Secure Key Generation from Biased PUFs

Roel Maes[1(✉)], Vincent van der Leest[1], Erik van der Sluis[1], and Frans Willems[2]

[1] Intrinsic-ID, Eindhoven, The Netherlands
{roel.maes,vincent.van.der.leest,erik.van.der.sluis}@intrinsic-id.com
[2] T.U. Eindhoven, Eindhoven, The Netherlands
f.m.j.willems@tue.nl

Abstract. PUF-based key generators have been widely considered as a root-of-trust in digital systems. They typically require an error-correcting mechanism (e.g. based on the code-offset method) for dealing with bit errors between the enrollment and reconstruction of keys. When the used PUF does *not* have full entropy, entropy leakage between the helper data and the device-unique key material can occur. If the entropy level of the PUF becomes too low, the PUF-derived key can be attacked through the publicly available helper data. In this work we provide several solutions for preventing this entropy leakage for PUFs suffering from i.i.d. *biased bits*. The methods proposed in this work pose no limit on the amount of bias that can be tolerated, which solves an important open problem for PUF-based key generation. Additionally, the solutions are all evaluated based on reliability, efficiency, leakage and reusability showing that depending on requirements for the key generator different solutions are preferable.

1 Introduction

A Physically Unclonable Function (PUF) implemented on an integrated circuit (IC) can be used as a hardware root-of-trust for a digital system, e.g. to generate and store the system's private master keys. These PUF-based key generators provide a secure and efficient alternative for protected non-volatile memories (e.g. Flash, EEPROM, antifuses, etc.). For such applications, a high-quality PUF is needed which is both unpredictable as well as reliable, i.e. PUF responses are random per instantiation but repeatable with limited noise over time and under all circumstances. To achieve this, both entropy extraction (for unpredictability) and error-correction coding (for reliability) are required in key generators. A secure key derivation function is used for entropy extraction to derive a cryptographic key from a random seed. For error-correction, a commonly used technique is the code-offset method that stores helper data during an enrollment phase, which is later used to correct bit errors that occur in the PUF response when reconstructing the key. This helper data should not provide any information about the key, because it is generally stored and/or transferred publicly.

It has recently been pointed out [9] that if a PUF does *not* have full entropy, information about the secret key material is leaked by the helper data. If this

T. Güneysu and H. Handschuh (Eds.): CHES 2015, LNCS 9293, pp. 517–534, 2015.
DOI: 10.1007/978-3-662-48324-4_26

entropy leakage becomes too large, the derived key will not have full entropy; this poses a serious threat to the security of the key generator. The main focus of this work is the development of (pre-)processing methods that prevent this leakage in case of biased PUFs. Using innovative approaches, the overhead of these algorithms on PUF size is minimized, while guaranteeing reliability of the system.

Related Work. PUF-based key generators based on error-correcting codes were firstly introduced in [13] for Arbiter PUFs and later on in [5] for SRAM PUFs, both using a configuration based on linear block codes. Efficiency optimizations were proposed based on code concatentation [1] and soft-decision decoding [10,17]. Later key generators based on ring oscillator PUFs were presented in [18,22]. Potential security issues can arise with these key generators when their input (i.e. the PUF response) does not have full entropy, see e.g. [6], which was recently emphasized strongly in [9].

Contributions. The primary contribution of this work is the introduction of a number of solutions that prevent entropy leakage between helper data and the secret of a PUF-based key generator, in case of i.i.d. but biased PUF response bits. The presented solutions are all scalable in that they can handle an arbitrary amount of PUF response bias, given that the available PUF response is large enough to provide a sufficient input bits. This solves an important open problem with existing PUF-based key generators, which was even hypothesized to be unsolvable in [9]. The introduced methods are all proven to be secure and are compared based on their reliability, efficiency, leakage and reusability. This comparison shows that depending on requirements for the key generator, different solutions are preferable. Additionally, this work provides a new model for entropy leakage due to PUF bias as well as a model for the relation between PUF bias and the bit error rate of the corresponding PUF. The first model is an improvement over existing models, e.g. as used in [9], while the second model is a new concept which is an extension on the models from [14].

2 PUF-based Key Generation and Bias

2.1 General Construction

Figure 1 shows a generic PUF-based key generator using the code-offset method from [4], with which many earlier proposed implementations (e.g. [1,5,10,17]) conceptually comply. $\mathsf{Encode}()$ and $\mathsf{Decode}()$ are the encoding and corresponding decoding function of an error-correcting code. In this work it is considered that the error-correcting code is a binary linear block code. $\mathsf{KDF}()$ is a key derivation function for generating a strong key from a random source with possibly reduced entropy. This can be a strong extractor [4] (for information-theoretic security), or a cryptographically secure key derivation function, see e.g. [11,12]. $\mathsf{KDF}()$ could also be applied on the PUF response X instead of on the random seed S. However, for analysis in this work, both variants are equivalent (see [16]). Key generators have the following two key properties:

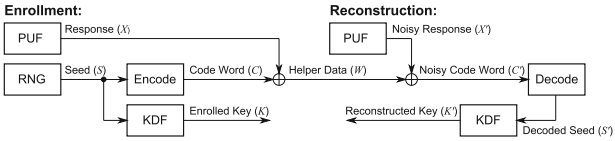


Fig. 1. PUF-based key generator based on the code-offset method.

1. *Reliability:* if the occurence of (bit) errors between X and X' is limited, then with high probability K' will be equal to K.
2. *Security:* if X is sufficiently unpredictable, then K is secure even to a party which observes W.

Reliability is accomplished by use of an error-correcting code able to cope with typically occuring amounts of bit errors. In a construction like Fig. 1 this results in a disclosure of information by W on X, since W is assumed to be public. However, the security property guarantees that *if there is sufficient entropy in* X, there will be enough left after this disclosure to derive a secure key. The security of the key is hence conditioned on the entropy of X. This aspect is the main subject of this work, where we will make this condition explicit and study its implications.

