PUF Interfaces and their Security

Marten van Dijk
University of Connecticut
vandijk@engr.uconn.edu

Ulrich Rührmair
Technische Universität Berlin
ruehrmair@ilo.de

***Abstract*—In practice, any integrated physical unclonable function (PUF) must be accessed through a logical interface. The interface may add additional functionalities such as access control, implement a (measurement) noise reduction layer, etc. In many PUF applications, the interface in fact hides the PUF itself: users only interact with the PUF's interface, and cannot "see" or verify what is behind the interface. This immediately gives rise to a security problem: how does the user know he is interacting with a properly behaving interface wrapped around a proper PUF? This question is not merely theoretical, but has strong relevance for PUF application security: It has been shown recently that a badly behaving interface could, e.g., log a history of PUF queries which an adversary can read out using some trapdoor, or may output "false" PUF responses that the adversary can predict or influence [20]. This allows attacks on a considerable number of PUF protocols [20]. Since we currently do not know how to authenticate proper interface behavior in practice, the security of many PUF applications implicitly rests on the mere assumption that an adversary cannot modify or enhance a PUF interface in a "bad" way. This is quite a strong hypothesis, which should be stated more explicitly in the literature.**

In this paper, we explicitly address this point, following and partly expanding earlier works [20]. We add to the picture the need for rigorous security which is characterized by some security parameter λ (an adversary has "$negl(\lambda)$ probability to successfully software clone/model a PUF"). First, this means that we need so-called Strong PUFs with a larger than $poly(\lambda)$ input/challenge space. In order to have scalable PUF designs (which do not blow up in chip surface or volume for increasing λ), we need PUF designs which constitute of a "algebraic" composition of smaller basic building blocks/devices. In such compositions the security relies on a less well-established computational hardness assumption which states that machine learning and other modeling methods with $poly(\lambda)$ runtime cannot reliably produce a software clone of the PUF. To provide rigorous security we argue that the PUF interface needs a one-way postprocessing of PUF responses such that the security can be reduced to the infeasibility of breaking the one-way property of the postprocessing. This leads to a set of interesting problems: how do we add noise reduction into this picture and how do we minimize or eliminate side channel leakage of computed intermediate values in the post processing?

I. Introduction

A PUF is a randomly structured and unclonable physical system P that can be excited with external stimuli or so-called challenges c. It reacts with corresponding responses r, which depend on the challenge and the unique random structure of the PUF. PUF P implements a statistical process that maps challenges c to responses $r \leftarrow P(c)$ ($P(c)$ defines a statistical distribution from which r is drawn). Tuples (c, r) are called the challenge-response pairs (CRPs) of P.

Due to their complex internal structure, PUFs can avoid some of the shortcomings of digital keys stored in non-volatile memory (NVM): It is usually harder to read out, predict, or derive PUF-responses than to obtain digital keys from NVM. The PUF-responses are only generated when needed, which means that the keys are not permanently present in an easily accessible digital form. Furthermore, certain PUFs are naturally tamper sensitive, as their exact behavior depends on minuscule manufacturing irregularities in different layers of the IC. Removing or penetrating these layers will automatically change the PUF's read-out values. These facts have been exploited in the past for different PUF-based security protocols: identification [16], [6], key exchange [16], and various forms of (tamper sensitive) key storage and applications thereof, such as intellectual property protection or read-proof memory [8], [10], [24].

In order to avoid an attack in which a software clone/model is produced by querying a PUF for all possible challenges, we need PUFs with a very large challenge set. This type of PUF sometimes has been referred to as *Physical Random Function* [6] or *Strong PUF* [8], [22], [21], [17] in the literature. [1] Its input-output behavior is assumed to be so complex that its response to a randomly chosen challenge cannot be predicted numerically and without direct physical measurement, not even by a person who had physical access to the Strong PUF at earlier points in time (here we do not mean physical access to the internals of the PUF, but access to the PUF in being able to query CRPs). Exactly this assumption of having a complex input-output behavior which is hard to machine learn or model is not well-established: Rather than relying on this assumption, we propose to rely on the PUF interface itself to postprocess responses by using a one-way function such that the security directly reduces to the hardness of breaking the one-way function. This reduces the PUF's role to that of an unclonable physical key storage (which was its original motivation).

[1] A Strong PUF is used similar to a *physical random oracle*, which can be passed on between parties in more advanced protocols (e.g. oblivious transfer, bit committment, key agreement etc.) and which can be read-out exactly by the very party who currently holds physical possession of it.

A PUF is always accessed through its logical interface. Unfortunately, in practice we cannot guarantee that the PUF interface is "good" in that the PUF+interface hardware has been faithfully generated and never been manipulated. A more realistic model is the "bad PUF model" [20], [14] which allows adversaries to manipulate the PUF interface: These manipulated "bad" PUFs look like a normal PUF from the outside, having a standard CRP-interface etc., but have extra properties that allow cheating. Even though the bad PUF model is more realistic in the worst-case scenario, for most applications we need to assume good PUFs (since we do not know how to deal with bad PUFs) for most PUF interactions (and discount worst-case scenarios in the applications' business models).

