On Foundation and Construction of Physical Unclonable Functions

J. Wu and M. O'Neill
ECIT, Queen's University Belfast
Belfast, UK
{j.wu, m.oneill}@ecit.qub.ac.uk

Abstract

Physical Unclonable Functions (PUFs) have been introduced as a new cryptographic primitive, and whilst a large number of PUF designs and applications have been proposed, few studies has been undertaken on the theoretical foundation of PUFs. At the same time, many PUF designs have been found to be insecure, raising questions about their design methodology. Moreover, PUFs with efficient implementation are needed to enable many applications in practice.

In this paper, we present novel results on the theoretical foundation and practical construction for PUFs. First, we prove that, for an ℓ-bit-input and m-bit-output PUF containing n silicon components, if $n < \frac{m2^\ell}{c}$ where c is a constant, then 1) the PUF cannot be a random function, and 2) confusion and diffusion are necessary for the PUF to be a pseudorandom function. Then, we propose a helper data algorithm (HDA) that is secure against active attacks and significantly reduces PUF implementation overhead compared to previous HDAs. Finally, we integrate PUF construction into block cipher design to implement an efficient physical unclonable pseudorandom permutation (PUPRP); to the best of our knowledge, this is the first practical PUPRP using an integrated approach.

1 Introduction

A physical unclonable function (PUF) is a physical object that can take inputs and generate unpredictable outputs; it is *unclonable* in that the input/output behaviour of a physical copy of one PUF will differ from that of the original one due to some uncontrollable randomness in the copying process. Ravikanth et. al [27] proposed the first PUF in literature, which is a piece of transparent medium containing randomly solved particles. When a laser beam propagates through the medium, it produces a speckle pattern that is affected by the particles. How the particles are solved in the medium is random and uncontrollable; therefore, two such PUFs produced using the same process will have different output speckle patterns. Ravikanth et. al's work demonstrated a novel way to implement an oneway function, which is a fundamental tool for many security application; in addition, the oneway function is physically unclonable and tamper-resistant, which may enable many new applications. After that, a large number of PUFs have been proposed [8, 9, 10, 11, 16, 15, 18, 22, 26, 30, 32]. Except for a few instances such as the Coating PUF [32] and the LC-PUF [11], most PUFs only use conventional silicon integrated circuit (IC) techniques so that the PUFs can be easily integrated in IC chips. Many PUF applications have been proposed that involve intellectual property (IP) protection [10], authenticated identification [6], and trusted computing [31], to name a few.

Some PUFs are designed to behave as pseudorandom functions: given an input, the output is unpredictable. To prevent the input/output from being exhausted, the PUFs should support a large input size, say, a 64-bits input. Examples of such PUFs include the controlled PUF (CPUF) [8, 9], the arbiter-PUF [16], and the lightweight PUF (LW-PUF) [21]. The most serious challenge for such a PUF is security: in a standard attacking model where an adversary is allowed to test the PUF by choosing any inputs and receiving the outputs, the adversary should be unable to predict the output for a fresh input. A second challenge is to achieve such security using low overhead. For example, it is desirable that a PUF does not rely on hash functions, since hash functions are unsuitable for extremely resource-constrained devices such as RFID tags [5].

Some other PUFs are designed as cryptographic key generators or random functions with small input size. A representative example is the SRAM-PUF [10] that use SRAM cells as raw bit generators. Since the raw bits are noisy and are not fully random, a Helper Data Algorithm (HDA) is needed to extract a reproducible, stable, and uniformly random secure key from the raw bits. Furthermore, an ℓ-bits-input and m-bit-output random function can be constructed by letting an ℓ-bits input choose m key bits out of $2^\ell \times m$ such key bits as the output. However, most HDAs use error-correcting codes which require significant hardware and computation overhead; in addition, the helper data of a HDA may leak information about the key in certain attacks. The main challenge for such PUF key generators is to design robust and secure HDA with low overhead.

One group of special PUFs, namely the *physical unclonable pseudorandom permutations* (PUPRPs), are of particulate interests because they can be used for encryption/decryption. A straightforward way to construct a PUPRP is to use a conventional block cipher with a key generated by PUF key generators. In this case, the security of the PUPRP is based on the security of the PUF key generator and the security of the block cipher separately. Armknecht et. al [2] proposed an integrated approach to construct PUPRPs, which combines PUFs into block cipher. The tight integration improves the tamper-resilience of the overall design. Armknecht et. al's construction is theoretical because its overhead increases exponentially as the input size of the PUPRP increases.

While there have been a lot of PUF constructions and applications, there is little study on the foundations of PUFs. All PUF constructions concern with implementation overhead. One fundamental question is that, what is the minimum overhead a PUF needs? Rührmair et. al [28] consider a PUF as an isolated physical system and discussed PUFs based on the maximal thermodynamic entropy of an isolated physical system. Rührmair et. al's work provides a new point of view for PUF study; however, the thermodynamic entropy is not directly related to any mathematical property of a PUF. Another fundamental question for PUFs is that, should there be some systematic design methodology for PUF construction? So far, the PUF designs in literature are ad hoc, somewhat resembling the cipher designs before Shannon [29] established diffusion and confusion as the principles for modern block cipher design. It would be interesting to investigate if some design principles also apply to PUF.

1.1 Our Contributions

In this paper, we study PUFs that are based on standard silicon components since they comprise the majority of existing PUFs. We systematically study the theoretical foundations and practical constructions for such silicon PUFs. Our contributions are as follows:

- We compute the information capacity, i.e., the information-theoretic entropy, of the PUFs, and use the information capacity to determine the maximum input/output size that a PUF can achieve as a random function. We further prove that diffusion and confusion are necessary to construct PUFs as secure pseudorandom functions.
- We propose a PUF key generator consisting of a new bit generator and a novel HDA. Compared to previous PUF key generators, ours significantly reduces the hardware overhead to achieve required robustness and security.
- We propose a practical PUPRP construction that integrates PUFs in block cipher designs. As the input size of the PUPRP increases, its overhead increase linearly as oppose to exponentially in the previous integrated PUPRP construction; as a result, the PUPRP has an efficient hardware implementation.

