Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs

Roel Maes[1], Pim Tuyls[1,2], and Ingrid Verbauwhede[1]

[1] K.U. Leuven ESAT/COSIC and IBBT, Leuven, Belgium
[2] Intrinsic-ID, Eindhoven, The Netherlands
{roel.maes,pim.tuyls,ingrid.verbauwhede}@esat.kuleuven.be

Abstract. Using a Physically Unclonable Function or PUF to extract a secret key from the unique submicron structure of a device, instead of storing it in non-volatile memory, provides interesting advantages like physical unclonability and tamper evidence. However, an additional Helper Data Algorithm (HDA) is required to deal with the *fuzziness* of the PUF's responses. To provide a viable alternative to costly protected non-volatile memory, the PUF+HDA construction should have a very low overhead. In this work, we propose the first HDA design using *soft-decision information* providing an implementation that occupies 44.8% less resources than previous proposals. Moreover, the required size of the used PUF can be reduced upto 58.4% due to the smaller entropy loss.

Keywords: Physically Unclonable Functions, Helper Data Algorithm, FPGA Implementation, Soft-Decision Decoder, Toeplitz Hash.

1 Introduction

The theoretical study of a cryptographic scheme aims to provide a well defined and quantitative understanding of its security. However, when the scheme enters the practical domain, more parameters join in the game. A security application does not only need to be as secure as possible, but also as inexpensive, fast, power-efficient and flexible as possible, which often means that the security is reduced in order to improve these practical characteristics. Moreover, the vast expansion of physical attacks on cryptographic implementations has shown that certain assumptions upon which the theoretical security of a scheme is based do not necessarily hold in practice, *e.g.* the existence of secure key storage. Private keys often need to be stored in publicly accessible devices, *e.g.* smart cards or RFID-tags, allowing adversaries to physically attack the implementation [1]. Tampering attacks [2,3], in which an attacker physically invades the device in order to extract sensitive information, are among the strongest known physical attacks and are in general always able to obtain the private key if no specific countermeasures are taken. Advanced techniques to detect and/or resist tampering in integrated circuits (*e.g.* [4,5]) are indispensable in security-sensitive applications, but unavoidably add to the *overhead* of the security aspect.

C. Clavier and K. Gaj (Eds.): CHES 2009, LNCS 5747, pp. 332–347, 2009.

Among the proposed tampering countermeasures, Physically Unclonable Functions or PUFs [6] take a special place because of their interesting properties and the cost-effective solutions they offer. A PUF implements a functionality that is highly dependent on the exact physical properties of the embedding device, down to a submicron level. PUFs on integrated circuits (ICs) take advantage of the intrinsic physical uniqueness of the device caused by unavoidable random deep-submicron manufacturing variations, which makes their behavior unique and unclonable. Moreover, since tampering attacks are bound to alter the physical integrity of the chip, they will also change the PUF's behavior [7] and PUFs can hence be used as a tamper detection mechanism. Their instance-specific unique behavior and their anti-tampering properties make PUFs on ICs ideal constructions for secure key storage, *i.e.* the PUF responses can be used to generate a unique and physically unclonable device key [8]. In addition, since the unique PUF behavior arises automatically, no non-volatile memory is needed for storing a key. A number of possible PUF implementations on ICs have been proposed, based on delay measurements [9,10] and power up values of memory elements [11,12]. In the latter category, SRAM-based PUFs exhibit convenient qualities: the power up states of SRAM cells are dependent on intrinsically present manufacturing variability which increases with shrinking technology nodes, SRAM cells are small, commonly used and available early in a new manufacturing process.

Since a PUF evaluation implies a physical measurement, the extraction of a key from the responses is not straightforward. Physical measurements are susceptible to noise and the measured random variables often come from a non-uniform source. On the other hand, we expect cryptographic keys to be highly reliable and to have full entropy to be secure. In order to bridge this gap, Helper Data Algorithms (HDAs) have been introduced [13,14], which are able to transform noisy and non-uniform variables into reliable and uniformly distributed bit strings using public *helper data*. This helper data, although it can be made public without disclosing any information about the extracted key, will always leak some entropy on the PUF responses. In short, one always needs to input more entropy into a HDA than the actual extracted key will contain, since part of it is leaked by the helper data. This *entropy loss* is a function of the noise levels of the input, and is an important characteristic of a HDA which should be minimized. In case of an SRAM PUF, the amount of entropy loss in the HDA relates directly to the number of SRAM cells needed in the PUF to extract a key and hence the size of the PUF on silicon. Since it is in our interest to minimize the implementation cost of the key storage, we would like this number to be as small as possible. The HDA itself also causes an overhead cost and its implementation should hence also be resource-optimized.

Contributions. In this work, we propose a new low-overhead design for a HDA that uses available soft-decision information. A practical FPGA implementation of the design is provided with a considerably lower implementation cost than previous HDA implementations [15], concerning both the required PUF size (58.4% smaller) and the resource usage of the HDA (44.8% smaller).

Related Work. SRAM PUFs were introduced in [11] and similar constructions are studied in [16,17,12]. They provide a practical PUF implementation because of the ease of use and general availability of SRAM cells on regular silicon devices. The concept of HDAs has been introduced as shielding functions in [14] and fuzzy extractors in [13]. A first efficient implementation on FPGA of a HDA for key extraction was proposed in [15]. We will refer regularly to this work and compare our results. The use of soft-decision information to improve performance is a long known result in channel coding and its usefulness for HDAs was first demonstrated in [18]. To the best of our knowledge, this work is the first to propose an efficient HDA implementation using soft-decision information.

