Achieving Side-Channel Protection with Dynamic Logic Reconfiguration on Modern FPGAs

Pascal Sasdrich*, Amir Moradi*, Oliver Mischke*†, Tim Güneysu*
*Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany
{pascal.sasdrich, amir.moradi, oliver.mischke, tim.gueneysu}@rub.de
†Infineon Technologies AG, Chip Card & Security Division, Munich, Germany
oliver.mischke@infineon.com

***Abstract*—Reconfigurability is a unique feature of modern FPGA devices to load hardware circuits just on demand. This also implies that a completely different set of circuits might operate at the exact same location of the FPGA at different time slots, making it difficult for an external observer or attacker to predict what will happen at what time. In this work we present and evaluate a novel hardware implementation of the lightweight cipher PRESENT with built-in side-channel countermeasures based on dynamic logic reconfiguration. In our design we make use of Configurable Look-Up Tables (CFGLUT) integrated in modern Xilinx FPGAs to nearly instantaneously change hardware internals of our cipher implementation for improved resistance against side-channel attacks. We provide evidence from practical experiments based on a Spartan-6 platform that even with 10 million recorded power traces we were unable to detect a first-order leakage using the state-of-the-art leakage assessment.**

I. Introduction

Side-channel analysis (SCA) exploits information leakage related to device internals, for example, by inspecting its power consumption [1]. This way, the theoretic security provided by a cryptographic primitive can be easily overcome if the device is not equipped with any SCA countermeasures. Many different countermeasures against SCA attacks have been already proposed that are typically classified as masking and hiding [2]. The main concept behind masking is to randomize the processed values by adding random masks so that it should become impossible for an attacker to predict intermediate values. With respect to signal-to-noise ratio other countermeasures aim at either increasing the noise by introducing noise generation resources [2], [3] or reducing the signal by e.g., equalizing the power consumption [4]. Despite the many proposals, only few of them are able to achieve the claimed level of security due to the presence of glitches inside the combinatorial masked circuits (for example see [5], [6]). Instead of masking combinatorial circuits, critical elements such as S-boxes can be realized as look-up tables that are dynamically randomized in memory. Such an approach based on scrambling block RAM (BRAM) contents of FPGAs has been presented in [3].

Contribution. In this work we present a novel implementation approach to randomize look-up tables which makes use of the dynamic reconfiguration feature of certain building blocks of FPGAs. In particular, our approach employs special LUT elements inside a Xilinx Spartan-6 FPGA as Configurable Look-Up Tables (CFGLUT) [7] that can change their configuration on the fly. This not only allows a more fine-grain but also by far a more flexible randomization of look-up tables on FPGAs compared to the previous approach using large and static 18 kBit BRAMs. As a case study we applied the new technique for dynamically randomized look-up tables on an implementation of the PRESENT cipher [8]. In addition, we decomposed the S-box into several stages and utilized register precharge to increase the overall resistance against SCA attacks. We tested the effectiveness of each countermeasure and their different combinations by applying the test vector leakage assessment (TVLA) methodology proposed by Goodwill et al. [9]. With all countermeasures enabled, we are unable to detect a first-order leakage on 10 million captured power traces from an SCA evaluation platform populated with a Spartan-6 FPGA. This significantly improves the results from [10] which is the only previous work that deals with using dynamic (partial) reconfiguration as SCA countermeasure.

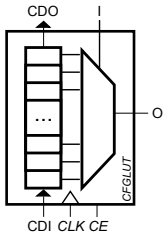
Outline. The remainder of this article is organized as follows: Section II explains how CFGLUTs of modern FPGAs can be used to create reconfigurable tables of any size. We also briefly introduce the reference architecture of PRESENT used for our experiments. Our portfolio of countermeasures, such as Boolean masking on LUTs, S-box decomposition, and register precharge, are presented in Section III. Evaluation results for our countermeasure is given in Section IV before we conclude our work in Section V.