2.2 Entropy Leakage

Given that $\mathsf{KDF}()$ is a secure key derivation function, the security of the derived key K depends on the unpredictability of the input of $\mathsf{KDF}()$. In a construction as in Fig. 1, the input of $\mathsf{KDF}()$ is a seed S which is randomly generated during the one-time enrollment, and reconstructed from W and a noisy PUF response X' during later reconstructions. Since W is considered public, S needs to be sufficiently unpredictable even when conditioned on W. In terms of entropy,[1] this is expressed as follows:

$$H(S|W) = H(S) - I\,(S;W), \tag{1}$$

or the conditional entropy of S given W is the original entropy of S reduced with the *entropy leakage* of S by W, which is expressed by the mutual information between S and W.[2] For the remainder of this work it is assumed that S is a fully random bit string of length k, i.e. $H(S) = |S| = k$. For a key generator design as in Fig. 1, deploying a linear block code with generator matrix $\mathbf{G}$ (where all rows of $\mathbf{G}$ are independent) and parity-check matrix $\mathbf{H}$, the entropy leakage is:

$$I\,(S; X \oplus S\mathbf{G}) = k - H(X) + H(X\mathbf{H}^\top), \tag{2}$$

[1] In this work, unpredictability of random variables is expressed by Shannon entropy, as is done in many earlier work on this subject, e.g. [7]. Note that Shannon entropy serves as a lower bound for average guesswork [19]. For a stronger (less practical) provable security notion, the more pessimistic min-entropy measure should be used.

[2] Note that $H(X|W) = H(S|W)$, see [16]. This shows the equivalence in security (in terms of entropy) for a key generated from S or X.

(proof, see [16]). This expression of entropy leakage is a known fact about code-offset schemes,[3] but is not very well-established in the context of PUF-based key generators.[4] Combining (1) and (2) gives:

$$H(S|W) = H(X) - H(X\mathbf{H}^\top). \tag{3}$$

The remaining entropy of S after observing W is hence equal to the entropy of the PUF response X reduced with the entropy of the *syndrome* of X under the used linear error-correcting block code.

If X has full entropy then the entropy leakage as given by (2) becomes zero and $H(S|W) = k$. Hence, in that case S remains fully random and can be used as a secure input for key derivation. However, if X does *not* have full entropy, the entropy leakage might no longer be zero and S might not be completely unpredictable after observation of W. In the following, we study the effect of reduced entropy of X on the security of S, and in particular what occurs when X suffers from *bias*.

2.3 Entropy Leakage Due to PUF Bias

The most common cause of reduced entropy of X is the presence of *global bias* on the bits of X, i.e. globally '0'-bits occur consistently more often than '1'-bits or vice-versa. We say that an n-bit PUF response X is p-biased ($0 \leq p \leq 1$) if the a-priori expected number of '1'-bits in X is $p \cdot n$, or p is the a-priori probability of a random bit of a PUF response evaluating to '1'. An unbiased PUF has $p = 50\,\%$, but from experiments it is clear for most PUFs p deviates slightly from $50\,\%$, or even significantly (see e.g. [7,8,13]).

We now investigate entropy leakage on S when X has reduced entropy caused *only* by global bias. In that case $H(X) = nh(p)$ with $h()$ the binary entropy function. From (2) and (3) it is evident that the quantity $H(X\mathbf{H}^\top)$ plays a central role in the entropy leakage. In any case, it holds that $H(X\mathbf{H}^\top) \leq |X\mathbf{H}^\top| = (n-k)$, which results in the lower bound:

$$H(S|W) \geq k - n(1 - h(p)). \tag{4}$$

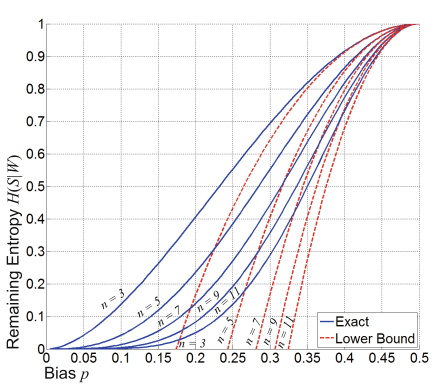
This is a known practical bound for constructing a secure PUF-based key generator (see e.g. [15]), but it needs to be stressed that this is a *lower bound* and hence any conclusions based on it could be overly pessimistic.[5]

We present two methods for calculating the entropy leakage exactly in the case of global bias for codes with certain properties:

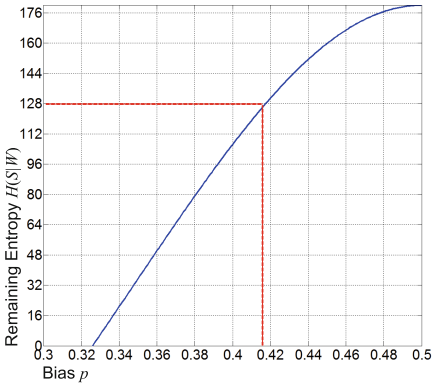
[3] E.g., a variant thereof appeared before in an early version of [21].

[4] This has led to some confusion and occasional misinterpretations, i.e. under- or overestimations of the leakage. A discussion on this is e.g. found in [3].

[5] Note that in particular for a too high bias this entropy bound even becomes negative, making it absolutely clear that this is a pessimistic lower bound.



(a) For repetition codes ($n = 3 \ldots 11$), calculated with the lower bound from (4) and exactly based on (5).



(b) For the key generator from [10], calculated with the lower bound formula following from plugging (6) into (3).

Fig. 2. $H(S|W)$ in case of a p-biased PUF response.

– For codes with a simple structure, a closed expression for $H(X\mathbf{H}^\top)$ can be derived. In particular for repetition codes it holds that:[6]

$$H(X\mathbf{H_{rep}}^\top) = -\sum_{t=0}^{n-1} \binom{n-1}{t} f(t;n,p) \log_2 f(t;n,p) \tag{5}$$

with $f(t;n,p) = p^t(1-p)^{n-t} + p^{n-t}(1-p)^t$.

– For non-trivial but relatively short codes (e.g. $n < 32$), the distribution of $X\mathbf{H}^\top$ can be determined exhaustively from the known distribution of X. $H(X\mathbf{H}^\top)$ then follows from the distribution of $X\mathbf{H}^\top$.