2 Preliminaries

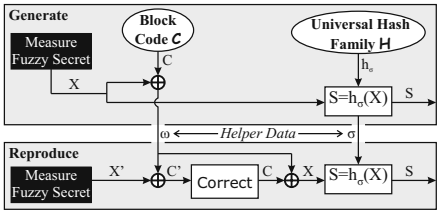
2.1 Helper Data Algorithms

A noisy and partially random variable, like a PUF response or a biometric, is often referred to as a *fuzzy secret*. Helper Data Algorithms (HDAs) are used to extract cryptographic keys from fuzzy secrets, and have been introduced as *fuzzy extractors* in [13] or *shielding functions* in [14]. We will use the formal definition of a fuzzy extractor from [13]:

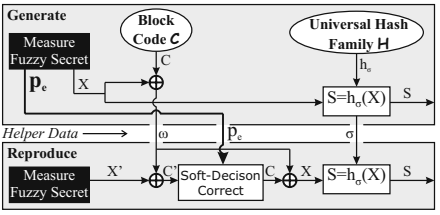
Definition 1 (Fuzzy Extractor). *A $(m, n, \delta, \mu, \epsilon)$-fuzzy extractor is a pair of randomized procedures,* generate *(Gen) and* reproduce *(Rep):*

1. *The generation procedure* Gen *on input $X \in \{0,1\}^m$ outputs an extracted string $S \in \{0,1\}^n$ and* helper data *$W \in \{0,1\}^*$.*
2. *The reproduction procedure* Rep *takes an element $X' \in \{0,1\}^m$ and a bit string $W \in \{0,1\}^*$ as inputs. The correctness property of fuzzy extractors guarantees that if the Hamming distance $\mathbf{dist}[X; X'] \leq \delta$ and S, W were generated by $(S, W) \leftarrow \mathsf{Gen}(X)$, then $\mathsf{Rep}(X', W) = S$.*
3. *The security property guarantees that for any distribution $\mathbb{D}$ on $\{0,1\}^m$ of min-entropy μ, the string S is nearly uniform even for those who observe W: if $(S, W) \leftarrow \mathsf{Gen}(X \leftarrow \mathbb{D})$, then it holds that the statistical distance $\Delta[(S, W); (U \leftarrow \mathbb{U}_n, W)] \leq \epsilon$, with $\mathbb{U}_n$ the uniform distribution on $\{0,1\}^n$.*

The used notions of min-entropy and statistical distance are described in Appendix A. The correctness property of fuzzy extractors takes care of possible noise in the fuzzy secret. As long as the distance between the fuzzy secret during generation and reproduction is limited, the same extracted output can be obtained. This is also known as *information reconciliation.* The security property tells us that the extracted output is very close to uniform as long as the fuzzy secret contains a sufficient amount of min-entropy, even when the helper data is observed. This is called *privacy amplification.* The important contribution of HDAs is the ability to extract a *private key* from a fuzzy secret if a *public helper channel* is available. It is however important to guarantee the integrity of the helper data [19]. The information reconciliation and privacy amplification



(a) Classical helper data algorithm using the code offset technique and a universal hash function.



(b) Proposed soft-decision helper data algorithm. The additional *soft-decision information*, available as the bit error probabilities p_e, is transfered as helper data.

Fig. 1. Constructions of a helper data algorithm

functionality of a HDA are typically implemented by two separate algorithms. We elaborate on a common construction for both:

Information Reconciliation with the Code Offset Technique [13]. A binary linear block code $\mathcal{C}$ with parameters $[n, k, d]$ contains code words of length n, dimension k and minimal Hamming distance d and is able to correct at least $t = \lfloor (d-1)/2 \rfloor$ bit errors occurring in a single code word. The code offset technique picks a uniformly random code word, denoted as $C \leftarrow \mathcal{C}$, in the generation phase and calculates the offset between the fuzzy secret X and C: $\omega = X \oplus C$. This offset ω is made publicly available as helper data. In the reproduction phase, a new version X' of the fuzzy secret is measured and $C' = X' \oplus \omega$ is calculated. If $\mathbf{dist}\,[X; X'] \equiv \mathbf{dist}\,[C; C'] \leq t$ then C' can be corrected to C: $C = \mathsf{Correct}(C')$, which allows the reproduction of $X = C \oplus \omega$. As observed in [13], publishing ω amounts to a min-entropy loss of $n-k$, *i.e.* $\widetilde{\mathbf{H}}_\infty\,(X|\omega) = \mathbf{H}_\infty\,(X) - n + k$. We aim to minimize this loss while maintaining an acceptable level of error-correction.

Privacy Amplification with Universal Hash Functions [20]. A universal hash family $\mathcal{H}$ with parameters $[a, b]$ is a set of functions $\{h_i : \{0,1\}^a \rightarrow \{0,1\}^b\}$ such that the collision probability on two distinct inputs is at most 2^{-b} for a randomly picked function from $\mathcal{H}$: $\mathbf{Pr}\,(h_R(x) = h_R(x')) \leq 2^{-b}, \forall x \neq x' \in \{0,1\}^a$ and $h_R \leftarrow \mathcal{H}$. The *left-over hash lemma* [21] states that universal hash functions can act as a "magnifying glass" for randomness: when taking a random variable with limited min-entropy as input, the output distribution will be close to uniform. Upon generation, a function $h_\sigma \leftarrow \mathcal{H}$ is randomly selected and applied to X to obtain a random output $S = h_\sigma(X)$. The index σ is made available as helper data such that the same hash function can be used in the reproduction procedure to reproduce S from X' after information reconciliation.

