

# A Soft Decision Helper Data Algorithm for SRAM PUFs

Roel Maes

ESAT: SCD/COSIC and IBBT

Katholieke Universiteit Leuven

Leuven, Belgium

Email: Roel.Maes@esat.kuleuven.be

Pim Tuyls

Intrinsic-ID

Eindhoven, The Netherlands

Email: Pim.Tuyls@intrinsic-id.com

Ingrid Verbauwhede

ESAT: SCD/COSIC and IBBT

Katholieke Universiteit Leuven

Leuven, Belgium

Email: Ingrid.Verbauwhede@esat.kuleuven.be

**Abstract**—In this paper we propose the idea of using soft decision information in helper data algorithms (HDA). We derive and verify a distribution for the responses of SRAM-based physical unclonable functions (PUFs) and show that soft decision information becomes available without loss in min-entropy of the fuzzy secret. This significantly improves the implementation overhead of using an SRAM PUF + HDA for key generation compared to previous constructions.

## I. INTRODUCTION

### A. Physical Unclonable Functions

Physical Unclonable Functions or PUFs [9] have been introduced as a new security primitive that makes it possible to inseparably bind digital keys and identifiers to actual hardware implementations. The functionality of a PUF depends on unique intrinsic properties of the implementation at a sub-micron level. Due to the practical infeasibility of controlling physical parameters at this scale, the exact functionality implemented by a PUF is deemed *unclonable*. In practice, a *challenge* can be applied to a PUF implementation and a corresponding *response* is produced. The unique challenge-response behavior of a particular PUF bares a strong resemblance to biometric identifiers from human beings.

PUFs can be used as identifiers, *e.g.* to act as a counter-measure for counterfeiting, because their challenge-response behavior is unclonable, even to the genuine manufacturer. They can moreover be used to generate keys for cryptographic purposes, effectively binding the key to the hardware. Since they are able to regenerate the same key, they can be used to store keys on devices without conventional non-volatile memory. The strong dependence on the internal parameters makes a PUF a highly tamper evident key storage.

Like biometrical identifiers, PUF responses are not reliably reproducible. This is due to noise present in the response measurement. Moreover, PUF responses are often not uniformly distributed. This poses problems when a PUF is used to generate a cryptographic key, since this needs to be robust and have full entropy. Post-processing is required in that case. Helper data algorithms (HDA) have been introduced as a primitive for turning *fuzzy*, non-uniform data into cryptographic keys.

### B. Previous Work

A number of different PUF implementations are described in literature. Optical PUFs [9] are based on the complex interaction between randomly solved particles in a transparent medium and an incident laser beam. A Coating PUF [11] measures the capacitance of randomly placed dielectric grains in the top coating of an integrated circuit. More interesting implementations of PUFs use the random variations that are *intrinsic* to a particular production process, hence avoiding the explicit introduction of random particles. These intrinsic variations are called *manufacturing variability*, and *e.g.* in the case of ICs, they have a noticeable effect on the operation. Arbiter PUFs [7] and Ring Oscillator PUFs [4] measure the effect of random manufacturing variability on the delay of a digital circuit. SRAM PUFs [5] and Butterfly PUFs [6] use the manufacturing dependency of the settling state of bistable circuits.

*Helper data algorithms* for generating cryptographic keys from fuzzy and non-uniform secrets were first introduced by Linnartz *et al.* [8] as *shielding functions* and Dodis *et al.* [3] as *fuzzy extractors*. Both constructions are largely equivalent. The secure use of fuzzy extractors was discussed in [2]. A first efficient hardware implementation of a HDA is given in [1].

