Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis

Jeroen Delvaux, Dawu Gu, Dries Schellekens, and Ingrid Verbauwhede, *Fellow, IEEE*

***Abstract*—Security-critical products rely on the secrecy and integrity of their cryptographic keys. This is challenging for low-cost resource-constrained embedded devices, with an attacker having physical access to the integrated circuit (IC). Physically, unclonable functions are an emerging technology in this market. They extract bits from unavoidable IC manufacturing variations, remarkably analogous to unique human fingerprints. However, post-processing by helper data algorithms (HDAs) is indispensable to meet the stringent key requirements: reproducibility, high-entropy, and control. The novelty of this paper is threefold. We are the first to provide an in-depth and comprehensive literature overview on HDAs. Second, our analysis does expose new threats regarding helper data leakage and manipulation. Third, we identify several hiatuses/open problems in existing literature.**

***Index Terms*—Helper data algorithm (HDA), key generation, physically unclonable function (PUF).**

I. Introduction

MAINTAINING the secrecy and integrity of cryptographic keys at a low manufacturing cost is challenging, especially for resource-constrained embedded devices, with an attacker having physical access to the integrated circuit (IC). Traditionally, keys are stored in nonvolatile memory (NVM), encapsulated in the IC. Mature technologies are the following: EEPROM and its successor Flash, battery-backed SRAM, and fuses. However, EEPROM/Flash requires floating-gate transistors, increasing the manufacturing cost with respect to a regular CMOS process. Batteries are costly as well. Furthermore, NVM technologies tend to be vulnerable to physical attacks due to their permanent robust electrical nature. Finally, additional circuitry to protect against physical attacks is rather expensive. Consider protective coatings or sensors to detect invasion.

Physically unclonable functions (PUFs) are a promising alternative for secure low-cost key generation. Essentially, they are binary functions of which the behavior depends on IC manufacturing variations. Therefore, they are rather analogous to unique human biometrics. Most PUFs are compatible with a regular CMOS process, benefiting manufacturing cost. Furthermore, the secret is hidden in the physical structure of an IC, which is a much less readable format. Also, invasive attacks might damage the PUF and destroy the secret. Unfortunately, output bits cannot be used directly as a key. Post-processing by helper data algorithms (HDAs) is required to meet the stringent key requirements: reproducibility, high-entropy, and control.

A. Contribution

The novelty of this paper is threefold.

1) We are the first to provide an in-depth overview on HDAs, comprehending a decade of research. Various schemes for entropy compression, error-correction, bit selection, and manipulation detection are reviewed. This significantly supersedes the glance in [44]. PUFs are regarded as a black-box, making this paper generic.
2) Our analysis reveals new threats regarding helper data leakage and manipulation, most importantly the following. We derive an exact formula for the leakage of repetition codes in case of bias, demonstrating the well-known $(n-k)$ upper bound to be overly pessimistic. We demonstrate that the leakage of soft-decision coding has been underestimated. We describe divide-and-conquer manipulation attacks for parallel codes, concatenated codes, and soft-decision codes. We disprove the intuitive assumption that bit selection schemes have no leakage.
3) We identify hiatuses/open problems in existing literature, offering a foundation for future work.

B. Overview

The remainder of this paper is organized as follows. Section II provides preliminaries. Section III describes an accurate PUF reliability model, serving as a reference. Section IV provides a high-level framework for PUF HDAs. Entropy compression, error correction, bit selection, and manipulation detection are discussed in Sections V–VIII, respectively. Section IX identifies hiatuses/open problems in existing literature. Section X concludes this paper.

Manuscript received June 30, 2014; accepted October 22, 2014. Date of publication November 24, 2014; date of current version May 20, 2015. This work was supported in part by the European Commission through the ICT Programme under Contract FP7-ICT-2011-317930 HINT, in part by the Research Council of KU Leuven: GOA TENSE under Grant GOA/11/007, in part by the Flemish Government under Grant FWO G.0550.12N, in part by the Hercules Foundation under Grant AKUL/11/19, and in part by the National Major Development Program for Fundamental Research of China (973 Plan) under Grant 2013CB338004. The work of J. Delvaux is supported by IWT-Flanders under Grant SBO 121552. This paper was recommended by Associate Editor K. Mai.

J. Delvaux is with the Department of Electrical Engineering/Computer Security and Industrial Cryptography (COSIC), KU Leuven, Leuven B-3001, Belgium, also with iMinds, Brussels, Belgium, and also with the Department of Computer Science and Engineering, Lab of Cryptology and Computer Security (LoCCS), Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: jeroen.delvaux@esat.kuleuven.be).

D. Gu is with the Department of Computer Science and Engineering, LoCCS, Shanghai Jiao Tong University, Shanghai 200240, China.

D. Schellekens is with Septentrio Satellite Navigation, Leuven B-3001, Belgium.

I. Verbauwhede is with the Department of Electrical Engineering/Computer Security and Industrial Cryptography, KU Leuven, Leuven B-3001, Belgium, and also with iMinds, Brussels, Belgium.

Digital Object Identifier 10.1109/TCAD.2014.2370531

II. Preliminaries

A. Notation

Vectors are denoted with a bold lowercase character, e.g., **c**. All vectors are row vectors. Matrices are denoted with a bold uppercase character, e.g., **H**. Functions are printed in *italic*, e.g., Hamming weight $HW(\mathbf{e})$. Random variables are denoted with an uppercase character, e.g., $\mathcal{X}$.

B. Probability Distributions

Random variables are described by their probability density function (PDF) or cumulative distribution function (CDF), denoted by f and F, respectively. Equation (1) considers a normal distribution with mean zero and standard deviation $\sigma = 1$. Equation (2) considers a binomial distribution with k successes for n Bernoulli trials, each having success probability p

$$F_{\mathcal{N}}\left(x'\right) = \int_{-\infty}^{x'} f_{\mathcal{N}}\left(x\right)\ \mathrm{d}x = \int_{-\infty}^{x'} \frac{\exp\left(-\frac{x^2}{2}\right)}{\sqrt{2\pi}}\ \mathrm{d}x \tag{1}$$

$$F_{\mathcal{B}}\left(k'; n, p\right) = \sum_{k=0}^{k'} f_{\mathcal{B}}\left(k; n, p\right) = \sum_{k=0}^{k'} \binom{n}{k} p^k (1-p)^{n-k}. \tag{2}$$

C. Entropy

Consider a secret binary variable $\mathcal{X}$, which is either a bit or a bit vector. We define its entropy using logarithms to the base 2, as it is the most natural choice. Equation (3) represents the Shannon entropy and quantifies the average-case resistance against a brute-force attack. However, we will focus on the more conventional notion of min-entropy, as given by (4), corresponding to the worst-case resistance

$$H(\mathcal{X}) = -E_{\mathcal{X}}\left[\log_2(P\left(\mathcal{X}\right))\right] \tag{3}$$

$$H_{\infty}(\mathcal{X}) = -\log_2\left(\max_{\mathcal{X}}(P\left(\mathcal{X}\right))\right). \tag{4}$$

Assume now that an attacker obtains additional information about $\mathcal{X}$, as described by the variable $\mathcal{Y}$. We, therefore, define the conditional entropy of $\mathcal{X}$ given $\mathcal{Y}$. An attacker has typically no control over $\mathcal{Y}$, so we use the average-case resistance here. Equations (5) and (6) correspond with the Shannon and min-entropy, respectively. We will focus again on the latter

$$\widetilde{H}(\mathcal{X}|\mathcal{Y}) = -E_{\mathcal{Y}}\left[E_{\mathcal{X}}\left[\log_2\left(P\left(\mathcal{X}|\mathcal{Y}\right)\right)\right]\right] \tag{5}$$

$$\widetilde{H}_{\infty}(\mathcal{X}|\mathcal{Y}) = -\log_2\left(E_{\mathcal{Y}}\left[\max_{\mathcal{X}} P\left(\mathcal{X}|\mathcal{Y}\right)\right]\right). \tag{6}$$

D. PUFs: Black-Box Description

The binary input and output of a PUF are referred to as challenge **c** and response **r**, respectively. The challenge-response behavior is different for each IC due to manufacturing variations. Unfortunately, there are discrepancies with a random oracle. First, the response bits are not perfectly reproducible. Transistor-level noise sources are the main responsible. Environmental fluctuations, e.g., IC supply voltage and outside temperature, make things worse. Second, the response bits are nonuniformly distributed. The entropy is nonmaximum due to bias and correlations. Bias means that 0 and 1 do not occur with equal probability.

One often distinguishes between two categories of PUFs, depending on their scalability. Weak PUFs have an array structure, with each cell producing one or more bits.[1] Addressing the array provides a challenge-response mechanism. The total bit-content scales linearly with the circuit area. Correlations are primarily of spatial nature. Strong PUFs have a small response space but a large challenge space, e.g., 1 bit and 128 bits, respectively. The total bit-content is enormous and scales exponentially with the circuit area. However, correlations are much more severe: they originate from the functional behavior which is composed from a limited number of circuit elements only.

III. PUF Reliability Model

There is considerable advantage in having a generic but accurate PUF reliability model. First, a model provides more insights than raw experimental data. Second, one can comprehend the full spectrum of PUFs rather than a single instance only, by modifying model parameters. Third, it facilitates the design and analysis of HDAs. We describe a well-validated reliability model, serving as a reference later-on, although our results do not fully depend on its assumptions. A probability distribution describes the error rate of individual response bits $\widetilde{r}$. But first we provide a warning regarding the use of "averaged" simplified models.

A. Naive, Homogeneous

The most simple reliability model assigns an identical error rate to each response bit, as used in [4] and [41]. This naive homogeneous approach does not correspond with PUF reality. Nevertheless, one might be able to obtain accurate formulas for the average-case device failure rate within a certain working range. We consider this a dangerous practice though, especially for small-dimensioned constructions, operating on a limited number of bits. Furthermore, the model does not acknowledge that the device failure rate will show a spread among ICs: yield issues arise in industry where customers expect products to have a fixed quality level. Slightly better is a split between stable and unstable bits, although one does not always mention there to be a continuous transition rather than a strict separation.

B. Accurate, Heterogeneous

A heterogeneous approach draws the error rate P_e for each response bit $\widetilde{r}$ from a certain probability distribution. We describe a comprehensible reliability model, first employed in [36] and later validated with experimental PUF data in [10] and [24]. Error rate P_e is assumed to be independent and identically distributed (i.i.d.) among response bits. Two hidden variables are incorporated, representing manufacturing variability and noise, respectively. Their contributions

[1] This structure is not limited to memory-based PUFs, such as the SRAM PUF. Also coating PUFs [39] map to an array. And also the ring oscillator PUF [37], considering the most usable read-out modes such as neighbor chaining.