Complete helper data algorithm. Using the two techniques mentioned above, a complete HDA can be constructed, as shown in Figure 1(a). The helper data W consists of the code offset ω and the hash function index σ: $W = (\omega, \sigma)$. The two main blocks to be implemented in order to perform this HDA are an error-correcting decoder and a universal hash function. In this work, we carefully

select the parameters of these blocks and provide a resource-optimized design and implementation for reconfigurable hardware devices.

2.2 Soft-Decision Error Correction

A classic method to increase the performance of an error-correcting decoder, and hence decrease the code redundancy $(n-k)$, is using *soft-decision information* in the decoding algorithm. This technique could equivalently lower the entropy loss of a HDA. Soft-decision decoding is possible when *reliability measures* for received bits are available, which is called the soft-decision information. Two well known soft-decision decoding algorithms are the Viterbi algorithm for convolutional codes [22] and the belief propagation algorithm for LDPC codes [23]. However, both types are inappropriate for use in the code offset technique since they require very long data streams to work efficiently while the length of a fuzzy secret is often limited. We would like to use a soft-decision decoding algorithm for rather short linear block codes ($n \leq 2^8$) in order to maintain efficiency. We discuss two such decoders:

Soft-decision Maximum-Likelihood Decoding (SDML) is a straightforward algorithm that selects the code word that was most likely transmitted based on the bit reliabilities. SDML achieves the best error-correcting performance possible, but generally at a decoding complexity exponential in the code dimension k. Repetition codes ($k = 1$) can still be efficiently SDML decoded. Conversely, if $k = n$, SDML decoding degenerates to making a hard decision on every bit individually based on its reliability, and if $k = n - 1$, the block code is a parity check code and SDML decoding is done efficiently by flipping the least reliable bit to match the parity. This last technique is known as Wagner decoding [24].

Generalized Multiple Concatenated Codes (GMC). An r-th order Reed-Muller code $\mathsf{RM}_{r,m}$, is a linear block code with parameters $n = 2^m$, $k = \sum_{i=0}^{r} \binom{m}{i}$ and $d = 2^{m-r}$. It is well known that $\mathsf{RM}_{r,m}$ can be decomposed in the concatenation of two shorter inner codes, $\mathsf{RM}_{r-1,m-1}$ and $\mathsf{RM}_{r,m-1}$, and a simple length-2 block code as outer code. This decomposition can be applied recursively until one reaches $\mathsf{RM}_{0,m'}$, which is a repetition code, or $\mathsf{RM}_{r'-1,r'}$ (or $\mathsf{RM}_{r',r'}$), which is a parity check (or degenerated) code, all of which can be efficiently soft-decision decoded with SDML. This technique, known as Generalized Multiple Concatenated decoding (GMC) [25], yields a much lower decoding complexity then SDML, but only a slightly decreased error-correcting capability.

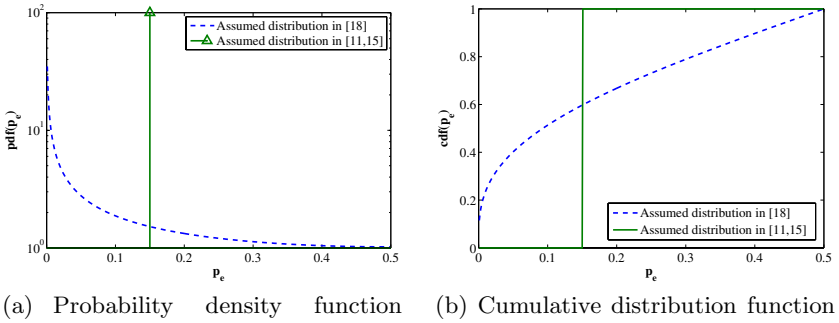
2.3 SRAM PUFs and Soft-Decision Helper Data

Extensive experiments in [11] show that the power up value of a randomly selected SRAM cell is random over $\{0, 1\}$, but tends to take the same value at every power up. This is due to the random manufacturing mismatch between the electrical parameters defining the cell's behavior. The power up values of SRAM cells can hence be used as PUF responses, and the function taking an

SRAM cell's address as challenge and returning its power up value as response is called an SRAM PUF. Occasionally, a cell is encountered with no distinct preference toward 0 or 1, introducing noisy bits.

Previous proposals concerning key extraction from an SRAM PUF [11,15] assume that the bit error probability of a response is constant, *i.e.* every response bit has the same probability of being measured incorrectly. However, experimental data shows that this is not quite the case, as most cells only very rarely produce a bit error while a minority of cells are faulty more often. In fact, the error probability of a randomly selected cell is itself a random variable drawn according to a certain distribution, and hence not a constant value. A theoretical derivation of this distribution, based on a model for the manufacturing variability in SRAM cells, was given in [18] and is summarized in Appendix B. It is clear that using a block code adapted to the *average* bit error rate, as in [11,15], is overly pessimistic for the majority of the bits, as most of them have an error probability much smaller than the average. The distribution of the error probability in both cases is shown in Figure 2, and from Figure 2(b) it is clear that in this case, around 60% of the bits have an error probability which is smaller than the assumed *fixed* average. A construction that takes into account the specific error probability of the individual bits would achieve a better overall performance, needing less redundancy and hence causing a smaller min-entropy loss. This is precisely what soft-decision decoding achieves. A HDA based on soft-decision decoding in the code-offset technique is shown in Figure 1(b).

In Section 3.1 we present a hardware design for an information reconciliation algorithm that uses the individual error probabilities of the response bits as soft-decision information. The bit error probability is measured during the generation phase and made publicly available as helper data. It is hence important to know the min-entropy leakage caused by revealing the error probabilities. It turns out that revealing P_e does not leak any min-entropy on the response X, *i.e.* $\widetilde{\mathbf{H}}_\infty(X|P_e) = \mathbf{H}_\infty(X)$. A proof for this statement is given in [18]. Measuring the bit error probability amounts to performing multiple measurements of every



(a) Probability density function (pdf).