II. Preliminaries

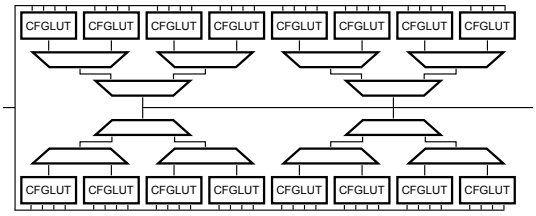
In the following we briefly describe the structure and function of Configurable Look-Up Tables (CFGLUT) and how any generic n-input and m-output Boolean logic function can be realized based on our Reconfigurable Function Table (RFT) concept. Afterwards, we introduce our basic architecture of the lightweight block cipher PRESENT.

A. Configurable Look-Up Table (CFGLUT)

Starting with the Virtex-5 device family, Xilinx devices are equipped with elements named Configurable Look-Up Tables that enable fast dynamic logic reconfiguration during runtime. Instead of modifying the configuration by full or partial



(a)



(b)

Fig. 1: (a) Configurable Look-Up Table as a basic module for Reconfigurable Function Tables (b) Exemplary design structure of a (6×4) Reconfigurable Function Table

bitstream reconfiguration, the function of Configurable LUTs can be loaded and replaced instantaneously without external intervention. Internally, each CFGLUT consists of a distributed memory block that can operate as a 16-bit Shift Register Look-Up Table (SRL16). Hence, even for older devices that do not provide dynamic logic reconfiguration, SRL16 instances can be used with slight effort to achieve the same functionality.

Figure 1a outlines the internal structure of a Configurable LUT that is composed of a 16-bit shift configuration memory and a subsequent multiplexer stage. Due to the restricted configuration memory size of 16 bits, each CFGLUT can realize at most a 4-input and 1-output Boolean function. In order to implement any $(n \times m)$ Boolean function, we now introduce the general concept for *Reconfigurable Function Tables*.

B. Reconfigurable Function Table (RFT)

In general, any $(n \times 1)$-output Boolean function can be realized using multiple (Configurable) Look-Up Tables combined by a multiplexer cascade. If this structure is instantiated m times with shared inputs, we call this a $(n \times m)$ Reconfigurable Function Table (RFT). Internally, each RFT consists of $m \cdot \lceil 2^{n-4} \rceil$ CFGLUT instances structured as shown in Figure 1b.

Using the example of an AES S-box the structure of a (8×8) RFT can be illustrated: an 8-input, 1-output Boolean function can be realized using 16 Configurable Look-Up Tables connected by cascading 15 multiplexers (each 2-to-1)[1]. This structure is instantiated 8 times while sharing the input signals. Thus, the entire AES S-box Reconfigurable Function Table could be built of 128 CFGLUT.

[1]This overhead can be reduced by utilizing LUTs as multiplexer.

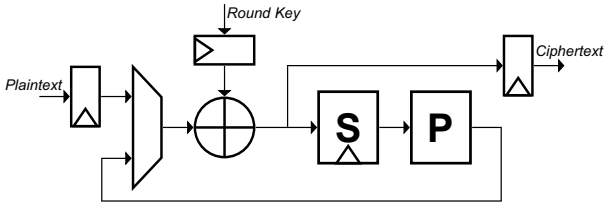


Fig. 2: Full-parallel round-based PRESENT encryption architecture

Obviously, for large RFTs this structure becomes inefficient since e.g., (excluding the necessary multiplexers) a non-reconfigurable AES S-box can be realized by 32 LUT6 compared to 128 CFGLUT which can be mounted into 64 LUT6. Therefore, we primarily propose RFTs for lightweight ciphers or those employing (4×4) S-boxes that can be realized more easily by an RFT of 4 CFGLUTs. In this work we have chosen to implement the PRESENT cipher as a case study.

C. Basic Architecture

PRESENT is a symmetric lightweight block cipher with a block size of 64 bits. The encryption scheme is based on a Substitution-Permutation (S/P) network encrypting a plaintext within 31 rounds using 32 sub-keys. Each sub-key is derived from an initial 80-bit (or 128-bit) key. Figure 2 provides an overview of our architecture design implemented on an FPGA. We opted to implement the PRESENT encryption scheme in a round-based architecture that requires two clock cycles per round and derives sub-keys on the fly. The datapath has a width of 64 bits and the substitution layer consists of 16 parallel S-boxes including the state registers. The permutation is applied bitwise and can be realized in hardware by plain routing resources. The total overhead of our protected design compared to a simple round based architecture is shown in Table I.

