. Gutmann [16, 17] writes that SRAM memories can retain some data that has been previously stored and that this phenomenon can also affect the startup values. How long the data is retained varies with temperature. Skorobogatov in [36] presents experimental evidence that show that retained data in SRAM memory is rapidly lost in a small amount of time (few msec) after startup. We have performed measurements to test the behavior of SRAM startup values after "normal memory usage". We simulated this usage by writing zeros or ones into the memory and maintaining this memory state for over 10 minutes at a time. The SRAM startup values were then read out after restarting the FPGA. Figure 6 shows the fractional Hamming distance between the bit string of a reference measurement and bit strings of startup values measured shortly after writing zeros and ones into the SRAM memory. The figure shows that storing zeros or ones into the memory has very little influence in the SRAM start-up values. The fractional Hamming distance between bit strings from an enrollment (reference) measurement and any of the other measurements does not exceed 4.5% in this test.

Identification Performance. The fractional Hamming distance between bit strings of different SRAM blocks (and different FPGAs) should be close to 50%, such that each SRAM block (and thus each FGPA) can be uniquely identified. In order to get an idea of how well the start-up bit strings from different memory blocks can be distinguished from each other, we have investigated the distribution of Hamming distances between bit strings of length 8190 bytes derived from different SRAM blocks (inter-class distribution). A histogram of inter-class Hamming distances is depicted in Fig. 7. The startup bit values of seventeen different SRAM blocks were used to create this graph. Our analysis shows that the inter-class fractional Hamming distance distribution closely matches a normal distribution with mean 49.97% and a standard deviation of 0.3%. Figure 7 also shows the histogram of intra-class Hamming distance measurements. This histogram was created by comparing 92 repeated measurements of the same SRAM

block. The intra-class fractional Hamming distance distribution of startup bit strings has an average of 3.57% and a standard deviation of 0.13%.

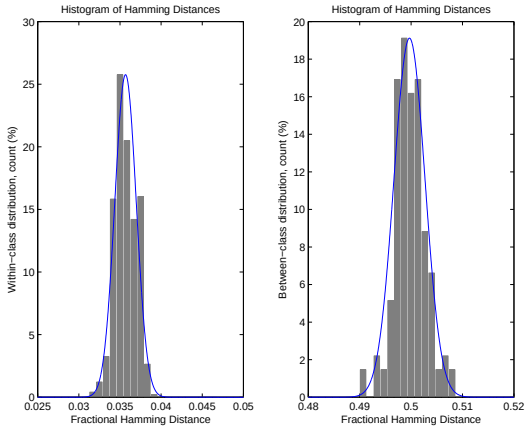

Fig. 7. Histogram of intra-class (left) and inter-class (right) Hamming distances between startup bit strings of SRAM blocks and their approximating normal distributions.

6 On the Cost of Extracting a 128-bit Key

It is well known that due to the noisy nature of PUFs a fuzzy extractor is required. A fuzzy extractor, as explained in Sect. 2.2, provides error correction capabilities to take care of the noisy measurements and privacy amplification to guarantee the uniform distribution of the final secret. We refer to Sect. 2.2 for the details but, in general, we will need to choose an error correcting code which accepts efficient decoding, implement its decoding algorithm on the FPGA, and implement a universal hash function, chosen at random from a set $\mathcal{H}$ during enrollment. Notice that only the Rep procedure must be implemented on the FPGA since the generation of the helper data is performed during enrollment. The next subsection describes the choices that can be made to derive a 128-bit key, which could be used in combination with an IND-CPA encryption scheme and corresponding MAC in the protocols proposed in Sect. 4.

Secrecy Rate. The fuzzy extractor derives a key K from the SRAM startup bits R by compressing these bits with a hash function h_i. The minimal amount of compression that needs to be applied by the hash function is expressed in the secrecy rate S_R, see [18]. The maximum achievable secrecy rate S_R is given by the mutual information between bit strings derived during enrollment and reconstruction, written $\mathbf{I}(R, R')$. In [18], a method was presented for estimating

this secrecy rate using a universal source coding algorithm called the Context-Tree Weighting Method [40]. We have applied this method to the SRAM startup values. By estimating the mutual information $\mathbf{I}(R, R')$ between repeated measurements of the same memory block, we find an average secrecy rate of 0.76 bits per SRAM memory bit. That means that to derive a secret of size N, we need at least $\lceil 1.32N \rceil$ source bits.