(b) Cumulative distribution function (cdf).

Fig. 2. Distributions of the bit error probability as assumed respectively in [18] and [11,15]. The expected value for P_e is set equal for both cases: $\mathbb{E}[P_e] = 15\%$.

response bit and estimating the most-likely value, which could be inefficient. Our simulations show that an estimate based on a limited amount of measurements, in the order of 10 to 100, already greatly improves the decoder performance. Moreover, this measurement should be performed only once for every PUF.

3 Designing a Soft-Decision HDA for FPGA

This section provides the main contribution of this work, *i.e.* a low-overhead design of a soft-decision helper data algorithm for a reconfigurable hardware device. A first efficient HDA implementation for FPGAs was given in [15]. We will build upon this work and try to improve their results. Sections 3.1 and 3.2 describe the respective design choices for the information reconciliation and the privacy amplification algorithm that we choose to implement.

3.1 Soft-Decision Information Reconciliation Design

The code offset technique as described in Section 2.1 is an efficient technique for turning an error-correcting decoder into an information reconciliation algorithm. As motivated in Section 2.3, we will use a soft-decision decoder to reduce the min-entropy loss of the information reconciliation.

Representing the Soft-Decision Information. As stated in Section 2.3 and shown in Fig. 1(b), the error probability p_{e_i} of an individual SRAM cell i will be used as soft-decision helper data. In general, p_{e_i} takes real values in $]0, \frac{1}{2}[$ and we need to determine a way of representing p_{e_i} in binary format, such that it can be efficiently used in a soft-decision decoder. We denote a codeword C from a block code $\mathcal{C}$ of length n as $C = (C_0, \ldots, C_{n-1})$, an n-bit SRAM PUF response as $X = (X_0, \ldots, X_{n-1})$ and the corresponding vector with error probabilities as $p_e = (p_{e_0}, \ldots, p_{e_{n-1}})$. When receiving an n-bit possibly noisy code word $C' = X' \oplus \omega$, with ω the code offset helper data as defined in 2.1, a soft-decision decoder tries to find the corrected code word C^* which maximizes the (log-)likelihood:

$$\begin{aligned} C^* &= \underset{C \in \mathcal{C}}{\operatorname{argmax}} \prod_{i=0}^{n-1} (1 - p_{e_i})^{(C'_i \oplus C_i)} \cdot p_{e_i}^{(1 \oplus C'_i \oplus C_i)}, \\ &= \underset{C \in \mathcal{C}}{\operatorname{argmax}} \sum_{i=0}^{n-1} (-1)^{C_i} \cdot (-1)^{C'_i} \cdot \left(\log_\beta (1 - p_{e_i}) - \log_\beta (p_{e_i}) \right), \end{aligned}$$

with $\beta > 1$ a design parameter. For convenience, we work with the log-likelihood and choose the soft-decision helper data s_i of an SRAM PUF response bit i to be:

$$s_i \stackrel{\text{def}}{=} \lfloor \log_\beta (1 - p_{e_i}) - \log_\beta (p_{e_i}) \rfloor, \tag{1}$$

which is a deterministic function of the error probability and an integer approximation of the magnitude of the log-likelihood of bit i. For a noisy PUF

Algorithm 1. SDML-DECODE-Repetition$_n(L)$ with *soft output*

$L^* := \sum_{i=0}^{n-1} L_i$
return $(L^*, \ldots, L^*)_n$

Algorithm 2. GMC-DECODE-RM$_{r,m}(L)$ with *soft output*

define $F(x,y) := \operatorname{sign}(x \cdot y) \cdot \min\{|x|, |y|\}$
define $G(s,x,y) := \lfloor \frac{1}{2}(\operatorname{sign}(s) \cdot x + y) \rfloor$
if $r = 0$ **then**
  $L^* = \text{SDML-DECODE-Repetition}_{2^m}(L)$
else if $r = m$ **then**
  $L^* = L$
else
  $L_j^{(1)} = F(L_{2j-1}, L_{2j}), \forall j = 0 \ldots 2^{m-1} - 1$
  $L^{(1)*} = \text{GMC-DECODE-RM}_{r-1,m-1}(L^{(1)})$
  $L_j^{(2)} = G(L_j^{(1)*}, L_{2j-1}, L_{2j}), \forall j = 0 \ldots 2^{m-1} - 1$
  $L^{(2)*} = \text{GMC-DECODE-RM}_{r,m-1}(L^{(2)})$
  $L^* = \left(F(L_0^{(1)*}, L_0^{(2)*}), L_0^{(2)*}, \ldots, F(L_{2^{m-1}-1}^{(1)*}, L_{2^{m-1}-1}^{(2)*}), L_{2^{m-1}-1}^{(2)*}\right)$
end if
return L^*

response X', the soft-decision information that enters the decoder is calculated as: $L_i = (-1)^{X'_i \oplus \omega_i} \cdot s_i$. The decoder tries to find the code word $C^* = \operatorname{argmax}_{C \in \mathcal{C}} \sum_{i=0}^{n-1} (-1)^{C_i} \cdot L_i$. In the remainder of the text, s_i and L_i will be represented by 8-bit signed (2's-complement) integers $\in [-128, 127]$. The log-base β is a design parameter that is chosen large enough to avoid overflows in the decoder algorithm, but as small as possible to keep the approximation error small.