## II. HELPER DATA CONSTRUCTIONS

### A. Fuzzy Extractor Definition

The most commonly used formal definition of a HDA is that of a fuzzy extractor from [3]:

**Definition 1:** An  $(\mathbf{M}, m, \ell, t, \epsilon)$  fuzzy extractor is given by two procedures (Gen, Rep):

- 1) Gen is a probabilistic generation procedure, which on input  $w \in \mathbf{M}$  outputs an "extracted" string  $R \in \{0, 1\}^\ell$  and a public string  $P$ . We require that for any distribution  $W$  on  $\mathbf{M}$  of min-entropy  $m$ , if  $\langle R, P \rangle \leftarrow \text{Gen}(W)$ , then we have  $\text{SD}(\langle R, P \rangle, \langle U_\ell, P \rangle) \leq \epsilon$ , with  $U_\ell$  the uniform distribution on  $\{0, 1\}^\ell$ .
- 2) Rep is a deterministic reproduction procedure allowing to recover  $R$  from the corresponding public string  $P$  and any vector  $w'$  close to  $w$ : for all  $w, w' \in \mathbf{M}$  satisfying  $\text{dist}[w; w'] \leq t$ , if  $\langle R, P \rangle \leftarrow \text{Gen}(w)$ , then we have  $\text{Rep}(w', P) = R$ .  $\text{dist}[\cdot; \cdot]$  is a distance measure on  $\mathbf{M}$ .

In the *generation phase*, a fuzzy source produces an amount of fuzzy data  $w$  from which a (secret) key  $R$  and (public) *helper data*  $P$  are extracted. In the *reproduction phase* the same data is measured, but due to the fuzzy nature of the source, the produced response  $w'$  will not be exactly the same. However, if  $w'$  lies sufficiently close to  $w$ , the reproduction procedure is able to reproduce the secret key  $R$  with the help of the previously produced helper data  $P$ . A fuzzy extractor also provides some security guarantees on the extracted key. If the fuzzy secret  $w$  comes from a distribution  $W$  with sufficient min-entropy  $m$ , then the distribution of the extracted key  $R$  is  $\epsilon$ -close to the uniform distribution. Moreover, this condition still holds when the helper data  $P$  is publicly known. This means that the helper data  $P$  can only leak a limited amount of information about the fuzzy secret  $w$ . This leakage is expressed in terms of min-entropy and called *min-entropy loss*.

A fuzzy extractor consists of two stages. In the first stage or *information reconciliation*, possible bit errors in the fuzzy data are corrected to form a robust bit string. The second stage or *privacy amplification* compresses this robust bit string to obtain a full entropy key.

#### B. Information Reconciliation with the Code-offset Technique

The *code-offset technique* [3] is a method for performing information reconciliation based on error-correcting block codes. It can reconstruct a secret bit string  $w$  from a fuzzy version  $w'$  of the bit string, as long as the Hamming distance between  $w$  and  $w'$  is limited to  $t$ . In the generation phase,  $w \in \{0, 1\}^n$  is measured and a codeword  $c$  is picked at random from a linear block code  $C_{n,k}$ <sup>1</sup>. The bit offset between  $w$  and  $c$  becomes the helper data,  $h = c \oplus w$ , which is made publicly available. In the reproduction phase, a fuzzy version  $w' \in \{0, 1\}^n$  is obtained and  $c' = w' \oplus h$  is calculated. If  $\text{dist}[c; c'] \equiv \text{dist}[w; w'] \leq t$ , then  $c'$  can be corrected to  $c$ . The original secret  $w = c \oplus h$  can be calculated. From this construction it is clear that the min-entropy of  $w$  decreases by  $n - k$ , by making the helper data  $h$  publicly available. The remaining min-entropy in  $w$  after information reconciliation is  $m' = m - n + k$ .

#### C. Privacy Amplification with 2-Universal Hash Functions

By compressing the non-uniform but partially secret  $w$ , an  $\epsilon$ -uniform and fully secret key  $R$  can be obtained, as long as the remaining min-entropy  $m'$  of  $w$  is large enough. According to the left-over hash lemma, this can be done using 2-universal hash functions. Since the focus of this work is on the information reconciliation stage, we don't go into detail here.