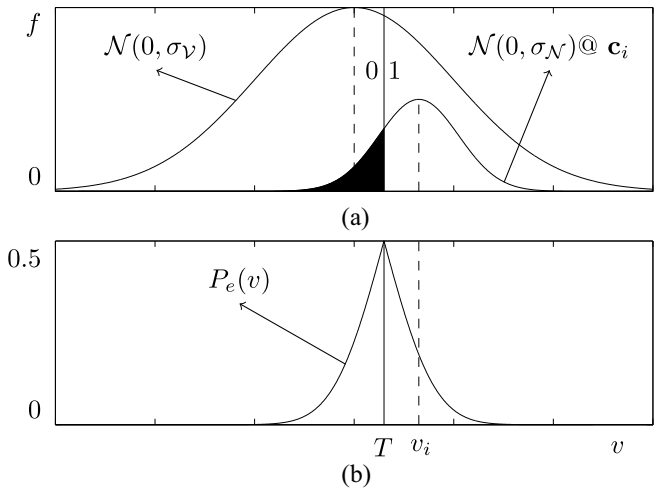

Fig. 1. Heterogeneous PUF reliability model. (a) Noise PDF is centered around the variability component v_i for a certain challenge $\mathbf{c}_i$. The black shaded area corresponds with P_e. (b) Further from T, the lower P_e.

are assumed to be additive and independent. Furthermore, both variables are assumed to be normally distributed, as empowered by the central limit theorem: each represents a complicated accumulation of local circuit effects.

Variability component v has mean zero and a certain standard deviation $\sigma_\mathcal{V}$ among response bits, as shown in Fig. 1(a). Also, noise component $\widetilde{n}$ has mean zero and a certain standard deviation $\sigma_\mathcal{N}$, among an infinite set of evaluations. A threshold value T implements an analog-to-digital conversion. Equation (7) represents both the nominal and instantaneous value of a certain response bit. A nonzero mean of either v or $\widetilde{n}$ could be incorporated in T. The bit error rate is given in (8) and graphically represented by Fig. 1(b). The closer to the threshold, the more often a bit does flip

$$r = \begin{cases} 1, & \text{if } v > T \\ 0, & \text{otherwise,} \end{cases} \qquad \widetilde{r} = \begin{cases} 1, & \text{if } v + \widetilde{n} > T \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

$$P_e = P\left(\widetilde{r} \neq r\right) = F_\mathcal{N}\left(\frac{-|v-T|}{\sigma_\mathcal{N}}\right). \tag{8}$$

C. Entropy

Two variables uniquely quantify the reliability characteristics of a PUF: $T/\sigma_\mathcal{V}$ and $\sigma_\mathcal{N}/\sigma_\mathcal{V}$. An attacker is typically assumed to know their values. This is a not unreasonable, given the many potential clues: dimensioning of the HDA, measurement of system failure rates, helper data, etc. A nonzero threshold value T does introduce bias, as defined by (9). Equation (10) represents the min-entropy. However, there is some ambiguity: one could define an instantaneous bias as well, as in (11). Although discrepancies might not be significant, care is advised when comparing various work: former and latter bias represent a best-case and worst-case, respectively[2]

$$B = P\left(r = 1\right) = 1 - F_\mathcal{N}\left(\frac{T}{\sigma_\mathcal{V}}\right) \tag{9}$$

$$H_\infty(r|B) = -\log_2(\max(B, 1-B)) \tag{10}$$

[2]Assuming v and n to be fully independent over their whole range.

$$\widetilde{B} = P\left(\widetilde{r} = 1\right) = \frac{1}{\sigma_\mathcal{V}} \int_{-\infty}^{\infty} f_\mathcal{N}\left(\frac{v}{\sigma_\mathcal{V}}\right) F_\mathcal{N}\left(\frac{v-T}{\sigma_\mathcal{N}}\right) \mathrm{d}v. \tag{11}$$

D. Model Limitations

Each model has its limitations. We summarize effects which are not taken into account. However, even for the basic heterogeneous model, HDA analysis is still lacking, as demonstrated later-on.

Environmental perturbations are not taken into account. One should distinguish between dc and ac variations. The latter is a form of noise injection and could be incorporated by increasing $\sigma_\mathcal{N}$. DC variations are more complicated, as they modify the nominal response behavior of the PUF. The error rate is typically defined with respect to the response at a certain dc reference environment. Increasing $\sigma_\mathcal{N}$ might capture this effect well, as implicitly assumed in [11] and experimentally validated for various PUFs using temperature and supply voltage variations. Alternatively, Maes [23] introduces an additional hidden variable, again additive, and normally distributed, with experimental validation for various PUFs using temperature variations.

Spatial and functional correlations are not included. However, correlation among v would imply correlation among $P_e(v)$ as well. Furthermore, for weak PUFs with a high bit-content, and hence spanning a large IC area, a single set of model parameters might be insufficient.

Temporal correlations are not included either. Response bit samples $\widetilde{r}$ are not necessarily independent, if the measurement interval is short. This effect originates from low-frequency noise. Or also, low-frequency perturbations of the environment.

PUF behavior might evolve over time due to transistor-aging effects (negative bias temperature instability, positive bias temperature instability, hot carrier injection, etc.) [26]. The reference response is typically defined at manufacturing time. Therefore, the error rate is expected to increase gradually.

IV. Framework for PUF-Based Key Generation

Fig. 2 represents a framework for PUF-based key generation. We consider a low-cost resource-constrained embedded device, prone to both physical and protocol attacks, as depicted in Fig. 2(a). Key storage in NVM comprehends two phases, as shown in Fig. 2(b). The first phase is a one-time enrollment in a secure environment after device manufacturing. A unique key $\mathbf{k}$ is then defined and programmed in the device. The second phase is in-the-field deployment, where an attacker might be waiting.

PUF technology might alleviate the two drawbacks of NVM: vulnerability to physical attacks and a high manufacturing cost. We assume the PUF to generate a lengthy string of response bits $\mathbf{r}^{IN}$. For weak PUFs, this corresponds with an array read-out. For strong PUFs, this would require the evaluation of a list of challenges, provided by a pseudorandom number generator starting from a fixed seed for instance. The same two phases are present again. Helper data $\mathbf{p} = Gen(\mathbf{r}^{IN})$ is generated during enrollment. *Gen* can be

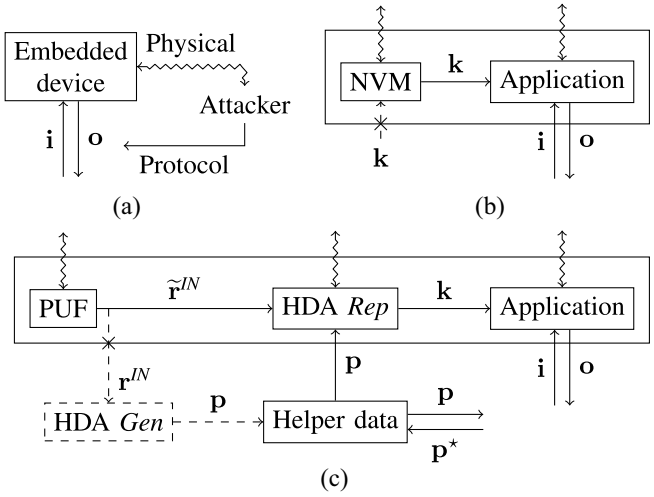

Fig. 2. (a) Embedded device under attack. (b) Traditional key storage in NVM. (c) Framework for PUF-based key generation. Enrollment steps are drawn dashed. Physical attack pathways are drawn zigzag. One-time interfaces, destructed after enrollment, are marked by the symbol $\times$.

implemented off-chip to save resources. In-the-field, the IC performs a reconstruction $\mathbf{k} = \mathit{Rep}(\widetilde{\mathbf{r}}^{IN}, \mathbf{p})$. Helper data can be stored in insecure (off-chip) NVM, or by another party.

We avoid the regularly used term syndrome to indicate helper data. This would be confusing since one popular HDA component actually employs the syndrome of an error-correcting code, as clarified later-on. We also stress that fuzzy extractors [12], [13] comprehend only a subset of all HDAs. Their definition offers two guarantees. First, correctness of reconstruction, given $HD(\mathbf{r}^{IN}, \widetilde{\mathbf{r}}^{IN}) \leq t$, with t a fixed parameter. Second, output $\mathbf{k}$ is nearly uniform, assuming an attacker to observe helper data $\mathbf{p}$.

A. Key Requirements

We define five key requirements. They are straightforward to satisfy for the traditional NVM scenario. For PUFs, this is very different, implicating the need for a HDA. Most work on HDAs considers the first two requirements only.

1) *KeyReq1: Reproducibility*. This is always required. It is a common practice to define a maximum failure rate for the key reconstruction phase, e.g., $P_{FAIL} \leq 10^{-6}$. This is coupled to a certain manufacturing yield: PUF noisiness and hence also P_{FAIL} shows a spread among devices.
2) *KeyReq2: Uniformly Distributed*. Keys are assumed to have maximum entropy. For some applications, entropy loss can be forgiven, given the use of a longer key, for example, for the computation of a HMAC, the application penalty might be low.
3) *KeyReq3a: PUF Independency*. One might have to program a key which does not depend on the PUF. This does not necessarily imply full control over the key bits: the HDA might perform additional hashing for instance. Consider symmetric key communication between two PUF devices with the same HDA. Or the replacement of a malfunctioning device with preservation of the key.
4) *KeyReq3b: Mathematical Restrictions*. The key might have to satisfy certain mathematical properties. Consider the primality test of RSA.
5) *KeyReq3c: Full Control*. One might need the ability to program any given key in the device. Consider symmetric key communication with a legacy device. This requirement supersedes *KeyReq3a* and *KeyReq3b*.

A trivial solution for *KeyReq3* is to encrypt/decrypt application keys with a PUF-derived key [33], [40]. The encrypted application keys are stored in insecure (off-chip) NVM, or by another party. For *KeyReq3b* in particular, one could also use the PUF-derived key as a seed for a deterministic *keygen* routine [38]. However, this might consume a lot of resources. Various HDA components offer intrinsic support for *KeyReq3a*, as clarified later-on.

B. Enrolled Reference: Bit Error Rate

Equation (8) describes the error rate of $\widetilde{r}$, considering the nominal value r as a reference. This is the best-case scenario, as the latter is the most likely value of the former. Therefore, it is very beneficial to perform a majority vote for the enrolled response $\mathbf{r}^{IN}$, in order to approximate the nominal value. This significantly relieves the burden, i.e., implementation overhead, of *Rep*. Voting has no impact on the IC footprint, as it can happen off-chip. Although the enrollment will take more time, it is a one-time effort only. Equation (12) considers an enrolled bit r^{IN} for an odd number of votes q. Equation (13) represents the error rate for $\widetilde{r}^{IN}$: the more votes, the better. Most formulas in this paper only depend on the averaged error rate, as given by (14)

$$r^{IN} = \begin{cases} 1, & \text{if } \widetilde{r}^{(1)} + \widetilde{r}^{(2)} + \ldots + \widetilde{r}^{(q)} > \frac{q-1}{2} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

$$P_e^{IN} = P\left(\widetilde{r}^{IN} \neq r^{IN}\right) = \begin{cases} P_e, & \text{if } r = r^{IN} \\ 1 - P_e, & \text{otherwise} \end{cases} \tag{13}$$

with

$$P\left(r = r^{IN}\right) = F_{\mathcal{B}}\left(\frac{q-1}{2}; q, P_e\right)$$
$$E_{\mathcal{V}}\left[P_e^{IN}\right] = \frac{1}{\sigma_{\mathcal{V}}} \int_{-\infty}^{\infty} f_{\mathcal{N}}\left(\frac{v}{\sigma_{\mathcal{V}}}\right) P_e^{IN}(v)\, \mathrm{d}v. \tag{14}$$

C. Public Helper Data: Attack Scenarios

We stick to the widely accepted notion that helper data should be public. Imposing constraints implicates the need for secure on-chip NVM, as in Fig. 2(b), undermining the potential benefits of PUF technology. An attacker can perform both read and write operations: the threats are identified as leakage and manipulation, respectively. The latter supersedes the former: if blindly writing a 0 does not result in a failure, then the previous value of the bit must have been a 0 too.

