Testing PUF-Based Secure Key Storage Circuits

Mafalda Cortez Gijs Roelofs Said Hamdioui
Delft University of Technology
Faculty of EE, Mathematics and CS
Mekelweg 4, 2628 CD Delft, The Netherlands
{A.M.M.O.Cortez, S.Hamdioui}@tudelft.nl

Giorgio di Natale
LIRMM, Universit Montpellier II
161 Rue Ada, 34392 Montpellier Cedex 5, France
diNatale@lirmm.fr

***Abstract*—Design for test is an integral part of any VLSI chip. However, for secure systems extra precautions have to be taken to prevent that the test circuitry could reveal secret information. This paper addresses secure test for Physical Unclonable Function based systems. In particular it provides the testability analysis and a secure Built-In Self-Test (BIST) solution for Fuzzy Extractor (FE) which is the main component of PUF-based systems. The scheme targets high stuck-at-fault (SAF) coverage by performing scan-chain free functional testing, to prevent scan-chain abuse for attacks. The scheme reuses existing FE sub-blocks (for pattern generation and compression) to minimize the area overhead. The scheme is integrated in FE design and simulated; the results show that a SAF fault coverage of 95.1% can be realized with no more than 50k clock cycles at the cost of a negligible area overhead of only 2.2%. Higher fault coverage is possible to realize at extra cost.**

Keywords: PUF-based systems, Fuzzy Extractor, Secure Testing, Scan-chain free test

I. Introduction

Physical Unclonable Functions (PUFs) based systems are becoming popular solutions for secure key storage against physical attacks; they use the unique, random, uncontrollable and intrinsic physical properties of *Integrated Circuits* (ICs) to derive a cryptographic key. The robustness of such a system is evaluated by means of its reproducibility, i.e., ability of the system to recover the cryptographic key from the same IC, and its uniqueness; i.e., the ability of the system to generate a unique cryptographic key for each IC [1,2]. A *Fuzzy Extractor* (FE) is one of the main components of a PUF-based system; its responsibility is to assure the system's reproducibility and uniqueness [3,4]. Hence, FE flawless operation is essential for the robustness of PUF-based systems. Testing a PUF-based system, and FE in particular, is a challenge. Testability demands excellent accessibility and observability, while security demands poor/no accessibility and observability to the chip, especially during the operation mode where an attacker could easily retrieve partial or complete cryptographic key. The trade-off between testability and security is the main challenge.

Design-for-Test and testability of secure devices have recently gained a lot of attention [5–11]. Overall, the published schemes can be classified into two classes: enhanced scan-chains [5–8] and functional based *Built-In Self Test* (BIST) [10,11].

Enhanced scan-chains target the protection of chains from being misused by attackers. In [5], B. Yang *et al.* developed a test solution for crypto cores based on a type of register that cannot be scanned out during test mode until being reset. In [6], A. Das *et al.* developed a test wrapper for secure test that authenticates legitimate testers. In [7], D. Hely *et al.* introduced spy flip-flops in the scan-chain that detect malicious shifts. In [8], J. Lee *et al.* applied a technique that makes the scan-chain operate unpredictably for untrusted users. However, the industry strongly believes that enhanced scan-chains cannot provide 100% secure IC and therefore they are reluctant to include them in designs targeting secure applications [9].

On the other hand, functional test based BIST targets the enhancement of security, although reaching a very high fault coverage with these schemes is a major challenge. In [10], M. Doulcier *et al.* presented a technique to reuse an *Advanced Encryption Standard* (AES) for self-testing. The work showed that AES cores have enough randomness to be used as test pattern generators and used this property to seft-test the AES core in a loop fashion. In [11], di Natale *et al.* proposed a generic self-test scheme for crypto cores. The work is an extension of the work presented in [10]; it performs the same analysis but for a *Data Encryption Standard* (DES). However, both [10] and [11] are not suited for testing PUF-based systems for two main reasons. First, AES/DES crypto cores are not available in all PUF-based systems and second, PUF-based systems comprise, on top of the crypto cores, error correction blocks which make it more challenging to test functionally.