### III. SRAM PUF RESPONSE MODEL

#### A. SRAM PUF Operation

Static Random Access Memory or SRAM is a type of volatile memory abundantly used in digital devices. In its popular form, one SRAM cell occupies six transistors that

<sup>1</sup> $C_{n,k}$  is a linear block code of length  $n$  and dimension  $k$  and minimum distance  $d \geq (2t + 1)$ .

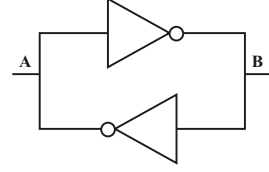


Fig. 1. Logical circuit implementing an SRAM cell.

constitute a logic circuit with two cross coupled inverters, as shown in Figure 1. Such a circuit can assume two logically stable states, *i.e.*  $(A, B) = (0, 1)$  and  $(A, B) = (1, 0)$ , and hence stores one binary digit. External connections can inspect the current state of the cell (read) or alter it (write). Many of these cells are stacked in large arrays, able to store millions of bits.

When powered up, the behavior of an SRAM cell is undefined. Ideally, consisting of two physically identical inverters, the SRAM cell will be in a logically undetermined and unstable state right after power-up. In reality, due to physical mismatch on the inverters and due to electrical noise, the cell will quickly converge to one of the two stable states. Since both the inverter's mismatch and the electrical noise are governed by stochastic processes, the power-up state of an SRAM cell will be random. The mismatch between the inverters is random, but fixed once the cell is manufactured. The electrical noise is a different random component for every cell and at every moment in time. Experiments done in [5] show that, although the power-up state of a random SRAM cell chosen from a large memory is random, one particular cells always tends to start up in the same state. Some cells always power up storing a '0', others always power up storing a '1'. A minority of cells have no distinct preference and power up as '0' sometimes and as '1' other times. It is clear that the power-up preference of a single cell toward '0' or '1' is determined by the static random inverter mismatch inside the cell. For some cells, this static mismatch is small, and the dynamic random noise at the time of power-up will determine the power-up state.

The power-up state of an SRAM cell hence gives a (noisy) measure for the physical mismatch intrinsically present in the cell. Since this mismatch is caused by uncontrollable factors during the production process, it is a form of manufacturing variability. The power-up state of an SRAM cell can hence be considered a PUF response. An SRAM PUF is constructed as follows: a binary challenge selects a specific address range inside a large SRAM memory, and the responses are the power-up states of the cells at these addresses. The presence of noise in the measurement introduces bit errors on the response.

#### B. Response Model

The power-up state of an SRAM cell is influenced by two stochastic components, *i.e.* the mismatch between the electrical parameters of the inverters and the amplitude of the electrical noise at the time of power-up. We describe both components using stochastic variables, respectively  $\mathcal{M}$  and  $\mathcal{N}$ . A value

$m_i \leftarrow \mathcal{M}$  is i.i.d. sampled every time a new SRAM cell  $i$  is manufactured. A value  $n_i^{(j)} \leftarrow \mathcal{N}$  is i.i.d. sampled at the  $j$ -th power-up of SRAM cell  $i$ <sup>2</sup>. The power-up state of cell  $i$  at the  $j$ -th power-up is written as  $r_i^{(j)}$  and follows from  $m_i$  and  $n_i^{(j)}$ :

$$r_i^{(j)} = \begin{cases} 0 & , \text{ if } m_i + n_i^{(j)} > T, \\ 1 & , \text{ if } m_i + n_i^{(j)} \leq T, \end{cases} \quad (1)$$

with  $T$  a *threshold parameter* for a specific SRAM technology. It is reasonable to assume a normal distribution for both  $\mathcal{M}$  and  $\mathcal{N}$ .  $\mathcal{M}$  follows a normal distribution  $\varphi_{\mu_{\mathcal{M}}, \sigma_{\mathcal{M}}}$  with mean  $\mu_{\mathcal{M}}$  and standard deviation  $\sigma_{\mathcal{M}}$ .  $\mathcal{N}$  follows a normal distribution  $\varphi_{0, \sigma_{\mathcal{N}}}$  with mean 0 and standard deviation  $\sigma_{\mathcal{N}}$ .