Helper data $\mathbf{p}$ unavoidably leaks information about the response $\mathbf{r}^{IN}$. This entropy loss can be compensated at the cost of more PUF bits, as clarified later. So it is mainly a matter of accurately quantifying the leakage in order to fully mitigate this threat. One typically assumes a single exposure of the helper data. It is not unthinkable, however, that there is more leakage when repeatedly exposing helper data for noisy variants of $\mathbf{r}^{IN}$ [5], assuming an on-chip implementation of *Gen*. Especially, in combination with fault injection attacks

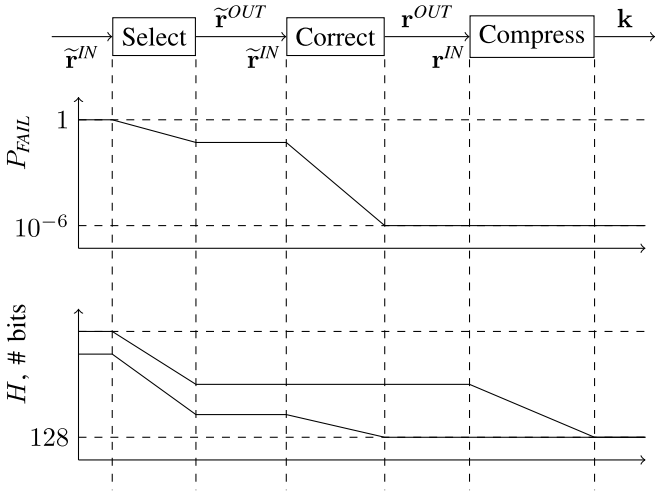

Fig. 3. Consecutive steps of a HDA. Typical profiles for failure rate P_{FAIL}, the number of bits and their total entropy.

on $\widetilde{\mathbf{r}}^{IN}$. Preventing this would require some sort of one-time interface to disable *Gen*.

Helper data manipulation comprehends two categories of threats. First, it can facilitate physical attacks on either the HDA or the application, as exploited in [28]. Second, protocol attacks can be mounted via the I/O interface of the application, as exploited in [8], [9], and [17]. By observing **o**, one aims to test hypotheses regarding the key **k** for a certain malicious string $\mathbf{p}^{\star}$. Below, we distinguish two types of applications, in terms of vulnerability. Section VIII will provide an overview of generic countermeasures, not depending on the HDA.

1) AppWeak: Output **o** depends exclusively on the key **k** and known data, given a series of commands via **i**. Consider an IC which returns $\mathbf{a} \leftarrow \mathit{Encrypt}(\mathbf{k}, \mathbf{n})$ for a user-defined nonce **n**. This would allow for IC authentication, by matching **a**. An attacker can then distinguish between keys: $\mathbf{o}_1 \neq \mathbf{o}_2$ if $\mathbf{k}_1 \neq \mathbf{k}_2$. This is relevant, as helper data might allow to (partly) reprogram the key.

2) AppStrong: Output **o** is persistently indifferent if any other key than **k** is being processed. Consider a secure boot application, decrypting processor code. A failure is bound to occur if **k** is not reconstructed correctly. However, observing failure rates might still be very informative for certain manipulations.

D. Assembling HDA

A practical HDA is an assembly of components rather than a single building block. One can typically distinguish three consecutive steps,[3] as represented in Fig. 3. First, bit selection, discarding the least reliable bits. This alleviates the burden of the second step: error-correction. An interaction of former steps results in a reasonable failure rate P_{FAIL}, but the outgoing bits have nonmaximum entropy. A third step performs entropy compression. We discuss the steps in backwards order, describing and analyzing individual proposals.

[3]In various works, PUF-specific helper data is employed, especially for the ring oscillator PUF and its variants [8]. These constructions are considered to be out-of-scope here.

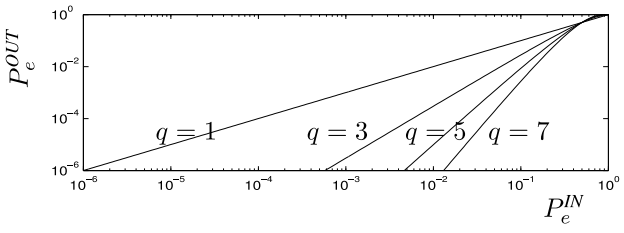

Fig. 4. Temporal majority voting.

V. Entropy Compression

The entropy of $\mathbf{r}^{IN}$ is nonmaximum because of two reasons. First, correlations and bias of the PUF. Second, leakage of the HDA. Entropy compression ensures the key **k** to be nearly uniform. This step is also referred to as privacy amplification, although the latter term might be more applicable to its original biometric context [22]. The total amount of entropy is preserved, but the average per bit increases by having more input than output bits. A hash function is the well-established solution, as first proposed in [15]. One computes $\mathbf{k} = \mathit{Hash}(\mathbf{r}^{IN})$.

One mostly opts for a lightweight hash function. A popular choice is SPONGENT, implemented in [17], [18], and [27]. A more bulky hash could be fine as well, especially if the application requires its implementation anyway, e.g., SHA-1 has been implemented in [38].[4] According to the fuzzy extractor definition [12], [13], one should opt for a universal hash function. Helper data then indicates the randomly selected function instance. The universal Toeplitz hash, which conveniently maps to an LFSR-based architecture, has been implemented in [4], [25], and [28]. There is no stringent need for this, however, although it would allow for a simple but secure update of the key **k**.

VI. Error-Correction Schemes

Error-correction, also referred to as information reconciliation, ensures the key to be reproducible. We discuss various constructions.

A. Temporal Majority Voting

Majority voting, previously discussed for the enrollment, can be performed during reconstruction as well [2]. However, then it is not a one-time effort anymore and additional IC-hardware (counters) is required. Equation (15) represents the error propagation for an odd number of votes q. Low bit error rates are successfully suppressed, but high rates remain high, as shown in Fig. 4. Therefore, the method is never sufficient by itself: further error-correction or prior bit-selection is advised

$$P_e^{OUT} = 1 - F_{\mathcal{B}}\left(\frac{q-1}{2}; q, P_e^{IN}\right) \quad \text{with } q \text{ odd.} \tag{15}$$

B. Exhaustive Search

The IC could perform an exhaustive search for the error pattern [30]. However, this might consume too much resources

[4]More recent cryptanalytic results are mainly for collisions, so SHA-1 could still be used for entropy compression.

to be practical. The overhead also includes a secure check for correctness, as a stopping criterion.

C. Secure Sketch: Code-Offset and Syndrome Construction

Secure sketches, as defined in [12] and [13], are the workhorse of most HDAs. They allow for the construction of a fuzzy extractor. Despite the rather generic definition, two constructions dominate the implementation landscape, as specified below. Both the code-offset and syndrome construction employ a binary $[n, k, t]$ block code $\mathcal{C}$, with t the error-correcting capability.[5] A variant of the code-offset construction allows to fulfill *KeyReq3a* [41]. Instead of outputting the error-corrected response, one could also feed $\mathbf{w}$ into the entropy compression hash. Or alternatively $\mathbf{m}$, the message corresponding to the codeword, as it would require less compression. The syndrome construction requires a linear block code, as it employs the parity check matrix $\mathbf{H}$. Successful reconstruction is guaranteed for both constructions, given $HW(\mathbf{e}) \leq t$.

	Gen	*Rep*
code-offset	Random $\mathbf{w} \in \mathcal{C}$ $\mathbf{p} \leftarrow \mathbf{r}^{OUT} \oplus \mathbf{w}$	$\widetilde{\mathbf{w}} \leftarrow \widetilde{\mathbf{r}}^{IN} \oplus \mathbf{p} = \mathbf{w} \oplus \mathbf{e}$ Error-correct $\widetilde{\mathbf{w}}$ to $\mathbf{w}$ $\mathbf{r}^{OUT} \leftarrow \mathbf{p} \oplus \mathbf{w}$
syndrome	$\mathbf{p} \leftarrow \mathbf{r}^{OUT} \cdot \mathbf{H}^T$	$\mathbf{s} \leftarrow \widetilde{\mathbf{r}}^{IN} \cdot \mathbf{H}^T \oplus \mathbf{p} = \mathbf{e} \cdot \mathbf{H}^T$ Determine $\mathbf{e}$ $\mathbf{r}^{OUT} \leftarrow \widetilde{\mathbf{r}}^{IN} \oplus \mathbf{e}$

The code-offset construction is employed in [4], [24], [25], and [41]. The syndrome construction is employed in [27], [38], and [42]. BCH codes in particular are very popular: they are implemented in [27], [38], [42], [45], and [46]. Also repetition codes are rather popular, due to their ease of implementation [4], [25]. The latter is fundamentally different from temporal majority voting, although there is some similarity for the number of errors they can correct. Other codes have been implemented as well, such as Reed–Muller [4], [25], [41] and Golay [4], [41].

1) Failure: Equation (16) represents the averaged failure rate of both secure sketch constructions. The formula holds under the i.i.d. assumption, as in the heterogeneous reliability model of Section III. A proof is provided in the Appendix. An identical formula, when omitting both expected value operators, could be derived easily from the naive homogeneous reliability model [4]. However, as such one does not acknowledge P_{FAIL} to show a spread among ICs

$$E_{\mathcal{V}}[P_{FAIL}] = 1 - F_{\mathcal{B}}\left(t; n, E_{\mathcal{V}}\left[P_e^{IN}\right]\right). \tag{16}$$

2) Leakage: Leakage of the code-offset construction (and its variant) can be understood as a one-time pad imperfection. The secret $\mathbf{r}^{OUT}$ is XORed with a vector which is not fully random: codeword $\mathbf{w}$ has entropy $k < n$. For the syndrome construction, helper data can be understood as direct leakage: each helper bit reveals a linear equation of response bits. As proven in [12] and [13], the min-entropy loss is at most $(n-k)$ bits for both secure sketch constructions[6]

$$\widetilde{H}_\infty\left(\left(\mathbf{r}^{OUT}|B^{OUT}\right)|\mathbf{p}\right) \geq H_\infty\left(\mathbf{r}^{OUT}|B^{OUT}\right) - (n-k). \tag{17}$$

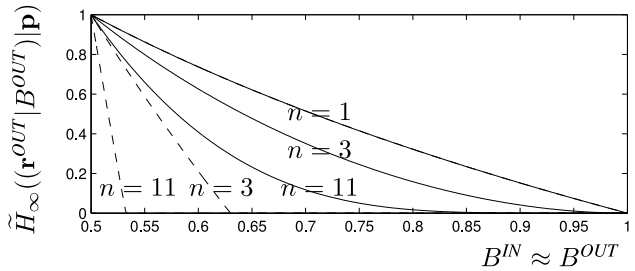

Fig. 5. Min-entropy loss for repetition code REP[11,1,5], REP[3,1,1], and the degenerate case REP[1,1,0]. Solid curves correspond with (18). Dashed curves correspond with (17). Due to the symmetry around $B = 1/2$, only half of the spectrum is shown.

