SCIS 2013 The 30th Symposium on
Cryptography and Information Security
kyoto, Japan, Jan. 22 - 25, 2013
The Institute of Electronics,
Information and Communication Engineers

The Implementation of Fuzzy Extractor is Not Hard to Do : An Approach Using PUF Data

Hyunho Kang * Yohei Hori * Toshihiro Katashita * Manabu Hagiwara *

Abstract— The extraction of a stable signal from noisy data is very useful in applications that aim to combine it with a cryptographic key. An approach based on an error correcting code was proposed by Dodis et al., which is known as a fuzzy extractor. Physical unclonable functions (PUFs) generate device-specific data streams, although PUFs are noisy functions. In this paper, we describe a method for preparing a PUF key during fuzzy extractor implementation. The experimental results showed that all possible combinations of input message length and the number of correctable errors were tested for a BCH code with codeword length N, which was the length of the PUF responses.

Keywords: Arbiter PUF, Fuzzy Extractor, Physical Unclonable Function (PUF)

1 Introduction

The fuzzy extractor scheme defined in [1] can derive reliable bit strings from noisy data and it is a very useful approach for applications that aim to combine it with a cryptographic key. Physical unclonable functions (PUFs) generate device-specific data streams by using manufacturing variations of each LSI. High security authentication is possible during secret key generation using PUFs, if a system requires the best extraction scheme.

Experimental studies of fuzzy extractors [2][3][4] have received considerable attention since this approach was proposed by Dodis et al., 2004. However, it appears to be difficult to implement initially. Thus, a test and an illustration of how to produce a key may facilitate a better understanding of a practical fuzzy extractor. We report the results of some implementation examples using PUF data and we present a detailed implementation diagram. We hope that this paper will help users to understand the implementation of this scheme.

The remainder of this paper is organized as follows. Section 2 presents the implementation approach and a detailed diagram. The experimental results and conclusions are provided in Sections 3 and 4, respectively.

2 Implementation approach

The key result provided by Dodis et al. [1] demonstrated that fuzzy extractors can be built from secure sketches using strong randomness extractors, as shown in Fig. 1. During the generation procedure, **SS** is applied to noisy data **w** and a random input message **r** is used to obtain **s**, while a strong extractor **Ext** with

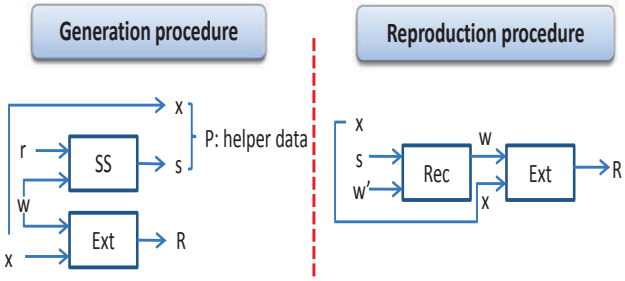

Figure 1: Typical scheme for a fuzzy extractor.

randomness of **x** to **w** is used to obtain almost uniform randomness **R**. The pair **(x, s)** is stored as helper data, **P**. During the reproduction procedure, the helper data is used to regenerate the output **R** from new noisy data **w′** based on **Rec(w′,s)** and **Ext(w,x)**.

During the implementation of this scheme, there are two important considerations, i.e., a combination of information reconciliation and privacy amplification. The information reconciliation step guarantees the elimination of noise from the measured noisy data. Privacy amplification guarantees the uniform distribution of the derived key bits. A BCH code and SHA-256 hash function were used to address these two basic requirements.

In this paper, I examine the fuzzy extractor performance of our Arbiter PUF by presenting results for all possible combinations of the message length and the number of correctable errors for a BCH code with a fixed codeword length (i.e., 127, 255, and 511).

Figure 2 shows the implementation diagram for Fuzzy extractors using the BCH code and hash function (N = 127). This diagram helps us to understand how the system operates.

