Intrinsic Physical Unclonable Functions in Field Programmable Gate Arrays[1]

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

Information and System Security Group
Philips Research Europe, Eindhoven, Netherland
Email: {jorge.guajardo | sandeep.kumar | klaus.kursawe | geert.jan.schrijen | pim.tuyls}@philips.com

Abstract

In today's globalized economy, it has become standard business practice to include third party Intellectual Property (IP) into products. However, licensing IP to third parties forces IP vendors to ensure that they can generate revenue from their internally developed IP blocks. This is only guaranteed if designs are properly protected against theft, cloning, and grey market overproduction. In this paper, we describe a solution for the IP protection problem on Field Programmable Gate Arrays (FPGAs) based on the use of Physical Unclonable Functions (PUFs). Our solution includes optimizations at the protocol level, making the resulting protocols more efficient than previously proposed ones. In addition, we show how SRAM memory blocks present in current FPGAs can be used as a PUF. This leads to a solution which allows unique identification of FPGAs without requiring significant additional hardware resources, and to ensure code can only run on authorized platforms.

1 Introduction

Begining in the 1980s, there has been a continuos move towards outsourcing of non-key activities within companies. This is particularly true in the semiconductor industry where the costs of building a state-of-the-art high-volume CMOS digital logic fab range anywhere from 1 to 2 billion US dollars [Brun99], thus making it not cost-effective for single companies to build such a fab. This has led to the emergence of foundries, which can invest large amounts of capital and service several customers at one time. Parallel to this trend, IP developers have also recognized that IP developed in-house can be a source of additional revenue if licensed to external parties. The previous developments have led many companies to disclose internally developed IP to external parties and as a consequence to face the counterfeiting challenge. It is estimated that as much as 10% of all high tech products sold globally are counterfeit [KPMG05]. This translates into a conservative estimate of US$100 billion of global IT industry revenue lost due to counterfeiting [KPMG05]. The same paper advises to employ anti-counterfeiting technologies to mitigate the effects of counterfeiters. In this paper, we deal explicitly with one such technology and its implementation on Field Programmable Gate Arrays (FPGAs).

[1] Appears in Information Security Solutions Europe – ISSE 2007, September 25-27, 2007, Warsaw, Poland.

FPGAs are devices containing programmable logic blocks and programmable interconnect. The programmable blocks give an FPGA the ability to instantiate virtually any logic function whereas the programmable interconnect allows connecting different logic blocks in the FPGA. Since the mid 90s, FPGAs have steadily increased their logic resources. This has lead to an increasing number of implementations and products using FPGAs. Most FPGAs in use today are volatile SRAM-based devices. This means that upon power-up, a configuration file or bitstream, stored in external non-volatile memory, needs to be loaded onto the FPGA providing it with the desired functionality. This flexibility is also the reason why FPGA designs can be easily copied. An attacker can easily tap the bus between non-volatile memory and FPGA on an authentic board, obtain the configuration file, copy it onto a different board, and obtain exactly the same functionality as in the original board. Such an attack is called a *cloning* attack and it results in counterfeited products.

From a security perspective, the counterfeiting threat can be best explained as an authentication problem. In general, we can identify the following security services required by different parties in the overall IP protection chain:

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 execute.

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

Using bitstream encryption with a key that is specific to a particular FPGA would provide the means to solve most of the problems. This observation is due to Kean [Kean02], 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 or (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.

In this paper, we will focus on providing services S1, S2 and S3. In particular, we propose new and improved protocols for IP protection on FPGAs. We show that the protocols of [SiSc06] (which deals with S1 and S2), can be considerably simplified. We describe simplifications in terms of communication complexity, assumptions, and number of encryptions performed. In addition, our protocols 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. The cost of TTP-privacy is a public-key (PK) based operation. The public-key operation does not affect the resource requirements of the FPGA implementation when compared to the work in [SiSc06]. This is achieved by performing the PK operation during the *online* phase of the protocol. Finally, we describe 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. The actual implementation and analysis of this FPGA intrinsic PUF is described in detail in [GKST07].