Especially for repetition codes REP[n, 1, $(n-1)/2$], with n odd, there is an imminent threat: at most 1 bit of min-entropy can remain. The devastating power of bias has been illustrated in [43]. More recently, this warning has been repeated in [21], although we consider their conclusions as exaggerated. One seems to ignore that $(n-k)$ is an upper bound only. We are the first to derive an exact formula for the min-entropy loss of REP, in the case of bias. Equation (18) is valid for both secure sketch constructions, as proven in the Appendix. Fig. 5 illustrates that (17) is overly pessimistic

$$\widetilde{H}_\infty\left(\left(\mathbf{r}^{OUT}|B^{OUT}\right)|\mathbf{p}\right) = -\log_2\left(F_{\mathcal{B}}\left(\frac{n-1}{2}; n, \min\left(B^{OUT}, 1-B^{OUT}\right)\right)\right). \tag{18}$$

3) Manipulation: No generic manipulation attacks have been published so far, but we make a few observations for the code-offset construction. First, omitting the entropy compression hash would enable related-key attacks (*AppWeak* only). For the original method, one can inject a malicious helper string $\mathbf{p}^\star \leftarrow \mathbf{p} \oplus \mathbf{e}$, with $HW(\mathbf{e})$ small. This would result in a related key $\mathbf{k}^\star \leftarrow \mathbf{k} \oplus \mathbf{e}$. For the variant, assuming the conventional usage of linear codes, one can add a codeword: $\mathbf{p}^\star \leftarrow \mathbf{p} \oplus \mathbf{w}$, resulting in a related key $\mathbf{k}^\star \leftarrow \mathbf{k} \oplus \mathbf{w}$.

Second, one can derive estimates of individual bit error rates P_e^{IN}. This requires the *AppWeak* and *AppStrong* assumption for the original and variant, respectively. The potential threats are described later in Section VI-F, as this exposure is an integral part of soft-decision coding. We introduce a synthetic error for one particular bit: $\mathbf{p}^\star \leftarrow \mathbf{p} \oplus \mathbf{e}$, with $HW(\mathbf{e}) = 1$. By measuring failure rate P_{FAIL} before and after, one can derive an estimate. A lot of measurements will be required though to obtain a reasonable accuracy.

D. Codes in Parallel

Very often, it is not feasible to process all response bits with a single code, due to the decoding complexity [4], [25], [27].

[5]The first use of error-correcting codes in a PUF context was in [14] and [15].

[6]The leakage upper bound is more generally applicable than bias only.

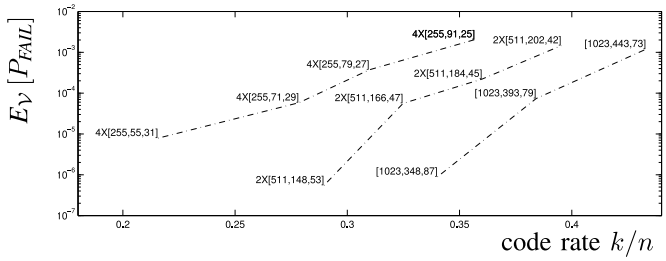

Fig. 6. Failure rate versus code rate for BCH codes, with $E_{\mathcal{V}}[P_e^{IN}] = 5\%$. Increasing x worsens the trade-off.

Algorithm 1: Attack for Nonlinear Code-Offset

Input: List of codewords $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{k'}$
Helper data $\mathbf{p}$
Output: Response section $\mathbf{r}$
$I \leftarrow \{1, 2, \ldots, k'\}$
for $i \leftarrow 1$ ***to*** k' **do**
  $j \leftarrow mod\,(i, k') + 1$
  Modify helper data: $\mathbf{p}^\star \leftarrow \mathbf{p} \oplus \mathbf{w}_i \oplus \mathbf{w}_j$
  if *key reconstruction failed* **then**
    $I \leftarrow I \backslash \{i, j\}$
$\mathbf{r} \leftarrow \mathbf{p} \oplus \mathbf{w}_I$

This is resolved by subdividing $\mathbf{r}$ in x nonoverlapping sections of length n, processed independently by a smaller code. To save area and power, sections are decoded one-by-one.

1) Failure: Equation (19) is a trivial extension of (16). Introducing sections is not without a penalty. More PUF bits will be required to obtain the same failure rate and key length, taking leakage into account. An illustration is provided in Fig. 6

$$E_{\mathcal{V}}[P_{FAIL}] = 1 - \left(F_{\mathcal{B}}\left(t; n, E_{\mathcal{V}}\left[P_e^{IN}\right]\right)\right)^x. \tag{19}$$

2) Leakage: Former leakage results can be applied to each parallel instance separately.

3) Manipulation: A novel divide-and-conquer manipulation attack aims at retrieving the section responses one-by-one. The brute-force effort is drastically reduced: roughly $x2^k$ instead of 2^{xk}, when omitting entropy compression. Linear codes, which are consistently used in literature, are found to be secure. Less conventional codes, which have not been employed in literature so far, not necessarily. We provide some examples for the code-offset construction, under the *AppStrong* assumption.

Consider a nonlinear code having at least three codewords. Algorithm 1 can retrieve the section responses. It does not matter whether $\mathbf{0} \in \mathcal{C}$. Set I is expected to contain a single element in the end. Depending on the code, false positives might occur in the occasional case that the sum of three codewords is again a codeword. This can easily be resolved with an extension of the algorithm.

Codes with some sort of nonuniformity for t would be in danger as well. Consider the following scenarios. First, $1 \rightarrow 0$ and $0 \rightarrow 1$ errors might not be equivalent [47]. Second, t might differ per codeword. Third, the error-correcting capability for each codeword might be correlated with the position of the errors. In all former cases, an attacker can iterate again over all potential codewords and test hypotheses via P_{FAIL}.

E. Concatenated Codes

Concatenated codes, in the context of PUFs, have first been proposed in [4]. They have also been adopted in [27]. They are particularly useful when $E_{\mathcal{V}}[P_e^{IN}]$ is high. Consider the concatenation of two block codes: $[n_2, k_2, t_2] \circ [n_1, k_1, t_1]$, with n_1 an integer multiple of k_2. A good design is as follows. Code $\mathcal{C}_2$ should be able to correct many errors (a high ratio t_2/n_2). Repetition codes in particular are very popular. Code $\mathcal{C}_1$ only needs to correct a few errors (a low ratio t_1/n_1), but therefore a high k_1 to maintain entropy.

1) Failure: P_{FAIL} depends on the decoder characteristics of $\mathcal{C}_2$. We distinguish between repetition codes and all other codes, as represented by (20) and (21), respectively. As before, they are the more accurate equivalent of a similar formula in [4]. We assume the general case of x concatenated codes in parallel. For repetition codes, the decoding behavior is trivial: if the number of errors exceeds t_2, a single error will propagate always. For other codes, we assume that half of the outgoing bits is expected to flip if t_2 is exceeded

$$E_{\mathcal{V}}[P_{FAIL}] = 1 - \left(F_{\mathcal{B}}\left(t_1; n_1, 1 - F_{\mathcal{B}}\left(t_2; n_2, E_{\mathcal{V}}\left[P_e^{IN}\right]\right)\right)\right)^x \tag{20}$$

$$E_{\mathcal{V}}[P_{FAIL}] = 1 - \left(F_{\mathcal{B}}\left(t_1; n_1, \frac{1 - F_{\mathcal{B}}\left(t_2; n_2, E_{\mathcal{V}}\left[P_e^{IN}\right]\right)}{2}\right)\right)^x. \tag{21}$$

2) Leakage: Leakage formulas can again be applied independently. The outgoing entropy of each parallel instance can never exceed k_1.

3) Manipulation: Divide-and-conquer manipulation attacks are again a threat. There is inherently a parallel structure for $\mathcal{C}_2$. And typically $x > 1$, endangering $\mathcal{C}_1$ too.

F. Soft-Decision

Soft-decision decoding, in the context of PUFs, has been introduced conceptually in [24], with a subsequent implementation in [25]. The error-correcting capabilities improve with respect to traditional hard-decision decoding. Less PUF bits are required for the same failure rate and key length, taking leakage into account. The original proposal comprehends a collaboration with the code-offset method, hereby exposing all bit error rates P_e^{IN} as public helper data.[7]

Unfortunately, the decoding effort increases significantly. Soft-decision maximum-likelihood (SDML) decoding offers the best performance, by iterating over all codewords and computing the likelihood each time. The computational complexity is hence exponential with the code dimension k. There are

[7]A variation has been proposed in [41]. Several PUF bits are clustered and averaged to form soft-decision data. Measurement of P_e and its accompanying exposure as helper data is not required, at the cost of performance loss.

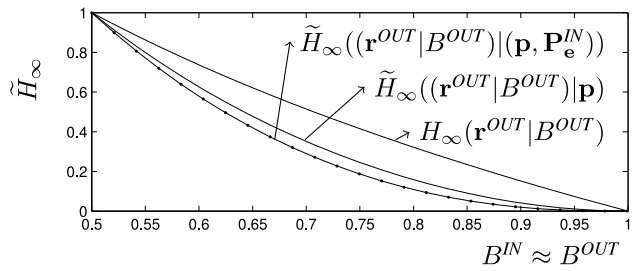

Fig. 7. Soft-decision leakage for a [3, 1, 1] repetition code. Solid lines represent (18) and (38). Dots represent independent simulation results to verify the correctness of the latter.

faster procedures for some codes at the cost of reduced performance. Consider generalized multiple concatenated codes: one exploits that a large Reed–Muller code can be split recursively in two smaller Reed–Muller codes.

1) Leakage: Exposing a vector $\mathbf{P}_\mathbf{e}^{IN}$ of bit error rates is stated not to result in additional min-entropy loss [24]. A proof of (22) has been derived for the heterogeneous reliability model of Section III-B. Before arguing that the leakage threat has been underestimated, we confirm that the proof by itself is correct and provide some additional insights

$$\widetilde{H}_\infty\left(\left(\mathbf{r}^{OUT}|B^{OUT}\right)|\mathbf{P}_\mathbf{e}^{IN}\right) = H_\infty\left(\mathbf{r}^{OUT}|B^{OUT}\right). \quad (22)$$

Exposure of P_e allows for a twofold "inversion" of the curve in Fig. 1(b). This arms an attacker with an individual bias B_i^{OUT} for each bit, as represented by (23), in contrast to an averaged bias B^{OUT}. We observe that $E_\mathcal{V}[B_i^{OUT}] = B^{OUT}$, a fact which is implicitly embedded in the proof in [24]. We also observe that (22) does not extend to Shannon-entropy: the average-case brute-force attack is accelerated

$$\begin{aligned} B_i^{OUT} &= \frac{f_\mathcal{N}\left(\frac{T+|v_i-T|}{\sigma_\mathcal{V}}\right)}{f_\mathcal{N}\left(\frac{T-|v_i-T|}{\sigma_\mathcal{V}}\right)+f_\mathcal{N}\left(\frac{T+|v_i-T|}{\sigma_\mathcal{V}}\right)} \\ &= \frac{1}{1+\exp\left(\frac{2T|v_i-T|}{(\sigma_\mathcal{V})^2}\right)}. \end{aligned} \quad (23)$$

Unfortunately, one does not incorporate the interaction with the code-offset method. Exposure of $\mathbf{P}_\mathbf{e}^{IN}$ might increase the leakage of $\mathbf{p}$, although the former does not leak by itself, as represented by (24). We demonstrate this for repetition codes in particular in Fig. 7

$$\widetilde{H}_\infty\left(\left(\mathbf{r}^{OUT}|B^{OUT}\right)|\left(\mathbf{p},\mathbf{P}_\mathbf{e}^{IN}\right)\right) \leq \widetilde{H}_\infty\left(\left(\mathbf{r}^{OUT}|B^{OUT}\right)|\mathbf{p}\right). \quad (24)$$

Finally, exposure of $\mathbf{P}_\mathbf{e}^{IN}$ can facilitate modeling attacks, assuming the use of a strong PUF. Reliability data has been demonstrated to be a modeling asset for easy targets such as the arbiter PUF [10], [11].