Figure 2(a) shows $H(S|W)$ for repetition codes with a p-biased PUF.[7] Both the lower bound (4) and the exact calculation (5) are shown. It is clear that for $p \not\approx 50\,\%$, the lower bound is not tight and significantly underestimates the remaining entropy; e.g. for $n = 5$ the lower bound reaches zero for $p = 24\,\%$, while exact calculation shows that about 0.35 bit of entropy is actually still left. The lower bound is hence rather pessimistic for biases not close to 50 % and a more exact calculation is preferable. This also partially refutes the so-called *"repetition code pitfall"* as stated in [9] which was solely based on the pessimistic conclusions from (4).

On the other hand, it is clear from Fig. 2(a) that a biased PUF response still severely reduces the remaining entropy of the seed S. This could be problematic, in particular for key generators deploying concatenated codes with a repetition code as the inner code, as shown next.

[6] See [16] for the derivation of this formula and similar for min-entropy in [3].
[7] Only $p \leq 0.5$ is shown; entropy-vs-bias graphs are symmetrical around $p = 0.5$.

2.4 Effect of PUF Bias on a PUF-based Key Generator

In [10], one of the most efficient key generators to date[8] was proposed using a concatenation of $r = 15\times$ a $(24, 12)$-Golay code word for the outer code and a $(8, 1)$-repetition code as inner code. The decoder consists of a hard-in-soft-out repetition decoder and a soft-in-hard-out Golay decoder. This construction extracts a 128-bit key with a failure rate $<10^{-6}$, from 2880 PUF response bits with average bit error rate $\leq 15\,\%$.[9] As a safety measure for reduced PUF entropy, the seed S from which the key is derived has a length of $15 \times 12 = 180$ bits.[10] We will study the effect of global PUF bias on this construction, and how much protection this implemented safety measure offers.

Since the concatenation of two linear block codes forms a new block code, all the entropy leakage results from Sects. 2.2 and 2.3 remain valid. Assuming the bits of X are i.i.d., then it holds for $r\times$ a generic (n_2, k_2)-block code concatenated with a $(n_1, 1)$-repetition code that:

$$H(X\mathbf{H}^\top) \leq r \cdot \Big(n_2 \cdot H(X_{1:n_1}\mathbf{H_{rep}}^\top) + H(X_{1:n_2}\mathbf{H_2}^\top)\Big), \tag{6}$$

with $X_{1:n_1}$ and $X_{1:n_2}$ vectors of n_1 and n_2 bits from X.[11] This is an upper bound ($\leq$) since the entropy contributions of both right terms in (6) could partially overlap,[12] but the same entropy cannot be leaked twice. We evaluate (6) exactly for the example key generator by calculating the repetition code entropy term using (5), and the Golay code term using the exhaustive method. The result is plugged into (3) and shown in Fig. 2(b).

From Fig. 2(b) it is clear that bias significantly affects the remaining seed entropy for this realistic key generator. For $p < 41.8\,\%$ ($p > 58.2\,\%$), the remaining entropy (lower bound) even falls below 128 bits, and the input of the key derivation function has potentially less than 128 bits of entropy. Hence one can no longer claim that the derived 128-bit key has full entropy.[13] Hence, the safety measure of using a 180-bit seed effectively keeps this key generator secure for PUFs with a bias $41.8\,\% \leq p \leq 58.2\,\%$.

In case of a seed of only 128 bits, the key's security would be reduced for any PUF with even the slightest bias. On the other hand, in order to cope with even more bias, overhead on the seed length will have to be increased even

[8] Efficient in terms of PUF size, while following the design of Fig. 1 and using only a single enrollment measurement per derived key.

[9] The key generator from [10] is based on an SRAM PUF, but in this work we make abstraction of the actual PUF used. Our analysis and solutions apply to all PUF types with i.i.d. response bits suffering from bias.

[10] [10] aims for a seed of 171 bits, but this is rounded up to 180 for practicality. The need for having 171-bit seeds originated in [5], but the reasoning is not fully clear.

[11] Since bits of X are assumed i.i.d., which particular bits from X are considered for the entropy calculation is of no importance.

[12] $X_{1:n_1}\mathbf{H_{rep}}^\top$ and $X_{1:n_2}\mathbf{H_2}^\top$ are not necessarily independent.

[13] Note that this does not directly imply that the key becomes predictable, just that it is potentially less unpredictable than it should be according to its length.

Table 1. Effect of scaling the seed length of the key generator from [10].

Code words r	Seed Length $\|S\|$	PUF Size $\|X\|$	Failure Rate	$H(S\|W) \geq 128$ for $p \in$
15	180	2880	$4.8 \cdot 10^{-7}$	$50\% \pm 8.2\%$
20	240	3840	$6.4 \cdot 10^{-7}$	$50\% \pm 10.8\%$
25	300	4800	$8.0 \cdot 10^{-7}$	$50\% \pm 12.1\%$
30	360	5760	$9.6 \cdot 10^{-7}$	$50\% \pm 13.0\%$

further. In Table 1 we scaled the seed length (and PUF size and failure rate) with the same code construction but increasing the number of Golay code words r. Unfortunately, the resistance to bias does not scale accordingly. The extra bias this generator can handle by increasing the seed gradually becomes very small. For the used code construction it cannot increase much beyond $50\,\% \pm 13.0\,\%$[14]. Also, the cost for achieving this (slightly) increased bias resistance is a *doubling* of the PUF size.

The conclusions about the studied key generator from [10], as summarized by Fig. 2(b) and Table 1, can be generalized to all key generators of the same design. The details differ slightly depending on the used codes, but the tendencies are always the same: global bias on PUFs relatively quickly reduces the remaining seed entropy, and increasing seed length has only a limited effect on the bias resistance and comes at a high cost in PUF size. This restricts the efficient use of key generators like Fig. 1 to PUFs with a limited global bias, roughly in the order $50\,\% \pm 10\,\%$. Since many experimentally studied PUF constructions have a global bias within this range, this is not necessarily problematic. However, for other PUFs with larger bias this key generator design cannot be used, and it was hypothesized (e.g. in [9]) that secure key derivation from such PUFs is impossible. In the following we will counter this by presenting a number of solutions that efficiently generate secure keys from PUFs with arbitrarily large global bias. These solutions are generic, so they could be used to deal with other (than global) types of bias with only minor modifications.