1.2 Organization

In Section 2, we review existing silicon PUFs. In Section 3, we present the foundations for PUFs, including a PUF model, the PUF information capacity, and the necessary conditions for secure PUF construction. In Section 4, we present the new bit generator and the helper data algorithm. In Section 5, we present the PUPRP and its security analysis. Section 6 concludes the work.

2 Related Work

One line of PUF designs aims to support large input length, say, 64-128 bits. Examples include the controlled PUF (CPUF) [8, 9], the arbiter-PUF [16], and the lightweight PUF (LW-PUF) [21]. Gassend et al. [8, 9] introduce the concept of silicon PUFs and construct Controlled PUFs (CPUF). A CPUF contains a “weak” PUF consisting of a ring oscillator (RO) and a counter. The choice of the delay path of the RO is controlled by an ℓ bits input. The counter counts the number of rounds of the RO within a certain time interval. For the same input, the counting may vary from one PUF instance to another, depending on the periods of their ROs. One hash function is placed before the PUF and one hash function after to scramble the input/output. Since the output is determined by the path delay that is affected by temperature and power supply voltage, the CPUF is sensitive to environmental influences.

To reduce the environmental influences, Lim et al. [16] propose an arbiter-based PUF using a differential structure. Instead of measuring one delay value, arbiter-based PUFs compare the delays of two identical paths. Since the variation of temperature or power supply affects both paths, their effects may cancel each other, and hence increases the stability of the output. A basic arbiter-based PUF can be represented using a linear model that can be solved from multiple input/output pairs. To prevent this attack, a feed-forward structure is added to provide nonlinearity in the arbiter-PUF. However, Majzoobi et al. [21] and Rührmair et al. [28] show that the feed-forward arbiter-PUF (without input/output hashing) can be learned by machine learning algorithms; therefore it is not secure.

To restore the security of the arbiter-based PUF without using input/output hashing, Majzoobi et al. [21] proposed a lightweight PUF (LW-PUF), which uses linear transformation for the input/output preprocessing and postprocessing; however, Rührmair et al. [28] show that this LW-PUF is also not secure.

Another line of PUF design focuses on PUF key generators, or equivalently, PUFs with small input length. A representative example is the SRAM-PUF [10]. A SRAM-PUF with an ℓ-bits input and an m-bits output consists of $2^{\ell} \times m$ SRAM cells. Each cell is a bi-stable circuit that settles to 0 or 1 after power-up. Such a SRAM-PUF uses $m2^{\ell}$ cells, therefore it is not intended to provide a large number of challenge-response pairs; instead, the $m2^{\ell}$ bits are often used to generate a cryptographic key.

A SRAM cell may settle to different states in different power-ups due to circuit noises. To obtain stable outputs, several helper data algorithms are proposed [10], [7], [2], [19]. Since most FPGAs set their SRAM memories to a known state upon power-up, several methods have been developed to simulate the SRAM cells by using standard digital logic on a FPGA. Kumar et al. [15] propose Butterfly PUFs based on cross-coupled latches. Maes et al. [18] propose Flip-Flop PUFs based on the power-up values of the flip-flops present on FPGAs.

3 PUF Foundations

3.1 Preliminary

First we define some notations that will be used throughout the paper:

$|\mathcal{S}|$: number of elements in a finite set $\mathcal{S}$.

$x \overset{D}{\longleftarrow} \mathcal{S}$: x is chosen from a set $\mathcal{S}$ according to a probability distribution D over $\mathcal{S}$.

$x \overset{U}{\longleftarrow} \mathcal{S}$: x is chosen from $\mathcal{S}$ according to a uniform probability distribution over $\mathcal{S}$.

(ℓ, m)-f: a function f with an ℓ-bits input and an m-bits output.

$\mathcal{Z}$: the set of all functions that map a domain $\{0,1\}^{\ell}$ to a range $\{0,1\}^{m}$ (Fact: $|\mathcal{Z}| = 2^{m2^{\ell}}$).

$\Phi(x)$: cumulative function of a standard Gaussian distribution.

$\Phi^{-1}(x)$: inverse of the cumulative function of a standard Gaussian distribution.

$\varphi(x)$: density function of a standard Gaussian distribution..

$\mathbf{B}(n,p,x,y)$: $\mathbf{B}(n,p,x,y) = \sum_{i=x}^{y} \binom{n}{i} p^i (1-p)^{n-i}$.

When it is not specified, a function has an ℓ-bits input and an m-bits output. We assume that the functions can be computed within a reasonable time for practical ℓ and m values.

Next we review the distinguishing game based on which pseudorandomness will be defined:

Definition 3.1 *Let $\mathcal{F}$ be a set of functions (i.e., a function family) and D be a probability distribution over $\mathcal{F}$. Let E be a distinguisher. A challenger C chooses a random bit $b \overset{U}{\longleftarrow} \{0,1\}$. If $b = 0$, then $f \overset{U}{\longleftarrow} \mathcal{Z}$ is returned to E; otherwise, $f \overset{D}{\longleftarrow} \mathcal{F}$ is returned to E. Within time t and querying f up to q times, E returns a bit b'. The advantage of E to distinguish $f \overset{D}{\longleftarrow} \mathcal{F}$ from $f \overset{U}{\longleftarrow} \mathcal{Z}$ is defined as*

$$Adv_{\mathcal{F},D}^{prf}(E) = \left| \Pr[b = b'] - \frac{1}{2} \right| . \qquad (1)$$

An insecure function associated to $f \overset{D}{\longleftarrow} \mathcal{F}$ is defined as the maximum advantage over all E:

$$Adv^{prf}_{\mathcal{F},D}(q,t) = \max_{E} \left\{ Adv^{prf}_{\mathcal{F},D}(E) \right\}.$$

The above definition is an extension of that by Bellare et al [3]. In the original definition, D is a uniform distribution over $\mathcal{F} \subset \mathcal{Z}$, but here we allow D to be any distribution to precisely model PUF below.