* Research Institute for Secure Systems (RISEC), National Institute of Advanced Industrial Science and Technology (AIST), 1-1-1, Umezono, Tsukuba-shi, Ibaraki 305-8568, Japan ({h-kang, hori.y, t-katashita, hagiwara.hagiwara}@aist.go.jp)

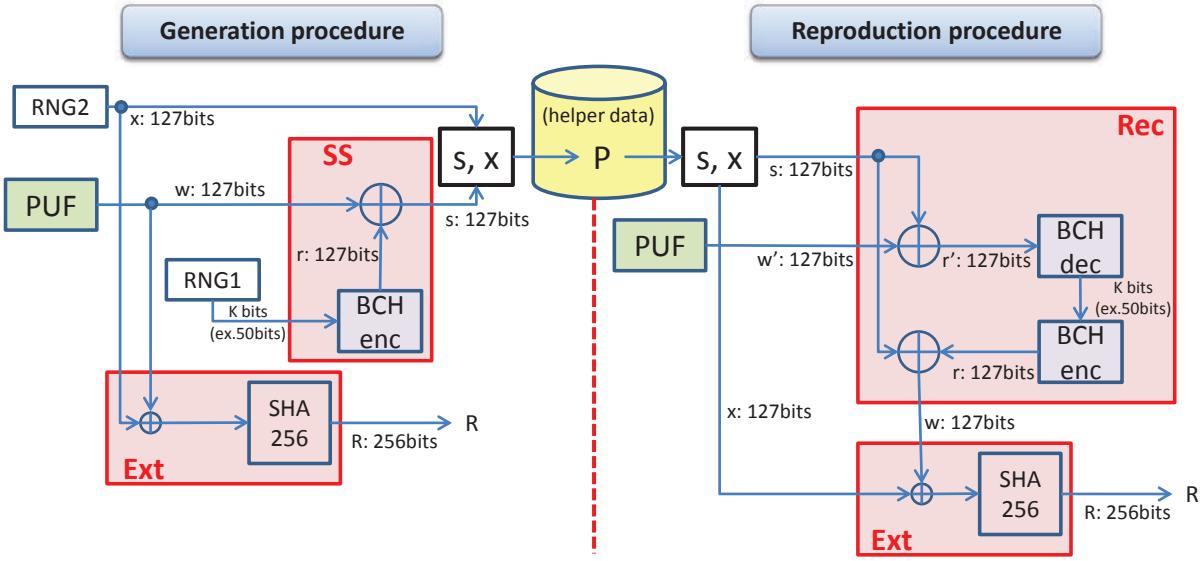

Figure 2: Implementation diagram for our fuzzy extractor (N = 127).

3 Experimental results

The FPGA used in this experiment was a Xilinx Virtex–5LX (xc5vlx30–ffg324), which operated on a SASEBO–GII evaluation board [5][6]. All of the performance results in this paper were generated using MATLAB. For example, the BCH code implementation is readily available in the Communications System Toolbox in MATLAB.

3.1 Performance of the two PUFs tested

In this section, we discuss the performance of the two Arbiter PUFs that we tested, before moving onto the fuzzy extractor performance. Reliability and uniqueness are commonly used to evaluate the performance of PUFs. In this study, we selected challenge response pair data for 100 test iterations using 500 IDs from each tested Arbiter PUF (two different FPGAs were implemented using the same circuit structure, as shown in Fig. 3).

3.1.1 Reliability

A comparison of the *SC Intra* and *DC Intra* of PUF1 in Figure 4 shows that the HD of the PUF was divided into two distinct classes, depending on the properties of the challenge. The peaks of the two sets of histograms were clearly separated, which indicated that there were no errors in terms of the false acceptance rate and false rejection rate.

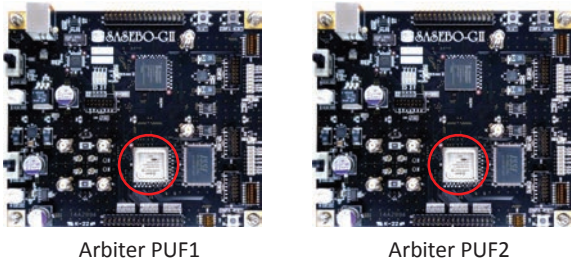

Figure 3: The two PUFs tested on the SASEBO-GII.

3.1.2 Uniqueness

We tested the uniqueness of the two Arbiter PUFs by finding all of the *SC Intra* and *SC Inter* HDs. As shown in Figure 5, there were no identification errors because there were no overlaps in the Intra and Inter SC distributions. To maintain stable security, it is desirable to separate the two distributions adequately. Thus, three types of response length were used to test the performance variation.