2) Manipulation: We are the first to warn against divide-and-conquer manipulation attacks, although their relevance would depend on implementation details. Due to the decoding complexity, there is strong tendency for $x > 1$, even more than for its hard-decision counterpart. We limit ourselves to a conceptual illustration in Fig. 8. We assume SDML decoding with $k > 1$ and the *AppStrong* scenario. For ease of understanding, assume that the response bits to be perfectly reproducible, although extension to a more realistic setting is not that hard. By setting $P_e^{IN} \leftarrow 1/2$, bit r does not contribute to the likelihood computation anymore. By observing failures for a few $\mathbf{P}_\mathbf{e}^{IN}$ patterns, one might be able to discard all-but-one codewords. Note that also repetition codes ($k = 1$) might be vulnerable, by setting $\mathbf{P_e} \leftarrow (1/2\ 1/2\ \ldots\ 1/2)$: some implementations might then select the first (or last/second) codeword always.

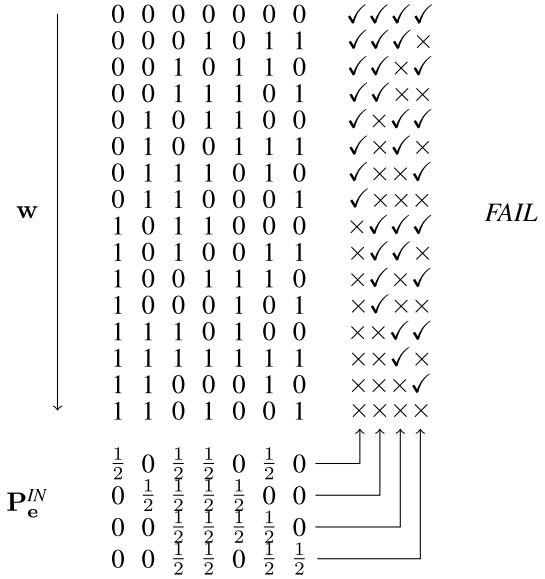

Fig. 8. Soft-decision manipulation attack, illustrated for a [7,4,1] code. We assume SDML decoding, deterministically iterating over all codewords $\mathbf{w}$, top-to-bottom as indicated by the arrow. We also assume that the first codeword is selected in case of a likelihood tie.

G. Convolutional Codes

The popularity of block codes does not exclude other possibilities. The use of convolutional codes has been proposed in [17] and [18]. They implement a hard-decision Viterbi algorithm.

H. Substring Matching

An alternative for error-correcting codes has been proposed in [31] and [32]. We limit ourselves to the basic idea only. Consider a lengthy string of PUF bits $\mathbf{r}$. During enrollment, a shorter substring $\mathbf{r}_{SUB}$ is selected at random, possibly considering $\mathbf{r}$ to be circular, and exposed as public helper data. Former selection procedure is repeated for several strings: substring indices are combined to obtain a key $\mathbf{k}$ of sufficient length. During reconstruction, substrings are shifted along newly generated strings $\widetilde{\mathbf{r}}$. The correct indices are retrieved via their low Hamming distance. The scheme is able to fulfill *KeyReq3a* and possibly also *KeyReq3b* and *KeyReq3c* if substring indices are simply concatenated to form the key.

1) Failure: Failure behavior has been studied in [9], so we limit ourselves to qualitative insights. First, the longer the

substrings (while upscaling $\mathbf{r}$ as well to maintain its number of indices), the lower P_{FAIL}: the gap in Hamming distance between correct and incorrect indices relatively increases. Unfortunately, this is accompanied with a direct efficiency overhead. Second, the shorter $\mathbf{r}$ (while maintaining the length of $\mathbf{r}_{SUB}$), the lower P_{FAIL}: there is less competition between indices then. However, more substrings are required to obtain a key of equal length.

2) Leakage: The leakage concept is slightly different that for error-correcting codes. Before, we examined the direct entropy loss of a secret response $\mathbf{r}$. The challenge list $\langle \mathbf{c} \rangle$ is known: PUF correlations would amplify the helper data leakage, although this effect has not been quantified in this and previous work. Now, one should examine to which extent an attacker can retrieve the link between a known list $\langle \mathbf{c} \rangle$ and a nonsecret substring of the response $\mathbf{r}_{SUB}$. For a perfect noncorrelated PUF, the leakage would be zero, unlike before. However, correlations are by no means hidden and hence easy-to-exploit: there is imminent danger for the most vulnerable strong PUFs.

3) Manipulation: Helper data manipulation attacks have been applied successfully in [9], working under either the *AppStrong* or *AppWeak* assumption. They gradually append bits to the substrings $\mathbf{r}_{SUB}$, in terms of exposure. The danger is twofold. First, exploitation of PUF correlations is facilitated. Second, if $\mathbf{r}$ is noncircular, substring indices can be retrieved directly, independent of the PUF.

VII. Bit Selection Schemes

Bit selection is the crude (or lightweight) version of soft-decision coding: the least reliable bits are simply discarded. This lowers the burden of the subsequent error-correction step. There is a good symbiosis with temporal majority voting: only low bit error rates sustain selection, which can be dealt with effectively [2]. For high loss ratios, it might even be possible to omit the error-correction step [3]. The overhead of bit selection greatly differs for weak and strong PUFs. The former case is more problematic: there is a direct cell/area loss. For strong PUFs, the challenge generator only needs to make a few skips.

We are the first to disprove the intuitive assumption that bit selection does not leak. We demonstrate there to be an amplification of bias. Furthermore, we derive formulas describing the drop in $E_{\mathcal{V}}[P_e]$, given the heterogeneous reliability model of Section III. Previously derived formulas for the error-correction step are compatible. We now discuss the proposals one-by-one and compare them afterwards.

A. Global Thresholding

Imposing a global threshold for P_e^{IN} is the most intuitive idea. This has first been proposed in [36], with employment later-on in [2], [3], [17], [19], and [37].

1) Failure: We discard all bits with $|v - T| \leq \delta v$. Equation (25) represents the ratio of discarded bits. Equation (26) is the averaged bit error rate, hereby embedding (13)

$$Loss = \frac{1}{\sigma_{\mathcal{V}}} \int_{T-\delta v}^{T+\delta v} f_{\mathcal{N}}\left(\frac{v}{\sigma_{\mathcal{V}}}\right) \mathrm{d}v \tag{25}$$

$$E_{\mathcal{V}}\left[P_e^{OUT}\right] = \frac{1}{\sigma_{\mathcal{V}}(1 - Loss)} \left(\int_{-\infty}^{T-\delta v} f_{\mathcal{N}}\left(\frac{v}{\sigma_{\mathcal{V}}}\right) P_e^{IN}(v)\,\mathrm{d}v + \int_{T+\delta v}^{\infty} f_{\mathcal{N}}\left(\frac{v}{\sigma_{\mathcal{V}}}\right) P_e^{IN}(v)\,\mathrm{d}v\right). \tag{26}$$

2) Leakage: Equation (27) represent the bias of the outgoing bits. We later illustrate graphically that there is an entropy loss: $|B^{OUT} - 1/2| > |B - 1/2|$

$$B^{OUT} = \frac{1 - F_{\mathcal{N}}\left(\frac{T+\delta v}{\sigma_{\mathcal{V}}}\right)}{1 - F_{\mathcal{N}}\left(\frac{T+\delta v}{\sigma_{\mathcal{V}}}\right) + F_{\mathcal{N}}\left(\frac{T-\delta v}{\sigma_{\mathcal{V}}}\right)}. \tag{27}$$

3) Manipulation: The scheme is highly vulnerable to helper data manipulation, even under the *AppStrong* assumption. Assume each selected bit to have discarded neighbors to its left and right. By shifting a selected index and observing the failure rate P^{FAIL}, one can check whether bits are equal. Repeated exploitation of this principle can reveal all response bits, except for one degree of freedom. The former has been described in [17], although one does not recognize there to be a problem if *Loss* is small: clusters of selected bits counteract the attack. However, in combination with other HDA components, or under the *AppWeak* assumption, it might be resolvable.

4) Format: The helper data format comprehends a trade-off between its size and its on-chip interpretation effort. We argue that one should make a choice based on *Loss*. In [3], one assigns a dedicated helper bit to each response bit. This is efficient if $Loss \approx 50\%$. In [17], a list of helper data indices represents the relative distances between consecutive selected bits, which if efficient if *Loss* is high. If *Loss* is low, one can represent distances between consecutive discarded bits instead. However, representing distances with a fixed number of bits is not necessarily efficient. As a resolution, the same authors later proposed a run-length encoding scheme in [18].

5) Extension: An extension to satisfy *KeyReq3a* has been proposed in [17]. The selected vector $\mathbf{r}^{OUT}$ is XORed with another vector stored as helper data. We make the following observations. First, this operation could be postponed and executed after the error-correction or hash as well. In the latter case, there is less helper data and *KeyReq3b*/*KeyReq3c* could have been satisfied as well. Second, if the application keys are large, it is more efficient to encrypt/decrypt applications keys with a PUF-derived key.

B. Local Thresholding: 1-Out-of-n

A local equivalent of global thresholding has been proposed in [37]. Response $\mathbf{r}^{IN}$ is subdivided in nonoverlapping sections of length n. For each section, only the most reliable bit is retained. Equation (28) represents the ratio of discarded bits. A potential decrease in helper data size would be the main advantage with respect to global thresholding

$$Loss = \frac{n-1}{n}. \tag{28}$$

1) Failure: The failure behavior can be analyzed via order statistics [1]. We reorder section bits according to their variability component: $v_{(1)} < v_{(2)} < \ldots < v_{(n)}$. Equation (29)

represents the joint PDF of $v_{(1)}$ and $v_{(n)}$, assuming the nondegenerate case $n > 1$. Equation (30) represents the averaged bit error rate, hereby embedding (13)

$$f(v_{(1)}, v_{(n)}) = \frac{n(n-1)}{(\sigma_{\mathcal{V}})^2} f_{\mathcal{N}}\left(\frac{v_{(1)}}{\sigma_{\mathcal{V}}}\right) f_{\mathcal{N}}\left(\frac{v_{(n)}}{\sigma_{\mathcal{V}}}\right) \left(F_{\mathcal{N}}\left(\frac{v_{(n)}}{\sigma_{\mathcal{V}}}\right) - F_{\mathcal{N}}\left(\frac{v_{(1)}}{\sigma_{\mathcal{V}}}\right)\right)^{n-2} \quad (29)$$

with

$$v_{(1)} < v_{(n)}$$

$$E_{\mathcal{V}}\left[P_e^{OUT}\right] = \int_{-\infty}^{\infty}\int_{-\infty}^{T-|v_{(n)}-T|} f(v_{(1)}, v_{(n)}) P_e^{IN}(v_{(1)})\, \mathrm{d}v_{(1)}\mathrm{d}v_{(n)} + \int_{-\infty}^{\infty}\int_{T+|v_{(1)}-T|}^{\infty} f(v_{(1)}, v_{(n)}) P_e^{IN}(v_{(n)})\, \mathrm{d}v_{(n)}\mathrm{d}v_{(1)}. \quad (30)$$

2) Leakage: Equation (31) represent the bias of the outgoing bits. We will again demonstrate graphically that there is a bias amplification and hence entropy loss

$$B^{OUT} = \int_{T}^{\infty}\int_{2T-v_{(n)}}^{v_{(n)}} f(v_{(1)}, v_{(n)})\, \mathrm{d}v_{(1)}\mathrm{d}v_{(n)}. \quad (31)$$

3) Manipulation: As for global thresholding, bit equalities can be extracted, although limited within a section. In case of bias ($B \neq 1/2$), there is an additional entropy loss.