3.2 PUF Model

Our PUF model and related notations and facts are described as follows: each PUF design determines a probability distribution D over $\mathcal{Z}$; each *PUF instance* implements a sample $f \overset{D}{\longleftarrow} \mathcal{Z}$. D is called a *PUF distribution.* The *function family* of D is defined as the subset of $\mathcal{Z}$ in which each function is sampled with a non-zero probability. If $\mathcal{F}$ is the function family of D, then random process $f \overset{D}{\longleftarrow} \mathcal{Z}$ is identical to $f \overset{D}{\longleftarrow} \mathcal{F}$.

We call a PUF instance $f \overset{U}{\longleftarrow} \mathcal{Z}$ a *physical unclonable random function PURF*, and we call a PUF instance $f \overset{D}{\longleftarrow} \mathcal{F}$ a *physical unclonable pseudorandom function PUPRF* if its insecure function defined in (1) is small enough (under practically large q and t) to be negligible in practice. If the PUPRF is a permutation, then we call it a *PUPRP.*

Among previous PUFs, the SRAM-PUF and its variants, the butterfly-PUF and the Flip-flop-PUF, can be considered as PURF. Other PUFs with a large input length, including the optical-PUF, the CPUF, the arbiter-PUF, and the LW-PUF, are designed as PUPRFs. A block cipher contains PURFs to generate a key can be considered as a PUPRP.

3.3 PUF Information Capacity

Since a PUF is a silicon object that implements a deterministic function, it has to contain the information that determines the function. We consider what is the maximum information capacity that a PUF can achieve. Suppose that a PUF contains n silicon components. Their delay values are n continuous random variables. We assume that their probability distribution is Gaussian distribution with a mean value T and a variance ${\sigma_X}^2$. The information source used to determine the function is the n random variables. Measuring a continuous variable can obtain any number of bits of information if the measurement is accurate enough. However, due to the noise in silicon circuits, the information capacity of a continuous variable is bounded. We assume that the jitters of delays caused by noises follow a Gaussian distribution with a mean value 0 and a variance ${\sigma_X}^2$. The information capacity of one such variable can be computed using the following theorem: [12, §8,§9].

Lemma 3.2 *Let X be a continuous random variable with Gaussian distribution with a variance ${\sigma_X}^2$, let N be a Gaussian noise with a variance ${\sigma_N}^2$, and let $x_1, \ldots, x_r$ be r samples of X. On average, the maximum information capacity of the r samples is*

$$c \;=\; \frac{1}{2}\log_2\left(1 + \frac{r^2{\sigma_x}^4}{{\sigma_N}^2(r{\sigma_x}^2 + {\sigma_N}^2)}\right)$$

bits.

Then, the information capacity of a PUF with n components is obtained in the next theorem:

Lemma 3.3 *We assume that 1) a PUF contains n components, and $X_1, \ldots, X_n$ be their delays, 2) the delays are continuous random variables with Gaussian distribution with a mean value T and a variance ${\sigma_X}^2$, and 3) the jitter of a delay follows a Gaussian distribution with a mean value 0 and a variance ${\sigma_N}^2$. It holds that, within a time period $rT, r \geq 1$, the maximum information capacity that the PUF can obtain is*

$$I_{max}(X_1, \ldots, X_n) \quad \leq \quad nc$$

bits.

Proof (Sketch). Sampling a delay with mean value T takes a time T on average. Then, for each X_i, within time rT, r samples can be made on average. The information capacity of the n random variables is at most the sum of the information capacity of each random variables (equality holds only when the random variables are independent). □

3.4 PURF Resource Requirement

We show that the input size of a PURF is limited by its information capacity:

Lemma 3.4 *Let $\mathcal{F}$ be the function family of a PUF distribution D over $\mathcal{Z}$. If each PUF instance has n components and $n < \frac{m2^\ell}{c}$, then it holds that $|\mathcal{F}| < |\mathcal{Z}|$.*

Proof. The maximum information capacity of a PUF instance is at most nc. With nc bits information, at most 2^{nc} distinct objects can be identified. It holds that

$$|\mathcal{F}| \leq 2^{nc}.$$

With $n < \frac{m2^\ell}{c}$ and $|\mathcal{Z}| = 2^{m2^\ell}$, it holds that

$$|\mathcal{F}| \leq 2^{nc} < 2^{m2^\ell} = |\mathcal{Z}| \tag{2}$$

□

Corollary 3.5 *If a PUF instance has n components where $n < \frac{m2^\ell}{c}$, then it is not a PURF.*

The result indicates that it may be impractical to construct a PURF with large input size, since the circuit size of the PUF increases exponentially as ℓ increases.

A (ℓ, m)-SRAM-PUF uses $m2^\ell$ SRAM cells (plus some control logic). If the power-up state of the cells settle to 0 and 1 with equal probability, then the (ℓ, m)-SRAM-PUF is a (ℓ, m)-PURF. Therefore, $n = \Theta(m2^\ell)$ can be considered as the necessary and sufficient condition for building a (ℓ, m)-PURF.

3.5 Confusion and Diffusion in PUPRF

When $|\mathcal{F}| < |\mathcal{Z}|$, a PUF $f \overset{D}{\leftarrow} \mathcal{F}$ is not a random function; but it may be indistinguishable from a random function, i.e., a pseudorandom function. In this section, we consider how to construct PUPRF.

Shannon [29] introduced diffusion and confusion as two principles to design symmetric key ciphers where the key length is less than the message length. Let $\mathcal{F}$ be the function family of a PUF distribution D over $\mathcal{Z}$. We review diffusion and confusion in terms of PUFs.

Definition 3.6 *Diffusion requires that, for $y = f(x), f \in \mathcal{F}$, each bit of x affects all bits of y.*

Next, we review confusion. To identify if $f \in \mathcal{F}$, $\log_2 |\mathcal{F}|$ bits of information are sufficient. Therefore, it holds that $f(x) = F(K, x)$ for some function F and a key K of $\log_2 |\mathcal{F}|$ bits. Let $b_j(y)$ be the j^{th} bit of y; after querying f for q times, one obtains qm pairs of $(x_i, b_j(y_i))$ that correspond to qm Boolean function equations

$$b_j(y_i) = b_j(F(K, x_i)), \qquad \begin{array}{l} 0 \le i \le m-1 \\ 0 \le j \le q-1. \end{array} \tag{3}$$

When $qm \ge \log_2 |\mathcal{F}|$, K is uniquely determined by the qm $(x_i, b_j(y_i))$ pairs. Confusion requires that, although the q pairs of (x, y) uniquely determine K, it is infeasible to find out K.