3.2 Fuzzy extractor performance of our Arbiter PUFs

As mentioned earlier, all possible combinations were used as parameters of the BCH code in each response length to examine the fuzzy extractor performance of our Arbiter PUFs.

Figure 6∼8 show the Hamming distance between two extracted keys when the two tested PUFs were the same, which demonstrates the dependency of the number of correctable errors on the testing index. Figure 6 shows that the response errors in all tests were corrected from an index of 5. In Fig. 7 and 8, the errors were corrected from indices of 10 and 17, respectively.

(Note: the test index of the enrolled PUF response occurs first, as shown in Fig. 6∼8.)

Figure 9∼11 show the Hamming distance between two extracted keys when two different PUFs were tested. Figure 9 shows that the response errors in all tests were corrected from an index of 17 because of the authentication of different Arbiter PUFs. In Fig. 10 and 11, the errors were corrected from indices of 33 and 56, respectively.

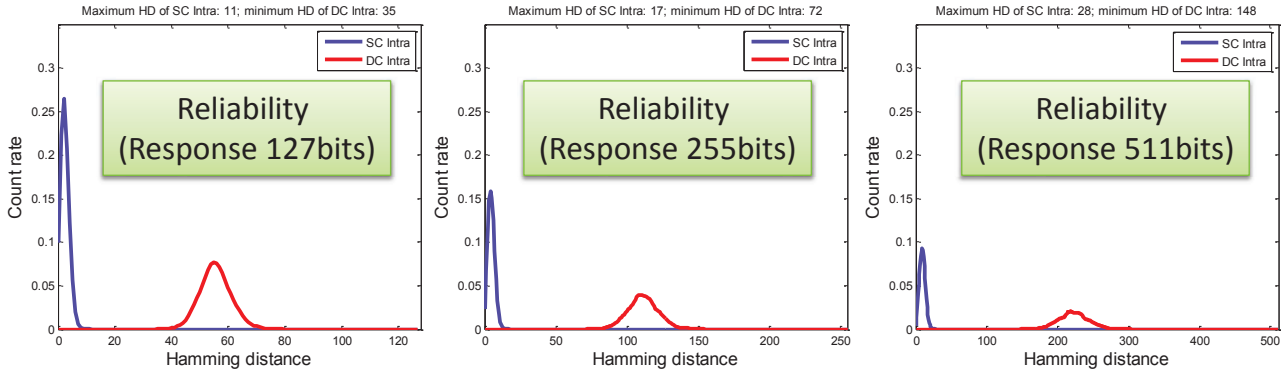

Figure 4: Reliability.

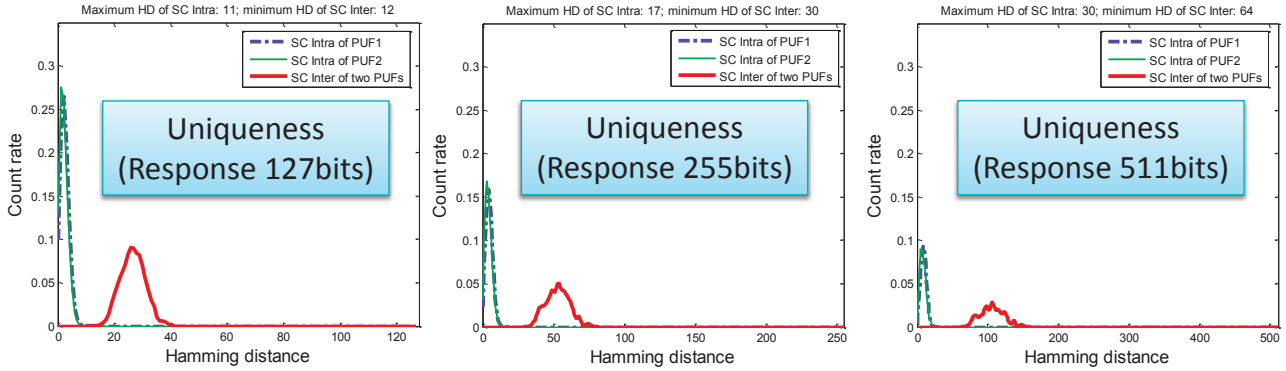

Figure 5: Uniqueness.