2 Physical Unclonable Functions, Fuzzy Extractors and Helper Data Algorithms

Physical Unclonable Functions (PUFs), also referred to as Physically Random Functions, consist of inherently unclonable physical systems. PUFs are functions embedded in a separate physical structure or extracted from physical properties inherently present in a hardware device (intrinsic PUFs) that map challenges to responses (identifiers). 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. In general, PUFs have three main properties:

- **Easy to evaluate**. It is easy and cheap to challenge the PUF and measure the response. This implies that the whole evaluation procedure can be carried out with minimal time delay and minimal cost.
- **Hard to characterize**. An attacker, who is in possession of the device containing the PUF can only obtain a negligible amount of knowledge about the PUF. Hence, he is unable to manufacture a device with similar properties. In other words, a realistic attacker (one who does not have infinite resources) cannot clone the PUF.
- **Tamper evidence**. Under a physical attack the PUF gets damaged to such an extent that its challenge-response behaviour changes substantially and the extracted identifiers are destroyed. In addition to the properties previously described, PUFs can be inseparably bound to the device. This means that any attempt to remove the PUF from the device leads to the destruction of the PUF (and/or the device). Hence, an attack aiming at removing the PUF will destroy the data being extracted from it.

PUFS can be used to generate unclonable identifiers and to generate and to securely store cryptographic keys in a device. 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. Extracting the data from the PUF is only possible by reading it in the "proper" way, i.e., 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 challenge-response pairs such that an attack (performed during a limited amount of time) based on exhaustively measuring all responses only has a negligible probability of success. 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; in the extreme case, a weak PUF may only have one challenge. Due to noise in the measurement process, the PUF responses may contain some errors, which need to be compensated and corrected. As PUF responses are noisy (as explained above) and may show statistical correlations, a Fuzzy Extractor or Helper Data Algorithm [LiTu03, DoRS04] is needed to extract usable data from the PUF responses. 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 is generated during the *enrollment phase*, which happens once in the lifetime of the device in a trusted environment. Later, during the *reconstruction phase*, the data is reconstructed based on a noisy measurement and the helper data.

3 PUF Constructions

This section describes some known PUF constructions including: optical PUFs [PRTG02], silicon PUFs [GCDD02a] and coating PUFs [TSS+06]. Although coating PUFs are very cheap to produce they still need a small additional manufacturing step. For the FPGA protection, we use an intrinsic PUF (IPUF) [GKST07], i.e., a PUF that is inherently present in the device due to its manufacturing process and no additional hardware has to be added for embedding the PUF.

3.1 Optical PUFs and Silicon PUFs.

Pappu et al. [PRTG02] introduce the idea of a Physical One-Way Function. They use a bubble-filled transparent epoxy wafer and shine a laser beam through it leading to a response interference pattern. This kind of optical PUF is hard to use in the field because of the difficulty to have a tamper resistant measuring device. Gassend et al. introduce Silicon Physical Random Functions (SPUF) [GCDD02a] which use manufacturing process variations in ICs with identical masks to uniquely characterize each chip. 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. Silicon PUFs are very sensitive to

environmental variations like temperature and voltage. Lim et al. [LLG+05] introduce *arbiter-based* PUFs which use a differential structure and an arbiter to distinguish the difference in the delay between the paths. Gassend et al. [GCDD02b] also define a Controlled Physical Random Function (CPUF) which can only be accessed via an algorithm that is physically bound to the randomness source in an inseparable way. This control algorithm can be used to measure the PUF but also to protect a "weak" PUF from external attacks. Recently, Su et al. [SuHO07] present 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.

3.2 Coating PUFs

In [TSS+06], Tuyls et al. present coating PUFs 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. It is shown in [TSS+06] that it is possible to extract up to three key bits from each sensor in the IC leading to approximately 600 bits per mm^2. A key observation in [TSS+06] is that the coating can be used to store keys (rather than as a CRP repository as in previous works) and that these keys are not stored in memory. Rather, whenever an application requires the key, the key is generated on the fly. This makes it much more difficult for an attacker to compromise key material in security applications. Finally, Tuyls et al. [TSS+06] show that active attacks on the coating can be easily detected, thus, making it a good countermeasure against probing attacks.

