On the (re)design of an FPGA-based PUF

P. Grabher, D. Page and M. Wójcik
Department of Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road, Bristol, BS8 1UB, UK.
{*grabher,page,wojcik*}*@cs.bris.ac.uk*

***Abstract*—Physically Unclonable Functions (PUFs) represent a promising basis for solutions to problems such as secure key storage, and delivery of higher-level applications such as authentication. Although effective PUF designs exist for CMOS-based technologies (e.g., arbiter PUFs), their implementation on FPGAs remains a challenge (e.g., because of their routing characteristics). With this in mind, Anderson described a PUF design specifically tailored towards FPGAs. In this paper we identify and analyse a flaw in said design which renders it impractical for security-critical use. We describe two alternative solutions (relating to different trade-offs) that eliminate this flaw.**

***Keywords*-FPGA; PUF**

I. Introduction

A vast range of challenges and opportunities has emerged as a result of the (ongoing) proliferation of mobile and embedded computing. On one hand, an increase in computational and storage capability has enabled more complex application classes on which we now routinely depend. On the other hand, various supporting technologies and techniques (efficient implementation, for example) need to keep pace with such developments. In particular, the omnipresent specters of power consumption and security represent central problems. With respect to the latter, cryptography can help to ensure the secrecy and authenticity of data; in practice however, a range of modern attack techniques mean concrete guarantees are still difficult to achieve. Exemplars are captured by the field of physical security, including active fault injection and passive side-channel attacks [7].

Our focus in this paper is on secure cryptographic key storage and device authentication within such a context. In short, the issue is that modern cryptosystems are implemented according to Kerckhoffs's principle: security should rely on secrecy of key material, rather than of the algorithm design and/or implementation. For embedded computing in particular this can represent a problem, and development of robust solutions (e.g., which ensure the key is protected from manipulation or leakage via a physical attack) remains a non-trivial and ongoing issue.

Physically Unclonable Functions (PUFs) represent an emerging but promising solution for these problems (and many others). PUFs represent an example of the larger field of hardware intrinsic security. The underlying idea is that behaviour of such components should intrinsically depend on unique, physical features in a given instance. This can be explained via analogy. The form of a fingerprint, for example, depends on genetics plus unique, physical influence such as wear; likewise, the behavioural characteristics of an Integrated Circuit (IC) depend on a design plus unique, physical variation during fabrication. As a result, PUFs (potentially with suitable post-processing) can support the delivery of key material derived from the intrinsic properties of an instance, rather than being stored in a conventional sense.

A PUF design is genuinely intrinsic whenever relies only on components that already exist within the underlying platform. FPGAs are interesting as a result, since they offer a (more) diverse range of such components plus the usual benefit of flexibility that affords algorithm agility. Although other options exist [9], [10], [2], [5] Anderson [1] described a PUF design specifically tailored towards resources on the Xilinx Virtex-5 family of FPGA devices.

Section II presents an overview of PUFs, and the FPGA-based design of Anderson which forms the basis for our contribution. Section III identifies and then analyses a flaw in [1] which renders the associated design impractical for security-critical use. This flaw can be resolved in one of (at least) two ways, which we outline in Section III-C and Section III-D. We conclude in Section IV with some general discussion on FPGA-based PUF design, and highlight some areas for further work.

II. Background

A. PUFs

1) Terminology and notation: We refer the reader to [6, Chapter 1] for a comprehensive overview of PUFs from both a theoretical and practical perspective.

Basic concepts: As the name suggests, a PUF is a function in the sense it simply produces output given some input. However, since a PUF is better described as a physical influenced process (or algorithm) than a mathematical function, it is important to distinguish between the design (or specification) and a physical instance of it.

Let Π_i to denote the i-th distinct instance of a given PUF design. An instance can be sampled (cf. function evaluated) using an n-bit challenge c as input; the m-bit response (or signature) $r = \Pi_i(c)$ is produced as output. A set $\{(c_j, r_j = \Pi_i(c_j)) \mid 0 \leq j < l\}$ of the resulting Challenge-Response Pairs (CRPs) describes the instance behaviour (non-exhaustively, depending on l):

crucially, both the design *and* the process of instantiating it contribute to said behaviour.