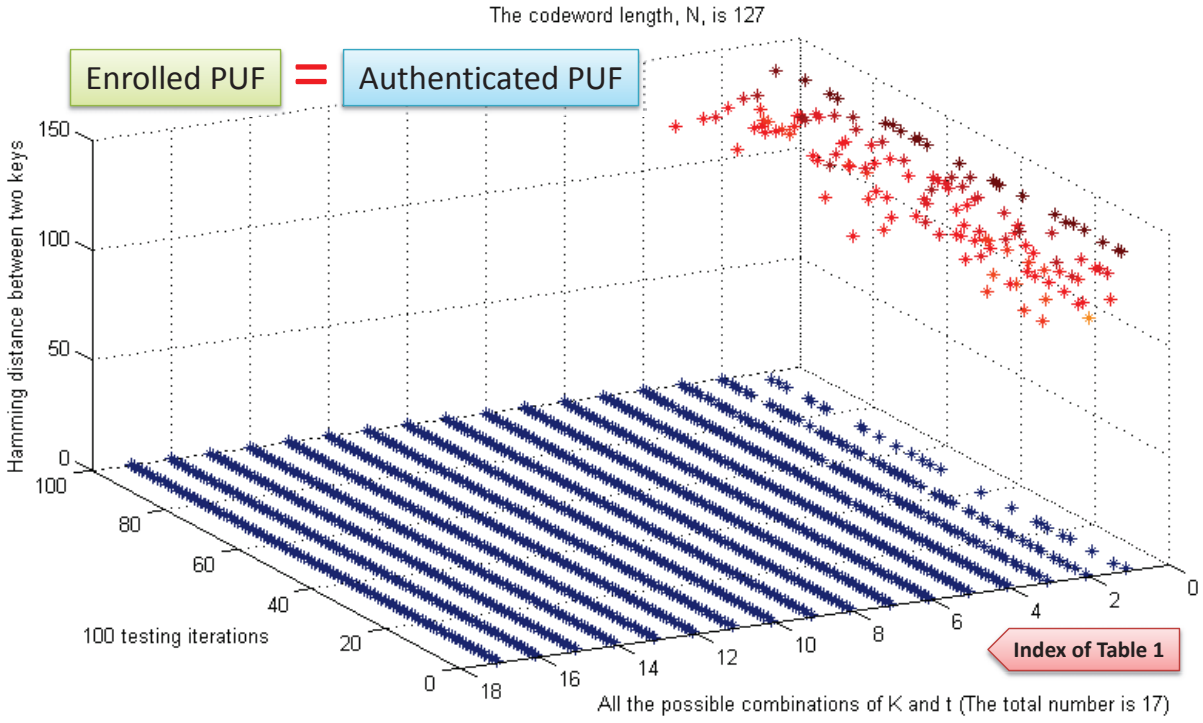

Figure 6: Hamming distance between two extracted keys (N=127).