Definition 3.7 *Confusion requires that, for practical ℓ and m values, given $f \overset{D}{\longleftarrow} \mathcal{F}$, within a reasonable time, it is infeasible to query f for q times and find K that satisfies the qm pairs of $(x_i, b_j(y_i))$ in (3).*

Next, we show that, if $|\mathcal{F}| < |\mathcal{Z}|$, then diffusion and confusion are necessary to achieve low $Adv^{prf}_{\mathcal{Z},D}(q, t)$.

Lemma 3.8 *Let D be a distribution over $\mathcal{F}$. It holds that*

$$Adv^{prf}_{\mathcal{F},D}(q, t) \ge Adv^{prf}_{\mathcal{F},U}(q, t). \tag{4}$$

Informally, Lemma 3.8 says that, for a uniform distribution U over $\mathcal{F}$ and a non-uniform distribution D over $\mathcal{F}$, $f \overset{U}{\longleftarrow} \mathcal{F}$ is more random than $f \overset{D}{\longleftarrow} \mathcal{F}$. We skip the proof of Lemma 3.8.

Lemma 3.9 *Let $\mathcal{F} \subset \mathcal{Z}$. We assume that computing $f(x), f \in \mathcal{F}$ takes a time $O(\ell m)$. If diffusion is not satisfied, then it holds that, for a time $t = O(\ell m q)$*

$$Adv^{prf}_{\mathcal{F},U}(q, t) > \frac{1}{2}\left(1 - \left(\frac{1}{2}\right)^q\right). \tag{5}$$

Proof (sketch). If diffusion is not satisfied, then there exist i and j such that, for $f \in \mathcal{F}$,

$$\begin{aligned} & b_i(f(x_0, \ldots, x_j, \ldots, x_{\ell-1})) \\ = \; & b_i(f(x_0, \ldots, \bar{x}_j, \ldots, x_{\ell-1})). \end{aligned} \tag{6}$$

A distinguisher E queries f for $2q$ times with q pairs of $(x_0, \ldots, x_j, \ldots, x_{\ell-1})$ and $(x_0, \ldots, \bar{x}_j, \ldots, x_{\ell-1})$. If (6) holds for all pairs, then E concludes that $f \overset{U}{\longleftarrow} \mathcal{F}$, and $f \overset{U}{\longleftarrow} \mathcal{Z}$ otherwise.

For $f \in \mathcal{F}$, equation (6) holds for all q pairs of selected (x, y). For $f \overset{U}{\longleftarrow} \mathcal{Z}$, which is a random function, the probability that (6) holds for all q pairs of selected (x, y) is $\frac{1}{2^q}$. It is easy to verify that (5) holds. □

Lemma 3.10 *Let $\mathcal{F} \subset \mathcal{Z}$. We assume that computing $f(x), f \in \mathcal{F}$ takes a time $O(\ell m)$. Suppose that confusion is not satisfied so that a distinguisher E can query f and find out K that satisfies (3) with in time t'. We also assume that, after obtaining K, E knows if $f \in \mathcal{F}$. Then, for some time $t = t' + O(q\ell m)$, it holds that*

$$Adv^{prf}_{F,U}(q, t) > \frac{1}{2}\left(1 - \left(\frac{|\mathcal{F}|}{|\mathcal{Z}|}\right)\right). \tag{7}$$

Proof (sketch). E queries f for q times and solves (3) for K. If K is obtained in time t and it holds that $F(K, x) \in \mathcal{F}$, then E concludes that $f \overset{U}{\leftarrow} \mathcal{F}$.

E is wrong only when $f \overset{U}{\leftarrow} \mathcal{Z}$ and $f \in \mathcal{F}$. It is easy to show that (7) holds. $\square$

Combining (2), (4), (5), and (7), we have the following result:

Proposition 3.11 *If each PUF instance has n components where $n < \frac{m2^\ell}{c}$, then confusion and diffusion are necessary for the PUFs to be secure PUPRFs.*

Remark. Equation (6) defines a specific relation between x and $y = f(x), f \in \mathcal{F}$. In fact, any deterministic relations between x and $y = f(x), f \in \mathcal{F}$ would invalidate $\mathcal{F}$ as pseudorandom function family. Searching for such relations is an important approach for block cipher cryptanalysis. Linear cryptanalysis [23] and differential analysis [4] are two such examples; they are addressed in more details in Section 5.3.

4 PUF Key Generator and PURF

4.1 Construction

In this section, we present a PUF key generator consisting of a new bit generator and a novel helper data algorithm that we call MVBB (majority-voting-bright-bits). The bit generator is a bi-stable ring of two NAND gates as shown in Figure 1.When the CTL signal is 0, both the outputs of the two NANDs are 1. When CTL is 1, the two NANDs are configured as a bi-stable ring, and the output O settles to either 1 or 0. Compared to the previous Butterfly-PUF [15], the NAND-loop bit generator is more efficient in hardware cost and takes less response time. We note that the NAND-loop can be implemented on an FPGA by properly configuring the FPGA synthesis tool; this is contrary to the thought that combinational loops cannot be created on an FPGA [15].

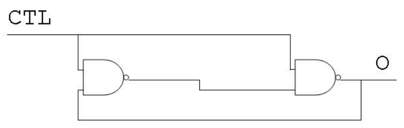

Figure 1: Bit generator.