In what follows we focuss on PUF interfaces: We first explain how they are used for providing extra functionality within the good PUF model. Next we discuss how an extra interface layer (a one-way postprocessing) can provide rigorous security guarantees and we discuss the corresponding security assumption.

II. Logical Interfaces

In practice a PUF is wrapped inside an interface which enables extra functionality, in particular:

1) By using measurement-noise reduction techniques such as fuzzy extraction, privacy amplification, or error correction in the form of erasure decoding using "helper information" or majority voting, the PUF becomes (with high probability) a function (rather than a statistical process that samples a distribution).
2) By using pre-processing a challenge seed can be expanded into a sequence of challenges that are each individually evaluated by the PUF to provide a sequence of response bits. In order to maximize the entropy in the sequence of response bits, the pre-processing generates a sequence of challenges such that each two challenges within the sequence have a large "distance" between one another.
3) Generally, pre- and post-processing allows more complex access control mechanisms (by binding challenges and responses to user/program supplied information). E.g., as in controlled PUFs, or its generalization which allows proofs-of-execution. One could also construct *Logically Erasable PUFs* by letting the control logic maintain some record of the previously applied challenges and of the erased challenges (e.g., in the form of an authenticated hash tree). [2]

The PUF interface as described above does not extend the Trusted Computing Base (TCB) of the PUF itself in that an adversary is allowed to see the internal values computed within the interface: the security does not depend on whether these values get known by the adversary or not. The PUF interface as described above only extends the TCB in that the PUF can only be accessed through this interface.[3] In other words we need to assume an adversary who is not able to circumvent the interface in order to access the PUF. This means that the PUF with interface is assumed to behave in a "proper" way and that "bad" behavior is excluded. As has been shown in recent research [20], [14] this security assumption may not hold: PUF interfaces may very well be modified/enhanced with bad behavior such as (1) circumpassing PUF behavior and use a predictable (to the adversary) algorithm instead or (2) logging past measured challenge response pairs (which can be retrieved by an adversary). Since we do not know how to authenticate proper interface behavior, we simply need to assume an adversary who cannot modify or enhance a PUF interface in a "bad" way. This is a strong security assumption needed for most applications which should be explicitly stated.

III. The Need for Post-Processing

In what follows we argue that in order to be scalable to smaller dimensions, we need PUFs which rely on the assumption that no machine learning (or other) method can be used to construct a software clone of the PUF. This assumption is not well-established in the machine learning community and for this reason we propose an extra interface layer to provide rigorous security:

We realize that PUFs are composed of smaller basic devices (building blocks). The (software) unclonability of the PUF depends on the complexity of this composition since each device individually can easily be modelled based on a small number of its input output pairs. E.g., a PUF may be a parallel composition of n devices with an interface which (based on a challenge) reads the outputs of λ out of the n devices, where λ represents the security parameter. Since each challenge response pair teaches a non-negligible amount of information about λ devices, $O(n/\lambda)$ challenge response pairs is sufficient to construct a reliable software clone of the PUF. Since λ is the security parameter, n must be sub-exponential in λ. As an example, SRAM PUFs use a parallel composition with a sub-exponential number of SRAM cells placed in a cross-bar memory layout. We conclude that a strong PUF based on parallel composition needs a sub-exponential number of devices leading to a large surface.

To be scalable to smaller dimensions, we need PUFs that use devices in different compositions. For example, devices are laid out in a sequential way where the outputs of one

[2] Also Logically Reconfigurable PUFs (LR-PUFs) as introduced by Katzenbeisser et al. [9] can be an option in this context. They allow the manufacturer of the PUF to collect a CRP-list that remains valid even after many reconfiguration operations.

[3] The original TCB assumes that the PUF can only be accessed as a black box which can be querried by an adversary without the adversary allowing access to internal values computed within the PUF. The extended TCB limits access through the PUF interface, but allows the adversary to learn iternally computed values within the interface.

device constitute the inputs of the next device as in Arbiter-based PUFs. This leads to a smaller number of devices per PUF (hence, a small surface per PUF). The security now relies on the difficulty of using machine learning methods to create a software clone. It has been shown that at least ≈ 8 Arbiter PUFs need to be XORed in order for this assumption to hold for currently know machine learning methods [21]. We must find the right balance in our composition: complex enough to allow for a hardness assumption and simple enough not to amplify noise generated by the individual devices in the composition.

Rather than finding a composition of devices that leads to a secure, reliable and scalable PUF, another solution is to build a logical interface layer that post-processes the composition's output using a one-way function such that the irreversibility of the one-way function itself is the computational hardness assumption that prohibits an attacker from learning any of the actual input-output behavior of the devices within the composition. Since a one-way function is input sensitive, the logical interface must first apply noise reduction techniques before evaluating the one-way function. We must add noise reduction in such a way that corresponding helper information (e.g. in the form of parity symbols) cannot be used to break the one-way function.