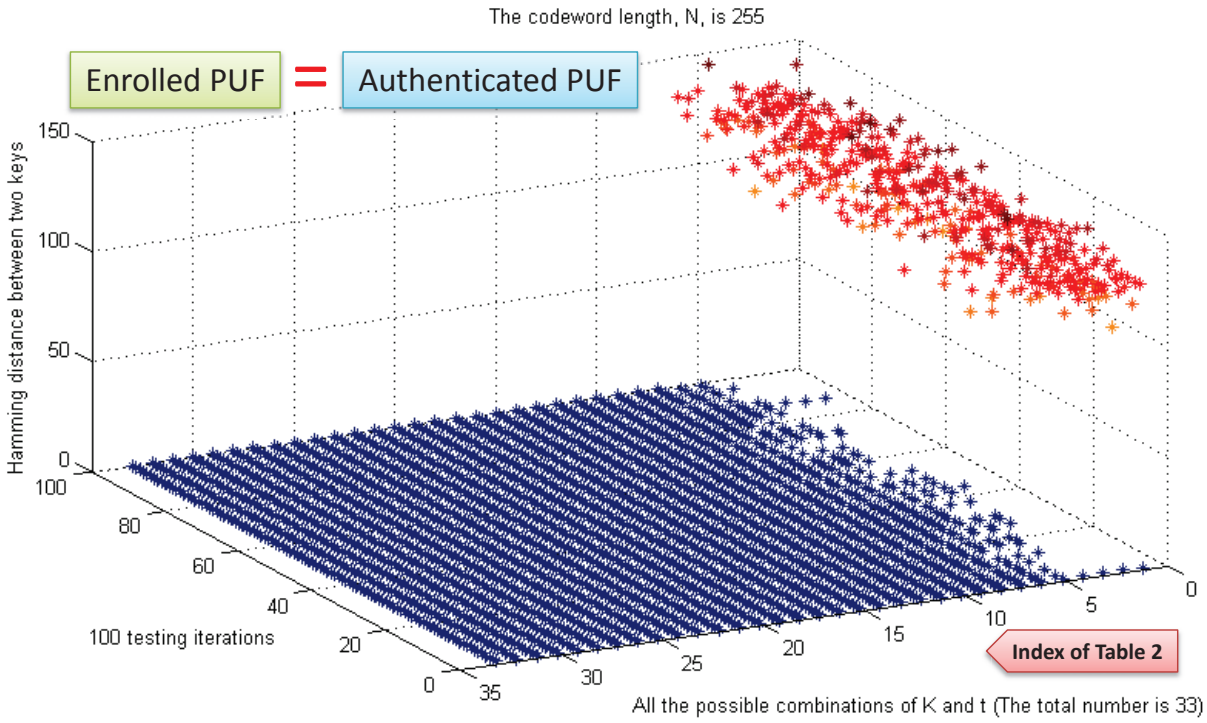

Figure 7: Hamming distance between two extracted keys (N = 255).

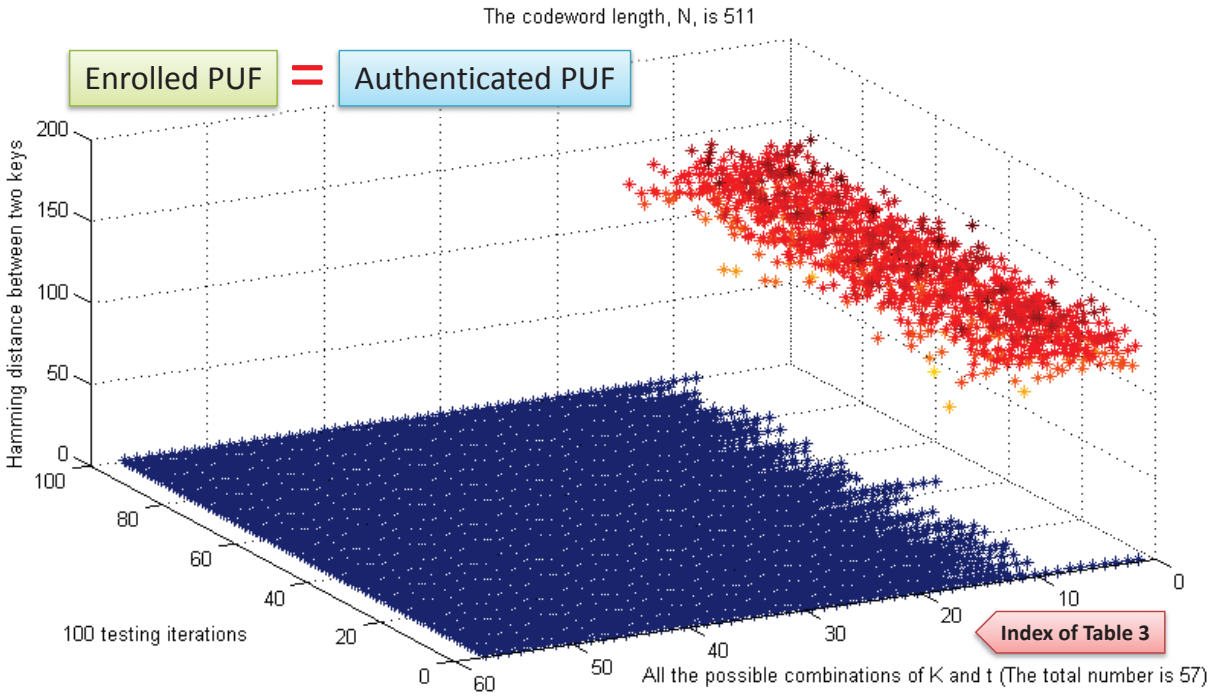

Figure 8: Hamming distance between two extracted keys (N = 511).