Choosing a Soft-Decision Decoder Algorithm. Among the linear block codes, Reed-Muller codes have a relatively high error-correcting performance similar to BCH codes, and are easier to decode. As explained in Section 2.2, there exists also a relatively efficient algorithm for soft-decision decoding of Reed-Muller codes based on GMC. Bösch et al. [15] demonstrate that using *code concatenation*, where the decoded words from an *inner code* form a code word from an *outer code*, can substantially reduce the min-entropy loss. A balanced concatenation of two different codes, *e.g.* a repetition code and a Reed-Muller code, will achieve a better performance than the case were only a single code is considered. Taking all this into account, we decide to implement a soft-decision decoder as a concatenation of a SDML repetition decoder for the inner code and a GMC Reed-Muller decoder for the outer code.

SDML repetition decoding of soft-decision information L amounts to calculating $L^* = \sum_{i=0}^{n-1} L_i$. The most-likely transmitted code word was all zeros if $L^* > 0$ and all ones if $L^* < 0$. Moreover, the magnitude of L^* gives a reliability for this decision which allows to perform a second soft-decision decoding for the

outer code. Algorithm 1 outlines the simple operation for the SDML decoding of a repetition code. As an outer code, we use a $\mathsf{RM}_{r,m}$ code and decode it with an adapted version of the soft-decision GMC decoding algorithm as introduced in [25]. The soft-decision output of the repetition decoder is used as input by the GMC decoder. The operation of the GMC decoder we use is given by Algorithm 2. Note that this a recursive algorithm, calling itself twice if $0 < r < m$.

Decoder Design. We propose a hardware architecture to efficiently execute the soft-decision decoders given by Algorithms 1 and 2. Since our main design goal is providing an as small as possible HDA implementation, we try to minimize the used hardware resources. As a general architecture, we opt for a highly serial execution of the algorithms using a small 8-bit custom datapath. Looking at the algorithms, we identify the following major operations:

- Algorithm 1 performs a summation of n 8-bit integers. We implement this serially using an 8-bit signed accumulator.
- To evaluate the function $F(x, y)$ in Algorithm 2, we propose a 3-cycle execution. In the first two cycles, $x + y$ and $x - y$ are computed and their signs c_+ and c_- are stored. In the third cycle, the output is computed as $F(x, y) = c_+ \cdot (x \cdot (c_+ \neq c_-) + y \cdot (c_+ = c_-))$. This last operation amounts to choosing between x and y and possibly changing the sign based on the values of c_+ and c_- and can be done with an adder/subtracter with one of the inputs set to zero.
- For $G(s, x, y)$ in Algorithm 2, we propose a 2-cycle execution. In the first cycle, the sign of s is loaded and in the second cycle, $G(s, x, y)$ can be calculated as an addition or subtraction of x and y based on sign (s), followed by a bit shift.

To be able to execute these operations, we propose the arithmetic unit (AU) depicted in gray in Figure 3. The signed adder/subtracter can change the sign of any of its inputs, or put them to zero. The sign bits of the two previous AU

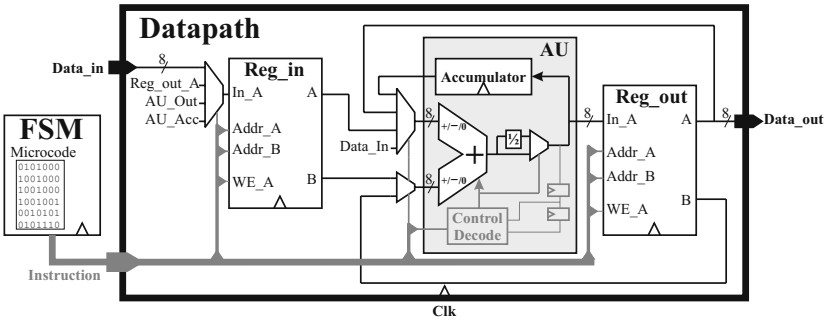
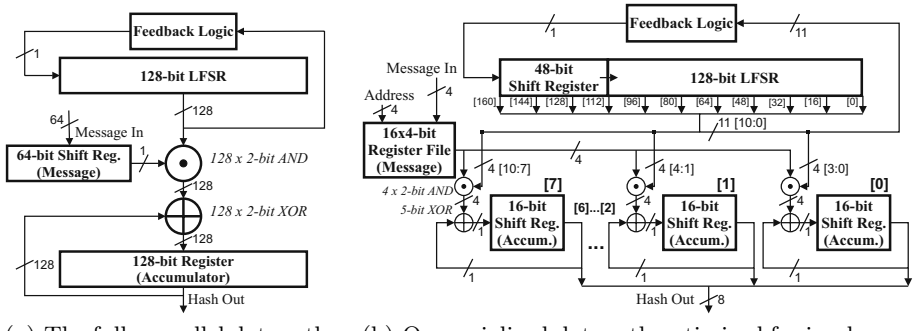


Fig. 3. Details of the soft-decision decoder architecture. The datapath consists of an Arithmetic Unit (AU) and an input and output register file. The controller contains the microcode to execute the decoder algorithm.

outputs are used as control signals. The AU is combined with an input and output dual port register file into a custom 8-bit datapath as shown in Figure 3. Dual port register files can be efficiently implemented on an FPGA using SRAM-based Lookup Tables (LUTs). The depth of the register files depends on the choice of the decoder parameters. The algorithm execution is controlled by an FSM applying the consecutive algorithm steps that are stored as microcode. An operational example of a soft-decision decoder using this design is presented in Section 4, providing detailed implementation parameters and performance results.