3 Debiasing Solutions

3.1 Basic Concept

In Sect. 2.4 we have shown that classic code-offset based key generators can only cope with a limited amount of PUF bias. In order to overcome this, we propose to extend the key generator design with a *debiasing step* prior to generating the code-offset helper data, as shown in Fig. 3. The idea is that the debiased PUF response Y which is actually enrolled is less biased than the original X, and hence the entropy leakage due to bias is reduced, or ideally zero. This seems a rather

[14] Note that we cannot increase beyond $r = 31$, without increasing the length of the repetition code, otherwise the failure rate gets too large.

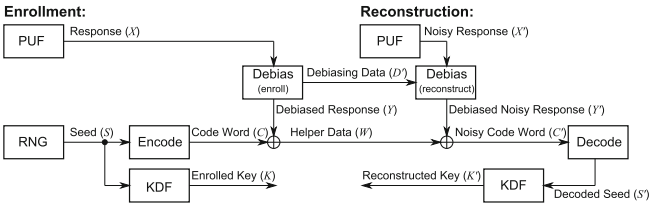


Fig. 3. *Debiasing* PUF-based key generator based on the code-offset method.

straightforward extension of the classic key generator design from Fig. 1, but there are a number of important and non-trivial points to consider when doing this:

Reliability. A debiasing step should not compromise the reliability of the key generator. This requirement excludes many potential debiasing options that blow up the bit error rate of the enrolled bit string. On the other hand, certain debiasing solutions allow for an intelligent combination of debiasing and error-correction, as shown next.

Efficieny. A debiasing step typically compresses or discards part of the PUF response, hence introducing a *debiasing overhead*. It will become clear that basic debiasing methods have a rather high overhead. We propose innovative optimizations specifically tailored for PUF-based key generation which significantly reduce overhead.

Leakage. Debiasing typically also produces side information during enrollment which is required during reconstruction. This *debiasing data* D hence needs to be stored/transfered publicly with the code-offset helper data W, potentially introducing new entropy leakage. For each presented debiasing method, we will prove that the combined entropy leakage of the debiasing data D and helper data W is zero.

Reusability. The classic code-offset scheme from Fig. 1 is *reusable* as shown in [2]. This means that the same PUF can be enrolled many times, each time producing a different key K and helper data W, without leaking more entropy than under just one single enrollment. This property does not necessarily hold when a debiasing step is used. We will investigate the reusability of each proposed debiasing method.

We will now introduce and discuss a number of increasingly more sophisticated debiasing methods and investigate these properties.

3.2 CVN: Classic von Neumann (VN) Debiasing

The classic randomness extractor as proposed by von Neumann [20] considers consecutive pairs of bits. If both bits are equal they are discarded, if they are opposed then the first bit is retained as a debiased bit. It is well known that if the input bits are i.i.d. but globally biased, then the output bits are perfectly random. Figure 4 shows how the *classic von Neumann* (CVN) extractor can be

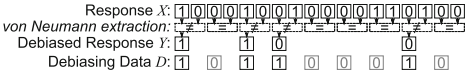


Fig. 4. Debiasing with a classic von Neumann extractor (enrollment only).

used as a debiasing step in a key generator. During enrollment (shown), the output of the CVN extractor becomes the debiased response Y which is used to calculate the code-offset helper data on. Also, for each considered bit pair of X a selection bit is used to show whether (1) or not (0) the first bit of that pair was retained in Y. These selection bits are the *debiasing data* D which is transfered alongside the helper data W. During reconstruction, the retained bits Y' from the noisy PUF reponse X' are selected with the bits in D.

Reliability. A nice feature of debiasing methods like CVN is that they hardly affect the error rate of the PUF response bits, contrary to other methods such as hash functions or XOR-combiners.[15] This is *the* main motivation to use a von Neumann-like extractor as a debiasing step in a PUF-based key generator. Therefore, all following proposed debiasing solutions are variants of this classic von Neumann debiasing.

Efficiency. If X is a p-biased n-bit PUF response, then the number of unbiased bits retained by CVN is binomially distributed with parameters $(\lfloor \frac{n}{2} \rfloor, 2p(1-p))$. The debiasing overhead of CVN is hence very high; even when the input X is already unbiased, CVN will still discard on average $3/4$ of the bits. However, in practical situations this ratio is even lower; to obtain $|Y|$ unbiased bits with a maximum failure rate p_{fail}, n needs to be large enough to meet:

$$F^{-1}_{\text{bino}}\big(p_{\text{fail}}; \lfloor \tfrac{n}{2} \rfloor, 2p(1-p)\big) \geq |Y|. \tag{7}$$

E.g. for $|Y| = 1000$, $p = 50\,\%$ and $p_{\text{fail}} = 10^{-6}$, this yields $n \geq 4446$, or a debiasing overhead factor of about 4.4. If X is actually biased this even becomes worse; e.g. for $p = 30\,\%$, $n \geq 5334$ or a factor of 5.3.

Leakage. Due to the properties of CVN, Y will be perfectly random regardless of the bias on X, and hence according to (2) we have $I(S; W) = 0$. However, since CVN debiasing also produces public information D, we need to consider $I(S; (W, D))$ instead. It can be shown that $I(S; (W, D)) = 0$ for CVN (proof, see full version [16]), hence the combination of (W, D) leaks no information on the seed S.

Reusability. A key generator using CVN for debiasing is *not* reusable. An example of the insecurity from enrolling the same PUF more than once is shown in Fig. 5. Here one learns from the helper data $W^{(1)}$ of the first enrollment that the first and fifth bit of $X^{(1)}$ are equal since they are both XOR-ed with the same 2-bit repetition code word. In the second enrollment, $X^{(2)}$ a noisy version of $X^{(1)}$ is enrolled, with differing bits marked in black.

[15] Von Neumann extractors have a small effect on bit error rate, shown in Sect. 4.1.

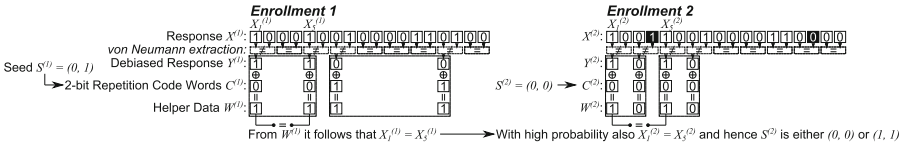


Fig. 5. Insecurity of CVN in case of reuse.