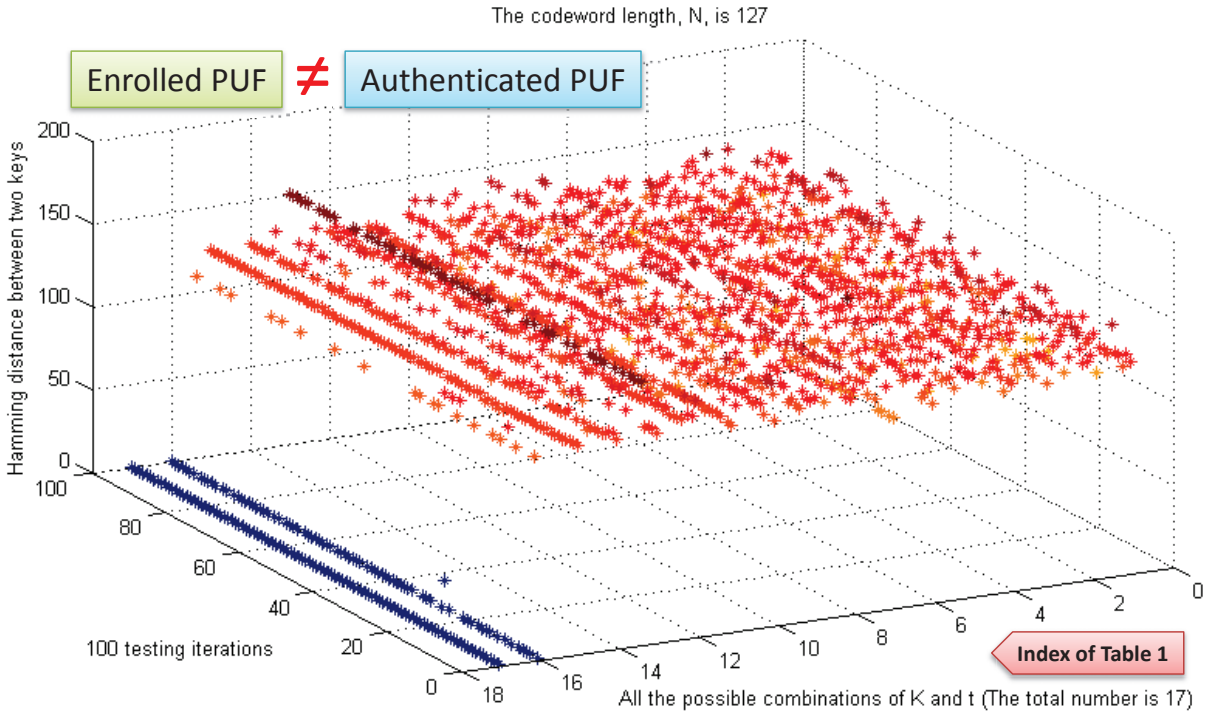

Figure 9: Hamming distance between two extracted keys (N = 127).