3.2 Privacy Amplification Design

In Section 2.1, it was mentioned that privacy amplification amounts to applying a universal hash function. Krawczyk [26] proposed an LFSR-based Toeplitz universal hash algorithm which performs the multiplication of a random Toeplitz matrix with the hash argument. As Krawczyk already showed, this algorithm can be efficiently implemented in hardware, since the columns of the pseudorandom Toeplitz matrix can be generated by an LFSR, and the resulting product of each column with the hash input is accumulated to calculate the full matrix product. This construction is shown in Figure 4(a) and was implemented on an FPGA in [15]. However, the need for an LFSR and an accumulator register of the same size as the key (*e.g.* 128 bit) and an input register of the argument size (*e.g.* 64 bit) yields a relatively expensive implementation on an FPGA, when regular flip-flops are used to implement them. This is because the number of available flip-flops on typical FPGAs is rather low. In [15], this results in the hash algorithm occupying the major part of the used resources for the HDA. More resource-efficient methods for implementing shift registers on FPGAs exist [27], however, they cannot be used directly in Krawczyk's algorithm,



(a) The fully parallel datapath as proposed in [26] and implemented in [15].

(b) Our serialized datapath optimized for implementation with 16-bit shift registers.

Fig. 4. Datapath for (64-bit in/128-bit out) Toeplitz hash implementation

since parallel access to all the bits in the LFSR is required. The implementation from [27] only allows parallel access to every 16th bit of the LFSR state. We reworked the algorithm such that it can be executed in a serial way, using the resource-efficient shift register implementations. This required some modifications to the datapath, but the functional behavior of the algorithm is preserved. The basic idea behind the serialization is that, in stead of accumulating an entire 128-bit product in every cycle, only a partial product (using the accessible bits) is calculated and accumulated in 16-bit rotation shift registers. The resulting datapath is shown in Figure 4(b). This drastically decreases the resource usage of the FPGA implementation as shown by the implementation results in Section 4.

4 Implementation Parameters and Results

In this section, details of a full HDA implementation are provided and compared to the results from [15]. In order to make a fair comparison, the same values for the *average* bit error probability (15%), the amount of min-entropy in the SRAM PUF responses (78%) and for the decoder failure rate ($\leq 10^{-6}$) are chosen.

Determining the decoder parameters. We simulated our decoder proposal in software with SRAM PUF responses sampled according to the proposed distribution from [18]. The bit error probabilities are estimated from 64 measurements. We compared the number of SRAM PUF response bits that were necessary to obtain $\lceil 128/0.78 \rceil = 171$ non-redundant bits after decoding with a failure rate $\leq 10^{-6}$ for different parameters (n, r, m) of the decoder. The best decoder we tested is the one with code parameters $(n = 3, r = 2, m = 6)$ and the design parameter $\beta = 1.8$, and uses $\lceil 171/22 \rceil \times 3 \times 64 = 1536$ SRAM PUF response bits.

FPGA implementation. We described our design in VHDL and synthesized and implemented it on a Xilinx Spartan-3E500 FPGA using Xilinx ISE Design Suite 10.1. The implementation results concern the routed netlist. The functional correctness and the cycle count is tested by applying test benches with ModelSim.

Soft-Decision Decoder: The decoder takes 192×8-bit log-likelihoods as input and every three consecutive values are accumulated (repetition decoded) to obtain 64×8-bit inputs for the $\mathsf{RM}_{2,6}$-decoder which outputs a 64-bit error-corrected code word. To execute Algorithm 2, the input and output register file size are respectively set to 64×8-bit and 32×8-bit. The instructions to carry out Algorithm 2 are stored as embedded microcode. The FPGA implementation occupies 164 slices and 2×16-kbit Block RAMs. The critical path is 19.9ns and one complete decoding cycle (input + decode + output) finishes in 1248 cycles.

LFSR-based Toeplitz hash: The universal hash function accepts 64-bit message blocks and hashes them in a 128-bit value. Our implementation occupies 59 slices. The critical path is 9.2ns and one complete hash cycle (input seed + input message + hash + output) finishes in 432 cycles.

Table 1. Implementation and performance results on a Xilinx Spartan-3E500 FPGA compared to the results from [15]. The given results concern HDA implementations which take SRAM PUF response bits with a 15% average error probability and 78% min-entropy as an input and produce a full-entropy 128-bit key with failure rate $\leq 10^{-6}$.

(1) The soft-decision HDA implementation as proposed in this section.
(2) The HDA implementation from [15] with the lowest SRAM usage.
(3) The HDA implementation from [15] with the lowest HDA resource overhead.

		(1)	(2)	(3)
Decoder (1 round)	Slices	164	580	110
	Block RAMs	2	?*	?*
	Cycles	1248	1716	855
	SRAM Usage	192 bit	264 bit	176 bit
Toeplitz Hash (1 round)	Slices	59	327	319
	Cycles	432	96	64
Complete HDA	**Slices**	**237**	**$\geq$ 907**	**$\geq$ 429**
	(Spartan-3E500)	(5.1%)	($\geq$ 19.5%)	($\geq$ 9.2%)
	Block RAMs	2	?*	?*
	(Spartan-3E500)	(10%)	(?*)	(?*)
	Critical Path	19.9 ns	6.6 ns	5.7 ns
128-bit Key Extraction	Rounds	8	14	35
	Cycles	10298	$\geq$ 24024	$\geq$ 29925
	Performance	205 μs @ 50.2 MHz	$\geq$ 159 μs @ 151.5 MHz	$\geq$ 171 μs @ 175.4 MHz
	SRAM Usage	**1536 bit**	**3696 bit**	**6160 bit**
	Helper Data Size	13952 bit	3824 bit	6288 bit

* The results from [15] for (2) and (3) do *not* include the resources for the controller, hence the number of Block RAMs needed for algorithm control cannot be compared.