Although the research in hardware security including test is getting more attention due to the importance of the field, there is almost nothing published on testing PUF-based systems. This topic is addressed in this paper. In particular, it targets testing and testability analysis of Fuzzy Extractors (FEs), which are the main blocks of such systems; FEs are challenging to test as they comprise not only a crypto core, but also error correction blocks, which are typically hard to test functionally. The paper proposes an efficient *scan-chains free* secure test scheme that realizes a high test quality based on pattern generation for stuck-at-faults by performing functional testing. The proposed solution reuses FE existing sub-blocks (for pattern generation and compression) to minimize the area overhead.

The rest of the paper is organized as follows: Section II

978-3-9815370-2-4/DATE14/©2014 EDAA

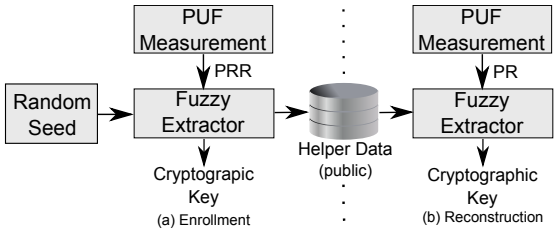


Fig. 1: PUF-based Key Storage System

briefly reviews the background on PUF based-systems. Section III analyzes the testability of FE sub-blocks. Section IV defines the test requirements, proposes a secure test method, presents and discusses the results and provides a generic list of step-by-step instructions on FE secure testing. Finally, Section V concludes the paper.

II. PUF-based Secure Systems

In this section, we first briefly show how PUFs are deployed in a key storage system. Thereafter, we briefly describe the Fuzzy Extractor, which is the main block of such a system and the primary focus of this work.

A. Key-storage based on PUFs

Fig. 1 shows the flow of a PUF-based key storage system [1,2] implemented with a *Fuzzy Extractor* (FE) [3,4], which typically consists of two phases:

(a) **Enrollment:** a cryptographic key is generated from a PUF. First, a PUF measurement is taken and used as *PUF Reference Response* (PRR). Next, PRR and *Random Seed* are processed by the FE into a cryptographically strong *Cryptographic Key*, and helper data is generated as a FE byproduct. Finally, the helper data is stored in an external *Non-Volatile Memory* (NVM); hence, it becomes public information.

(b) **Reconstruction:** the earlier enrolled *Cryptographic Key* is reliably recovered. First, a PUF measurement is taken and used as *PUF Response* (PR). Typically, some bits of PR are different from the original PRR; hence, PR is a noisy version of PRR. Next, PR is processed by the FE in combination with the helper data which is retrieved from the external NVM. If the noisy PR is close enough to the PRR measured during enrollment (i.e., the PUF response is reproducible up to a limited amount of noise), then the FE succeeds in reliably reconstructing the enrolled *Cryptographic Key*.

B. Fuzzy Extractor

A Fuzzy Extractor (FE) is the fundamental component of a PUF-based key storage system; it has two main functions. (a) Error correction: it uses the helper data to correct errors on the measured PUF response; and (b) Privacy amplification: considering that the helper data contains information on the PRR, privacy amplification is needed to make sure that the helper data does not reveal any information on the derived cryptographic key; the FE compresses the resulting data into a cryptographic key with maximum entropy making it hard for the attacker to retrieve the key [3,4]; it also removes any biasing (unequal distribution of zeros and ones) in the error-corrected PUF response.

III. Fuzzy Extractor and its testability analysis

Fig. 3 shows the six main blocks of a Fuzzy Extractor; this implementation is based on the one used for the UNIQUE project [17]. The Peripheral Circuitry has two main functions; it selects between both functional modes and it performs an XOR function between either the PUF and the output of the Repetition Encoder (*RE_O*) generating the Helper Data, if during enrollment or, between the stored Helper Data and PUF generating the input of the Repetition Decoder (*RD_I*), if during reconstruction. The other five blocks are mostly computation intensive and are responsible for enrollment and reconstruction. Each of the five blocks is explained next.