### C. Response Distribution

For an SRAM cell  $i$ , we define the *one-probability* as:

$$p_{r_i} \stackrel{\text{def}}{=} \Pr_j \left( r_i^{(j)} = 1 \right), \quad (2)$$

Using Eq. (1) and the normal distribution assumption for  $\mathcal{N}$ , this can be written as:

$$p_{r_i} = \Phi \left( \frac{T - m_i}{\sigma_{\mathcal{N}}} \right), \quad (3)$$

with  $\Phi$  the standard normal cumulative distribution function. It is clear that this one-probability is itself a sample of a stochastic variable  $\mathcal{P}_r$  when considering the whole population of cells. We derive the distribution of this variable by starting from the definition of its cumulative distribution function or  $\mathbf{cdf}_{\mathcal{P}_r}$  and using Eq. (3):

$$\begin{aligned} \mathbf{cdf}_{\mathcal{P}_r}(x) &\stackrel{\text{def}}{=} \Pr_i (p_{r_i} \leq x) = \Pr(\mathcal{P}_r \leq x), \\ &= \Pr \left( \Phi \left( \frac{T - \mathcal{M}}{\sigma_{\mathcal{N}}} \right) \leq x \right), \\ &= \Phi(\lambda_1 \cdot \Phi^{-1}(x) - \lambda_2), \end{aligned} \quad (4)$$

with the parameters  $\lambda_1 = \frac{\sigma_{\mathcal{N}}}{\sigma_{\mathcal{M}}}$  and  $\lambda_2 = \frac{T - \mu_{\mathcal{M}}}{\sigma_{\mathcal{M}}}$ . The probability density function or  $\mathbf{pdf}_{\mathcal{P}_r}$  is easily derived by differentiation of Eq. (4):

$$\mathbf{pdf}_{\mathcal{P}_r}(x) = \frac{\lambda_1 \cdot \varphi(\lambda_2 - \lambda_1 \cdot \Phi^{-1}(x))}{\varphi(\Phi^{-1}(x))}. \quad (5)$$

The one-probability of an SRAM cell  $i$  can be estimated by rebooting the cell several times and counting the number of times it powers up as a '1'. Figure 2 shows a plot of the histogram of such a measurement done for a large number of cells from a real SRAM memory. Also shown are plots of the fitted distribution given by Eqs. (5) and (4). The proposed model fits the measured values closely. This observation justifies the assumptions we made for the SRAM PUF response model in Section III-B.

<sup>2</sup>It is implicitly assumed that the random noise component follows the same distribution for every cell. We can hence use a single stochastic variable  $\mathcal{N}$  to describe the noise for all cells.

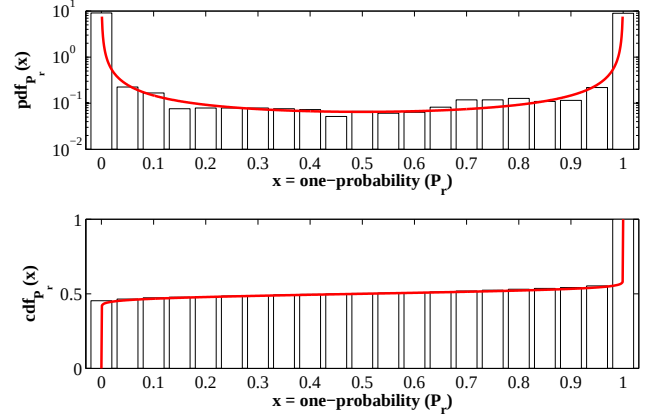


Fig. 2. Histogram and cumulative histogram of the one-probabilities measured on a large number of real SRAM cells. The solid lines show the plots of  $\mathbf{pdf}_{\mathcal{P}_r}(x)$  and  $\mathbf{cdf}_{\mathcal{P}_r}(x)$  for the fitted parameters  $\lambda_1 = 0.065$  and  $\lambda_2 = 0.000$ .