Error Correction. In order to choose an adequate error correcting code, we first consider the number of bits of information, which have to be at least $\lceil 1.32N \rceil$ bits, which for $N = 128$ is 171. Assuming that all bits are independent, the probability that a string of S bits will have more than t errors, denoted by P_{total}, is given by $\sum_{i=t+1}^{S} \binom{S}{i} p_b^i (1-p_b)^{S-i} = 1 - \sum_{i=0}^{t} \binom{S}{i} p_b^i (1-p_b)^{S-i}$, where p_b denotes the bit error probability. Notice that the maximum number of errors that we have experimentally seen is about 12%. Thus, assume that we have a bit error probability $p_b = 0.15$, to be conservative and that we are willing to accept a failure rate of $P_{total} = 10^{-6}$. Since, we are assuming that the errors are independent, a binary BCH code is a good candidate (see for example [8, 30]) with N-bit code words and a minimum distance at least $d = 2t + 1$, t the number of errors that $\mathcal{C}$ can correct. Since we need to generate in the end at least 171-bits of information, it becomes an optimization problem to choose the best code in terms of hardware resources, number of SRAM bits required, performance, etc. For example, using $[511, 19, t = 119]$-BCH, we would need $9 \times 511 = 4599$ bits to generate 171 information bits. On the other hand, if we assume $p_b = 0.06$ (i.e. assume that we only need to operate at $20°C$), then we could use the binary $[1023, 278, t = 102]$-BCH code, which requires only 1023 bits of SRAM memory to generate 278 bits of information.

Privacy Amplification. A universal hash function, introduced by Carter and Wegman in [9], is a map from a finite set A of size a to a finite set B of size b. For a given hash function h and two strings x, x' with $x \neq x'$, we define the function $\delta_h(x, x')$ as equal to 1 if $h(x) = h(x')$ and 0 otherwise. For a finite set (or family) of hash functions $\mathcal{H}$, $\delta_{\mathcal{H}}(x, x')$ is defined to be $\sum_{h \in \mathcal{H}} \delta_h(x, x')$. In other words, $\delta_{\mathcal{H}}(x, x')$ counts the number of functions $h \in \mathcal{H}$ for which x and x' collide. For a random $h \in \mathcal{H}$ and any two distinct x, x', the probability that $h(x) = h(x')$ is $\delta_{\mathcal{H}}(x, x')/|\mathcal{H}|$, where $|\mathcal{H}|$ denotes the size of the set $\mathcal{H}$. There has been extensive research on universal hash functions (see for example [33, 27]). However, their suitability for hardware implementations has not been thoroughly investigated. To our knowledge, the work of [22] and the recent work of Kaps et al. [20] are the only ones that consider their hardware implementation. However, no one seems to have considered their implementation on FPGAs. Thus, we will consider what the best architecture for FPGAs is in future work.

7 Conclusions

In this paper, we have proposed new and efficient protocols for the IP-protection problem. In addition, we have introduced a new PUF construction which is unique in the sense that it is intrinsic to FPGAs and thus, it does not require

modification of the hardware or the manufacturing process to be used. We have tested this construction on FPGAs with embedded block RAM memories which are not reset at power-up. We have seen similar phenomena in ASICs and expect similar behavior on any other device which contains uninitialized SRAM memory. At present, we have identified other properties of SRAM memory, which have the potential to be used as a PUF-source. This will be investigated in future work. We will also explore in the future the exact complexity of implementing a fuzzy extractor on an FPGA. Finally, we notice that the unique identifiers derived from the PUFs could be useful for tracking purposes.

References

1. J. H. An and M. Bellare. Does Encryption with Redundancy Provide Authenticity? In B. Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 512–528. Springer, May 6-10, 2001.
2. A. Bellaouar and M. I. Elmasry. *Low-Power Digital VLSI Design. Circuits and Systems*. Kluwer Academic Publishers, first edition, 1995.
3. M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. In N. Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *LNCS*, pages 1–15. Springer, August 18-22, 1996.
4. M. Bellare, J. Kilian, and P. Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.
5. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In T. Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, December 3-7, 2000.
6. M. Bellare and P. Rogaway. Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In T. Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 317–330. Springer, December 3-7, 2000.
7. A. J. Bhavnagarwala, X. Tang, and J. D. Meindl. The Impact of Intrinsic Device Fluctuations on CMOS SRAM Cell Stability. *IEEE Journal of Solid-State Circuits*, 36(4):658–665, April 2001.
8. R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley Publishing Company, first edition, 1985.
9. L. Carter and M. N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.
10. B. Cheng, S. Roy, and A. Asenov. The impact of random doping effects on CMOS SRAM cell. In *European Solid State Circuits Conference*, pages 219–222, Washington, DC, USA, 2004. IEEE Computer Society.
11. Y. Dodis, M. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology —- EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer-Verlag, 2004.
12. D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography (Extended Abstract). In *ACM Symposium on Theory of Computing — STOC'91*, pages 542–552. ACM, May 6-8, 1991.