Bit generator model We review a model for the bi-stable ring from [20] with slight change to describe the hebaviour of the bi-stable ring based on propagation delays and jitters. Let X_1 and X_2 be the delays of the two NAND gates and $\Delta = X_1 - X_2$, and let N be the jitter. Without loss of generality, we assume that the NAND ring settles to 1 if $\Delta + N > 0$, and settles to 0 otherwise. Let σ_N be the standard deviation of the delays and σ_N be the standard deviation of jitters. Let p be the probability that a bit generator outputs 1 and let $\lambda = \frac{\sigma_N}{\sqrt{2}\sigma_X}$. It holds that

$$\Pr[p \le x] = \Phi(\lambda \Phi^{-1}(x))$$

and

$$\Pr[p = x] = \frac{\lambda \varphi(-\lambda \Phi^{-1}(x))}{\varphi(\Phi^{-1}(x))}$$

where $\Phi(), \Phi^{-1}()$, and $\varphi()$ are the the cumulative function, the inverse of the cumulative function, and the density function of a standard Gaussian distribution [20].

MVBB algorithm In the MVBB algorithm, each bit generator is *dark*, *grey*, or *bright* according to its p value, the probability that it outputs 1. Let each bit generator generates an odd number of n outputs. We specify two two parameters d_g (distance of grey) and d_d (distance of dark), $0 < d_g < d_d < n/2$. A bit generator is *dark* if

$$\frac{d_d}{n} \le p \le \frac{n - d_d}{n};$$

the generator is *grey* if

$$\frac{d_g}{n} \le p < \frac{d_d}{n} \text{ or } \frac{n - d_d}{n} < p \le \frac{n - d_g}{n};$$

and the generator is *bright* if

$$0 \le p < \frac{d_g}{n} \text{ or } \frac{n - d_g}{n} < p \le 1.$$

Figure 2 illustrates $\Pr[p = x], 0 \le x \le 1$, and regions of the bright, grey, and dark bits.

Suppose that a PUF uses k bit generators. The MVBB algorithm consists of two algorithms defined as follows.

- Helper data generating: for each generator, MVBB measures the probability, p, that the generator outputs a value of 1. A string α of k bits is generated and output as the helper data to indicate which key bits are dark and which are non-dark. A k bit key, K', is also generated. If $p > \frac{1}{2}$, then the corresponding key bit is 1; otherwise the key bit is 0. The key bits of dark generators are set to 0.

  In practice, the probability measurement can be approximated by $\frac{x}{n}$ where x is the number of 1s in the generator outputs. In the analysis, we will assume that p is accurately obtained.

- Key reproducing: MVBB receives a helper data string β. For each bit generator, MVBB generates n outputs and record x, the number of 1s. MVBB estimates $p = \frac{x}{n}$. If a generator is bright according to p but is marked as dark in β, then MVBB stops key producing. Otherwise, MVBB generates a k bit key K. For bit generators marked as dark in β, the key bits are set to 0. For a bit generator marked as non-dark in β, the key bit is 1 if $p > \frac{1}{2}$, or 0 otherwise.

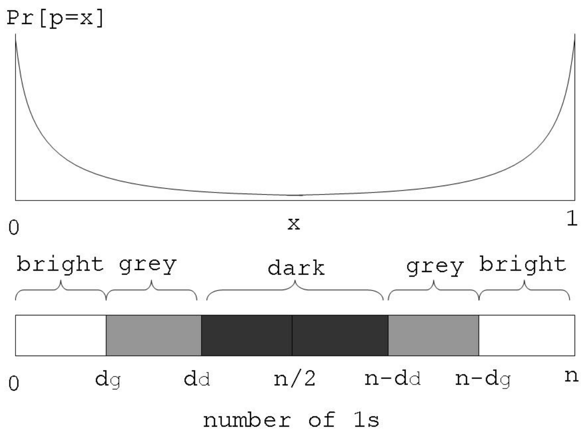

Figure 2: Bright, grey, and dark bit generators.

For efficient hardware implementation of the logic, we choose $n = 2^i - 1, d_g = 2^{i_g}$, and $d_d = 2^{i_d}$ where i, i_g, and i_d are integers and $i_g < i_d < i$. The key bit is the most significant bit of the counter. If the $i - i_g$ most significant bits of the counter are all 1s or 0s, then the bit generator is bright; if the $i - i_d$ most significant bits of the counter are all 1s or 0s, then the bit generator is bright or grey.

Privacy amplification K is a non-uniform random string because of the dark generators, and because the output of a bit generator may output 0 or 1 with unequal probability. We implemented 1024 generators on Xilinx FPGA. In our experiment, about 60% bit generators output 0, so the entropy of a secure key bits is about 0.97 bits. The entropy of the k-bits key, K, is $0.97n_b$. K can be compressed to generate a $0.97n_b$ bits (both in length and in entropy) cryptographic key using 2-universal hash functions (see, e.g., [7]).

4.2 Robustness

A HDA should be robust so that the same key will be reproduced in different key generating rounds. For the MVBB algorithm, we require that, if the helper data is not altered, then $K = K'$. We have the following result:

Theorem 4.1 *Let K' and K be the keys generated in MVBB helper data generating and MVBB key reproducing respectively. If the helper data is not altered, i.e., $\alpha = \beta$, then it holds that*

$$\Pr[K \neq K'] \leq \Pr[E_b] + \Pr[E_g].$$

where

$$\Pr[E_b] \leq 1 - \mathbf{B}\left(n, \frac{d_g - 1}{n}, 0, d_d - 1\right)^n$$

and

$$\Pr[E_g] \leq 1 - \mathbf{B}\left(n, \frac{d_d - 1}{n}, 0, \frac{n-1}{2}\right)^n$$

Proof. When the helper data are not changed, there are two cases that result in $K \neq K'$: 1) the number of 1s generated by any bright generator changes from a value less than d_g to a value greater than d_d or from a value greater than $n - d_g$ to a value less than $n - d_d$ (an event denoted as E_b), and 2) the number of 1s generated by any grey generator changes from a value between d_g and d_d to a value greater than $n/2$ or from a value between $n - d_d$ and $n - d_g$ to a value less than $n/2$ (a event denoted as E_g). Then the results follow immediately. □

We give an example to illustrate the robustness of MVBB. Based on results on delay variances and jitters in circuits ([17], [13]), we assume that $\frac{\sigma_X}{\sigma_N} = 16$, so we set $\lambda = 0.05$. Suppose that K has a length of 100 bits. We choose $n = 127$, and set $d_g = 8$ and $d_d = 32$. Then it holds that $\Pr[K \neq K'] \leq 3.1 \times 10^{-8}$.