### D. Response Error Distribution

We define what we assume to be the *correct* response  $r_i^C$  of an SRAM cell  $i$ :

$$r_i^C \stackrel{\text{def}}{=} \mathbf{mode}_j \left( r_i^{(j)} \right) \stackrel{\text{def}}{=} \argmax_{r \in \{0, 1\}} \left\{ \Pr_j \left( r_i^{(j)} = r \right) \right\}, \quad (6)$$

An observation  $j$  of an SRAM PUF response of cell  $i$  is a Bernoulli trial, with probability of success equal to the one-probability  $p_{r_i}$  of that cell. This means that  $r_i^C = 0$ , if  $p_{r_i} < \frac{1}{2}$ , and  $r_i^C = 1$ , if  $p_{r_i} > \frac{1}{2}$ .

The *error-probability* of an SRAM cell  $i$  is defined as:

$$p_{e_i} \stackrel{\text{def}}{=} \Pr_j \left( r_i^{(j)} \neq r_i^C \right). \quad (7)$$

Using Eq. (6), this can be written as:

$$p_{e_i} = \min\{p_{r_i}, 1 - p_{r_i}\}. \quad (8)$$

When considering the whole population of cells, we can say  $p_{e_i} \leftarrow \mathcal{P}_e$ , with  $\mathcal{P}_e$  a stochastic variable. We derive the distribution  $\mathbf{cdf}_{\mathcal{P}_e}$  of this variable by using Eq. (8):

$$\begin{aligned} \mathbf{cdf}_{\mathcal{P}_e}(x) &\stackrel{\text{def}}{=} \Pr_i (p_{e_i} \leq x) = \Pr(\mathcal{P}_e \leq x), \\ &= \mathbf{cdf}_{\mathcal{P}_r}(x) + 1 - \mathbf{cdf}_{\mathcal{P}_r}(1 - x). \end{aligned} \quad (9)$$

By differentiation of Eq. (9), we find  $\mathbf{pdf}_{\mathcal{P}_e}$ :

$$\mathbf{pdf}_{\mathcal{P}_e}(x) = \mathbf{pdf}_{\mathcal{P}_r}(x) + \mathbf{pdf}_{\mathcal{P}_r}(1 - x). \quad (10)$$

## IV. SOFT DECISION HELPER DATA ALGORITHM

### A. Motivation

The implementation and timing overhead of using a PUF + HDA for key generation should be very low. The important design criteria for an efficient HDA are hence using as little fuzzy data (*i.e.* number of SRAM PUF cells) as possible at a low algorithm complexity. In this section, we aim to reduce the number of SRAM cells that are needed in the HDA, by improving the performance of the used error-correcting codes.

We moreover try to lower the decoding complexity by using shorter codes than previous proposals.

In [5], [1], it is assumed that all SRAM PUF cells have the same *average* bit error probability, estimated by counting the number of erroneous bits in a large number of responses. The proposed implementations of HDAs, *e.g.* in [1], use the code-offset technique with linear block codes<sup>3</sup> and hard decision decoding. The block code parameters are chosen to make the bit error-probability small enough (typically  $\leq 10^{-6}$ ). In Section III-D we have demonstrated that the error-probability of an SRAM PUF is not a constant, but a random variable over the cell population, with a distribution given by Eq. (10). This is moreover an asymmetric distribution, meaning that there are a few cells with an error-probability (much) larger than the average, but there are many cells with an error-probability that is smaller. For the majority of the cells, the used error-correcting code is hence overkill. A more efficient construction would take into account the specific error-probability of individual cells. This is precisely what a soft decision decoder does.