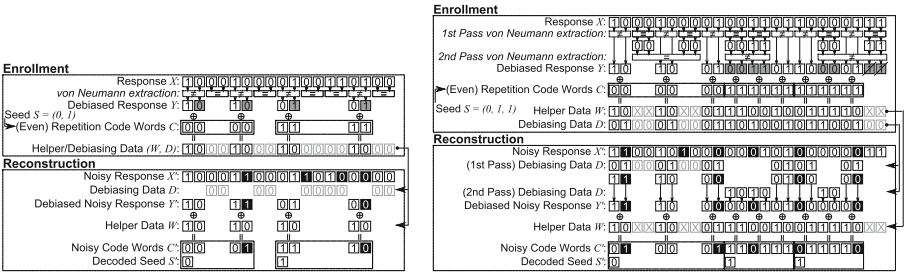
Because of these few differing bits, the first and fifth bit of $X^{(2)}$ are now enrolled in two different code words. However, from the first enrollment one has learned that these two bits are equal. From this knowledge, one can deduce that the two code words in the second enrollment, and their corresponding seed bits, are also equal (with high probability). The 2-bit seed $S^{(2)}$ hence has only two possible values instead of four. In general, one can say that $I\left(S^{(2)}; (W^{(1)}, D^{(1)}, W^{(2)}, D^{(2)})\right) > I\left(S; (W, D)\right) = 0$ and hence there is (more) leakage when the PUF is enrolled more than once. The reason for this reuse insecurity is the stochastic nature of the CVN step which is caused by random bit errors in between enrollments. Due to these differing bits, enrolled bits can shift between code words in between enrollments, which causes this particular type of leakage.

Summarizing, by using CVN as a debiasing step in a design like Fig. 3, one can build a PUF-based key generator which leaks no information on the secret seed S even if X is biased. Note that CVN poses no limit on the amount of bias on X that can be tolerated; in theory X can have an arbitrarily high bias, the leakage will always be zero. However, the efficiency restriction as expressed by (7) will pose a limitation in practice, since the PUF size n cannot become arbitrarily large. Nonetheless, this is still a major advancement in PUF-based key generators, since it shows that a secure key can be derived from a PUF with arbitrary bias, whereas the classic design of Fig. 1 was limited to PUFs with a bias in the range $50\,\% \pm 10\,\%$ as discussed in Sect. 2.4. The cost paid for this advancement is an increase in the PUF's size (overhead factor > 4) and the loss of the reusability property. In the following we will address these issues, first by proposing overhead optimizations in Sect. 3.3, and next by proposing a debiasing solution which retains the reusability property in Sect. 3.4.

3.3 Pair-Output (2O-VN) and Multi-Pass Tuple-Output VN Debiasing (MP-TO-VN)

The first proposed optimization (shown in Fig. 6(a)) consists of two minor modifications to the CVN debiasing solution of Sect. 3.2:

1. Instead of using only the first bit of each selected pair as in CVN, both bits of a selected pair are retained, hence the name *pair-output von Neumann* or 2O-VN debiasing.
2. The most inner code in the code-offset scheme is an even-length repetition code (e.g. 4 bits in Fig. 6(a)).



(a) Debiasing with a pair-output von Neumann extractor.

(b) Debiasing with a multi(2)-pass tuple-output von Neumann extractor.

Fig. 6. Key generators with improved efficiency VN debiasing. Extra retained bits are marked in grey. Bit errors during reconstruction are marked in black.

The second modification ensures that each pair retained by the first modification is used within the same repetition code word. This is an important condition for the security of this construction. In Fig. 6(a) the helper data W and the debiasing data D are combined in a single bit string (W, D), where a value 00 signifies that a pair is not retained, whereas a non-zero value (01 or 10) is the code-offset helper data of a retained pair.[16]

Reliability. The reliability analysis of 2O-VN is comparable to CVN and is explained in more detail in Sect. 4.1.

Efficiency. It is clear that the debiasing overhead of 2O-VN is about half that of CVN. The constraint on n now becomes:

$$F_{\text{bino}}^{-1}\big(p_{\text{fail}}; \lfloor \tfrac{n}{2} \rfloor, 2p(1-p)\big) \geq \tfrac{|Y|}{2}. \tag{8}$$

E.g. for $|Y| = 1000$, $p = 50\,\%$ and $p_{\text{fail}} = 10^{-6}$, this yields $n \geq 2322$, or a debiasing overhead factor of about 2.3, and for $p = 30\,\%$, $n \geq 2794$ or an overhead factor of 2.8.

Leakage. It can be shown that $I\,(S; (W, D)) = 0$ still holds for 2O-VN debiasing.This appears counterintuitive since Y is no longer i.i.d. but contains explicit dependencies, i.e. for each pair $(Y_{2i-1}, Y_{2i})_{i>0}$ it holds that $Y_{2i-1} = \overline{Y_{2i}}$ or the parity is odd. However, note that the code-offset helper data of a repetition code anyway discloses the parities of all bit pairs of Y used within the same code word. Hence if a bit pair is used within the same repetition code word, as guaranteed by the 2O-VN modifications, it is no problem that its parity is known since it would have been disclosed anyway. More intuitively, one can see (e.g. in Fig. 6(a)) that the known dependencies in Y do not help an outsider in predicting anything about S from (W, D).

Reusability. 2O-VN debiasing is not reusable for the same reasons as CVN debiasing (see Sect. 3.2).

[16] This is just one possible exemplary representation of (W, D).

Extension to Multi-pass Tuple Output (MP-TO-VN). The efficiency of 2O-VN can be further improved by reconsidering the discarded bits in a second pass, shown in Fig. 6(b). Bits are now grouped as quadruplets, and the extractor compares the first half of a quadruplet to the second half. More passes are possible (not shown in Fig. 6(b)) where in general the i-th pass considers tuples of 2^i bits which were not retained by any of the previous passes. Hence the name *multi-pass tuple-output von Neumann* or MP-TO-VN. For (M=2)P-TO-VN, the constraint on n becomes:

$$\sum_{a=0}^{|Y|-1} \sum_{b=0}^{\lfloor \frac{n}{4} \rfloor} f_{\text{bino}}\left(\tfrac{a-4b}{2}; \lfloor \tfrac{n}{2} \rfloor, 2p(1-p)\right) \cdot f_{\text{bino}}\left(b; \lfloor \tfrac{n-(a-4b)}{4} \rfloor, \tfrac{2p^2(1-p)^2}{(p^2+(1-p)^2)^2}\right) < p_{\text{fail}} \qquad (9)$$

E.g. for $|Y| = 1000$, $p = 50\,\%$ and $p_{\text{fail}} = 10^{-6}$, this yields $n \geq 1538$, or an overhead factor of 1.5, and for $p = 30\,\%$, $n \geq 2068$ or a factor of 2.1. For more passes similar constraints can be derived and the overhead will reduce even further. However, the extra reduction for each additional pass quickly becomes small and almost negligible for more than three passes.