Design Component	**Unprotected**		**Protected**	
	Logic (LUT)	*Memory* (FF)	*Logic* (LUT)	*Memory* (FF)
Keyschedule	48	85	48	85
Round Function	128	64	224	128
Key Addition	64	-	64	-
Single S-box	4	4	10	8
Countermeasure Instance	-	-	1236	388
Decomp. + Masking	-	-	1172	260
Precharge	-	-	64	128
Latency	*Time* (Clock Cycles)		*Time* (Clock Cycles)	
S-box Decomposition	-		16*	
Reconfiguration	-		16	
Encryption	31		62	

*This time can be avoided if a new decomposition is computed in parallel to a previous encryption.

TABLE I: Comparison of Resource Utilization and Time Overhead for Unprotected and Protected PRESENT Designs

III. Countermeasures

The evaluation of hardware countermeasures is a tricky problem due to the many different reasons for side-channel leakages in the design space, including the routing and placement of resources. However, in order to fairly investigate and compare the effectiveness of our proposed countermeasures, we apply a modular approach. More precisely, each countermeasure can be separately enabled for individual and joint evaluation with others. We now delve into the details how we integrated the countermeasures into the basic architecture presented in Section II-C.

A. S-box Random Decomposition

Side-channel leakages can often be found in power traces on critical transitions in the combinatorial circuit after a change in a driving register. Since most side-channel attacks on symmetric block ciphers target the output of the non-linear substitution layer, it might be beneficial to avoid the storage of the S-box outputs into such registers. Therefore, we introduce the idea of a random S-box decomposition. By moving the state register into the substitution layer, we split the standard S-box up into two (random) mappings. Hence, we never store a correct S-box output into a register but only (randomly) mapped values.

The two mappings, surrounding the state register, are built of two (4×4) RFTs in order to be able to update their configuration for every encryption. The first RFT is configured to realize a randomly selected bijection R_1. Hence, the second RFT should be configured to implement the bijection R_2 in such a way that $\forall x,\ R_2(R_1(x)) = S(x)$. This means that only the application of both mappings results in the correct S-box output which, however, is never stored to a register. Instead, the intermediate register only holds a value $R_1(x)$ which is unpredictable for an attacker.

The configuration of both RFTs (of R_1 and R_2) is computed randomly prior to every encryption within 16 clock cycles. R_1 is computed by swapping two random elements of the identity configuration while R_2 is computed using the equation $R_2(R_1(x)) = S(x)$. All 16 S-boxes of the round function share the same configuration and before the encryption of a plaintext can start, the derived configuration is loaded into the RFTs within another 16 clock cycles. It is possible to compute new configurations in parallel to an encryption, but to evaluate our design in a worst-case scenario (i.e., the best an attacker can achieve) we avoided such parallelism.

Note that if S-box decomposition is disabled, the configuration of R_1 is the standard PRESENT S-box, and hence R_2 realizes the identity mapping.

B. Boolean Masking

Algorithmic Boolean masking is intended to randomize the intermediate values of a cipher implementation by additively introducing random masks. Since the modification of intermediate values affects the computation, Boolean masking requires some adaptions for the implementation. In particular, the non-linear substitution layer, i.e., the S-box configuration of our cipher implementation, has to be updated depending on the randomly selected mask. In order to keep the masking countermeasure compatible to the S-box decomposition, we mask both RFTs (R_1 and R_2) independently based on m_1 and m_2 respectively. To keep the masked tables constant for every round we update the masks only prior to each encryption. The configuration of each table, i.e., R_1 and R_2, is then recomputed as follows:

$$R_1'(x) = R_1(x \oplus m_1) \oplus m_2 \tag{1}$$

$$R_2'(x) = R_2(x \oplus m_2) \oplus P^{-1}(m_1), \tag{2}$$

where P denotes the PRESENT PLayer. This means that each nibble of the round state is masked by a 4-bit mask m_1. After the computation of the first mapping R_1 (regardless of whether R_1 is a random mapping or the standard S-box), the mask is changed to m_2 and the masked state is stored to the register stage. Afterwards, the second mapping is applied, changing the mask for every nibble from m_2 to the inverse permutation of m_1. This means that after the linear permutation layer of PRESENT, the round state is again masked with the initial mask m_1. Thus, the masking is kept constant for all rounds of one encryption.