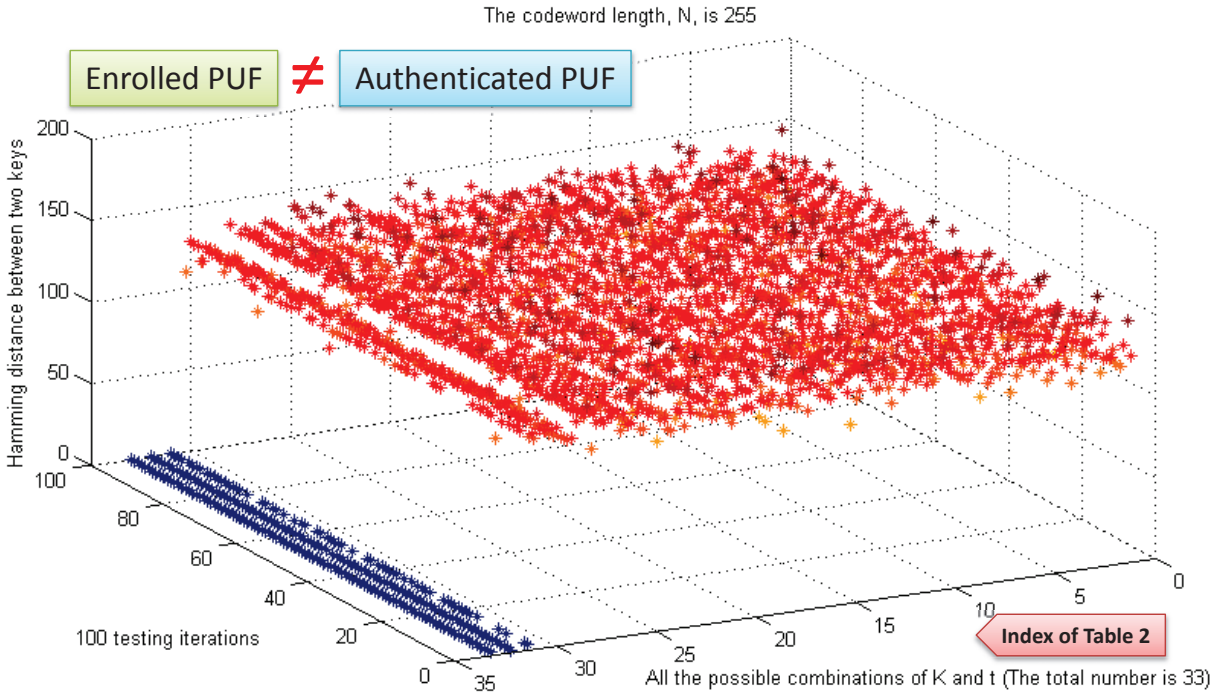

Figure 10: Hamming distance between two extracted keys (N = 255).

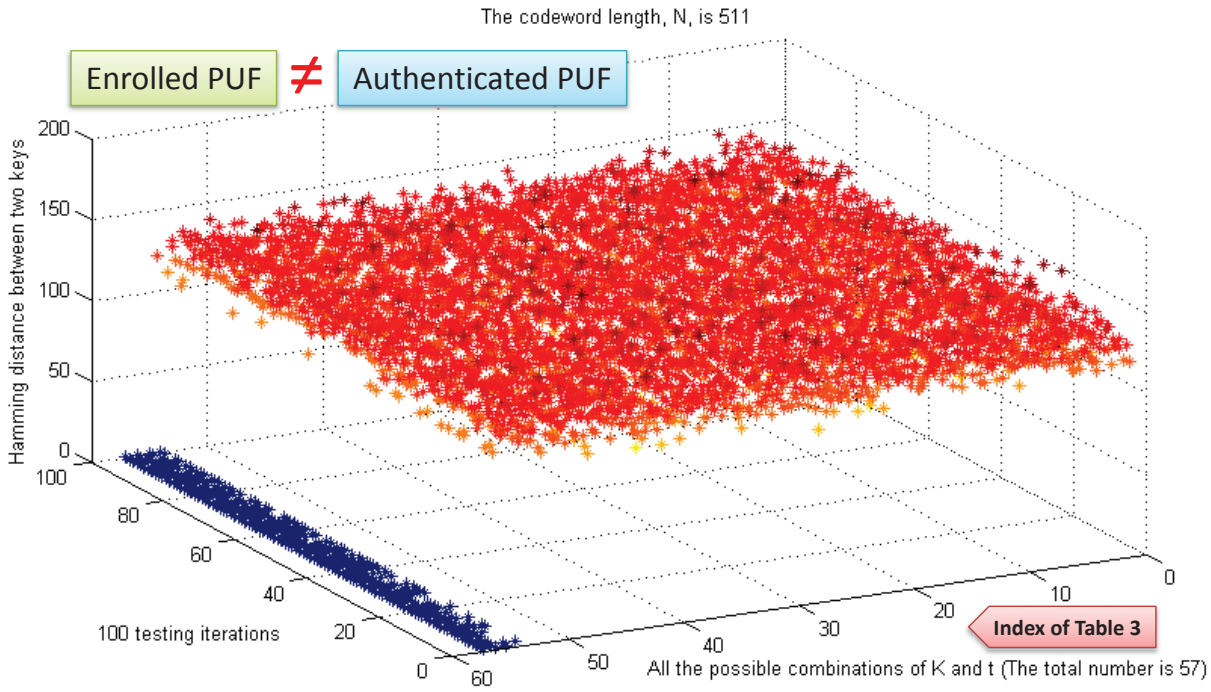

Figure 11: Hamming distance between two extracted keys (N = 511).