4.3 Security

For a HDA, we need to consider if the key, K, is secure under both *passive attacks* and *active attacks*. In passive attacks, an adversary tries to learn K without changing the helper data; in active attacks, an adversary tries to learn K by manipulating the helper data.

Active attacks against previous HDAs We note that the majority-voting-dark-bit (MVDB) algorithm [2] is insecure against active attacks when it is used to generate keys. In MVDB, bit generators are marked as dark or non-dark. In the key reproducing process, key bits produced by non-dark bit generators are appended to the key string. An active attack is described as follows. Suppose that the key generated by MVDB is used on a cipher, and the adversary has access to the helper data and can use the cipher as an oracle, but the adversary does not has access to the key. Let $\beta_1, \beta_2, \ldots$ be the non-dark bits in the helper data and let $k_1, k_2, \ldots$ be the key bits. The adversary changes b_1 in the helper data to dark and encrypts a message. In this case, the key used for encryption is $k_2 k_3 \ldots$. Then the adversary changes b_2 to dark but keep b_1 non-dark, and encrypt the same message again. This time, the key used for encryption is $k_1 k_3 \ldots$. By checking if the two cipher texts are the same, the adversary knows if $k_1 = k_2$. Repeating this process, the adversary can groups all key bits in two sets where each set contains the key bits with the same value. The adversary can find which set is 1 using at most two trials and fully recover the key. We do not find other HDAs such as [7] and [2] that are all based on error-correcting codes susceptible to the same active attacks.

Security of MVBB Regarding the security of MVBB, we have the following result.

Theorem 4.2 *The key bits in K produced by bright generators are secure under both passive attacks and active attacks.*

Proof. Helper data itself does not leak any information about a bright or grey key bit, i.e., $\Pr[k_i = 1] = \Pr[k_i = 1 | \beta_i]$, because by knowing that a generator is grey or bright does not leak if the generator outputs 1 or 0; therefore, the grey bits and bright bits are secure under passive attacks. In active attacks, there are three cases: 1) a non-dark bit β_i of a bright generator is changed to dark in β. In this case, the MVBB will stop, and the adversary knows that the corresponding bit generator is bright, but he obtains no information about if the key bit is 1 or 0; 2) a non-dark bit of a grey generator is changed to dark in β. In this case, the MVBB will regard corresponding bit generator as dark and set its key bit to 0. This may leak the true value of the key bit of this grey generator as in the active attack against MVDB above. However, since all bit generators are independent, the adversary does not obtain any information about the key bits of any bright bit generator; 3) a dark bit β_i is changed to non-dark in β. In this case, the MVBB will proceed to generate an incorrect key, but the adversary still obtains no information about the key bit of a bright or grey bit generator. In summary, the key bits of the bright bit generators are always secure. □

As an example, Suppose that K has a length of 100 bits. For $\lambda = 0.05$, $n = 127$, $d_g = 8$, and $d_d = 32$, it holds that $\Pr[n_b < 80] < 1.1 \times 10^{-7}$, i.e., the probability that less than 80 keys bits is secure is less than 1.1×10^{-7}.

4.4 Overhead

The MVBB does not use error-correcting codes; this makes it much simpler and more efficient than other error-correcting code based HDAs. For example, the soft decision HDA [19] is the most efficient HDA among previous error-correcting code based HDAs; its algorithm is executed using a arithmetic logic unit (ALU) controlled by an finite state machine (FSM) that applies the consecutive algorithm steps stored as microcode. As a comparison, the MVBB has a structural hardware and is more efficient in both hardware and computation.

4.5 PURF

Using a uniformly random key string of length $m2^\ell$ generated above, we can build an (ℓ, m)-PURF. The bits are arranged as a $2^\ell \times m$ matrix. For an ℓ bit input, one of the 2^ℓ rows is chosen. The m bits of the chosen row is the output of the PURF.

5 PUPRF Construction

5.1 PUPRF Design Approaches

Initially, in PUF design, a PUF is supposed to contain a large number random elements; the input to the PUF interacts with the elements in a complex process so that the output is unpredictable (e.g. the optical PUF [27]). In this case, the physical objects contains sufficient entropy, or the complex physical interaction between the input and the PUF provides confusion and diffusion. Lemma 3.3 shows that a silicon object cannot provide sufficient entropy for large input PUFs in practice; therefore, confusion and diffusion must be used. However, no existing *silicon* PUFs relying solely (or mainly) on physical processes such as the arbiter-PUF [16] and the LW-PUF [22] provide sufficient confusion to be secure [28]. How to design secure PUPRF based on *physical* diffusion and confusion mechanisms is still an open problem.

Some other PUFs use cryptographic primitives to provide confusion and diffusion. For example, CPUF uses hash functions. Another example of PUF is a block cipher using a key generated by PURFs. For these PUFs, confusion and diffusion are implemented by conventional hash or cipher algorithms.

Armknecht et. al [2] proposed a PUPRP construction using an integrated approach that integrate PUFs with the confusion/diffusion mechanism of a block cipher. The tight integration improves the tamper-resilience of the overall design. Armknecht et. al's construction is theoretical because its overhead increases exponentially as the input size of the PUPRP increases.

In this section, we propose a practical PUPRP construction using the integrated approach. As the input size of the PUPRP increases, its overhead increase linearly; as a result, the PUPRP has an efficient hardware implementation.

5.2 Feistel PUPRP

We use $(4,4)$-PURFs and a Feistel-network to construct a $(64,64)$-PUPRP as shown in Figure 3. The PUPRP uses 12 rounds of iterations. Each iteration uses a Feistel function (F function). The F function is a substitution-permutation network (SPN) as shown in Figure 4. The SPN consists of a layer of eight (4,4)-PURFs serving as substitute functions (S-Box) and a permutation layer $\pi()$ as defined in Table 1. The permutation layer is the same as the permutation network in DES [1]. In additional to the input x and output y, the PUPRP also receives helper data, α, and output helper data, β, as defined in the MVBB algorithm for each PURF it contains.