We need to emphasize that we do not reuse any masks at each round. In other words, for each state nibble, two independent 4-bit m_1 and m_2 are chosen from a uniform distribution. In total, for the masking our design requires 128 random bits prior to each encryption. The masking countermeasure can be disabled easily by setting all m_1 and m_2 to zero.

C. Register Precharge

Since in our design the masks do not change during one encryption, the round inputs (resp. outputs) are masked with the same masks. Hence, the state register introduces a leakage by storing the consecutive round inputs. Such a leakage can be easily detected by a Hamming distance model as

$$\mathrm{HD}(x \oplus m, y \oplus m) = \mathrm{HW}(x \oplus y).$$

Therefore, we expanded the single register stage into two registers to avoid such a leakage.

Depending on the initial content of the registers, the encryption rounds are always interleaved with another dummy encryption round. This technique avoids the aforementioned leakage but reduces the throughput by the factor of two. If this feature is disabled, the loop multiplexer (see Figure 2) passes the plaintext for two initial clock cycles, filling both registers with the same value.

The final design of our S-box, including all proposed countermeasures is shown in Figure 3. For our implementation, this S-box is instantiated 16 times as depicted in Figure 2.

IV. Practical Evaluation

We employed a SAKURA-G platform [11], i.e., a Spartan-6 FPGA, for practical SCA evaluations. The power consumption traces have been measured by means of a *pico Technology* digital oscilloscope (PicoScope 6402B) by monitoring the voltage drop over a $1\,\Omega$ resistor in the V_{dd} path. We have used

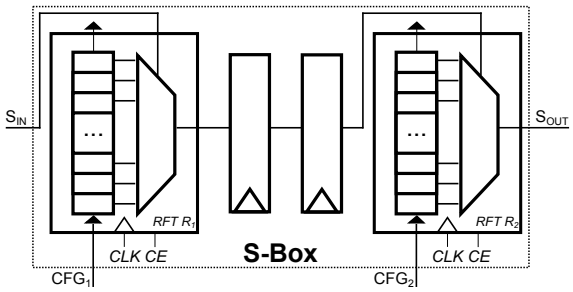


Fig. 3: 4-bit PRESENT S-box using Reconfigurable Function Tables

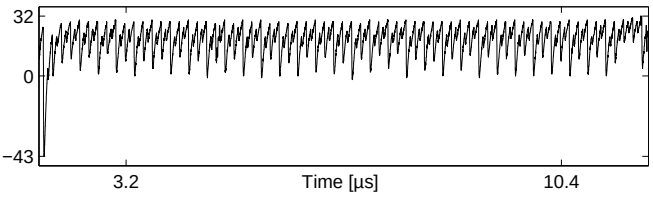


Fig. 4: Sample Power Trace

the embedded amplifier of the SAKURA-G and recorded the traces at a sampling rate of 625 MS/s while the design was running at a low clock frequency of 3 MHz to reduce noise caused by the overlap of power traces.

A. Specific Statistical t-test

In order to evaluate the resistance/vulnerability of our designs, we applied a *specific* statistical t-test [9]. The goal of such a scheme is not to examine a key-recover attack, rather – with respect to the underlying model – it provides an overview of the existence of a leakage which can be extracted by an attack. The concept of the specific t-test is based on the classical DPA attack of [1]. During the measurements the input (plaintext) is taken randomly while the key is fixed for all the collected traces. The masks are also drawn from a uniform distribution.