1) Golay Encoder: first block of the enrollment phase. Its responsibility is to prepare the data for the error correction. This block maps the input *Random Seed* (12 bits) to *GE_O* (24 bits) by appending twelve parity bits used for error correction. This makes it feasible for the Golay Decoder in Reconstruction phase to correct up to three bits [14,15]. The main core of this block comprises a loop that generates the Golay space (space of perfect code words). Our implementation of the Golay Encoder has a latency of two clock cycles and it comprises 6.5% of the total number of FE gates.

2) Repetition Encoder: second block of the enrollment phase. It adds extra robustness to the error capabilities of the Golay Encoder. The block replicates each of the *GE_O* 24 bits 11 times resulting in 264 bits serial output *RE_O*; it enables error correction up to five bits for the Repetition Decoder in reconstruction phase. The enrollment phase completes after 86 rounds, i.e., the computations described above are performed 86 times. At each time, the 12 bits of the 1032 bit *Random Seed* are used with a 264 bits fraction of the PUF to generate Helper Data. The total size of both PUF and Helper Data equals 2.8kB (264×86). The main part of the block comprises two counters. The first counter loops over the 24 bits of *GE_O*, while the second counter replicates each bit of *GE_O* 11 times. Our implementation of the Repetition Encoder has a latency of 267 clock cycles and it comprises 7.3% of FE gates.
The reconstruction phase starts with performing a new measurement of the PUF and XORing it with the Helper Data. The result of this operation is the serial input of the Repetition Decoder block.

3) Repetition Decoder: first block of the reconstruction phase. It is also the first stage of error correction. The block performs majority voting on each of the 24 groups (each of 11 bits) scanned serially via *RD_I*, and produce 24 bits at the output *GD_I*. This sub-block performs the inverse operation of the Repetition Encoder and its main core comprises three counters:

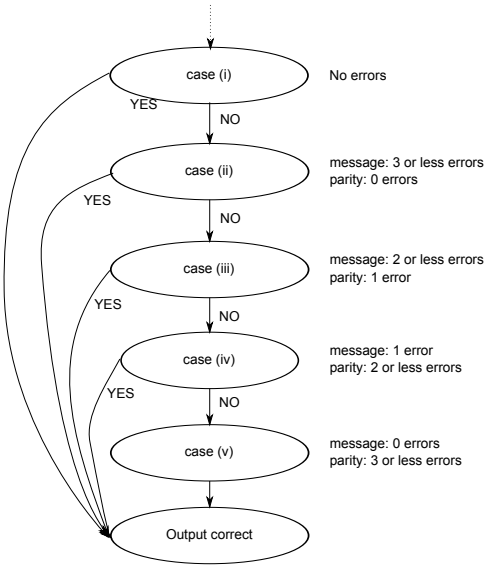


Fig. 2: Golay Decoder state-machine

one counter, *repetition counter* and *destination counter*. The one counter counts the number of ones in a chunk of input *RD_I* (see Fig. 3) and its value is reset after the repetition counter processed n=11 input bits. Next, a single output bit is written on the index provided by the destination counter which is subsequently incremented. The written output bit presents the majority voting result of the processed input chunk derived from the one counter. Our implementation of the Repetition Decoder has a latency of 290 clock cycles and comprises 6.5% of FE gates.

4) Golay Decoder: second block of the reconstruction phase responsible for error correction. The block recovers *Random Seed*, i.e., *HF_I* (12 bits), as long as the provided input *GD_I* is within the error capabilities of the error correction system. Also during the reconstruction phase, the Repetition Decoder and the Golay Decoder repeat their operations 86 times; each time, they serially process 264 bits generated based on PUF and Helper Data. The results of each iteration is a 12 bits buffered inside the Hash Function block. The Golay Decoder is the most complex block in the circuit. It contains a *Finite State-Machine* (FSM), with nine states for vector decoding. As stated previously, a Golay Decoder can correct up to three errors. Its input *GD_I* comprises 12 message bits and 12 parity bits (as the outcome of the Golay Encoder). Fig. 2 shows the states dedicated to error correction; these are selected depending on the location and number of errors in *GD_I*. Error wise, five different cases are possible, denoted in Fig. 2 as case (i) till (v).