For MP-TO-VN to be leakage-free, it needs to be ensured that bits which were retained as a 2^i-bit tuple are always used within the bounds of a single repetition code word for calculating the helper data. This could entail that retained tuples need to be reshuffled or that a retained tuple is cropped when a code word bound is reached; e.g. in Fig. 6(b) the final two bits are discarded, they cannot be used for the next code word. Depending on the method used, it could be needed to keep track in which pass a certain bit pair was retained. In Fig. 6(b), this is done by letting D have three possible values for each bit pair, i.e. 'not retained' (00), 'retained in first pass' (01) or 'retained in second pass' (10). The leakage and reusability analysis for MP-TO-VN are the same as for 2O-VN.

3.4 ϵ-2O-VN: Pair-Output VN Debiasing with Erasures

Now we propose (Fig. 7) a modification of the 2O-VN debiasing solution from Sect. 3.3 which makes the key generator reusable again:

1. During enrollment 2O-VN debiasing is applied, yet bit pairs which are not retained by 2O-VN are not completely discarded but replaced with *erasure symbols* (ϵ), hence the name *pair-output von Neumann debiasing with erasures* or ϵ-2O-VN. The length of Y (i.e. the number of symbols) is hence equal to the length of X.
2. The most inner code is again an even-length repetition code (e.g. 6 bits in Fig. 7). The code-offset helper data between the code words and Y is calculated with the ϵ-XOR operation denoted as $\oplus\!\!\!\!\bigcirc$, which is defined as the regular XOR ($\oplus$) if both operands are regular bits, but produces an ϵ if one or both of its operands is an ϵ.
3. During reconstruction, the noisy code words will also contain ϵ symbols. These are to be treated as regular bit erasures by the decoder, which hence needs to be able to handle both errors and erasures. Figure 7 uses a concatenated code of an inner 6-bit repetition code and an (exemplary) $(4, 2)$ outer code.

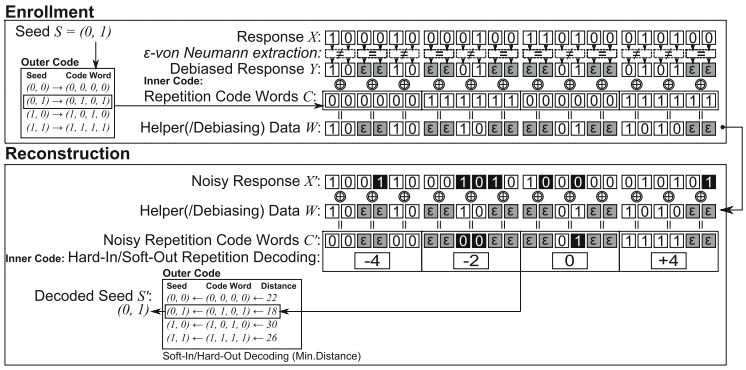


Fig. 7. Debiasing with a pair-output von Neumann extractor with erasures. Bit pairs discarded by 2O-VN are now retained as erasures (ϵ, marked in grey). Bit errors during reconstruction (w.r.t. enrollment) are marked in black.

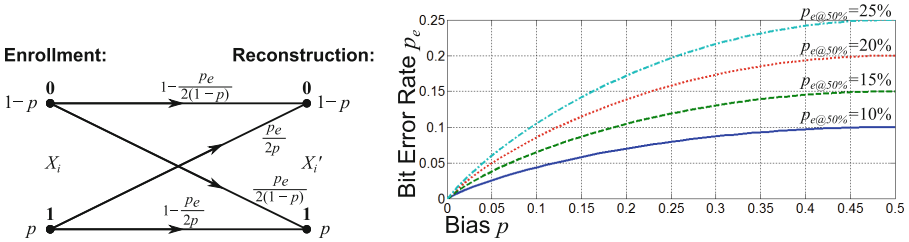
The repetition code is decoded with a hard-in-soft-out decoder which treats erasures as non-preferential code bits. The outer code is decoded with a soft-in-hard-out decoder (i.c. a trivial minimum-distance list decoder) to retrieve the seed S.

Reliability. It is evident that ϵ-2O-VN debiasing impacts the reliability of a key generator, since the used error-correcting code needs to be able to deal with bit errors caused by noise, as well as with erasures caused by bias. For an n-bit p-biased PUF response X, the probability of having $\frac{e}{2}$ erasures is binomially distributed with parameters $(\lfloor \frac{n}{2} \rfloor, p^2 + (1-p)^2)$. In Sect. 4.2, it is demonstrated how this affects the reliability/efficiency of a realistic key generator.

Efficiency. ϵ-2O-VN is very efficient in terms of debiasing overhead factor $\frac{|X|}{|Y|}$, since $|X| = |Y|$. However, the cost for ϵ-2O-VN sits in the fact that a more powerful error-correcting code (hence with a smaller code rate) needs to be used to account for the introduced erasures. For ϵ-2O-VN debiasing, reliability and efficiency need to be considered together, as demonstrated in Sect. 4.2.

Leakage. The ϵ-2O-VN debiasing method does not produce any explicit debiasing data D. All required information is contained in W which uses the symbols 0, 1 and ϵ. For leakage, we need to consider $I(S;W)$ again, but now in the new setting of ϵ-2O-VN. It can be shown that *classic* von Neumann debiasing with erasures is leakage-free (proof, see full version [16]). From this, the security of *pair-output* von Neumann debiasing with erasures follows in the same manner as for 2O-VN due to the fact that the most inner code is a repetition code.

Reusability. ϵ-2O-VN debiasing *is* reusable, i.e. $I\left(S^{(i)}; (W^{(1)}, W^{(2)}, \ldots)\right) = I(S;W) = 0$ (proof, see full version [16]). This means that the same PUF can be enrolled an arbitrary number of times without the combination of



(a) Channel model for a PUF with bias p and bit error rate p_e.