4 Conclusion

The main aim of this study was to investigate the fuzzy extractor performance of our Arbiter PUF. This test may be helpful to facilitate the understanding of fuzzy extractor implementation.

Acknowledgments

This work was funded in part by the Core Research for Evolutional Science & Technology (CREST) program of the Japan Science and Technology Agency (JST).

References

[1] Y. Dodis, R. Ostrovsky, L. Reyzin, A Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data,"(A preliminary version of this paper appeared in Eurocrypt 2004.) SIAM J. Comput., 38(1), pp. 97–139, 2008.

[2] P. Bulens, F.-X. Standaert, J.-J. Quisquater, "How to strongly link data and its medium: the paper case," IET Inf. Secur., Vol. 4, Iss. 3, pp. 125–136, 2010.

[3] Ya. N. Imamverdiev, L. V. Sukhostat, "A Method for Cryptographic Key Generation from Fingerprints," Automatic Control and Computer Sciences, Vol. 46, No. 2, pp. 66–75, 2012.

[4] V. van der Leest, E. van der Sluis, G-J, Schrijen, P. Tuyls, H. Handschuh, "Efficient Implementation of True Random Number Generator Based on SRAM PUFs," Cryptography and Security: From Theory to Applications, LNCS6805, Springer, pp. 300–318, 2012.

[5] A. Satoh, T. Katashita, H. Sakane, "Secure implementation of cryptographic modules–Development of a standard evaluation environment for side channel attacks–," Synthesiology-English edition, Vol. 3, No. 1, pp. 86–95, 2010.

[6] The download site of Side-channel Attack Standard Evaluation BOard support file in AIST (National Institute of Advanced Industrial Science and Technology), http://www.risec.aist.go.jp/project/sasebo/.

Table 1: Number of correctable errors in the BCH code, for N = 127

index	N	K	t	index	N	K	t
1	127	120	1	10	127	57	11
2	127	113	2	11	127	50	13
3	127	106	3	12	127	43	14
4	127	99	4	13	127	36	15
5	127	92	5	14	127	29	21
6	127	85	6	15	127	22	23
7	127	78	7	16	127	15	27
8	127	71	9	17	127	8	31
9	127	64	10				

Table 2: Number of correctable errors in the BCH code, for N = 255

index	N	K	t	index	N	K	t
1	255	247	1	18	255	115	21
2	255	239	2	19	255	107	22
3	255	231	3	20	255	99	23
4	255	223	4	21	255	91	25
5	255	215	5	22	255	87	26
6	255	207	6	23	255	79	27
7	255	199	7	24	255	71	29
8	255	191	8	25	255	63	30
9	255	187	9	26	255	55	31
10	255	179	10	27	255	47	42
11	255	171	11	28	255	45	43
12	255	163	12	29	255	37	45
13	255	155	13	30	255	29	47
14	255	147	14	31	255	21	55
15	255	139	15	32	255	13	59
16	255	131	18	33	255	9	63
17	255	123	19				

Table 3: Number of correctable errors in the BCH code, for N = 511

index	N	K	t	index	N	K	t
1	511	502	1	30	511	241	36
2	511	493	2	31	511	238	37
3	511	484	3	32	511	229	38
4	511	475	4	33	511	220	39
5	511	466	5	34	511	211	41
6	511	457	6	35	511	202	42
7	511	448	7	36	511	193	43
8	511	439	8	37	511	184	45
9	511	430	9	38	511	175	46
10	511	421	10	39	511	166	47
11	511	412	11	40	511	157	51
12	511	403	12	41	511	148	53
13	511	394	13	42	511	139	54
14	511	385	14	43	511	130	55
15	511	376	15	44	511	121	58
16	511	367	17	45	511	112	59
17	511	358	18	46	511	103	61
18	511	349	19	47	511	94	62
19	511	340	20	48	511	85	63
20	511	331	21	49	511	76	85
21	511	322	22	50	511	67	87
22	511	313	23	51	511	58	91
23	511	304	25	52	511	49	93
24	511	295	26	53	511	40	95
25	511	286	27	54	511	31	109
26	511	277	28	55	511	28	111
27	511	268	29	56	511	19	119
28	511	259	30	57	511	10	127
29	511	250	31				