Note that some care is required regarding terminology: a reconfigurable PUF (or rPUF) [4] is such that the behaviour of a specific instance can be (permanently) altered via some mechanism. With this in mind, we consider the combination of a specific FPGA device and a specific bit-stream as representing a distinct PUF instance[1]. If partial reconfiguration of the device is possible, this supports reconfigurable PUF designs via online alteration of a given instance.

Quality metrics: The unclonability of an instance is a result of the physical and inherently non-reproducible impact of instantiation, and (potential) lack of ability to inspect the resulting instance: invasive inspection of the instance may (by design, and usually permanently) alter the behaviour of or destroy the PUF.

Beyond this, the quality of a given PUF design is often evaluated in terms of inter- and intra-distance measures over example instances:

- For a challenge c and two PUF instances Π_1 and Π_2, inter-distance measures the difference between $r_1 = \Pi_1(c)$ and $r_2 = \Pi_2(c)$. Intuitively, one wants this difference to be large since this allows unique identification of instances.
- For a challenge c and a PUF instance Π_1, intra-distance measures the difference between $r_1 = \Pi_1(c)$ and $r_2 = \Pi_1(c)$. Intuitively, one wants this difference to be small since this allows reliable identification of instances.

The behaviour of a given instance may be influenced by the environment (e.g., temperature, voltage or ageing of the device). The associated impact is ideally minimised somehow (e.g., via compensation logic) to ensure robustness of the metrics above and help to prevent tampering-based attack techniques.

Generic design components: Per [3, Chapter 3], Figure 1 describes a complete PUF as three components. The sample step first takes a challenge and supplies it as input to the PUF, producing an output that is mapped onto a response; the latter steps may involve some form of error correction in order to cope with noise within samples.

The resulting component can be placed within various surrounding modes of operation. An important example are Controlled PUFs (CPUFs), whereby the PUF is combined with other cryptographic primitives. For example, a hash function could be used to post-process the response to provide appropriate statistical properties (such as uniformity in the distribution of responses); where the PUF response is used to derive key material, this addition is sometimes termed a key extractor.

B. FPGA-based PUFs designs

[1]This choice implies a change in either device or bit-stream (e.g., resynthesis, even if the design and hence HDL source code is the same), creates a new instance. However, any (re)configuration of the same device with the same bit-stream produces the same instance.

1) Supporting features and components: Roughly speaking, an FPGA fabric is a collection of logic resources (or blocks) organised in a two-dimensional mesh; the logic blocks are connected via configurable routing resources.

In Xilinx Virtex-5 devices, the central resource is a Configurable Logic Block (CLB) [11] which incorporates two slices. In turn, each slice contains four Look-Up Tables (LUTs), four flip-flops and a chain of four interconnected multiplexers (which permit fast carry propagation when arithmetic operations are implemented); the LUTs are SRAM-based, and can compute any 6-input, 1-output Boolean function once configured appropriately. In Virtex-5 devices, roughly a quarter of all slices can perform extra functionality and can be used for implementation of SRAM cells or shift registers; these are referred to as SLICEM, with normal slices referred to as SLICEL.

2) A concrete design: In this section, we introduce the PUF design due to Anderson [1] used as a basis for our contribution. From here on, we refer to said design as "the Anderson PUF" and use the short-hand 0 (resp. 1) to denote the value logic-0 (resp. logic-1) used in [1]. For brevity, we let C_i^s denote signal s related to the i-th instance of some component C (omitting either index where appropriate).