(b) Plot of $p_e = f_{\text{bias}}(p; p_{e@50\%})$

Fig. 8. Relations between PUF bias and error rate.

all produced helper data strings leaking anything about any of the enrolled seeds. A key generator with ϵ-2O-VN debiasing (re)gains this property since the debiasing is no longer stochastic: randomly differing bits between enrollments do no longer affect the selection of bit pairs since all bit pair positions are always retained. Unfortunately, ϵ-2O-VN cannot be extended to multiple passes in the same manner as MP-TO-VN without compromising the reusability property.

4 Objective Comparison of Debiasing Solutions

4.1 Relation Between PUF Bias and Bit Error Rate

Similarly to global bias, we define the *global bit error rate* to be p_e if the a-priori expected number of differing bits between two evaluations X and X' of the same n-bit PUF response is $p_e \cdot n$, or put otherwise p_e is the a-priori probability that $X_i \neq X'_i$ for a random bit i of a PUF.

Firstly, note that a biased PUF will have a different bit error rate for 0 and 1 bits; e.g. for a PUF which is biased towards 0, the probability of a bit error will be higher for a 1-bit than for a 0-bit. Such behavior is typically expressed as a channel model, shown in Fig. 8(a) for our situation. The assumptions are that the bias of X and X' are the same, even with bit errors, and that the *average* bit error rate is equal to p_e. Note that von Neumann-based debiasing methods will (negatively) affect this average bit error rate since the ratio of 0 and 1 bits which are retained will be changed, favoring the lesser occuring but more error-prone kind.

Secondly, one notices that the heavier the bias p of a PUF response, the smaller its bit error rate p_e; e.g. in the extreme case of a $p = 100\,\%$ biased PUF, the bit error rate p_e will be zero. To objectively compare PUFs with different bias levels, we need to make this relation between biases and error rates explicit; e.g. there is a big difference between a PUF with $p_e = 15\,\%$ but no bias $p = 50\,\%$ and another PUF with $p_e = 15\,\%$ and a heavy bias of $p = 30\,\%$. For objective comparison, we introduce the *fixed point* $p_{e@50\%}$ which is the (hypothetical) bit

Table 2. Comparison of debiasing solutions for the code-offset key generator from [10]. The *fixed point* bit error rate is set to $p_{e@50\%} = 15\,\%$. The key generator is failure rate $< 10^{-6}$.

Solution	Reusable?	Bias p	Rescaled Bit Error Rate p_e	Debias Overhead $\frac{\lvert X\rvert}{\lvert Y\rvert}$	Nr. of Golay Code Words r	Repetition Code Length n_{rep}	PUF Size $\lvert X\rvert$	Seed Length $\lvert S\rvert$	Seed Entropy $H(S\lvert(W,D))$	p_{fail} (Enroll)	p_{fail} (Reconstruct)	PUF Size Overhead w.r.t. *
No Debias	Yes	50.0%	15.0%	1.00	11	8	2112	132	132	0	5.61E-7	0.73
*No Debias**	Yes	41.8%	14.7%	1.00	15	8	2880	180	128	0	5.00E-7	1.00
No Debias	Yes	40.0%	14.5%	1.00	19	8	3648	228	135	0	2.74E-7	1.27
No Debias	Yes	37.0%	14.2%	1.00	30	8	5760	360	128	0	3.50E-7	2.00
2O-VN	No	40.0%	14.5%	2.31	11	8	4888	132	132	9.14E-7	9.86E-7	1.70
2O-VN	No	35.0%	13.9%	2.45	11	8	5168	132	132	9.42E-7	6.03E-7	1.79
2O-VN	No	30.0%	13.0%	2.66	11	8	5616	132	132	9.63E-7	9.57E-7	1.95
2O-VN	No	25.0%	11.9%	2.99	11	8	6314	132	132	9.90E-7	4.94E-7	2.19
2P-TO-VN	No	40.0%	14.5%	1.58	11	8	3334	132	132	9.98E-7	6.31E-7	1.16
2P-TO-VN	No	35.0%	13.9%	1.73	11	8	3650	132	132	9.44E-7	7.69E-7	1.27
2P-TO-VN	No	30.0%	13.0%	1.96	11	8	4142	132	132	9.53E-7	8.67E-7	1.44
2P-TO-VN	No	25.0%	11.9%	2.32	11	8	4890	132	132	9.92E-7	9.63E-7	1.70
ϵ-2O-VN	Yes	40.0%	14.5%	1.00	11	20	5280	132	132	0	8.67E-7	1.83
ϵ-2O-VN	Yes	35.0%	13.9%	1.00	11	22	5808	132	132	0	9.87E-7	2.02
ϵ-2O-VN	Yes	30.0%	13.0%	1.00	11	26	6864	132	132	0	6.12E-7	2.38
ϵ-2O-VN	Yes	25.0%	11.9%	1.00	11	28	7392	132	132	0	6.12E-7	2.57

error rate a PUF would have if it would have been unbiased ($p = 50\,\%$). For a PUF with a given $p_{e@50\%}$, a relation $p_e = f_{\text{bias}}(p; p_{e@50\%})$ can be derived based on the reliability model for PUFs from [14] (see derivation in full version [16]). This function is shown for different values of $p_{e@50\%}$ in Fig. 8(b). This graph should be interpreted as follows: if one wants to objectively compare the efficiency of a key generator for an unbiased PUF, e.g. with $p_e = 15\,\%$, to a key generator for a biased PUF, e.g. with $p = 30\,\%$, then the corresponding error rate for the biased PUF should be set to $f_{\text{bias}}(30\,\%; 15\,\%) = 13.0\,\%$.

4.2 Comparison of Debiasing Solutions

The different debiasing solutions proposed in Sect. 3 are evaluated and compared to each other in an objective manner. The evaluation is done for the key generator from [10], which uses a repetition code-Golay code concatenation. The results are shown in Table 2. Three different debiasing solutions are compared amongst

each other and against the case when no debiasing is used (see also Table 1)[17]. The row marked with a * is the best proposal from [10] and is used as reference case. To make the comparison objective, the effective bit error rate p_e for each simulation scales with the bias level according to $f_{\text{bias}}(p; p_{e@50\%})$ as shown in Fig. 8(b) with $p_{e@50\%} = 15\,\%$ corresponding to the error rate assumed in [10] and other works. For realistic simulations, the channel model from Fig. 8(a) is used, so 0-bits and 1-bits have different error probabilities in case of bias.