Complete HDA: The complete HDA executes the decoder $\lceil 171/22 \rceil = 8$ times and hashes the 8 corrected 64-bit words into a 128-bit key. The implementation of the full HDA + control occupies 237 slices and 2×16-kbit Block RAMs for the microcode. The critical path is 19.9ns and the complete key generation (initialize + 8$\times$ decode and hash + output) finishes in 10298 cycles.

Discussion with respect to previous results. The two main parameters we want to optimize are the SRAM usage of the SRAM PUF and the resource overhead of the HDA implementation. Table 1 compares our implementation results (1) to two different implementations from [15]: (2) the implementation with the lowest SRAM usage, implementing a concatenation of a Golay[24,13] code and a Repetition[11,1] code[1] and (3) the implementation with the lowest HDA resource overhead, implementing a concatenation of a $\mathsf{RM}_{1,4}$ code and a Repetition[11,1] code. It is clear from Table 1 that our soft-decision based implementation outperforms the previous implementations on both characteristics. The construction

[1] We remark that a decoder with an even lower SRAM usage than (2), but still higher than our implementation, is proposed in [15], based on BCH codes. However, no implementation is provided and no fair comparison can be made.

proposed in this section uses 58.4% less SRAM bits and over 44.8% less slices than the respective optimized implementations from [15]. These improvements come at the cost of an increased helper data size ($\times 3.6$) and the need to perform multiple measurement during generation to obtain the soft-decision information. On the other hand, more helper data is not necessarily a problem in many applications, since the helper data can be (externally) stored and transfered in plain without revealing information about the key, only its integrity should be guaranteed. The actual PUF+HDA implementation, *e.g.* residing on an embedded device, remains small. Measuring the error probability of the SRAM PUF cells can be done together with the regular functional testing of the IC right after manufacturing. Performing 10 to 100 measurements can be done relatively fast. Even when very few (< 10) measurements are available, the reconfigurable decoder allows to use stronger codes and remains more efficient than hard decision decoding. We also note that an average error of 15%, as assumed here and in [15] is very safe. Experiments on SRAM PUFs show error probabilities as low as 5%, requiring less initial measurements for the soft-decision decoder to be effective.

5 Conclusion

The bit error probability of an SRAM PUF is not a constant value, but a random variable for every individual response bit. This observation suggests the use of soft-decision information to lower the min-entropy loss of the helper data algorithm, resulting in a more efficient use of the SRAM PUF. We propose a design of a soft-decision helper data algorithm and implement it on an FPGA. A soft-decision Reed-Muller decoder is implemented using a small custom 8-bit datapath which can be easily reconfigured to work with different code parameters depending on the noise levels. The privacy amplification is performed by a serialized LFSR-based Toeplitz hash implementation that makes optimal use of the available FPGA resources. Both constructions constitute to a HDA which has a considerably lower implementation overhead than previous proposals and can even be of independent interest in other domains. The drawbacks of having to store more helper data and having to perform multiple initial measurements are no issue in many applications and should be considered as trade-offs. In any case, this work presents a new direction in the exploration of the design space of efficient helper data algorithms.

Acknowledgments

This work was supported by the IAP Program P6/26 BCRYPT of the Belgian State 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. Verbauwhede, I., Schaumont, P.: Design methods for security and trust. In: Proc. of Design Automation and Test in Europe (DATE 2008), NICE,FR, p. 6 (2007)
2. Anderson, R.J., Kuhn, M.G.: Low Cost Attacks on Tamper Resistant Devices. In: Christianson, B., Lomas, M. (eds.) Security Protocols 1997. LNCS, vol. 1361, pp. 125–136. Springer, Heidelberg (1998)
3. Skorobogatov, S.P.: Semi-invasive attacks - A new approach to hardware security analysis. University of cambridge, computer laboratory: Technical report (April 2005)
4. Yang, J., Gao, L., Zhang, Y.: Improving Memory Encryption Performance in Secure Processors. IEEE Trans. Comput. 54(5), 630–640 (2005)
5. Posch, R.: Protecting Devices by Active Coating. Journal of Universal Computer Science 4(7), 652–668 (1998)
6. Ravikanth, P.S.: Physical one-way functions. PhD thesis, Chair-Benton, Stephen, A. (2001)
7. Tuyls, P., Schrijen, G.-J., Škorić, B., van Geloven, J., Verhaegh, N., Wolters, R.: Read-Proof Hardware from Protective Coatings. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 369–383. Springer, Heidelberg (2006)
8. Tuyls, P., Batina, L.: RFID-Tags for Anti-Counterfeiting. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 115–131. Springer, Heidelberg (2006)
9. Gassend, B., Clarke, D., van Dijk, M., Devadas, S.: Silicon physical random functions. In: CCS 2002: Proceedings of the 9th ACM conference on Computer and communications security, pp. 148–160. ACM, New York (2002)
10. Lee, J.W., Lim, D., Gassend, B., Suh, G.E., van Dijk, M., Devadas, S.: A technique to build a secret key in integrated circuits for identification and authentication applications. In: VLSI Circuits, 2004. Technical Papers, pp. 176–179 (June 2004)
11. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA Intrinsic PUFs and Their Use for IP Protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
12. Kumar, S.S., Guajardo, J., Maes, R., Schrijen, G.J., Tuyls, P.: Extended abstract: The butterfly PUF protecting IP on every FPGA. In: IEEE International Workshop on Hardware-Oriented Security and Trust (HOST-2008), pp. 67–70 (2008)
13. Dodis, Y., Ostrovsky, R., Reyzin, L., Smith, A.: Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. SIAM Journal on Computing 38(1), 97–139 (2008)
14. Linnartz, J.P.M.G., Tuyls, P.: New shielding functions to enhance privacy and prevent misuse of biometric templates. In: Kittler, J., Nixon, M.S. (eds.) AVBPA 2003. LNCS, vol. 2688, pp. 393–402. Springer, Heidelberg (2003)
15. Bösch, C., Guajardo, J., Sadeghi, A.-R., Shokrollahi, J., Tuyls, P.: Efficient Helper Data Key Extractor on FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 181–197. Springer, Heidelberg (2008)
16. Su, Y., Holleman, J., Otis, B.: A Digital 1.6 pJ/bit Chip Identification Circuit Using Process Variations. IEEE Journal of Solid-State Circuits 43(1), 69–77 (2008)
17. Holcomb, D.E., Burleson, W.P., Fu, K.: Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In: Proceedings of the Conference on RFID Security (2007)
18. Maes, R., Tuyls, P., Verbauwhede, I.: A Soft Decision Helper Data Algorithm for SRAM PUFs. In: IEEE International Symposium on Information Theory (2009)