### B. Soft Decision Helper Data for Decoding

In addition to only the '0'- or '1'-value of a received bit, soft decision (SD) information also provides a reliability measure for this bit value. SD information becomes available, *e.g.* when a demodulator has to decide on the bit value of a received waveform. It will output the bit value it decided on, but also a confidence measure for this decision based on the waveform distortion. An error-correcting decoding algorithm can use this SD information to improve its performance, *i.e.* it can correct more errors or the code rate can be increased. Optimally, a SD decoder will output the codeword that was most likely transmitted, taking into account the reliability of the received bits. In case of HDAs, a better performing decoding algorithm implies a smaller min-entropy loss in the information reconciliation.

In case of an SRAM PUF cell  $i$ , SD information can be obtained in the form of its error-probability  $p_{e_i}$ . In the generation phase of the HDA,  $p_{e_i}$  is measured for every cell and added to the helper data. In the reproduction phase, the  $p_{e_i}$  contained in the helper data are used as SD information in the decoding algorithm of the code-offset technique. It is important that this additional helper data doesn't leak (much) information about the fuzzy secret, since this would induce an additional min-entropy loss.

*Theorem 1:* Revealing the error-probability  $p_{e_i}$  of the SRAM PUF cells on average does not leak any information about the expected response  $r_i^C$ , or  $\mathbf{H}_\infty(\mathcal{R}^C) = \tilde{\mathbf{H}}_\infty(\mathcal{R}^C|\mathcal{P}_e)$ .

*Proof:*

$$\begin{aligned} \text{Let } p_{r^C} &\stackrel{\text{def}}{=} \Pr_i(r_i^C = 1) = \Pr(\mathcal{R}^C = 1), \\ &= \Pr(\mathcal{M} \leq T) = \Phi(\lambda_2), \end{aligned}$$

<sup>3</sup>Other types of error correcting codes, such as convolutional codes and LDPC codes, are not well suited for use with a PUF to extract relatively short cryptographic keys, since they require longer data streams to work efficiently.

then the min-entropy of  $\mathcal{R}^C$  can be written as:

$$\mathbf{H}_\infty(\mathcal{R}^C) = -\log_2(\max\{p_{r^C}, 1-p_{r^C}\}) = -\log_2(\Phi(|\lambda_2|)).$$

The definition and notation for the conditional min-entropy come from [3]:

$$\tilde{\mathbf{H}}_\infty(\mathcal{R}^C|\mathcal{P}_e) \stackrel{\text{def}}{=} -\log_2\left(\mathbb{E}_{p_e \leftarrow \mathcal{P}_e}\left[2^{-\mathbf{H}_\infty(\mathcal{R}^C|p_e)}\right]\right). \quad (11)$$

$$\begin{aligned} \text{Let } p_{r^C}^*(p_e) &\stackrel{\text{def}}{=} \Pr_i(r_i^C = 1|p_{e_i} = p_e) = \Pr(\mathcal{R}^C = 1|\mathcal{P}_e = p_e), \\ &= \Pr(\mathcal{M} \leq T|\mathcal{M} = T \pm \sigma_n \Phi^{-1}(p_e)), \\ &= \frac{\varphi(\lambda_2 + \lambda_1 \Phi^{-1}(p_e))}{\varphi(\lambda_2 + \lambda_1 \Phi^{-1}(p_e)) + \varphi(\lambda_2 - \lambda_1 \Phi^{-1}(p_e))}. \end{aligned}$$

If  $\lambda_2 > 0$  then  $2^{-\mathbf{H}_\infty(\mathcal{R}^C|p_e)} = p_{r^C}^*(p_e)$ , and

$$\begin{aligned} \tilde{\mathbf{H}}_\infty(\mathcal{R}^C|\mathcal{P}_e) &= -\log_2\left(\int_0^{\frac{1}{2}} p_{r^C}^*(x) \cdot \mathbf{pdf}_{\mathcal{P}_e}(x) dx\right) \\ &= -\log_2\left(\lambda_1 \int_0^{\frac{1}{2}} \frac{\varphi(\lambda_2 + \lambda_1 \Phi^{-1}(x))}{\varphi(\Phi^{-1}(x))} dx\right) \\ &= -\log_2\left(\int_{-\infty}^{\lambda_2} \varphi(u) du\right) = -\log_2(\Phi(\lambda_2)) \end{aligned}$$