3.3 FPGA Intrinsic PUFs and SRAM Memories

The disadvantage of most of the previous approaches is the use of custom built circuits or the modification of the IC manufacturing process to generate a reliable PUF. In [GKST07], the authors approach the problem by identifying an *Intrinsic* PUF which is defined as a PUF already present in the device and that requires no modification to satisfy the security goals. We describe next how SRAM memories, which are widely available in almost every computing device including modern FPGAs, can be used as an Intrinsic PUF.

3.3.1 Basic Principles of SRAM PUFs

A CMOS SRAM cell is a six transistor device formed of two cross-coupled inverters and two access transistors connecting to the data bit-lines based on the word-line signal. The transistors forming the cross-coupled inverters are constructed particularly weak to allow driving them easily to 0 or 1 during a write process. Hence, these transistors are extremely sensitive to atomic level intrinsic fluctuations which are outside the control of the manufacturing process and independent of the transistor location on the chip (see [ChRA04]). In practice, SRAM cells are constructed with proper width/length ratios between the different transistors 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, different

SRAM cells will behave randomly and independently from each other. In [GKST07], the authors consider as a challenge a range of memory locations within a SRAM memory block. The response are the start-up values at these locations. Notice also that SRAM-based PUFs produce a binary string as result of a measurement, in contrast to other PUFs, 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. For our proof of concept, we use FPGAs which include dedicated RAM blocks. In order to be useful as a PUF, SRAM startup values should have good statistical properties and be robust over time, to temperature variations, and have good identification performance. These properties were studied in [GKST07]. Here we summarize their findings. Regarding robustness, 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. In [GKST07], the authors compared the Hamming distance between a first measurement and repeated measurements of the same SRAM block carried over approximately two weeks. The experiment was done with four different RAM blocks, located in two different FPGAs. The measurements showed that less than 4% of the startup bit values change over time. Similarly, 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. Finally, we notice that 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. The tests in [GKST07] indicate 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. The fractional Hamming distance between bit strings of different SRAM blocks and different FPGAs should be close to 50%, such that each FGPA can be uniquely identified. Reference [GKST07] investigated the distribution of Hamming distances between 8190-byte long strings derived from different SRAM blocks (inter-class distribution). The 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%. The intra-class fractional Hamming distance distribution of startup bit strings has an average of 3.57% and a standard deviation of 0.13%.

3.3.2 On the Cost of Extracting a 128-bit Key

Due to the noisy nature of PUFs, a fuzzy extractor is required to provide error correction capabilities on the noisy measurements as well as privacy amplification to guarantee the uniform distribution of the final data. In general, we will need to choose an error correcting code, implement its decoding algorithm on the FPGA, and implement an appropriate hash function. In the following, we describe the choices that can be made to derive a 128-bit key, which can be used in combination with symmetric-key cryptography and the protocols proposed in Section 4.

The fuzzy extractor derives a key K from the SRAM startup bits by first correcting any errors present in the raw data stream coming from memory and then compressing and making the resulting string uniformly distributed with a universal hash function. The minimal amount of

compression that needs to be applied by the hash function is expressed in the *secrecy rate* [ISS+06]. In [ISS+06], a method was presented for estimating this secrecy rate using a universal source coding algorithm called the Context-Tree Weighting Method. We have applied this method to the SRAM startup values. In repeated measurements of the same memory block, we find a secrecy rate of 0.76 bits per SRAM memory bit. That means that to derive a secret of size N, we need at least 1.32 N source bits, i.e., a secure 128 bit key requires 171 source bits to be fully random. In our experiments, the maximum number of errors that we have seen is about 12%. Thus, assume conservatively that we have a bit error probability of 0.15 and that we are willing to accept a failure rate of 10^{-6}. Since we are assuming that the errors are independent, a binary BCH [PeWe72] is a good candidate with N-bit code words and a minimum distance at least d=2t+1. Since we need to generate in the end at least 171 information bits, 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*511 = 4599 bits to generate 171 information bits. On the other hand, if we assume the error probability to be= 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.