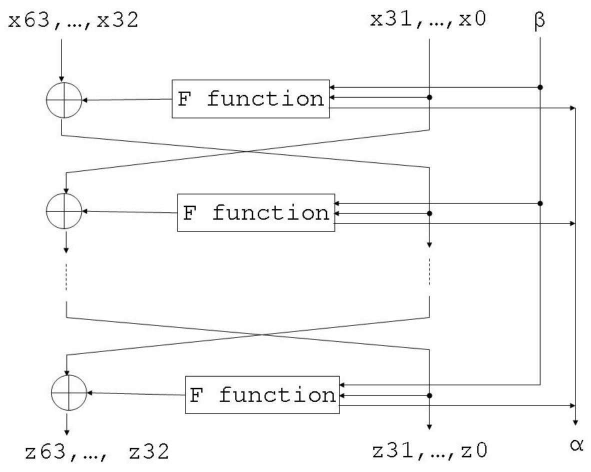

Figure 3: Feistel network PUPRF.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(i)$	8	16	22	30	12	27	1	17	23	15	29	5	25	19	9	0
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\pi(i)$	7	13	24	2	3	28	10	18	31	11	21	6	4	26	14	20

Table 1: Permutation network

Compared with a standard Feistel cipher using a SPN as the F function, e.g., the Feistel cipher model used in [14], the Feistel-PUPRF unrolls each round of the Feistel cipher, removes the key schedule model and key combining units, and replaces each S-box with a PURF. In principle, we can use any Feistel cipher and change it into a PUPRF with this approach.

Remark We note that a SPN block cipher (e.g., AES) cannot be turned into a PUPRP by directly replacing the S-boxes with PURFs. Suppose that we remove key schedule and key combining, and replace the S-Boxes of a SPN cipher with PURFs. Since a PURF is not a permutation, different

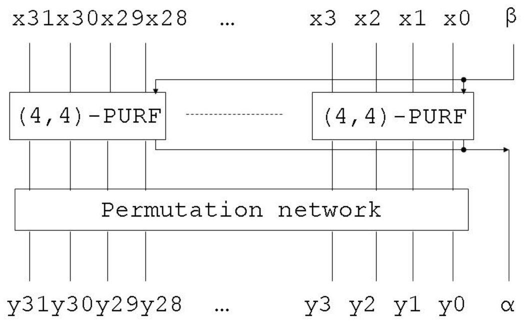

Figure 4: F function.

inputs may produce the same output. Therefore, with a probability $\frac{1}{2^n}$ (n is the input length of the S-Box), flipping one bit in the input of the cipher does not change its output. In Feistel cipher, this problem is avoided because of the XOR operations in each round.

We consider two version of Feistel-PUPRP. One is the full version described above. The other is a reduced version, where only one F function is repeatedly used in all rounds. The full version uses 96 $(4,4)$-PURFs and the reduced version uses eight $(4,4)$-PURFs.

5.3 Security Analysis

Confusion and diffusion First we consider whether the Feistel-PUPRP achieves sufficient confusion. For each $(4,4)$-PURF, assuming that its input is known, then each output bit of the $(4,4)$-PURF can be described as a linear Boolean function of 64 unknown variables. In the full version, each of the 32 left output bits of the PUPRP involves 12 rounds of F functions. In the last round, one PURF is involved. In all other 11 rounds, at least four PURFs are involved in each round. Therefore, the bit can be expressed as a Boolean function of at least $(11+1) \times 64 = 2880$ variables of degree 12. Similarly, each of the 32 left output bits can be expressed as a Boolean function of at least $(10+1) \times 64 = 2624$ variables of degree 11. In the reduced version, the PURFs used in each round are the same; each bit on the left is a Boolean function of 512 variables of degree 11, and each bit on the right is a Boolean function of 512 variables of degree 10. To reveal the unknown variables, one needs to solve a system of such equations. If the computation is infeasible in practice, then confusion is achieved. In fact, the difficulty of solving a system of nonlinear equations of a large number of variables are the basis of all block ciphers.

Next we consider diffusion. In each round, one output bit is affected by four inputs. We consider four consecutive rounds as a block. Approximately, for one block, every output bit is affected by all 64 inputs, and hence every input bit affects all 64 output bits. Three blocks (twelve rounds) appear to provide sufficient diffusion.

Linear and differential cryptanalysis Linear cryptanalysis [23] and differential cryptanalysis [4] are powerful attacking tools for block ciphers including Feistel ciphers. They exploit the relations

Table 2: NIST RNG testing for reduced version Feistel-PUPRP.

P-VALUE	Proportion	Statistical test	P-VALUE	Proportion	Statistical test
0.24	1.00	Frequency	0.19	0.99	OverlappingTemplate
0.26	0.97	BlockFrequency	0.81	0.97	Universal
0.80	0.99	CumulativeSums	0.32	0.96	ApproximateEntropy
0.83	0.99	Runs	0.60	1.00	RandomExcursions
0.72	1.00	LongestRun	0.77	1.00	RandomExcursionsVariant
0.83	0.99	Rank	0.85	0.98	Serial
0.04	1.00	NonOverlapping Template	0.40	1.00	LinearComplexity

between the input and output of each S-box, and then use the knowledge to further analyze the whole cipher. Security evaluation for differential and linear cryptanalyses for Feistel ciphers with SPN round function are available [14]. However, in the Feistel-PUPRP, each PURF is a random function, where the input-output relation is not defined as the S-boxes in conventional block ciphers. It appears that linear cryptanalysis or differential cryptanalysis is not applicable to the Feistel-PUPRP.

Random test We use the NIST RNG test suite [24] to test the randomness of bit strings generated by a software simulated Feistel-PUPRP in counter mode. For both the reduced version and the full version Feistel-PUPRP, 1000 of 1Mbits strings are generated and tested. The results for both versions indicate no deviation from random behaviour. A test result for the reduced version is shown in Table 2.

The above analysis shows no weakness of the Feistel-PUPRP. However, as a new cryptographic primitive, the Feistel-PUPRP needs more analysis and public scrutiny. The proposed PUPRF is more of a proof-of-concept than an off-the-shelf product.