19. Boyen, X.: Reusable Cryptographic Fuzzy Extractors. In: ACM CCS 2004, pp. 82–91. ACM Press, New York (2004)
20. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. In: STOC 1977: Proceedings of the 9th ACM symposium on Theory of computing, pp. 106–112. ACM Press, New York (1977)
21. Bennett, C.H., Brassard, G., Robert, J.-M.: Privacy Amplification by Public Discussion. SIAM J. Comput. 17(2), 210–229 (1988)
22. Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. on Information Theory 13(2), 260–269 (1967)
23. Gallager, R.G.: Low Density Parity-Check Codes. IRE Trans. Inform. Theory 8, 21–28 (1962)
24. Silverman, R.A., Balser, M.: Coding for Constant-Data-Rate Systems-Part I. A New Error-Correcting Code. Proceedings of the IRE 42(9), 1428–1435 (1954)
25. Schnabl, G., Bossert, M.: Soft-decision decoding of Reed-Muller codes as generalized multiple concatenated codes. IEEE Trans. on Information Theory 41(1), 304–308 (1995)
26. Krawczyk, H.: LFSR-based Hashing and Authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994)
27. George, M., Alfke, P.: Linear Feedback Shift Registers in Virtex Devices (April 2007)

A Measures of Randomness

We briefly describe some concepts from information theory which are used to quantify the notion of the *amount of randomness* present in a measured variable, *i.e.* statistical distance and min-entropy. Let X and Y be two (discrete) possibly correlated random variables taking values from a set $\mathcal{S}$. We define:

- The statistical distance between (the distributions of) X and Y as: $\Delta[X;Y] \stackrel{\text{def}}{=} \frac{1}{2}\sum_{s\in\mathcal{S}} |\mathbf{Pr}(X=s) - \mathbf{Pr}(Y=s)|$.
- The min-entropy of (the distribution of) X as: $\mathbf{H}_\infty(X) \stackrel{\text{def}}{=} -\log_2 \max\{\mathbf{Pr}(X=s) : s \in \mathcal{S}\}$.
- The average conditional min-entropy [13] of (the distribution of) X given Y as: $\widetilde{\mathbf{H}}_\infty(X|Y) \stackrel{\text{def}}{=} -\log_2 \mathbb{E}_y\left[2^{-\mathbf{H}_\infty(X|Y=y)}\right]$.

B SRAM PUF Response Model and Distribution

As is clear from the construction of an SRAM PUF as described in Section 2.3, and also in [11], the generation of an SRAM PUF response bit is determined by the stochastic mismatch of the electrical parameters in an SRAM cell. A simple model for this mismatch is proposed in [18] and summarized here. Let M and N be two normally distributed random variables with respective probability density functions φ_{μ_M,σ_M} and φ_{0,σ_N}. $\varphi_{\mu,\sigma}$ is the probability density function of a normal distribution with mean μ and standard deviation σ. A value $m_i \leftarrow M$ is i.i.d. sampled every time a new SRAM cell i is manufactured and represents the random device mismatch in the cell caused by manufacturing variation. A

value $n_i^{(j)} \leftarrow N$ is i.i.d. sampled at the j-th power up of cell i and represents the amplitude of the stochastic noise voltage acting on cell i at the time of the power up. The power up state of SRAM cell i after the j-th power up is denoted as $x_i^{(j)} \in \{0,1\}$, and it is assumed that $x_i^{(j)}$ is fully determined by m_i and $n_i^{(j)}$:

$$x_i^{(j)} = \begin{cases} 0 & \text{, if } m_i + n_i^{(j)} > T, \\ 1 & \text{, if } m_i + n_i^{(j)} \leq T, \end{cases} \tag{2}$$

with T a *threshold parameter* for a specific SRAM technology.

The power up behavior of an SRAM cell i is described by the probability p_{x_i} that this cell powers up as '1', and the related probability p_{e_i} that this cell produces a bit error. Both parameters are themselves random variables. They are sampled for a particular SRAM cell at manufacturing time according to their respective distributions:

$$\mathbf{pdf}_{P_r}(x) = \frac{\lambda_1 \cdot \varphi\left(\lambda_2 - \lambda_1 \cdot \Phi^{-1}(x)\right)}{\varphi\left(\Phi^{-1}(x)\right)}, \text{ and}$$
$$\mathbf{pdf}_{P_e}(x) = \mathbf{pdf}_{P_r}(x) + \mathbf{pdf}_{P_r}(1-x),$$

with $\lambda_1 = \sigma_N/\sigma_M$ and $\lambda_2 = (T-\mu_M)/\sigma_M$ and $\varphi = \varphi_{0,1}$. The derivation of these distributions and an experimental validation thereof are given in [18].