(i) *GD_I* is error-free; thus, the four states where the error correction takes place, i.e., case (ii) to case (v), are skipped.
(ii) there are three or less errors in the message bits of *GD_I* and none in the parity bits.

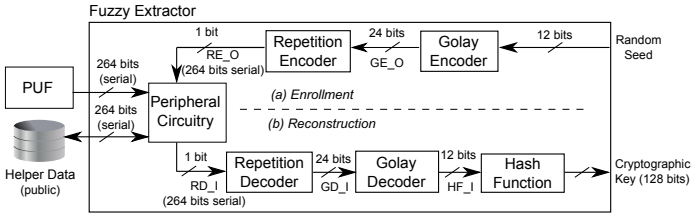


Fig. 3: Example of a Fuzzy Extractor

(iii) there are one or two errors in the message bits of *GD_I* and exactly one in the parity bits.
(iv) there is exactly one error in the message bits of *GD_I* and two or less in the parity bits.
(v) there are no errors in the message bits of *GD_I* and three or less in the parity bits.

The Golay Decoder has a variable latency depending on its input, with a maximum of 10 clock cycles and it comprises 61.5% of FE gates.

5) Hash Function: last block of the reconstruction phase. It performs privacy amplification. This block concatenates the 1032 bits (12×86 iterations) received from the Golay Decoder and applies the hash function on it to calculate the 128 bit Cryptographic Key.

Our Hash Function comprises three main components: an input buffer, a *Linear Feedback Shift Register* (LFSR) and an accumulator register. First, the input *HF_I* is copied to the input buffer. The input buffer is then analyzed bit per bit: if a bit is one, the current LFSR output (which updates itself each cycle based on its polynomial function) is added (XORed) with the accumulator; however, if the bit is zero, the accumulator keeps its value. When all input bits are analyzed the value of the accumulator register is propagated to the output. The Hash Function has a latency of 32 clock cycles and it comprises 18.2% of FE gates.

It is worth noting that the Fuzzy Extractor presented here is a *generic* and simplified construction of an industrial implementation [17]; therefore, any test method developed for this circuit can be applied also to any other implementation.