4 Offline HW/SW Authentication for FPGAs

In this section, we present two protocols to use the intrinsic PUF in an FPGA for IP protection. The first protocol assumes a trusted third party (TTP) that is allowed to see the IP block. 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.

As done in the protocol in [SiSc06], we assume 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 [SiSc06] 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.

Finally, we assume, as implicitly done in [SiSc06], that the circuit used to obtain challenge-response pairs 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 [SiSc06]) would be completely broken.

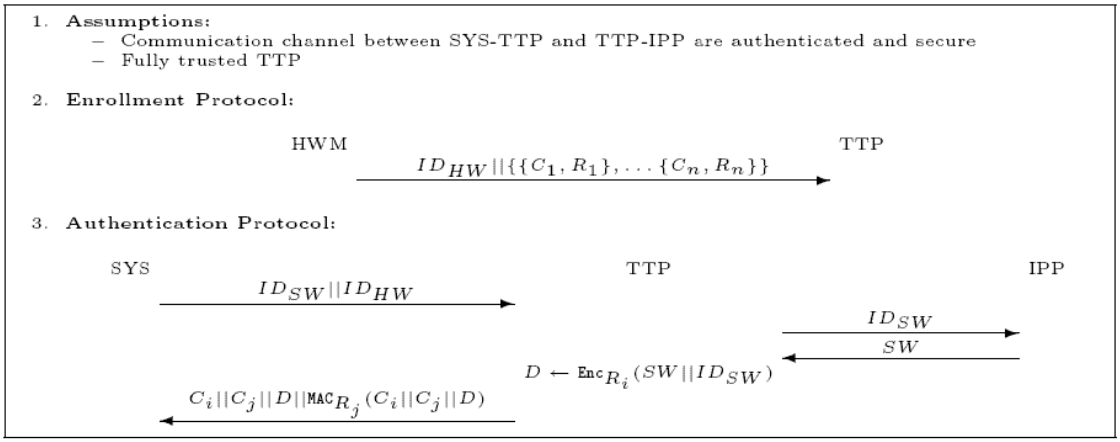
4.1 HW/SW Authentication Protocols for FPGAs

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'.

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 the corresponding helper data from memory, (v) using a fuzzy extractor to extract

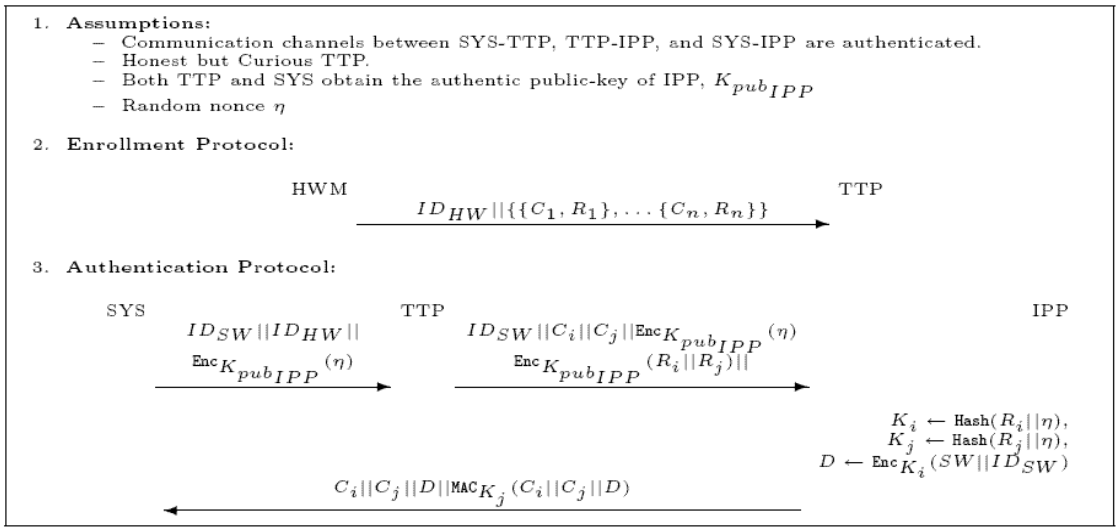
the key K from R_i' using the helper data (vi) decrypting the bitstream, and finally (vii) configuring the FPGA.