Following the concept of DPA, the traces T are categorized into two groups G_1 and G_2 with respect to the taken intermediate value. For such a categorization we considered three groups of models (in sum 160 models): 1) S-box output bits of one round (64 models), 2) XOR-result bits of the input of two consecutive rounds (64 models), 3) the 4-bit value of two S-Box outputs (each 16 models). For the sake of simplicity, we consider only one point of the collected traces in the following explanation.

Recall that Welch's (two-tailed) t-test is computed as

$$t = \frac{\mu(T \in G_1) - \mu(T \in G_2)}{\sqrt{\frac{\delta^2(T \in G_1)}{|G_1|} + \frac{\delta^2(T \in G_2)}{|G_2|}}},$$

where μ and δ^2 denote the sample mean and sample variance respectively, and $|.|$ stands for the cardinality. The t-test indeed examines the validity of the *null hypothesis* as the samples in both groups were drawn from the same population. If the null hypothesis is correct, it can be concluded with a high level of confidence that a corresponding DPA attack cannot distinguish the correct key.

For such a conclusion the Student's t-distribution function in addition to the degree of freedom is applied to determine the probability to reject the aforementioned hypothesis (for more information see [9]). For typical evaluations, a threshold as $|t| > 4.5$ is defined to reject the null hypothesis and conclude that a corresponding first-order attack is most likely feasible. This process is repeated at each sample point independently.

B. No Countermeasure

For comparison purposes we start our evaluation with a reference measurement, i.e., *Profile 0*, measuring one million encryption runs with random plaintexts and a fixed key as input and each of our proposed countermeasures disabled. Fig. 4 shows a sample power trace, where all 31 rounds, each taking 2 clock cycles, are clearly distinguishable. The results after the application of the specific statistical t-test for all 144 models are shown in Fig. 5a, 6a, 7a and 8a respectively. Obviously, for these measurements, the $|t|$ value exceeds the threshold of 4.5 for all models which means that all tests detect a first-order leakage.

C. Single Countermeasure

In the next step, we investigate all countermeasures insulated in order to assess the impact of each of them on the first-order leakage of our design. Therefore, the following three measurement profiles are evaluated:

1) *Profile 1*: S-box Random Decomposition
   This countermeasure was introduced to avoid buffering intermediate S-box values. The results of all applied tests, i.e., all 144 models, can be seen in Fig. 5b, 6b, 7b and 8b respectively. Although the S-box decomposition avoids the S-box output to be stored in a register, it is computed by the RFT of R_2, and its associated leakage is still detectable. According to the shown results, S-box random decomposition reduces the first-order leakage but is not sufficient to be solely applied.
2) *Profile 2*: Boolean Masking
   Masking of intermediate values aims at decorrelating power consumption and intermediate values using randomization of the processed values. The evaluation of this countermeasure based on 1 million power traces is shown in Fig. 5c, 6c, 7c and 8c. Due to the reuse of masks for all rounds, the Hamming distance between two consecutive values does not depend on the masks but only on the round outputs. Therefore, the evaluation results of Fig. 6c indicate a strong leakage for consecutive round values although round outputs are randomized.
3) *Profile 3*: Register Precharge
   An additional register stage and random initialization counteracts Hamming distance leakage because buffered data is not overwritten by consecutive results but random

values. The analysis results of this profile using 1 million power measurements can be seen in Fig. 5d, 6d, 7d and 8d. As expected, this countermeasure reduces the leakage related to the XOR between consecutive rounds' output (cf. Fig. 6d) but does not prevent the leakage associated to the S-box computation neither for the 64 models of the output bits nor for the 16 models of the output value.