4) Extension: A generalization of the scheme has been proposed in [37] as well. One selects k out of n bits, with $k \in [1,\ n]$. With n equal to the length of $\mathbf{r}^{IN}$, there would be an equivalency with global thresholding. Also, the manipulation threat increases for the nondegenerate case $k > 1$.

C. Local Thresholding: Index-Based Syndrome (IBS)

A variation on 1-out-of-n selection has been proposed in [43], with employment in [46] later-on. The IBS scheme is able to satisfy *KeyReq3a*. The most reliable 0 or 1 within each segment is selected, with equal probability. Or more accurately: the selected bit either minimizes or maximizes $(v - T)$, as the target value is not necessarily available. Equation (28) still represents the ratio of discarded bits.

1) Failure: Equation (32) represents the averaged bit error rate, again embedding (13) hereby

$$E_{\mathcal{V}}\left[P_e^{OUT}\right] = \frac{1}{2}\int_{-\infty}^{\infty} f\left(v_{(1)}\right) P_e^{IN}(v_{(1)})\ \mathrm{d}v_{(1)} + \frac{1}{2}\int_{-\infty}^{\infty} f\left(v_{(n)}\right) P_e^{IN}(v_{(n)})\ \mathrm{d}v_{(n)}$$

with

$$f\left(v_{(n)}\right) = \frac{n}{\sigma_{\mathcal{V}}} f_{\mathcal{N}}\left(\frac{v_{(n)}}{\sigma_{\mathcal{V}}}\right)\left(F_{\mathcal{N}}\left(\frac{v_{(n)}}{\sigma_{\mathcal{V}}}\right)\right)^{n-1}$$

and

$$f\left(v_{(1)}\right) = \frac{n}{\sigma_{\mathcal{V}}} f_{\mathcal{N}}\left(\frac{v_{(1)}}{\sigma_{\mathcal{V}}}\right)\left(1 - F_{\mathcal{N}}\left(\frac{v_{(1)}}{\sigma_{\mathcal{V}}}\right)\right)^{n-1}. \quad (32)$$

2) Leakage: As proven in [43], there is no leakage under the i.i.d. assumption. Stated otherwise: $B^{OUT} = 1/2$.

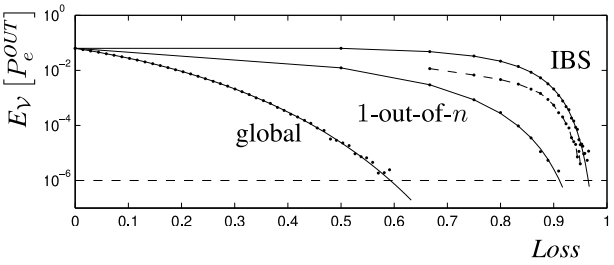

Fig. 9. Bit selection: trade-off between $E_{\mathcal{V}}[P_e]$ and *Loss*. The ingoing bits obey $\sigma_{\mathcal{N}}/\sigma_{\mathcal{V}} = 0.2$ and $T = 0$. We assume the enrollment majority vote to be ideal ($q = \infty$). All formulas are represented with solid lines, although only global thresholding is of continuous nature. Dots represent independent simulation results to confirm the correctness of the formulas. For C-IBS, we only have simulation results (codewords of length 3).

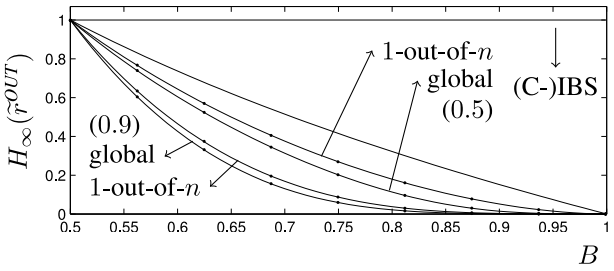

Fig. 10. Bit selection: leakage. The ingoing bits obey $\sigma_{\mathcal{N}}/\sigma_{\mathcal{V}} = 0.2$. Between brackets, *Loss* is indicated. Dots represent independent simulation results, to confirm the correctness of the formulas.

3) Manipulation: A similar manipulation threat as for 1-out-of-n selection is present.

4) Extension: An integration with local error-correction has been proposed in [16]. The trade-off between $E_{\mathcal{V}}[P_e^{OUT}]$ and *Loss* is improved somewhat, at the cost of increased implementation complexity. One selects a random codeword within each segment: either $(0\ 1\ \ldots\ 0)$ or $(1\ 0\ \ldots\ 1)$,[8] both having an odd number k of alternating bits. Each codeword is decoded to a single bit: (28) still represents the ratio of discarded bits. Up to $(k-1)/2$ bit errors can be corrected, without introducing helper data. Again, there is no leakage: $B^{OUT} = 1/2$.

D. Comparison

Global thresholding offers the best-trade-off between $E_{\mathcal{V}}[P_e^{OUT}]$ and *Loss*, as shown in Fig. 9. The 1-out-of-n selection scheme is significantly worse, although the k-out-of-n extension can lessen the gap. IBS has the lowest performance, although C-IBS can offer some improvement.

For leakage, the order of preference is reversed. IBS and C-IBS do not amplify bias, as shown in Fig. 10. Rather the opposite: they remove all bias. For the two other schemes: the larger *Loss*, the more bias amplification. Global thresholding amplifies bias the most.

VIII. Detect Manipulation and Nonmalicious Failures

The IC can employ a scheme to detect helper data manipulation and abort its operation prematurely. Four schemes are represented in Fig. 11. Minor variations can be applied to

[8]Choosing other codewords would lower the performance.

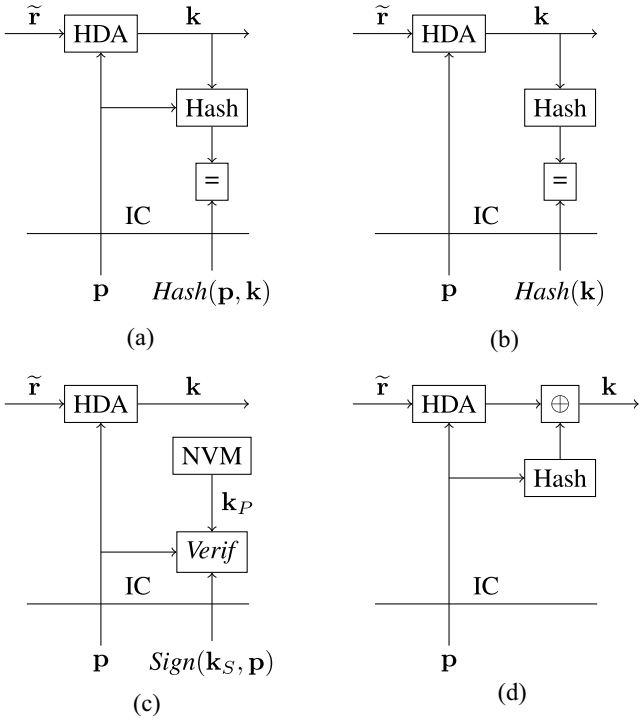

Fig. 11. Helper data manipulation detection. (a) keyed hash of the helper data. (b) Hashing the key. (c) Digital signature of the helper data. (d) Helper data directly affecting the key.

all, but our main purpose here is to provide an overview of the design flavors and associated issues. All require a cryptographic function, but resources can be shared with the application or the entropy compression step. There is a thin border with schemes for detecting nonmalicious failures, due to PUF noisiness. One can then either warn the application that the key is invalid or launch a retrial.

The scheme in Fig. 11(a) stores $Hash(\mathbf{k}, \mathbf{p})$ as public helper data [6]. It offers full protection against attacks via the application interface: any modification to $\mathbf{p}$ will be detected with extremely high probability. Instead of the key $\mathbf{k}$, one can also use the error-corrected response $\mathbf{r}$ as a secret hash input. The latter protects the entropy compression hash against manipulation-enhanced physical attacks. The former does not offer any protection. Both offer full failure detection: a modification in $\mathbf{k}$ is almost certainly detected.

Fig. 11(b) represents a simplification of the former scheme: only a hash of the key is stored [20], [31]. Unfortunately, there is no protection against application interface attacks which rely on the *AppStrong* assumption. Note that several HDAs have already been attacked under this assumption, as summarized in this paper. These attacks only require the measurement of failure probabilities for the original key (it does not matter whether the observable failure is generated by either the application or the detection scheme). Furthermore, no protection against manipulation-enhanced physical attacks is offered. There is full failure detection though.

Fig. 11(c) represents the use of digital signatures [39]. Although public key cryptography is perhaps not so very lightweight. One assumes tamper-proof storage of the public key (no manipulation), partly opposing the advantage of PUF technology. One can employ one-time programmable NVM as well as hard-wired read-only NVM, but the latter would imply the same key pair for many ICs: there are large-scale consequences then in case of compromise. During enrollment, one signs $\mathbf{p}$ with the private key. There is a full protection against manipulation-enhanced physical attacks. In the original proposal, there is a second signature for the key $\mathbf{k}$ too. This would provide full failure detection.

Fig. 11(d) represents a scheme where the key depends on the helper data [17]. There are no strong security guarantees under the *AppWeak* assumption. e.g., related-key attacks on the application. A minimum of helper data seems to be the main benefit. We note that the XOR could have been omitted by hashing $\mathbf{p}$ together with HDA output: this would consume less resources and perhaps offer better security (no related-key attacks). There is no detection of failures.

IX. Open Problems

We identify various topics with little coverage in open research so far. They are suggested as further work.

A. Global Optimization

HDA design comprehends method exploration as well as configuring parameters. Experience and insights of the designer seem to determine global decision making. We argue that computer-aided optimization might lead to better results. This implicates the automated evaluation of leakage, failure rates, bit loss, implementation complexity, etc., for a given design instantiation. Analytic formulas as well as simulations can be employed.

B. Secure Testing

For industrial applications, secure post-manufacturing testing of a HDA is a must. The traditional trade-off between testability and security still applies. However, the ability to correct errors is a major complication for obtaining a high fault coverage. A built-in self-test for a conventional fuzzy extractor has been designed in [7]. HDA components perform part of the testing functionality, to minimize overhead.