During Enrollment, the hardware manufacturer (HWM) measures the PUF with all relevant inputs, and sends those inputs and the corresponding responses to the trusted third party. The system developer (SYS) sends the identity of the software and the hardware platform to the Trusted Third party, which asks the intellectual property provider (IPP) for the actual software. The TTP encrypts the software with a key matching the PUF in the hardware and sends the encrypted version with the appropriate challenge to the PUF back to the system developer. In addition, the block is authenticated with a MAC using an independent challenge to the PUF.



4.2 IP Protection Protocols Providing Code 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 software we want to protect. 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 essential difference is that in this protocol, the intellectual property provider performs the encryption. By use of public key cryptography, we can assure that only the PUF knows the decryption key, even though the TTP still maintains the challenge response-lists.



5 Conclusion

In this paper, we describe efficient protocols for the IP-protection problem on FPGA code. In addition, we have also summarized existing PUF constructions. We pay particular attention to intrinsic PUFs as introduced in [GKST07]. This PUF construction 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

[Brun99] Richard Bruner, Catching the Outsourcing Wave - the boom in semiconductor foundries serving fabless semiconductor companies - Industry Trend or Event. Electronic News, April 12, 1999.

[ChRA04] B. Cheng, S. Roy, and A. Asenov, “The impact of random doping effects on CMOS SRAM cell,” in *European Solid State Circuits Conference*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 219–222.

[DoRS04] Y. Dodis, M. Reyzin, and A. Smith, “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” in *Advances in Cryptology —- EUROCRYPT 2004*, ser. LNCS, C. Cachin and J. Camenisch, Eds., vol. 3027. Springer-Verlag, 2004, pp. 523–540.

[GCDD02a] B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas, "Silicon physical unknown functions," in *ACM Conference on Computer and Communications Security — CCS 2002*, V. Atluri, Ed. ACM, November 2002, pp. 148–160.

[GCDD02b] 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*. Washington, DC, USA: IEEE Computer Society, 2002, p. 149.

[GKST07] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *Cryptographic Hardware and Embedded Systems — CHES 2007*, ser. LNCS. Springer, To appear 2007.

[ISS+06] 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*, pp. 499-503, Seattle, USA, July 2006.

[Kean02] T. Kean, "Cryptographic rights management of FPGA intellectual property cores," in *ACM/SIGDA tenth international symposium on Field-programmable gate arrays — FPGA 2002*, ACM, 2002, pp. 113–118.

[KPMG05] KPMG Electronics, Software & Services and Alliance for Gray Market and Counterfeit Abatement, "Managing the Risks of Counterfeiting in the Information Technology Industry, White Paper," 2005.

[LiTu03] J.-P. M. G. Linnartz and P. Tuyls, "New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates," in *Audio-and Video-Based Biometrie Person Authentication — AVBPA 2003*, ser. LNCS, J. Kittler and M. S. Nixon, Eds., vol. 2688. Springer, June 9-11, 2003, pp. 393–402.

[LLG+05] 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*, vol. 13, no. 10, pp. 1200–1205, October 2005.

[PeWe72] W. W. Peterson and E. J. Weldon, Jr. Error-Correcting Codes. The MIT Press, second edition, 1972.

[PRTG02] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 6, pp. 2026– 2030, 2002.

[SiSc06] E. Simpson and P. Schaumont, "Offline Hardware/Software Authentication for Reconfigurable Platforms," in *Cryptographic Hardware and Embedded Systems — CHES 2006*, ser. LNCS, L. Goubin and M. Matsui, Eds., vol. 4249. Springer, October 10-13, 2006, pp. 311–323.

[SuHO07] 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*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 406–408.

[TSS+06] 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, ser. LNCS, vol. 4249. Springer, October 10-13, 2006, pp. 369–383.