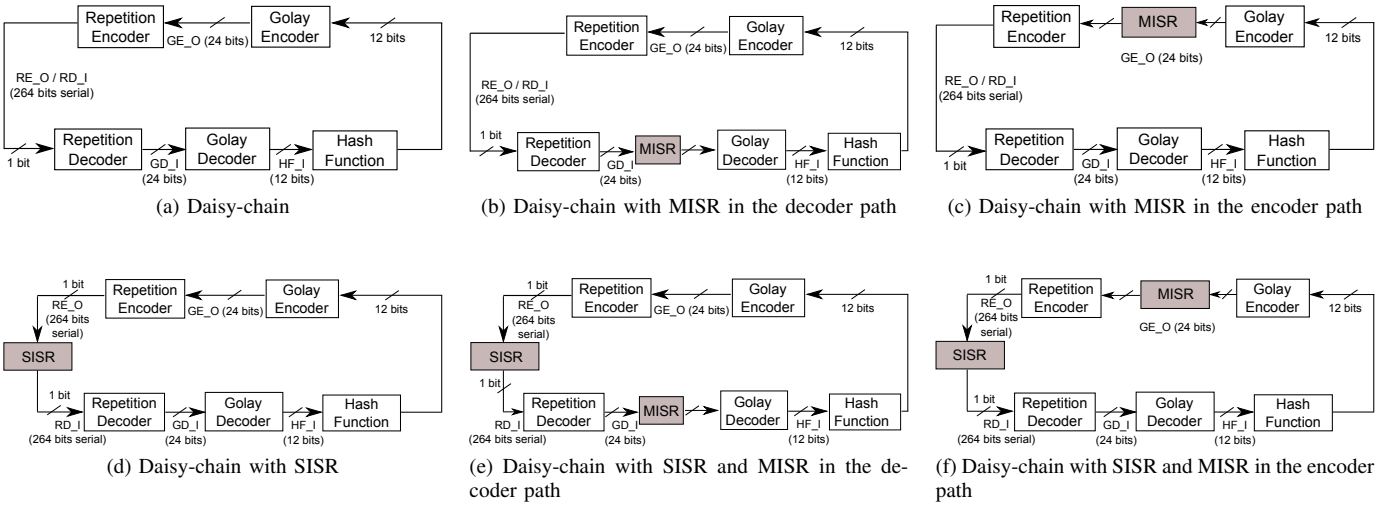
IV. Experiments Results

In this section, first, we define test and security requirements considered for the development of our test solution. Then, we present our test method and the experiments. Thereafter, we present the results. Finally, we provide a list of recommendations for secure testing of FE.

A. Test versus Security Requirements

Efficient test solutions for FE must prevent compromising the system security. The following requirements and assumptions apply:

(a) The signals *PUF*, *Random Seed* and *HF_I* (see Fig. 3) shall not be revealed at any time, partially nor fully.



(a) Daisy-chain

(b) Daisy-chain with MISR in the decoder path

(c) Daisy-chain with MISR in the encoder path

(d) Daisy-chain with SISR

(e) Daisy-chain with SISR and MISR in the decoder path

(f) Daisy-chain with SISR and MISR in the encoder path

Fig. 4: Daisy-chain for a FE - different constructions

An attacker learning this information might derive the *Cryptographic Key*, breaking the systems security.

(b) Helper data is assumed to be public knowledge and does not have to be secured.

(c) Reverse engineering the Fuzzy Extractor is not an issue. The Fuzzy Extractor uses algorithms that are standardized and publicly known.

(d) The PUF circuitry has its own internal test method, therefore it is outside the scope of this work.

B. Daisy-chain secure test method

Next, we propose and discuss a test method for FE, a daisy-chain based test method. The method is scan-chain free, which is a requirement in our case.

In this method, we propose to reuse the *Linear Feedback Shift Register* (LFSR) of the Hash Function block to create a random generator and test the FE in a loop-chain fashion, i.e., the outputs of each block are directly provided as inputs to next block as depicted in Fig. 4(a). This approach results in a negligible area overhead. However, a high fault coverage for the Golay Decoder cannot be guaranteed. This is because the Golay Decoder receives *always* error free input messages as provided by the Golay Encoder, which prevents the correct checking of all the decoder' states. Hence, reusing LFSR of hash function with daisy-chain approach *alone* will not provide the required test quality for Golay Decoder. To solve this problem, the randomness of the patterns provided at the Golay Decoder inputs (generated by the Golay Encoder) have to be improved, in order to trigger all states of the Golay Decoder FSM. This can be done by inserting a *Multiple-Input-Shift-Register* (MISR) at the input of the Golay Decoder as seen in Fig. 4(b). However, as the blocks are connected in a loop, the desired effect of randomness improvement can also be achieved by placing a MISR in any location between the Golay Encoder output and the Golay Decoder input (such as at the output of Golay Encoder in Fig. 4(c)), or a *Single-Input-Shift-Register* (SISR) in case the location is just a serial line; see Fig. 4(d). Moreover, a combination of MISR and SISR can be also used as shown in Fig. 4(e) and Fig. 4(f).

Comparing the cost (area overhead) and randomness of the several scenarios presented in Fig. 4 reveals that:

1) Scenario (d) results in the smallest area overhead.
2) Scenarios (b) and (c) as well as (e) and (f) are equivalent, reducing the number of scenarios to four.
3) Scenarios (e) and (f) could lead to higher fault coverage, as the combination of using SISR and MISR could improve the randomness.