C. Physical Attacks

HDAs are not necessarily secure against physical attacks. Side-channel analysis has been applied successfully to a code-offset secure sketch and Toeplitz hash function in [28]. A masking countermeasure for the code-offset secure sketch, taking benefit of code linearity, has been presented in [29]. One mixes a random codeword into the HDA input, by performing $\mathbf{p} \oplus \mathbf{w}_{\text{RAND}}$, with compensation later-on. The generation of $\mathbf{w}_{\text{RAND}}$ requires a secure true random number generator in addition to message encoding. Extension to Toeplitz hashing is possible due to internal linearity.

D. Reverse Concept

For certain applications, an IC might communicate with a resource-rich server, using symmetric key cryptography. A reverse fuzzy extractor [42] reduces the workload of the IC by shifting *Rep* to the server and implementing *Gen* on the

IC instead, as the latter consumes less resources. Although originally proposed as part of an authentication protocol, it directly applies to key generation. The IC could generate a nondeterministic key $\widetilde{\mathbf{k}} \leftarrow Hash(\widetilde{\mathbf{r}})$ and send helper data to the server for reconstruction. However, repeated helper data exposure might increase the leakage threat. Therefore, one recommends the syndrome secure sketch, as the $n-k$ leakage upper bound is provably still valid [5]. Extension to other HDA methods is to be studied.

E. Correlations

As in prior work, correlations are not taken into account properly. Quantifying correlations is hard and PUF-dependent. There is a work in the construction of integrated PUF models, capturing bit error rates, correlations, and possibly other effects. Furthermore, leakage of HDA components should be reevaluated: correlations are expected to increase the min-entropy loss. Finally, data-dependencies are not taken into account when correcting errors: there is margin to improve performance [35].

F. Leakage Reduction

Although leakage can be compensated, one might still aim to reduce it. We question the practical value of two rather unusual approaches.

In [45], the leakage of two HDAs is evaluated: the code-offset construction and IBS. Their calculation assumes an attacker to have knowledge of $\mathbf{r}$. Therefore, discarded IBS bits are regarded as leakage too, although helper data does not reveal their value. We argue that leakage gets irrelevant if cause and effect are reversed. A supposedly more secure version of IBS is proposed: candidate bits are discarded with a fixed probability (e.g., 50%) before applying the standard procedure based on reliability and nominal value. Under the conventional attacker model as described in Section IV, this would be an efficiency burden only, not improving security.

In [34], one spams the attacker with fake helper data instances as a form of obfuscation. These instances have the same probability distribution as the genuine one and are all sorted according to a random permutation. The IC is able to distinguish fake and genuine via reconstruction trial-and-error. Manipulation detection is crucial: otherwise an attacker can do the same, by modifying an instance and observing P_{FAIL}. The method might be relevant if the total entropy of the PUF is hardly sufficient to generate a secret key: the loss of valuable entropy is reduced. However, a typical PUF is not bothered by this limitation. The scheme is actually very costly compared to an increase in the number of PUF bits. The workload for the resource-constrained IC and the resource-rich attacker scales roughly the same in terms of the number of fake instances.

X. Conclusion

We provided a first in-depth overview on HDAs for PUF-based key generation, comprehending a decade of research. Furthermore, our analysis revealed various new threats regarding helper data leakage and manipulation. Finally, we identified hiatuses/open problems in existing literature, offering a foundation for future work.

Appendix

A. Proof of Failure Rate Using Secure Sketch

We derive a proof for (16), holding under the i.i.d. reliability assumption. All incoming bits will then obey a certain PDF $f(P_e^{IN})$. We make use of the Poisson-binomial distribution, as defined by (33). This generalizes the binomial distribution, with the success probabilities $p_1, p_2, \ldots, p_n$ of each trial not necessarily equal

$$\begin{aligned} &F_{\mathcal{P}}\left(k';\langle p_1,p_2,\ldots,p_n\rangle\right) \\ &\quad=\sum_{k=0}^{k'} f_{\mathcal{P}}\left(k;\langle p_1,p_2,\ldots,p_n\rangle\right) \\ &\quad=\sum_{k=0}^{k'}\frac{1}{k!}\frac{\mathrm{d}^k\prod_{i=1}^{n}(1-p_i(1-t))}{\mathrm{d}t^k}|_{t=0}. \end{aligned} \tag{33}$$

$$\begin{aligned} &E_{\mathcal{V}}\left[P_{FAIL}\right] \\ &\quad=1-\int_0^1\int_0^1\ldots\int_0^1\left(\prod_{i=1}^{n}f(P_{ei})\right) \\ &\qquad F_{\mathcal{P}}\left(t;\langle P_{e1},\ldots,P_{en}\rangle\right)\,\mathrm{d}P_{e1}\mathrm{d}P_{e2}\ldots\mathrm{d}P_{en} \\ &\quad=1-F_{\mathcal{P}}\left(t;\left\langle\int_0^1 f(P_{e1})\mathrm{d}P_{e1},\ldots,\int_0^1 f(P_{en})\mathrm{d}P_{en}\right\rangle\right) \\ &\quad=1-F_{\mathcal{P}}\left(t;\langle E[P_e],\ldots,E[P_e]\rangle\right) \\ &\quad=1-F_{\mathcal{B}}\left(t;n,E[P_e]\right). \end{aligned} \tag{34}$$

B. Proof of REP Leakage Using Secure Sketch

We derive a proof for (18). First, we apply definition (6) and simplify using Bayes's theorem

$$\begin{aligned} &\widetilde{H}_\infty\left(\left(\mathbf{r}^{OUT}|B^{OUT}\right)|\mathbf{p}\right) \\ &\quad=-\log_2\left(E_{\mathbf{p}}\left[\max_{\mathbf{r}^{OUT}}P\left(\left(\mathbf{r}^{OUT}|B^{OUT}\right)|\mathbf{p}\right)\right]\right) \\ &\quad=-\log_2\left(\sum_{\mathbf{p}}\max_{\mathbf{r}^{OUT}}P\left(\mathbf{r}^{OUT}|B^{OUT}\right)P\left(\mathbf{p}|\mathbf{r}^{OUT}\right)\right). \end{aligned} \tag{35}$$

For the code-offset construction, we assume the enrollment to be ideal: $P(\mathbf{w}|\mathbf{r}^{OUT}) = 1/2$. We assume the following codewords: $\left(0\,0\ldots0\right)$ and $\left(1\,1\ldots1\right)$. The leakage does not depend on this, but it makes the derivation more comprehensible. We elaborate (35) as shown below, introducing the variable $\Delta = HW(\mathbf{p})$

$$\begin{aligned} &=-\log_2\left(\sum_{\Delta=0}^{n}\binom{n}{\Delta}\frac{1}{2}\max\left(\left(B^{OUT}\right)^{\Delta}\left(1-B^{OUT}\right)^{n-\Delta}\right.\right. \\ &\qquad\left.\left.\left(1-B^{OUT}\right)^{\Delta}\left(B^{OUT}\right)^{n-\Delta}\right)\right) \\ &=-\log_2\left(F_{\mathcal{B}}\left(\frac{n-1}{2};n,\min\left(B^{OUT},1-B^{OUT}\right)\right)\right). \end{aligned} \tag{36}$$

For the syndrome construction, the enrollment is fully deterministic: $\max P(\mathbf{p}|\mathbf{r}^{OUT}) = 1$. We assume the following helper

data: $\mathbf{p} = \left(r_1 \oplus r_2 \; r_1 \oplus r_3 \ldots r_1 \oplus r_n\right)$. Again, the leakage does not depends on this, but it makes the derivation more comprehensible. We elaborate (35) as shown below, again introducing the variable Δ, and obtain an equality with (36)

$$= -\log_2 \left(\sum_{\Delta=0}^{n-1} \binom{n-1}{\Delta} \max \left(\left(B^{OUT}\right)^{\Delta} \left(1 - B^{OUT}\right)^{n-\Delta} \left(1 - B^{OUT}\right)^{\Delta} \left(B^{OUT}\right)^{n-\Delta} \right) \right). \quad (37)$$

C. REP Leakage Using Soft-Decision Coding

Equation (38) represents the min-entropy of outgoing soft-decision bits when including joint leakage of $\mathbf{p}$ and $\mathbf{P_e}$. Unfortunately, the expression does not simplify well

$$\begin{aligned}
&\widetilde{H}_{\infty} \left(\left(\mathbf{r}^{OUT} | B^{OUT} \right) | \left(\mathbf{p}, \mathbf{P_e} \right) \right) \\
&= -\log_2 \left(E_{\mathbf{p},\mathbf{P_e}} \left[\max_{\mathbf{r}^{OUT}} P \left(\left(\mathbf{r}^{OUT} | B^{OUT} \right) | \left(\mathbf{p}, \mathbf{P_e} \right) \right) \right] \right) \\
&= -\log_2 \left(\frac{1}{(\sigma_{\mathcal{V}})^n} \sum_{i=0}^{n} \binom{n}{i} \int_T^{\infty} f_{\mathcal{N}} \left(\frac{v_1}{\sigma_{\mathcal{V}}} \right) \ldots \right. \\
&\qquad \int_T^{\infty} f_{\mathcal{N}} \left(\frac{v_i}{\sigma_{\mathcal{V}}} \right) \int_{-\infty}^{T} f_{\mathcal{N}} \left(\frac{v_{i+1}}{\sigma_{\mathcal{V}}} \right) \cdots \int_{-\infty}^{T} f_{\mathcal{N}} \left(\frac{v_n}{\sigma_{\mathcal{V}}} \right) \\
&\qquad \max \left(B_1^{OUT} \ldots B_i^{OUT} \left(1 - B_{i+1}^{OUT}\right) \ldots \left(1 - B_n^{OUT}\right) \right. \\
&\qquad \left. \left(1 - B_1^{OUT}\right) \ldots \left(1 - B_i^{OUT}\right) B_{i+1}^{OUT} \ldots B_n^{OUT} \right) / \\
&\qquad \left(B_1^{OUT} \ldots B_i^{OUT} \left(1 - B_{i+1}^{OUT}\right) \ldots \left(1 - B_n^{OUT}\right) \right. \\
&\qquad + \left. \left(1 - B_1^{OUT}\right) \ldots \left(1 - B_i^{OUT}\right) B_{i+1}^{OUT} \ldots B_n^{OUT} \right) \\
&\qquad \left. \mathrm{d}v_n \mathrm{d}v_{n-1} \ldots \mathrm{d}v_2 \mathrm{d}v_1 \right). \quad (38)
\end{aligned}$$

REFERENCES

[1] M. Ahsanullah, V. B. Nevzorov, and M. Shakil, "An introduction to order statistics," in *Atlantis Studies in Probability and Statistics*, vol. 3. Amsterdam, The Netherlands: Atlantis Press, 2013.

[2] F. Armknecht, R. Maes, A.-R. Sadeghi, B. Sunar, and P. Tuyls, "Memory leakage-resilient encryption based on physically unclonable functions," in *Advances in Cryptology—ASIACRYPT* (LNCS 5912). Berlin, Germany: Springer, Dec. 2009, pp. 685–702.

[3] M. Bhargava and K. Mai, "An efficient reliable PUF-based cryptographic key generator in 65nm CMOS," in *Proc. Design Autom. Test Europe (DATE)*, Dresden, Germany, Mar. 2014, pp. 1–6.