The Anderson PUF was designed specifically with FPGA fabrics in mind, and even more specifically for the Xilinx Virtex-5 family. The design, which is outlined in Figure 2, uses specific resources within such devices and implies close integration of design and underlying technology; this means it is very efficient, in terms of resource utilisation, versus generic designs implemented on the same fabric. More specifically, each instance requires two SLICEM blocks and generates a 1-bit output (i.e., response): the output value depends intrinsically on the process variation dependent presence or absence of a glitch.

The output bit is held in a flip-flop (DFF), whose input DFF^{pre} is driven by a cascade of two multiplexers (MUX_0 and MUX_1); furthermore, DFF^Q is fed back into DFF^D so that the value is continuously latched by positive edges of a clock signal. When the device is initialised at power-on, DFF is initialised with 0. It remains in this state until $DFF^{pre} = 1$: this input is asynchronous and active high, meaning even a transient change (or glitch) will update the value held by DFF from 0 to 1 independently from the clock. The rest of the design is intended to generate said glitch, which controls DFF and hence determines the PUF output. We refer the reader to [1] for complete argumentation regarding the behavioural characteristics.

The perfect PUF design would have no bias: given m instances, the value of associated output bits would be uniformly distributed. Several points are worth stressing:

1) The PUF design does not permit an input, or challenge, in the same way discussed in Section II-A. Rather, the challenge is an intrinsic property of (or hard-coded in) the bit-stream. This fact implies security of the bit-stream is an important (though

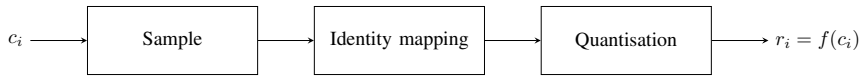


Figure 1: A high-level overview of components within a generic PUF design.

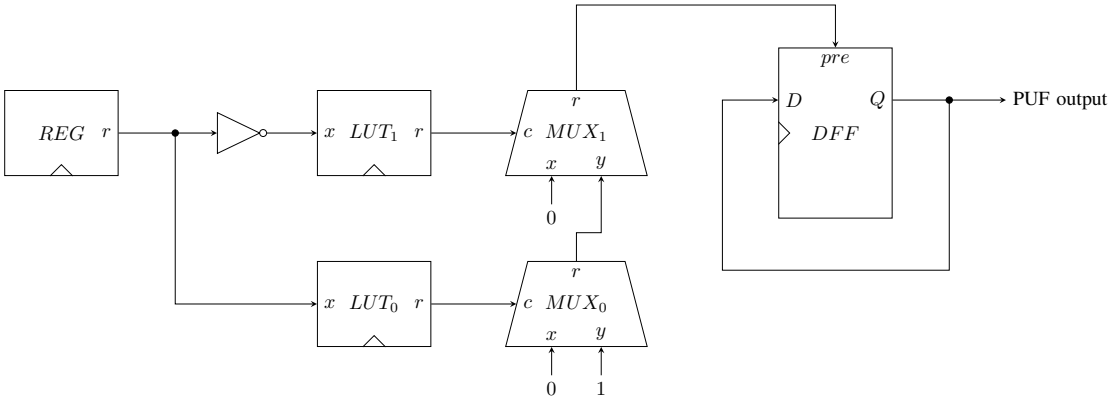


Figure 2: The Anderson PUF design, which realises the first (i.e., sampling) step in Figure 1 (note that clock signals driving REG, LUT_0, LUT_1 and DFF are omitted for clarity).

possibly dubious [8]) assumption for some use-cases.

2) The value held by DFF may be changed from 0 to 1 by a glitch on DFF^{pre}. However, there is *no* complementary operation: once the flip-flop holds 1, it remains in this state until the next power-on. As shown in Section III, this fact is important and plays a significant role in bias.
3) The "quality" of the glitch on DFF^{pre} is also important, and also plays a role in bias. Placement of the LUTs can influence said quality and, as such, different approaches to placement were studied in [1, Section III].