C. Experiments performed

We synthesize a Fuzzy Extractor described in VHDL in 0.35μm technology node with Synopsys Design Compiler. The design compiler outputs a verilog netlist that is used to extract a fault list with the *Automated Test Pattern Generation* (ATPG) tool Synopsys TetraMAX. We use LIFTING fault simulator optimized for functional BIST to analyze the fault coverage [25]. The results are analyzed with MATLAB.

To evaluate the quality of the proposed solutions in Fig. 4 in terms of fault coverage and test time, we perform the following experiments, as described next.

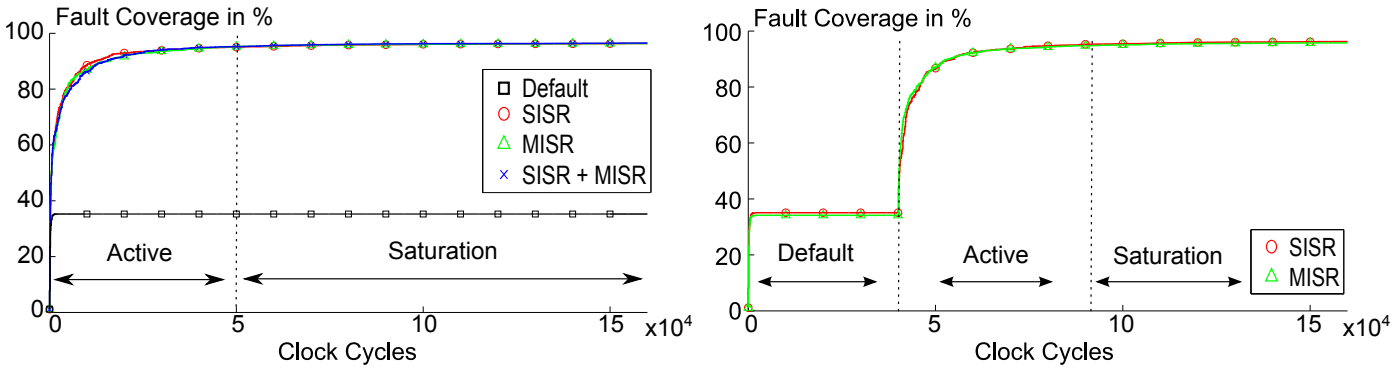
(i) **Default:** in this experiment, we simulate the circuit as in Fig. 4(a) for 15×10^4 clock cycles and analyze the fault coverage. This number of clock cycles is assumed to be our test time budget for all remaining experiments.

(ii) **MISR:** in this experiment, we simulate the circuit as in Fig. 4(b) (equivalent to Fig. 4(c)).

(iii) **SISR:** in this experiment, we simulate the circuit as in Fig. 4(d).

(iv) **SISR + MISR:** in this experiment, we simulate the circuit combining SISR and MISR as in Fig. 4(e) (equivalent to Fig. 4(f)).

(v) **Default + SISR:** in this experiment, we simulate the circuit in two stages. First, as in Fig. 4(a), we simulate



(a) Without default stage

(b) With default stage

Fig. 5: Fault coverage of Fuzzy Extractor Daisy-chain test method versus clock cycles

the FE using the default loop-chain for 25% of the test time budget. Second, as in Fig. 4(d), we include the SISR on the chain flow (between the Repetition Encoder and Repetition Decoder blocks) and analyze its fault coverage over the remaining 75% of the time. The goal of this experiment is to analyze the impact of *combining* the default scenario in Fig. 4(a) with the SISR in Fig. 4(d) on the fault coverage.

(vi) **Default + MISR:** in this experiment, we repeat the procedure of (v), but replacing the SISR with a MISR.

D. Results

Fig. 5 shows the fault coverage (y-axis) versus number of clock cycles (x-axis) of the experiments. Part (a) of the figure shows four first experiments, while part (b) shows the remainder experiments that include the default stage.