6 Conclusion and Discussion

We have systematically studied theoretical foundations and practical constructions for silicon PUFs. We have proposed a simple model that captures the core concept of PUF and enables us to study the security of PUFs in a strict mathematical model. We then presented an information capacity bound for silicon PUFs. Based on the model and the bound, we proved that silicon PUFs cannot be random functions with large input length, and confusion and diffusion are necessary to construct PUFs as pseudorandom functions. These results may provide some guidelines for PUF design and security analysis.

We proposed a new bit generator and a novel helper data algorithm to construct robust and secure physical unclonable random functions (PURFs). Compared with previous constructions, our algorithm significantly reduced hardware cost to achieve required robustness and secrecy. Then, using the PURFs and a Feistel network, we constructed a physical unclonable pseudorandom permutation (PUPRP). To the best of our knowledge, this is the first secure and practical silicon PUPRP construction without using a block cipher or a hash function as a building block. It also demonstrated a general approach to convert a Feistel cipher to a PUPRP.

Acknowledgement

The authors are grateful to Roger Woods for valuable comments and advices.

References

[1] Fips 46-3: The official document describing the DES standard. 1999.

[2] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Berk Sunar, and Pim Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 685–702. Springer, 2009.

[3] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[4] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, pages 2–21, London, UK, 1991. Springer-Verlag.

[5] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, and Y. Seurin. Hash functions and RFID tags: Mind the gap. In Oswald and Rohatgi [25], pages 283–299.

[6] Leonid Bolotnyy and Gabriel Robins. Physically unclonable function-based security and privacy in rfid systems. In *PERCOM '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications*, pages 211–220, Washington, DC, USA, 2007. IEEE Computer Society.

[7] Christoph Bösch, Jorge Guajardo, Ahmad-Reza Sadeghi, Jamshid Shokrollahi, and Pim Tuyls. Efficient helper data key extractor on FPGAs. In Oswald and Rohatgi [25], pages 181–197.

[8] Blaise Gassend, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 148–160. ACM, 2002.

[9] Blaise Gassend, Marten Van Dijk, Dwaine Clarke, Emina Torlak, Srinivas Devadas, and Pim Tuyls. Controlled physical random functions and applications. *ACM Trans. Inf. Syst. Secur.*, 10(4):1–22, 2008.

[10] Jorge Guajardo, Sandeep S. Kumar, Geert Jan Schrijen, and Pim Tuyls. FPGA intrinsic PUFs and their use for IP protection. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2007.

[11] Jorge Guajardo, Boris Skoric, Pim Tuyls, Sandeep S. Kumar, Thijs Bel, Antoon H. M. Blom, and Geert Jan Schrijen. Anti-counterfeiting, key distribution, and key storage in an ambient world via physical unclonable functions. *Information Systems Frontiers*, 11(1):19–41, 2009.

[12] L. P. Hyvärinen. *Information Theory for System Engineers.* Springer-Verlag, 1970.

[13] Keith A. Jenkins, Anup P. Jose, and David F. Heidel. An on-chip jitter measurement circuit with sub-picosecond resolution. In *Proceedings of ESSCIRC*, 2005.

[14] Masayuki Kanda. Practical security evaluation against differential and linear cryptanalyses for feistel ciphers with SPN round function. In *SAC '00: Proceedings of the 7th Annual International Workshop on Selected Areas in Cryptography*, pages 324–338, London, UK, 2001. Springer-Verlag.

[15] Sandeep S. Kumar, Jorge Guajardo, Roel Maes, Geert Jan Schrijen, and Pim Tuyls. The Butterfly PUF: Protecting IP on every FPGA. In Mohammad Tehranipoor and Jim Plusquellic, editors, *HOST*, pages 67–70. IEEE Computer Society, 2008.

[16] Daihyun Lim, Jae W. Lee, Blaise Gassend, G. Edward Suh, Marten van Dijk, and Srinivas Devadas. Extracting secret keys from integrated circuits. *IEEE Trans. VLSI Syst.*, 13(10):1200–1205, 2005.

[17] Ping Liu and Yong-Bin Kim. An accurate timing model for nano CMOS circuit considering statistical process variation. In *IEEE International SoC Design Conference(ISOCC)*, pages 269–272, 2007.

[18] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. Intrinsic PUFs from flip-flops on reconfigurable devices. In *3rd Benelux Workshop on Information and System Security (WISSec 2008)*, 2008.

[19] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2009.

[20] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. A soft decision helper data algorithm for SRAM PUFs. In *IEEE International Symposium on Information Theory (ISIT 2009)*, 2009.

[21] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Lightweight secure PUFs. In Sani R. Nassif and Jaijeet S. Roychowdhury, editors, *ICCAD*, pages 670–673. IEEE, 2008.

[22] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Techniques for design and implementation of secure reconfigurable PUFs. *TRETS*, 2(1), 2009.

[23] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 386–397, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[24] NIST. NIST random number generation and testing, 2006. `http://csrc.nist.gov/groups/ST/toolkit/rng/index.html`.

[25] Elisabeth Oswald and Pankaj Rohatgi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*. Springer, 2008.

[26] Erdinç Öztürk, Ghaith Hammouri, and Berk Sunar. Physical unclonable function with tristate buffers. In *ISCAS*, pages 3194–3197. IEEE, 2008.

[27] Pappu Srinivasa Ravikanth. *Physical one-way functions.* PhD thesis, 2001. Chair-Benton, Stephen A.

[28] Ulrich Rührmair, Jan Sölter, and Frank Sehnke. On the foundations of physical unclonable functions. Cryptology ePrint Archive, Report 2009/277, 2009. `http://eprint.iacr.org/`.

[29] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28:656–715, 1949.

[30] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *DAC*, pages 9–14. IEEE, 2007.

[31] G. Edward Suh, Charles W. O'Donnell, Ishan Sachdev, and Srinivas Devadas. Design and implementation of the aegis single-chip secure processor using physical random functions. *SIGARCH Comput. Archit. News*, 33(2):25–36, 2005.

[32] Pim Tuyls, Geert Jan Schrijen, Boris Skoric, Jan van Geloven, Nynke Verhaegh, and Rob Wolters. Read-proof hardware from protective coatings. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2006.