The proposed interface layer provides rigorous security if we assume an extended TCB where adversaries cannot obtain information about the internally computed values within the interface layer. In particular, this requires an implementation which avoids side channel leakage.

We notice that the proposed logical interface post-processes responses using a non-keyed one-way function f: Given a challenge c, the interface outputs $f(P(c))$ (we neglect error reduction for simplicity). Its behavior is the same as a keyed one-way function $f_K(c)$ where key K is replaced by PUF P. This means that into some extend our approach reduces PUFs to unclonable physical keys and nothing more.

IV. Conclusion

We conclude that PUFs, especially so-called Strong and Controlled PUFs [18], [20], should ideally be considered in combination with their logical interfaces. In order to achieve practical security, additional assumption about this interface have to made. This gives rise to a considerable research potential in this direction. One prime example are methods for the verification of the interface, and the underlying PUF behind the interface, in Strong PUF architectures [20]. Another example is the development of internal interfaces for Controlled PUFs that are side channel resilient (compare [1]).

References

[1] Georg T. Becker, Raghavan Kumar: *Active and Passive Side-Channel Attacks on Delay Based PUF Designs*. IACR Cryptology ePrint Archive 2014: 287 (2014)

[2] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, U. Rührmair: *The Bistable Ring PUF: A new architecture for strong Physical Unclonable Functions*. HOST 2011: 134-141

[3] M. van Dijk: *System and method of reliable forward secret key sharing with physical random functions.* US Patent No. 7,653,197, October 2004.

[4] B. Gassend, *Physical Random Functions*. MSc Thesis, MIT, 2003.

[5] B. Gassend, M. van Dijk, D.E. Clarke, E. Torlak, S. Devadas, and P. Tuyls: *Controlled physical random functions and applications.* ACM TISSEC 10(4), 2008.

[6] B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas: *Silicon physical random functions*. ACM CCS 2002.

[7] B. Gassend, D. Lim, D. Clarke, M. van Dijk, S. Devadas: *Identification and authentication of integrated circuits.* Concurrency and Computation: Practice & Experience, 2004.

[8] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls: *FPGA Intrinsic PUFs and Their Use for IP Protection*. CHES 2007.

[9] S. Katzenbeisser, C. Koabas, V. van der Leest, A.-R. Sadeghi, G. J. Schrijen, and C. Wachsmann: *Recyclable PUFs: Logically Reconfigurable PUFs*. Journal of Cryptographic Engineering 1(3): 177-186 (2011).

[10] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, and P. Tuyls: *The Butterfly PUF: Protecting IP on every FPGA*. HOST 2008.

[11] K. Kursawe, A. R. Sadeghi, D. Schellekens, P. Tuyls, and B. Skoric: *Reconfigurable physical unclonable functions – Enabling technology for tamper-resistant storage*. HOST 2009.

[12] J.-W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. *A technique to build a secret key in integrated circuits with identification and authentication applications.* In Proceedings of the IEEE VLSI Circuits Symposium, 2004.

[13] R. Maes, I. Verbauwhede: *Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions*. Section 1 in D. Naccache and A.-R. Sadeghi (Ed.), Towards Hardware-Intrinsic Security, Springer, 2010.

[14] R. Ostrovsky, A. Scafuro, I. Visconti, and A. Wadia: *Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions*. Eurocrypt 2013.

[15] R. Pappu: *Physical One-Way Functions*. PhD Thesis, Massachusetts Institute of Technology, 2001.

[16] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld: *Physical One-Way Functions*, Science, vol. 297, 2002.

[17] U. Rührmair, H. Busch, and S. Katzenbeisser: *Strong PUFs: Models, Constructions and Security Proofs.* In A.-R. Sadeghi, P. Tuyls (Editors): Towards Hardware Intrinsic Security: Foundation and Practice. Springer, 2010.

[18] U. Rührmair, S. Devadas, F. Koushanfar: *Security based on Physical Unclonability and Disorder*. In: M. Tehranipoor and C. Wang (Editors): Introduction to Hardware Security and Trust. Springer, 2011

[19] U. Rührmair and M. van Dijk: *Practical Security Analysis of PUF-based Two-Player Protocols*. CHES 2012.

[20] U. Rührmair and M. van Dijk: *PUFs in Security Protocols: Attack Models and Security Evaluations*. IEEE S&P 2013.

[21] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber: *Modeling Attacks on Physical Unclonable Functions*. ACM CCS, 2010.

[22] U. Rührmair, J. Sölter, and F. Sehnke: *On the Foundations of Physical Unclonable Functions.* Cryptology ePrint Archive, Report 2009/277, 2009.

[23] G. E. Suh and S. Devadas: *Physical Unclonable Functions for Device Authentication and Secret Key Generation.* DAC 2007.

[24] P. Tuyls, G. J. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, and R. Wolters *Read-Proof Hardware from Protective Coatings.* CHES 2006.

[25] P. Tuyls and B. Skoric: *Strong Authentication with Physical Unclonable Functions.* In: Security, Privacy and Trust in Modern Data Management, 2007.