From Fig. 5 (a) we can observe the following.

1) During the default stage the Fuzzy Extractor realizes a fault coverage of only 36%. This fault coverage is realized quickly in the first 2k clock cycles. The figure clearly shows that the fault coverage remains stable at 36% during the remaining clock cycles.
2) For the remaining schemes, after 5×10^4 clock cycles the fault coverage reaches 95.1%. The remaining 10×10^4 clock cycles lead to an increment of only 1.2% (from 95.1% up to 96.2%) fault coverage.
3) There is no major difference in fault coverage between SISR, MISR and their combination. All the techniques realize the same fault coverage in similar test time, i.e., 95.1% is achieved at 5×10^4 (50k) clock cycles.

From Fig. 5 (b) we can observe the following.

1) Switching to SISR/MISR after the default stage strongly increases the fault coverage, i.e, the fault coverage increases from 36% up to 95.1% in 6.4×10^4 clock cycles.
2) After switching to SISR/MISR, it takes 5.8×10^4 clock cycles to reach a fault coverage of 95.1%. Extending the test time to 15×10^4 increases the fault coverage to 96.3%.
3) The impact of combining the default stage with either a SISR or MISR on the fault coverage is negligible.

The area overhead is measured in 0.35μm technology with the following results: 2.2% for SISR, 6.80% for MISR and 8.0% for SISR and MISR combined.

Analyzing the results, we can see that (i) it is critical to randomize the output of the Golay Encoder block. (ii) the final obtained fault coverage in both figures is similar. From this we conclude that the default stage is superfluous. (iii) in terms of fault coverage and test time, there is no difference between a SISR, MISR and a combination of both. (iv) in addition, if we consider the area overhead, the most efficient solution is to use the SISR only.

The fault coverage of our method is in line with the fault coverage reported in other self-test methods [10] and [11]. However, due to the nature of the extra FE components were required (such as SISR/MISR) to increase the fault coverage. The area overhead of the proposed method is negligible, which is intrinsic to methods that reuse hardware.

E. Recommendations

We provide a generic step-by-step procedure for secure testing of FE based on our findings. These steps are:

- The first step is to identify each block and to deeply understand its functionality (which is critical for successful functional testing).
- Second, the characteristics of each block need to be assessed (e.g., its area overhead, if the block comprises a state-machine or not, state-machine complexity, etc).
- Third, perform testability analysis of each block by considering a random input source. Identify eventual challenges by testing a certain block with this method.
- Fourth, identify if there is need of extra components in order to increase the fault coverage, such as a SISR. If so, analyze the trade-off of such components in terms of its possible locations and of its impact on security, fault coverage, test time and area overhead.
- Fifth, optimize the test solution; e.g., by using a SISR or MISR to activate uncovered paths in a state-machine.
- Sixth, analyzed the FC test time and area overhead of

the test method separately and when combined with complementary schemes.
- Finally, determine and select the best test method combined with complementary schemes (such as SISR/MISR) to meet the design requirements.

Following these steps will enable a secure test scheme with low area overhead and high test quality for any construction of Fuzzy Extractor.

V. Conclusion

In this paper we demonstrated a secure test method for a Fuzzy Extractor, scan-chain free to make sure that secret information stays inside the module and cannot be read out. The secure test method is based on daisy-chains; it reuses Fuzzy Extractor blocks for test pattern generation and output compression. The results show that the method has an inherent low area overhead 2.2%, while it realizes a fault coverage of 95.1% using only 50k clock cycles. Finally, we provided a generic step-by-step procedure to test any given Fuzzy Extractor based on our findings.

Acknowledgments

The authors would like to thank Peter Simons and Geert-Jan Schrijen from Intrinsic-ID B.V. for all the useful discussions.

References

[1] J. Guajardo *et al.* , "FPGA Intrinsic PUFs and Their Use for IP Protection", *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pp.63-80, Sept. 2007.