[4] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient helper data key extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems* (LNCS 5154). Berlin, Germany: Springer, Aug. 2008, pp. 181–197.

[5] X. Boyen, "Reusable cryptographic fuzzy extractors," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, Washington, DC, USA, Oct. 2004, pp. 82–91.

[6] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky, and A. Smith, "Secure remote authentication using biometric data," in *Advances in Cryptology—EUROCRYPT* (LNCS 3494). Berlin, Germany: Springer, May 2005, pp. 147–163.

[7] M. Cortez, G. Roelofs, S. Hamdioui, and G. Di Natale, "Testing PUF-based secure key storage circuits," in *Proc. Design Autom. Test Europe (DATE)*, Dresden, Germany, Mar. 2014, pp. 1–6.

[8] J. Delvaux and I. Verbauwhede, "Key-recovery attacks on various RO PUF constructions via helper data manipulation," in *Proc. Design Autom. Test Europe (DATE)*, Dresden, Germany, Mar. 2014, Art. ID 74.

[9] J. Delvaux and I. Verbauwhede, "Attacking PUF-based pattern matching key generators via helper data manipulation," in *Topics in Cryptology—CT-RSA* (LNCS 8366). Cham, Switzerland: Springer, Feb. 2014, pp. 106–131.

[10] J. Delvaux and I. Verbauwhede, "Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise," in *Proc. Int. Symp. Hardw.-Orient. Security Trust (HOST)*, Austin, TX, USA, Jun. 2013, pp. 137–142.

[11] J. Delvaux and I. Verbauwhede, "Fault injection modeling attacks on 65nm arbiter and RO sum PUFs via environmental changes," in *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 61, no. 6, pp. 1701–1713, Jun. 2014.

[12] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology—EUROCRYPT* (LNCS 3027). Berlin, Germany: Springer, May 2004, pp. 523–540.

[13] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM J. Comput.*, vol. 38, no. 1, pp. 97–139, Mar. 2008.

[14] B. Gassend, "Physical random functions," Master's thesis, Dept. Electr. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, Feb. 2003.

[15] B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, Washington, DC, USA, Nov. 2002, pp. 148–160.

[16] M. Hiller, D. Merli, F. Stumpf, and G. Sigl, "Complementary IBS: Application specific error correction for PUFs," in *Proc. Int. Symp. Hardw.-Orient. Security Trust (HOST)*, San Francisco, CA, USA, Jun. 2012, pp. 1–6.

[17] M. Hiller, M. Weiner, L. R. Lima, M. Birkner, and G. Sigl, "Breaking through fixed PUF block limitations with differential sequence coding and convolutional codes," in *Proc. Int. Workshop Trustworthy Embedded Devices (TrustED)*, Berlin, Germany, Nov. 2013, pp. 43–54.

[18] M. Hiller and G. Sigl, "Increasing the efficiency of syndrome coding for PUFs with helper data compression," in *Proc. Design Autom. Test Europe (DATE)*, Dresden, Germany, Mar. 2014, pp. 1–6.

[19] M. Hofer and C. Böhm, "An alternative to error correction for SRAM-like PUFs," in *Cryptographic Hardware and Embedded Systems* (LNCS 6225). Berlin, Germany: Springer, Aug. 2010, pp. 335–350.

[20] T. Kevenaar *et al.*, "Robust and secure biometrics: Some application examples," in *Proc. Inf. Security Solutions Europe Conf. (ISSE)*, Rome, Italy, Oct. 2006, pp. 196–203.

[21] P. Koeberl, J. Li, A. Rajan, and W. Wu, "Entropy loss in PUF-based key generation schemes: The repetition code pitfall," in *Proc. Int. Symp. Hardw.-Orient. Security Trust (HOST)*, Arlington, VA, USA, May 2014, pp. 44–49.

[22] J.-P. Linnartz and P. Tuyls, "New shielding functions to enhance privacy and prevent misuse of biometric templates," in *Audio- and Video-Based Biometric Person Authentication (AVBPA)* (LNCS 2688). Berlin, Germany: Springer, Jun. 2003, pp. 393–402.

[23] R. Maes, "An accurate probabilistic reliability model for silicon PUFs," in *Cryptographic Hardware and Embedded Systems* (LNCS 8086). Berlin, Germany: Springer, Aug. 2013, pp. 73–89.

[24] R. Maes, P. Tuyls, and I. Verbauwhede, "A soft decision helper data algorithm for SRAM PUFs," in *Proc. Int. Symp. Inf. Theory (ISIT)*, Seoul, Korea, Jun. 2009, pp. 2101–2105.

[25] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Systems* (LNCS 5747). Berlin, Germany: Springer, Sep. 2009, pp. 332–347.

[26] R. Maes and V. van der Leest, "Countering the effects of silicon aging on SRAM PUFs," in *Proc. Int. Symp. Hardw.-Orient. Security Trust (HOST)*, Arlington, VA, USA, May 2014, pp. 148–153.

[27] R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: A fully functional PUF-based cryptographic key generator," in *Cryptographic Hardware and Embedded Systems* (LNCS 7428). Berlin, Germany: Springer, Sep. 2012, pp. 302–319.

[28] D. Merli, D. Schuster, F. Stumpf, and G. Sigl, "Side-channel analysis of PUFs and fuzzy extractors," in *Proc. Int. Conf. Trust Trustworthy Comput. (TRUST)* (LNCS 6740). Pittsburgh, PA, USA, Jun. 2011, pp. 33–47.

[29] D. Merli, F. Stumpf, and G. Sigl, "Protecting PUF error correction by codeword masking," in *Proc. IACR Cryptology ePrint Archive*, Buenos Aires, Argentina, 2013, p. 334.

[30] E. Öztürk, G. Hammouri, and B. Sunar, "Towards robust low cost authentication for pervasive devices," in *Proc. Int. Conf. Pervasive Comput. Commun. (PerCom)*, Hong Kong, Mar. 2008, pp. 170–178.
[31] Z. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching," in *Proc. Int. Symp. Hardw.-Orient. Security Trust (HOST)*, San Diego, CA, USA, Jun. 2011, pp. 128–133.
[32] Z. Paral, S. Devadas, and Verayo Inc., "Reliable PUF value generation by pattern matching," WO Patent 2 012 099 657, Jul. 2012.
[33] D. Schellekens, "Design and analysis of trusted computing platforms," Ph.D. dissertation, Dept. Electr. Eng., KU Leuven, Leuven, Belgium, 2013.
[34] B. Škorić and N. de Vreede, "The spammed code offset method," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 5, pp. 875–884, May 2014.
[35] B. Škorić and P. Tuyls, "An efficient fuzzy extractor for limited noise," in *Proc. Symp. Inf. Theory Benelux*, Eindhoven, The Netherlands, 2009, pp. 193–200.
[36] B. Škorić, P. Tuyls, and W. Ophey, "Robust key extraction from physical uncloneable functions," in *Applied Cryptography and Network Security* (LNCS 3531). Berlin, Germany: Springer, Jun. 2005, pp. 407–422.
[37] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. Design Autom. Conf. (DAC)*, San Diego, CA, USA, Jun. 2007, pp. 9–14.
[38] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas, "Design and implementation of the AEGIS single-chip secure processor using physical random functions," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Madison, WI, USA, Jun. 2005, pp. 25–36.
[39] P. Tuyls *et al.*, "Read-proof hardware from protective coatings," in *Cryptographic Hardware and Embedded Systems* (LNCS 4249). Berlin, Germany: Springer, Oct. 2006, pp. 369–383.
[40] P. Tuyls, B. Škorić, and T. Kevenaar, *Security With Noisy Data: Private Biometrics, Secure Key Storage and Anti-Counterfeiting*. Secaucus, NJ, USA: Springer, 2007.
[41] V. van der Leest, B. Preneel, and E. van der Sluis, "Soft decision error correction for compact memory-based PUFs using a single enrollment," in *Cryptographic Hardware and Embedded Systems* (LNCS 7428). Berlin, Germany: Springer, Sep. 2012, pp. 268–282.
[42] A. Van Herrewege *et al.*, "Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs," in *Financial Cryptography and Data Security* (LNCS 7397). Berlin, Germany: Springer, Feb. 2012, pp. 374–389.
[43] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 48–65, Jan. 2010.
[44] M.-D. Yu, D. M'Raïhi, S. Devadas, and I. Verbauwhede, "Security and reliability properties of syndrome coding techniques used in PUF key generation," in *Proc. Gov. Microcircuit Appl. Critical Technol. Conf. (GOMACTech)*, Las Vegas, NV, USA, Mar. 2013, pp. 1–4.
[45] M.-D. Yu, D. M'Raïhi, R. Sowell, and S. Devadas, "Lightweight and secure PUF key storage using limits of machine learning," in *Cryptographic Hardware and Embedded Systems* (LNCS 6917). Berlin, Germany: Springer, Sep. 2011, pp. 358–373.
[46] M.-D. Yu, R. Sowell, A. Singh, D. M'Raïhi, and S. Devadas, "Performance metrics and empirical results of a PUF cryptographic key generation ASIC," in *Proc. Int. Symp. Hardw.-Orient. Security Trust (HOST)*, San Francisco, CA, USA, Jun. 2012, pp. 108–115.
[47] H. Zhou, A. Jiang, and J. Bruck, "Nonuniform codes for correcting asymmetric errors in data storage," *IEEE Trans. Inf. Theory*, vol. 59, no. 5, pp. 2988–3002, May 2013.

Jeroen Delvaux received the master's degree in electrical engineering and the post-graduate degree in biomedical engineering, both from the Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 2010 and 2012, respectively. He is currently pursuing the Ph.D. degrees from the COSIC Research Group of the Electrical Engineering Department, KU Leuven, and the LoCCS Research Group of the Computer Science and Engineering Department, Shanghai Jiao Tong University, Shanghai, China.

He was a Research and Development Engineer at KLA-Tencor, Leuven, Belgium, for 19 months. His current research interests include the security and efficiency analysis of PUF-based systems.

Dawu Gu received the master's and Ph.D. degrees in cryptography from Xidian University, Xi'an, China, in 1995 and 1998, respectively.

He is a Professor with the LoCCS Research Group of the Computer Science and Engineering Department, Shanghai Jiao Tong University, Shanghai, China. His current research interests include cryptographic algorithms, side-channel attacks, and software security.

Dries Schellekens received the master's and Ph.D. degrees in electrical engineering from the Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 2002 and 2013, respectively.

He is a Security Expert with Septentrio Satellite Navigation, Leuven. He was a Post-Doctoral Researcher at the COSIC Research Group of the Electrical Engineering Department, KU Leuven.

Ingrid Verbauwhede (M'92–SM'00–F'13) received the Ph.D. degree from the Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, and Interuniversity Microelectronics Centre, Leuven, in 1991.

She is a Professor at the COSIC Research Group of the Electrical Engineering Department, KU Leuven, where she leads the embedded systems team. She is also an Associate Professor at the Electrical Engineering Department, University of California, Los Angeles, Los Angeles, CA, USA. She was a Post-Doctoral Visiting Researcher and a Lecturer at the University of California, Berkeley, Berkeley, CA, USA, and was a Principal Engineer at TCSI, Alameda, CA, USA, and Atmel, San Jose, CA, USA. Her current research interests include the design of secure embedded circuits and systems.