III. Analysis and redesign of FPGA-based PUF

A. Experimental platform

We used a control workstation, in conjunction with a remote controlled power supply, to conduct experiments on two SASEBO-GII development boards; all experiments were carried out under normal environmental conditions. Each such board houses a Xilinx Virtex-5 device (model XC5VLX50-1FFG324). A total of $28,800$ LUT cells are available, approximately a quarter of which can be used in shift register mode: this gives $\sim 7,200$ LUTs. Specific placement constraints (e.g., two LUTs placed in two different slices imply other cells within those slices are "blocked") and asymmetry of the FPGA fabric mean at most 960 PUF cells can be instantiated on each device. Following the approach of Anderson [1, Section IV], we internally divided each device into three regions: the first two hold PUF instances, while the third is reserved for an RS232 module that allows communication between board and control workstation. For clarity, we term R_i^j a virtual board resulting from the j-th region on the i-th physical board R_i (omitting either index where appropriate).

An existing[2] VHDL implementation of the design in Section II-B2 was used; synthesis was performed using the Xilinx ISE tool-chain, using associated placement constraints. Since each PUF instance produces a 1-bit response, the design was wrapped in a (parameterisable) module to permit m instances and hence an m-bit response. This allows use of Hamming distance for both inter- and intra-distance metrics.

B. Analysis of the Anderson PUF

Recall that each 1-bit PUF output is held by a flip-flop initialised to 0; the flip-flop content will irrevocably change to 1 whenever a glitch occurs on the asynchronous preset input. According to [1], the routing path from the output of the carry chain multiplexers to the flip-flop preset input can be regarded as a low-pass filter, i.e., glitches of too short a duration may not trigger the preset input of the flip-flop, which almost always will remain at 0.

If this assumptions holds, one would expect a stable PUF response (i.e., for the response to be constant over time). Figure 3a shows the response for a 320-bit Anderson PUF measured at different points in time. Notice that as time progresses, the number of bits irrevocably switching to 1 increases. In a sense, this is obvious: over time glitches continue to occur, and even if they are filtered, one will eventually trigger the preset input. This results in responses whose Hamming weight increases as time progresses, a phenomenon we term "saturation" of the response.

Figure 3b illustrates intra- and inter-distance metrics for a 320-bit Anderson PUF. The former gives an approximation of variation in PUF responses between samples taken at power-on and also 50 s later. We measured an intra-distance of roughly 90 bits; this corresponds to an

[2] http://www.eecg.toronto.edu/~janders/PUF/

(unacceptable) error of around 30%, and will further increase over time. This violates a basic requirement of PUFs, namely that the response should be reproducible (up to a small error) at any time.

C. Solution #1: measurement after specific delay

One approach to eliminate the saturation problem is to latch the PUF output at a single point of time: this captures a stable PUT response. Ideally, this would be performed immediately after power-on so an applications can make use of it straight away. However, Figure 3a shows there is considerable bias towards 0 in the PUF response at this time; the resulting non-uniformity makes the response unusable for direct use as key material.

The idea, therefore, is to capture the PUF response at a later point in time (potentially tuned to a specific device) so the response is close to being uniformly distributed without the need for key extraction. This approach implies a trade-off between latency (i.e., time before the response is captured and used) and quality.

Figure 3c shows the Hamming weight of responses from the enhanced 320-bit PUF design on four virtual boards; the results relate to sampling the PUF after 0.5 s after power-on, then using the response at a variety of times thereafter. The stability of responses is trivial, but confirms resolution of the original problem. Figure 3d shows intra- (with average $\mu_{\mathsf{intra}} = 4\%$) and inter-distance (with average $\mu_{\mathsf{inter}} = 45\%$) variation concerning one virtual board and the same experiment; both are clearly (more) in line with requirements than the original PUF design.

D. Solution #2: a "one-shot" approach

An alternative approach permits capture of a PUF response immediately at power-on. As mentioned in the previous section, there is considerable bias towards 0 at this point for the original PUF design. In the proposed [1] LUT placement, glitches tend to have a short duration, meaning they are "damped out" while passing the first time through the routing network acting as a low-pass filter.