D. Combination of Countermeasures

As a last step, we investigate all possible combinations of two or more countermeasures in order to find the best solution. This directly leads to the following four profiles:

1) *Profile 4*: Decomposition and Precharge
   In such a setting, precharging only prevents Hamming distance leakage, but the leakage of the S-box cannot be compensated. Fig. 5e, 6e,7e and 8e show the t-test results for all 144 models, but only for test group 2 the leakage could be reduced almost to the thresholds. For all other models still some leakage is detectable.
2) *Profile 5*: Masking and Precharge
   The evaluation results of the combination of Boolean masking and register precharge are shown in Fig. 5f, 6f, 7f and 8f. This combination already minimizes the leakage but still, in particular in Fig. 5f, some leakage is detectable using 1 million traces.
3) *Profile 6*: Decomposition and Masking
   The t-test results of the combination of S-box decomposition and Boolean masking can be seen in Fig. 5g, 6g, 7g and 8g. Apparently, this combination cannot compensate the problems of both countermeasures (leakage related to the XOR between consecutive rounds' output) and is not sufficient to prevent first-order leakage.
4) *Profile 7*: Decomposition, Masking and Precharge
   This last profile combines all our proposed countermeasures in order to benefit from their advantages. Fig. 5h, 6h, 7h and 8h show the result of the corresponding evaluations. In this case, we even recorded 10 million power traces, but could not detect any first-order leakage.

V. Conclusion

In this work we have presented a novel method to realize a first-order masking scheme based on the process of dynamic reconfiguration in modern FPGAs. Since the scheme uses precomputed masked S-boxes realized by Configurable Look-up Tables, its security is not affected by the known issues of masked hardware, e.g., glitches. By using reconfiguration we applied a dynamic decomposition of the S-boxes that includes precharging of registers as well.

We have practically examined all countermeasures and their combinations using the specific t-test leakage assessment methodology. We can report that even after recording 10 million power traces, we were not able to detect any first-order leakage when all countermeasures are active.

In short, we demonstrated an effective technique for FPGA-based platforms to achieve first-order SCA resistance. Compared to the known schemes, e.g., partial reconfiguration, our proposed solution has still reasonable overheads.

References

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology - CRYPTO '99, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.

[2] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[3] T. Güneysu and A. Moradi, "Generic Side-Channel Countermeasures for Reconfigurable Devices," in *Cryptographic Hardware and Embedded Systems - CHES 2011, Nara, Japan, September 28 - October 1, 2011. Proceedings*, ser. Lecture Notes in Computer Science, B. Preneel and T. Takagi, Eds., vol. 6917. Springer, 2011, pp. 33–48.

[4] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," in *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France*. IEEE Computer Society, 2004, pp. 246–251. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/DATE.2004.1268856

[5] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully Attacking Masked AES Hardware Implementations," in *Cryptographic Hardware and Embedded Systems - CHES 2005, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, ser. Lecture Notes in Computer Science, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 157–171.

[6] A. Moradi, O. Mischke, and T. Eisenbarth, "Correlation-Enhanced Power Analysis Collision Attack," in *Cryptographic Hardware and Embedded Systems, CHES 2010, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, ser. Lecture Notes in Computer Science, S. Mangard and F. Standaert, Eds., vol. 6225. Springer, 2010, pp. 125–139.

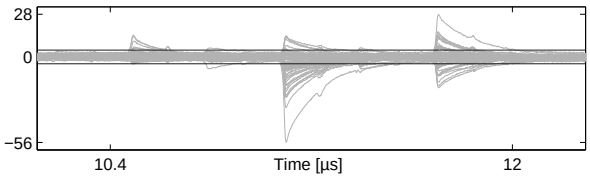
[7] Xilinx, "Spartan-6 Libraries Guide for HDL Designs (UG615 v 14.1)," Available via http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/spartan6_hdl.pdf, April 2012.

[8] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2007, Vienna, Austria, September 10-13, 2007, Proceedings*, ser. Lecture Notes in Computer Science, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, 2007, pp. 450–466.

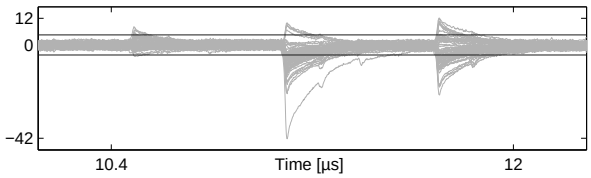
[9] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST Non-Invasive Attack Testing Workshop, Nara*, 2011. [Online]. Available: http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf

[10] N. Mentens, B. Gierlichs, and I. Verbauwhede, "Power and Fault Analysis Resistance in Hardware through Dynamic Reconfiguration," in *Cryptographic Hardware and Embedded Systems - CHES 2008, Washington, D.C., USA, August 10-13, 2008. Proceedings*, ser. Lecture Notes in Computer Science, E. Oswald and P. Rohatgi, Eds., vol. 5154. Springer, 2008, pp. 346–362.