The three debiasing methods have different properties. 2O-VN and 2P-TO-VN have a non-zero enrollment failure rate; it is possible that insufficient bits are retained after debiasing for successful enrollment. Also, for both of these systems the error-correcting code used does not change depending on the amount of bias (only the size of the PUF changes for maintaining reliability), while for ϵ-2O-VN the repetition length scales with the amount of bias and enrollment is always successful. The PUF size overhead in comparison to * also varies between the mehods. It is clear that the 2P-TO-VN is the most efficient method considering this parameter (only 1.70 times the amount of * is required to deal with the extreme case of 25 % bias), but this method does not allow reuse for enrolling multiple keys. If this property is required ϵ-2O-VN provides a strong alternative, which requires more PUF data (2.57× * at 25 % bias), but provides a combination of properties that was not known to date.

5 Conclusion

This work solves the open problem of secure key generation from *biased* PUFs using code-offset-based constructions. This is accomplished without compromising the secret key's security and for arbitrary bias levels. Existing conventional methods will lead to leakage on the secret key when the PUF is too biased, whereas our proposed debiasing techniques prevent this leakage, while maintaining the high reliability and for some solutions even the reusability of the key generator. This comes at a cost of PUF size overhead, but using innovative approaches we were able to limit this overhead and design a key generator based on the requirements at hand.

Remaining open questions and interesting future research directions include: how to further optimize the efficiency of these debiasing solutions, and how to prevent key leakage for PUFs which suffer from reduced entropy for reasons other than bias, e.g. because of bit correlations.

[17] Failure rates differ slightly from the results in Table 1 which were extrapolated from [10]. For objective comparison, the results of Table 2 are based on a new simulations, with the Hackett Golay decoder from [10] implemented in Matlab. The single Golay decoding failure rate $p_{\text{Golay-fail}}$ is estimated as the 95 %-confidence upper bound from the simulations; the actual values for $p_{\text{Golay-fail}}$ are hence likely smaller. The total reconstruction failure rate is computed as $1 - (1 - p_{\text{Golay-fail}})^r$.

References

1. Bösch, C., Guajardo, J., Sadeghi, A.-R., Shokrollahi, J., Tuyls, P.: Efficient helper data key extractor on FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 181–197. Springer, Heidelberg (2008)
2. Boyen, X.: Reusable cryptographic fuzzy extractors. In: ACM Conference on Computer and Communications Security–CCS 2004, pp. 82–91. ACM Press, New York (2004)
3. Delvaux, J., Gu, D., Schellekens, D., Verbauwhede, I.: Helper data algorithms for PUF-based key generation: overview and analysis. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **34**(1), 14 (2014)
4. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. SIAM J. Comput. **38**(1), 97–139 (2008)
5. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
6. Ignatenko, T., Willems, F.: Information leakage in fuzzy commitment schemes. IEEE Trans. Inf. Forensics Secur. **5**(2), 337–348 (2010)
7. Katzenbeisser, S., Kocabaş, U., Rožić, V., Sadeghi, A.-R., Verbauwhede, I., Wachsmann, C.: PUFs: myth, fact or busted? a security evaluation of physically unclonable functions (PUFs) cast in Silicon. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 283–301. Springer, Heidelberg (2012)
8. Koeberl, P., Li, J., Maes, R., Rajan, A., Vishik, C., Wójcik, M.: Evaluation of a PUF device authentication scheme on a discrete 0.13um SRAM. In: Chen, L., Yung, M., Zhu, L. (eds.) INTRUST 2011. LNCS, vol. 7222, pp. 271–288. Springer, Heidelberg (2012)
9. Koeberl, P., Li, J., Rajan, A., Wu, W.: Entropy loss in PUF-based key generation schemes: the repetition code pitfall. In: IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 44–49 (2014)
10. van der Leest, V., Preneel, B., van der Sluis, E.: Soft decision error correction for compact memory-based PUFs using a single enrollment. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 268–282. Springer, Heidelberg (2012)
11. Lily, C.: NIST Special Publication 800–108: Recommendation for Key Derivation Using Pseudorandom Functions (revised) (2009)
12. Lily, C.: NIST Special Publication 800–56C: Recommendation for Key Derivation through Extraction-then-Expansion (2011)
13. Lim, D., Lee, J., Gassend, B., Suh, G., van Dijk, M., Devadas, S.: Extracting secret keys from integrated circuits. IEEE Trans. Very Large Scale Integr. VLSI Syst. **13**(10), 1200–1205 (2005)
14. Maes, R.: An accurate probabilistic reliability model for silicon PUFs. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 73–89. Springer, Heidelberg (2013)
15. Maes, R.: Physically Unclonable Functions - Constructions, Properties and Applications. Springer, Heidelberg (2013)
16. Maes, R., van der Leest, V., van der Sluis, E., Willems, F.: Secure key generation from biased PUFs. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. xx–yy, Cryptology ePrint Archive, Report 2015/831, this is the full version of this work (including all appendices). Springer, Heidelberg (2015). http://eprint.iacr.org/

17. Maes, R., Tuyls, P., Verbauwhede, I.: Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 332–347. Springer, Heidelberg (2009)
18. Maes, R., van Herrewege, A., Verbauwhede, I.: PUFKY: a fully functional PUF-based cryptographic key generator. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 302–319. Springer, Heidelberg (2012)
19. Massey, J.L.: Guessing and entropy. In: IEEE International Symposium on Information Theory (ISIT), p. 204 (1994)
20. von Neumann, J.: Various techniques used in connection with random digits. In: Applied Math Series 12. National Bureau of Standards, USA (1951)
21. Skoric, B., de Vreede, N.: The spammed code offset method. Cryptology ePrint Archive, Report 2013/527 (2013). http://eprint.iacr.org/
22. Yu, M.-D.M., M'Raihi, D., Sowell, R., Devadas, S.: Lightweight and secure PUF key storage using limits of machine learning. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 358–373. Springer, Heidelberg (2011)