The idea, therefore, is to control the glitch width for transitions between LUT_0 and LUT_1, increasing it so the probability that a glitch occurs at the preset input of the flip-flop is also increased. This can realised simply by inserting an additional carry chain between the two LUTs. This approach implies a trade-off between area (i.e., an additional LUT is required, per instance, to support the extended carry chain) and flexibility (i.e., the range of variation possible).

Figure 3e shows the Hamming weight of responses from the enhanced 208-bit (fewer PUF instances due to the area overhead) PUF design on four virtual devices; the results relate to sampling the PUF immediately after power-on, then using the response at a variety of times thereafter. We stress that sampling at power-on is not a limitation, and one can defer the process should a use-case require: reinitialising both LUTs then sampling, or blocking input from the external shift register until sampling are both options. Figure 3f shows intra- (with average $\mu_{\mathsf{intra}} = 5\%$) and inter-distance (with average $\mu_{\mathsf{inter}} = 46\%$) variation concerning one virtual board and the same experiment.

IV. Conclusion

Design of cost efficient and secure solutions using FPGA devices is a non-trivial task. In this paper, we clearly underline this difficulty by identifying a flaw in the Anderson PUF design; we examined two approaches that resolve the flaw, transforming the design into a robust building block. We note that the result not only applies to stand-alone FPGA applications, but could also prove beneficial in systems that embed a tightly-coupled FPGA fabric. On the Zynq platform [12], for instance, the ARM micro-controller could utilise the PUF response produced on the integrated FPGA fabric.

Within this emerging field, a wide range of further work seems important. Fundamentally, for a robust set of results, experimentation would ideally be performed across a large sample of devices (to evaluate the impact of process variation). Without a large budget this is clearly problematic, contrasting with SRAM-based PUFs for example and especially where high-end FPGA devices are used. In our case, we have only four virtual devices: as such, rigorous evaluation of a larger sample, plus different environmental conditions, is an important next step. Then, methods for improving the PUF design demand a more rigorous analysis of PUF cell placement for example; again in comparison with other designs, esp. arbiter PUFs, investigation of whether similar modelling based approaches apply to FPGA-based designs seems vital.

It would be also worth to investigate, if any modification of the original Anderson design, such as changing a glitch value to 0 instead of 1 will lead to a more robust design with better properties. Further, investigation of whether the saturation effect itself is an intrinsic property of an FPGA device is also left as future work.

Acknowledgements

The work described in this paper has been supported in part by EPSRC grant EP/H001689/1.

References

[1] J.H. Anderson. A PUF design for secure FPGA-based embedded systems. In *ASP-DAC*, pages 1–6, 2010.

[2] J. Guajardo, S.S. Kumar, G.J. Schrijen, and P. Tuyls. FPGA intrinsic PUFs and their use for IP protection. In *Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, CHES '07, pages 63–80, 2007.

[3] I. Kim, A. Maiti, L. Nazhandali, P. Schaumont, V. Vignesh, and H. Zhang. From statistics to circuits: Foundations for future physical unclonable functions. In *Towards Hardware-Intrinsic Security*, pages 55–78, 2010.

[4] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Škorić, and P. Tuyls. Reconfigurable physical unclonable functions — enabling technology for tamper-resistant storage. In *HOST*, pages 22–29, 2009.

[5] R. Maes, P. Tuyls, and I. Verbauwhede. Intrinsic PUFs from flip-flops on reconfigurable devices. In *3rd Benelux Workshop on Information and System Security (WISSec 2008)*, 2008.

[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, 2010.

[7] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer, 2007.

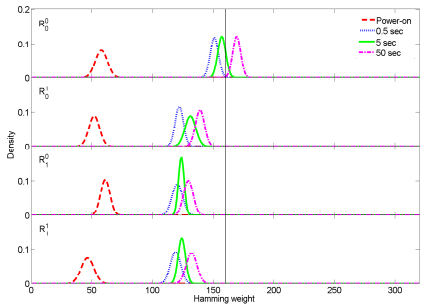
[8] A. Moradi, M. Kasper, and C. Paar. Black-box side-channel attacks highlight the importance of countermeasures: An analysis of the Xilinx Virtex-4 and Virtex-5 bitstream encryption mechanism. In *CT-RSA*, pages 1–18. LNCS 7178, 2012.