If  $\lambda_2 < 0$ , one can equivalently show that  $\tilde{\mathbf{H}}_\infty(\mathcal{R}^C|\mathcal{P}_e) = -\log_2(\Phi(-\lambda_2))$ , or regardless of the sign of  $\lambda_2$ :

$$\tilde{\mathbf{H}}_\infty(\mathcal{R}^C|\mathcal{P}_e) = -\log_2(\Phi(|\lambda_2|)) = \mathbf{H}_\infty(\mathcal{R}^C),$$

Theorem 1 tells us that using the error-probability of SRAM PUF cells as soft decision helper data does not induce any additional min-entropy loss.

## V. SIMULATIONS AND RESULTS

It is assumed that the error-probability  $p_{e_i}$  of each SRAM PUF cell  $i$  is known and through the helper data available in the reproduction phase of the HDA. This error-probability is combined with the response  $r_i^{(j)}$  of the cell and the code-offset helper data bit  $h_i$  to form the SD data  $s_i^{(j)}$  that enters the decoder:

$$s_i^{(j)} = (-1)^{h_i \oplus r_i^{(j)}} \cdot (\log(1 - p_{e_i}) - \log(p_{e_i})), \quad (12)$$

Generally, SD decoding is done with convolutional codes using a soft decision Viterbi decoder, or with LDPC codes using a belief propagation algorithm. However, to efficiently apply the code-offset technique, we would like to use relatively short linear block codes with SD decoding. Similarly to [1], we'll consider repetition, BCH and Reed-Muller codes, and concatenations thereof. Although less common, a number of SD decoding algorithms for these codes exist. The algorithms we consider are:

- *Soft Decision Maximum-Likelihood Decoding (SDML)* yields the best possible decoding performance for the

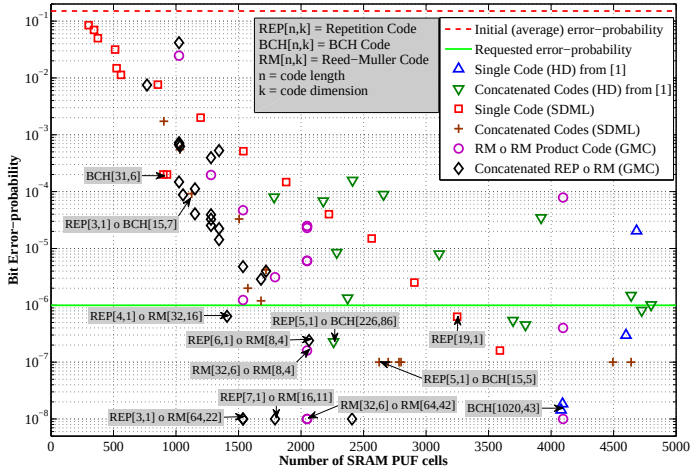


Fig. 3. Performance of different decoding algorithms from simulation. The plot shows the number of SRAM PUF cells that are needed to extract at least 171 robust information bits at the given error-probability.

used code. However, this goes at the cost of a decoding complexity that is exponential in the code dimension. The SDML algorithm simply goes through all the code words and returns the code word that was most likely transmitted based on the SD information:  $c^{(j,ML)} = \arg\max_{c \in C_{n,k}} \{\sum_{i=0}^{n-1} (-1)^{c_i} \cdot s_i^{(j)}\}$ . We simulated SDML decoding for repetition, short BCH and Reed-Muller codes, and for the concatenation of repetition codes with short BCH or Reed-Muller.

- A Reed-Muller code of length  $2^m$  and order  $r$ ,  $\mathbf{RM}_{r,m}$ , can be decomposed in the concatenation of two shorter inner codes  $\mathbf{RM}_{r-1,m-1}$  and  $\mathbf{RM}_{r,m-1}$  and a simple binary length-2 block code as outer code. This can be used to recursively decode a Reed-Muller code as *Generalized Multiple Concatenated codes (GMC)* with SD decoding, as introduced in [10]. This SD decoding technique yields a much lower complexity than straightforward SDML, with only a slightly decreased performance. We simulated GMC decoding for the concatenation of two Reed-Muller codes as a product code, and for the concatenation of repetition and Reed-Muller codes.