13. B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Controlled Physical Random Functions. In *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*, page 149, Washington, DC, USA, 2002. IEEE Computer Society.
14. B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas. Silicon physical unknown functions. In V. Atluri, editor, *ACM Conference on Computer and Communications Security — CCS 2002*, pages 148–160. ACM, November 2002.
15. S. Goldwasser and S. Micali. Probabilistic Encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
16. P. Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Sixth USENIX Workshop on Smartcard Technology Proceedings*, pages 77–89, San Jose, California, July 1996. Available at `http://www.cs.cornell.edu/people/clarkson/secdg/papers.sp06/secure_deletion.pdf`.
17. P. Gutmann. Data remanence in semiconductor devices. In *10th USENIX Security Symposium*, pages 39–54, August 2001. Available at `http://www.cryptoapps.com/~peter/usenix01.pdf`.
18. T. Ignatenko, G.J. Schrijen, B. Skoric, P. Tuyls, and F. Willems. Estimating the Secrecy-Rate of Physical Unclonable Functions with the Context-Tree Weighting Method. In *IEEE International Symposium on Information Theory*, pages 499–503, Seattle, USA, July 2006.
19. C. S. Jutla. Encryption Modes with Almost Free Message Integrity. In B. Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 529–544. Springer, May 6-10, 2001.
20. J.-P. Kaps, K. Y., and B. Sunar. Energy Scalable Universal Hashing. *IEEE Trans. Computers*, 54(12):1484–1495, 2005.
21. T. Kean. Cryptographic rights management of FPGA intellectual property cores. In *ACM/SIGDA tenth international symposium on Field-programmable gate arrays — FPGA 2002*, pages 113–118, 2002.
22. H. Krawczyk. LFSR-based Hashing and Authentication. In Y. Desmedt, editor, *Advances in Cryptology - CRYPTO '94*, volume 839 of *LNCS*, pages 129–139. Springer, August 21-25, 1994.
23. H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?). In J. Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *LNCS*, pages 310–331. Springer, August 19-23, 2001.
24. H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Internet RFC 2104, February 1997. Available at `http://www-cse.ucsd.edu/~mihir/papers/rfc2104.txt`.
25. D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205, October 2005.
26. J.-P. M. G. Linnartz and P. Tuyls. New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates. In J. Kittler and M. S. Nixon, editors, *Audio-and Video-Based Biometrie Person Authentication — AVBPA 2003*, volume 2688 of *LNCS*, pages 393–402. Springer, June 9-11, 2003.
27. W. Nevelsteen and B. Preneel. Software Performance of Universal Hash Functions. In J. Stern, editor, *Advances in Cryptology — EUROCRYPT'99*, volume 1592 of *LNCS*, pages 24–41. Springer, May 2-6, 1999.
28. R. S. Pappu. *Physical one-way functions.* PhD thesis, Massachusetts Institute of Technology, March 2001. Available at `http://pubs.media.mit.edu/pubs/papers/01.03.pappuphd.powf.pdf`.

29. R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297(6):2026–2030, 2002. Available at `http://web.media.mit.edu/~brecht/papers/02.PapEA.powf.pdf`.
30. W. W. Peterson and E. J. Weldon, Jr. *Error-Correcting Codes*. The MIT Press, second edition, 1972.
31. P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.
32. E. Seevinck, F. J. List, and J. Lohstroh. Static-Noise Margin Analysis of MOS SRAM Cells. *IEEE Journal of Solid-State Circuits*, 22(5):748–754, Oct 1987.
33. V. Shoup. On Fast and Provably Secure Message Authentication Based on Universal Hashing. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96*, volume 1109 of *LNCS*, pages 313–328. Springer, August 18-22, 1996.
34. E. Simpson and P. Schaumont. Offline Hardware/Software Authentication for Reconfigurable Platforms. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems — CHES 2006*, volume 4249 of *LNCS*, pages 311–323. Springer, October 10-13, 2006.
35. B. Skoric, P. Tuyls, and W. Ophey. Robust Key Extraction from Physical Uncloneable Functions. In J. Ioannidis, A. D. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security — ACNS 2005*, volume 3531 of *LNCS*, pages 407–422, June 7-10, 2005.
36. S. P. Skorobogatov. Low temperature data remanence in static RAM. Technical Report 536, University of Cambridge, Computer Laboratory, June 2002.
37. Y. Su, J. Holleman, and B. Otis. A 1.6pJ/bit 96% Stable Chip-ID Generating Cicuit using Process Variations. In *ISSCC '07: IEEE International Solid-State Circuits Conference*, pages 406–408, Washington, DC, USA, 2007. IEEE Computer Society.
38. P. Tuyls, G.-J. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters. Read-Proof Hardware from Protective Coatings. In *Cryptographic Hardware and Embedded Systems — CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 369–383. Springer, October 10-13, 2006.
39. D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). NIST Proposed Mode of Operation, June 2002. Available at `http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/ccm/ccm.pdf`.
40. F. Willems, Y.M. Shtarkov, and Tj. J. Tjalkens. The Context-Tree Weighting method: Basic Properties. *IEEE Trans. Inform. Theory*, IT-41:653–664, May 1995.