[9] S. Morozov, A. Maiti, and P. Schaumont. An analysis of delay based PUF implementations on FPGA. In *ARC*, pages 382–387, 2010.

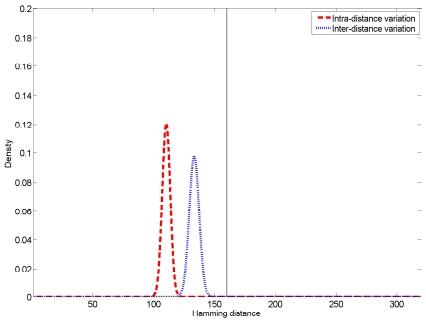
[10] D. Suzuki and K. Shimizu. The glitch PUF: A new delay-PUF architecture exploiting glitch shapes. In *Proceedings of the 12th international conference on Cryptographic hardware and embedded systems*, CHES'10, pages 366–382, 2010.

[11] Xilinx. Virtex-5 FPGA user guide. http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

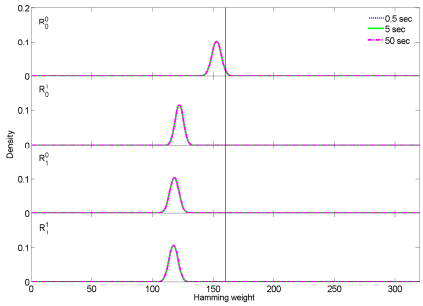
[12] Xilinx. Zynq-7000. http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000%-Overview.pdf.



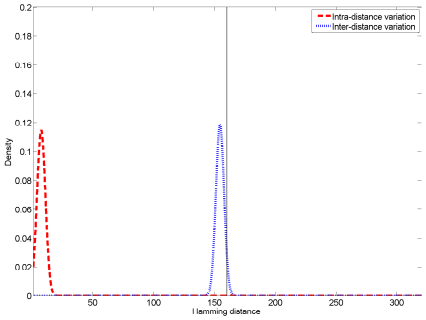
(a) Anderson PUF, saturation test.



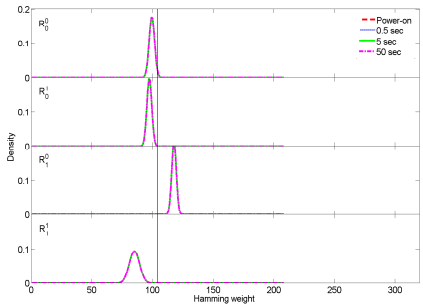
(b) Anderson PUF, intra- and inter-distance test.



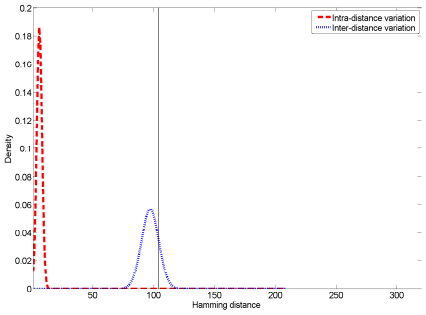
(c) Solution #1, saturation test.



(d) Solution #1, intra- and inter-distance test.



(e) Solution #2, saturation test.



(f) Solution #2, intra- and inter-distance test.

Figure 3: Experimental results for the Anderson PUF (top) and alterations relating to solution #1 (middle) and solution 2 (bottom); results for saturation (left), or Hamming weight of the PUF response over time, and inter- and intra-distance (right). Note the line marking the ideal average inter-distance (right) relates to $m/2$, i.e, half the total PUF response size; the ideal average intra-distance is clearly 0.