[2] B. Skoric, P. Tuyls, and W. Ophey, "Robust key extraction from Physical Unclonable Functions", *Applied Cryptography and Network Security*, vol.3531 of LNCS, pg.99135, Springer Berlin / Heidelberg, 2005.

[3] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data", *vol. 3027*, LNCS, 2004.

[4] J.-P. Linnartz and P. Tuyls, "New shielding functions to enhance privacy and prevent misuse of biometric templates", *Proc. of AVBPA03*, pp.393-402, Springer Berlin / Heidelberg, 2003.

[5] B. Yang, K. Wu and R. Karri, "Secure Scan: A Design-for-Test Architecture for Crypto Chips", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.25, no.10, pp.2287-2293, Oct. 2006.

[6] A. Das and Ünal Kocabaş and Ahmad-Reza Sadeghi and Ingrid Verbauwhede, "PUF-based Secure Test Wrapper Design for Cryptographic SoC Testing", Design, Automation and Test in Europe, 2012

[7] D. Hely, F. Bancel, M.-L. Flottes and B. Rouzeyre, "A secure Scan Design Methodology", Design, Automation and Test in Europe, 2006.

[8] J. Lee, M. Tehranipoor, C. Patel and J. Plusquellic, "Securing Scan Design Using Lock and Key Technique", *IEEE Int. Symp. on Defect and Fault Tolerance in VLSI System*, 2005

[9] J. da Rolt, G di Natale, M.-L. Flottes and B. Rouzeyre, "New security threats against chips containing scan chain structure", *IEEE Int. Symp. on Hardware-Oriented Security and Trust*, 2012

[10] M. Doulcier, M.L. Flottes and B. Rouzeyre, "AES-based BIST: self-test, test pattern generation and signature analysis", *IEEE International Symposium on Electronic Design, Test & Applications*, 2008

[11] G. di Natale, M. Doulcier, M.-L. Flottes and B. Rouzeyre "Self-Test Techniques for Crypto-Devices", *IEEE Trans. on VLSI Systems*, vol. 18, no. 2, Feb. 2010

[12] D.K. Pradhan and M. Chatterjee, "GLFSR-a new test pattern generator for built-in-self-test", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.18, no.2, pp.238-247, Feb. 1999.

[13] S. Boubezari and B. Kaminska, "A deterministic built-in self-test generator based on cellular automata structures", *IEEE Trans. on Computers*, vol.44, no.6, pp.805-816, Jun. 1995.

[14] "http://mathworld.wolfram.com/GolayCode.html"

[15] V. Pless, "Decoding the golay codes", *IEEE Trans. on Information Theory*, vol.32, no.4, pp.561-567, July 1986.

[16] C.V. Krishna, A. Jas and N.A. Touba, "Test vector encoding using partial LFSR reseeding", Int. Test Conference, pp.885-893, 2001.

[17] "www.unique-project.eu"

[18] A.A. Al-Yamani and E.J. McCluskey, "Built-in reseeding for serial BIST", *VLSI Test Symposium*, pp.63-68, 2003.

[19] S. Hellebrand, *et al.* , "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers" *IEEE Trans. on Computers*, vol.44, no.2, pp.223-233, Feb. 1995.

[20] L. Lei and K. Chakrabarty, "Test set embedding for deterministic BIST using a reconfigurable interconnection network", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.23, no.9, pp.1289-1305, Sept. 2004.

[21] N.A. Touba and E.J. McCluskey, "Bit-fixing in pseudorandom sequences for scan BIST", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.20, no.4, pp.545-555, Apr. 2001.

[22] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST", IEEE/ACM Int. Conf. on Computer-Aided Design, pp.337-343, 1996

[23] M. Arai *et al.* , "Seed selection procedure for LFSR-based BIST with multiple scan chains and phase shifters", *Asian Test Symposium*, 2004

[24] M. Bellos, D. Kagaris and D. Nikolos, "Test Set Embedding Based on Phase Shifters", *European Dependable Computing Conf.*, 2002.

[25] "http://www.lirmm.fr/"