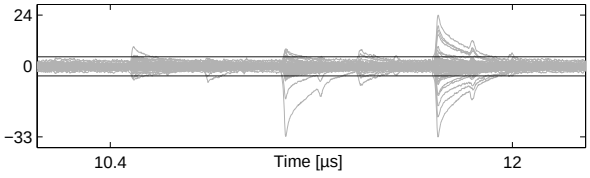
[11] H. Guntur, J. Ishii, and A. Satoh, "Side-channel AttacK User Reference Architecture SAKURA-G," in *GCCE 2014*. IEEE Computer Society, 2014, Further information are available via http://satoh.cs.uec.ac.jp/SAKURA/index.html.



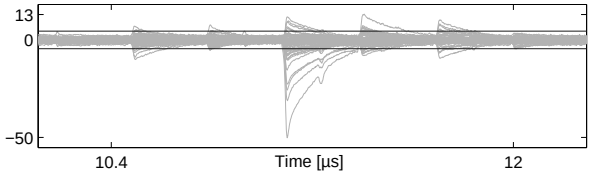
(a) *Profile 0*: No Countermeasure (1mio)



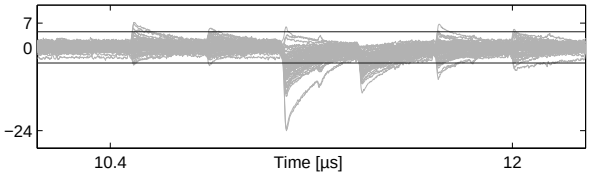
(b) *Profile 1*: S-box Decomposition (1mio)



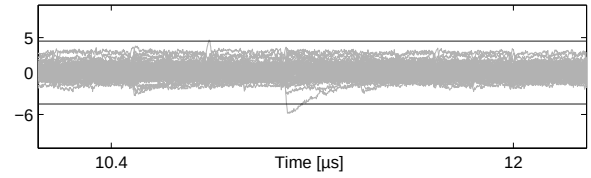
(c) *Profile 2*: Boolean Masking (1mio)



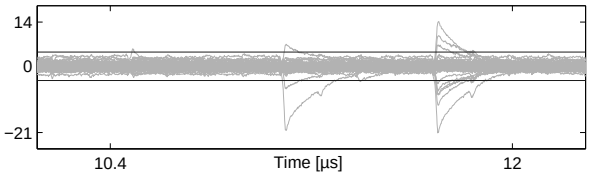
(d) *Profile 3*: Register Precharge (1mio)



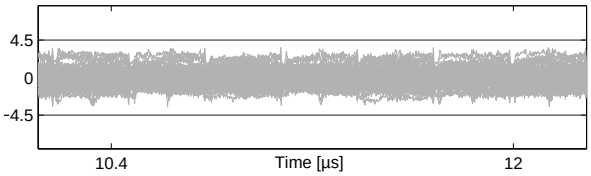
(e) *Profile 4*: Decomposition and Precharge (1mio)



(f) *Profile 5*: Masking and Precharge (1mio)

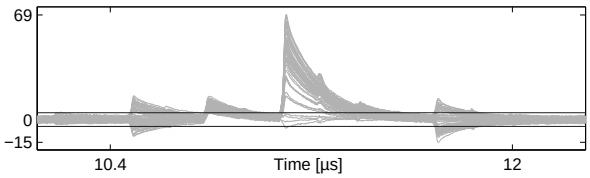


(g) *Profile 6*: Decomposition and Masking (1mio)