We compare our results to the results from [1], where an overall constant bit error-probability of the SRAM PUF responses of 15% is assumed. To make a fair comparison, we set the parameters of our model to  $\lambda_1 = 0.51$  and  $\lambda_2 = 0$ , which gives  $\mathbb{E}[\mathcal{P}_e] = 15\%$ . We make a comparison based on the number of SRAM PUF cells that are needed to obtain at least 171 information bits with an error-probability  $\leq 10^{-6}$  after information reconciliation. These 171 information bits will be compressed into a 128-bit cryptographic key in the privacy amplification step of the HDA.

The results from [1] and from our simulations with SD decoding using SDML and GMC are given in Figure 3. The following observations are made:

- In the case where only a single block code is used, the extremely simple 19-repetition code with SDML uses

20% less PUF cells than the complex BCH[1020, 43] hard decision decoding from [1].

- In the case of concatenated codes, the simple REP[5, 1] o BCH[15, 5] with SDML uses only 16% more cells than the more complex REP[5, 1] o BCH[226, 86] with hard decision from [1].
- We use GMC decoding for longer Reed Muller codes. The best observed result is obtained by the concatenation of REP[4, 1] o RM[32, 16], which uses 38% less cells than the best proposal from [1]. Also notable is the very simple concatenation of REP[6, 1] o RM[8, 4] which still outperforms the more complex proposal from [1] by 9%.

## VI. CONCLUSION

In this work, a mathematical model for the distribution of responses of SRAM PUFs is proposed and experimentally verified. The model suggests that soft decision information can be used to improve the efficiency of the HDAs that are needed to extract cryptographic keys from the SRAM PUF responses. Moreover, it is proven that revealing the soft decision information does not induce an additional min-entropy loss. A number of soft decision decoding techniques are simulated with the model. The best result from the simulation yields a HDA construction that uses 38% less SRAM cells at a lower decoding complexity than previous proposals.

## ACKNOWLEDGMENT

This work was supported by the IAP Program P6/26 BCRYPT of the Belgian State (Belgian Science Policy) and by K.U.Leuven-BOF funding (OT/06/04). The first author's research is funded by IWT-Vlaanderen under grant number 71369.

## REFERENCES

- [1] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient Helper Data Key Extractor on FPGAs," in *CHES*, 2008, pp. 181–197.
- [2] X. Boyen, "Reusable cryptographic fuzzy extractors," in *ACM CCS*, 2004, pp. 82–91.
- [3] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," in *EUROCRYPT*, 2004, pp. 523–540.
- [4] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *ACM CCS*, 2002, pp. 148–160.
- [5] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *CHES*, 2007, pp. 63–80.
- [6] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, and P. Tuyls, "The Butterfly PUF: Protecting IP on every FPGA," in *IEEE HOST*, 2008, pp. 67–70.
- [7] J. Lee, L. Daihyun, B. Gassend, G. Suh, M. van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *Symposium on VLSI Circuits*, 2004, pp. 176–179.
- [8] J.-P. Linnartz and P. Tuyls, "New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates," in *AVBPA*, 2003, pp. 393–402.
- [9] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 6, pp. 2026–2030.
- [10] G. Schnabl and M. Bossert, "Soft-decision decoding of Reed-Muller codes as generalized multiple concatenated codes," *IEEE Transactions on Information Theory*, vol. 41, no. 1, pp. 304–308, 1995.
- [11] P. Tuyls, G. J. Schrijen, B. Škorić, J. van Geloven, N. Verhaeghe, and R. Wolters, "Read-Proof Hardware from Protective Coatings," in *CHES*, 2006, pp. 369–383.