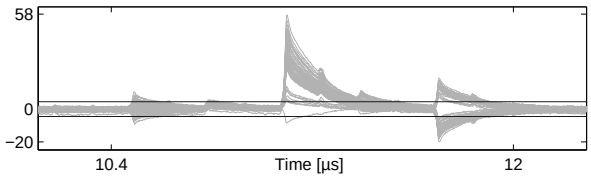


(h) *Profile 7*: Decomposition, Masking and Precharge

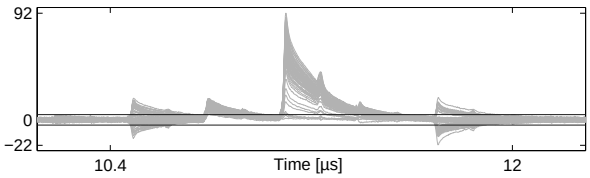
Fig. 5: Group 1 - S-box output bits (64 models)



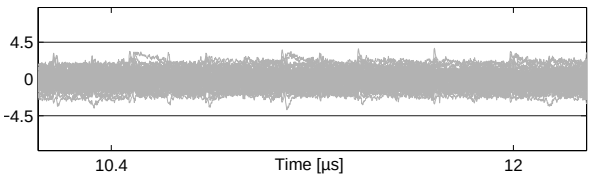
(a) *Profile 0*: No Countermeasure (1mio)



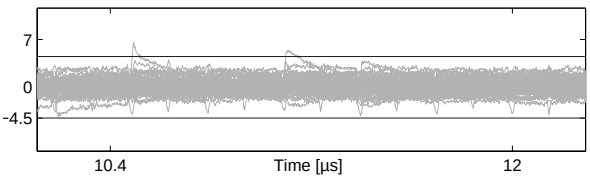
(b) *Profile 1*: S-box Decomposition (1mio)



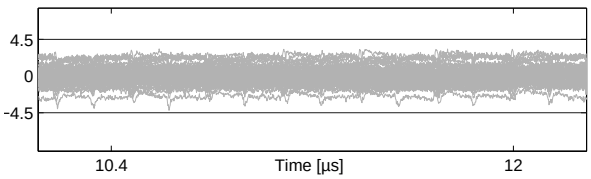
(c) *Profile 2*: Boolean Masking (1mio)



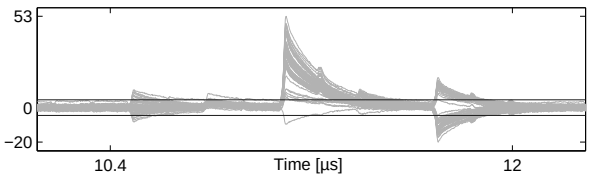
(d) *Profile 3*: Register Precharge (1mio)



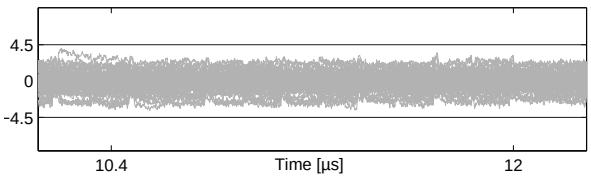
(e) *Profile 4*: Decomposition and Precharge (1mio)



(f) *Profile 5*: Masking and Precharge (1mio)

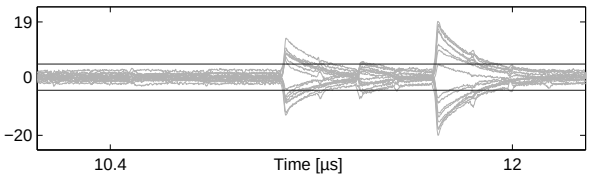


(g) *Profile 6*: Decomposition and Masking (1mio)

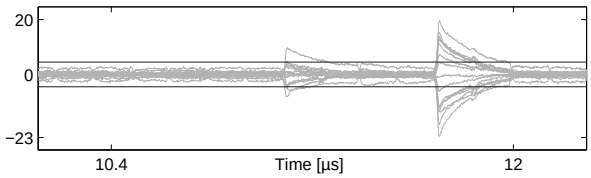


(h) *Profile 7*: Decomposition, Masking and Precharge (10mio)

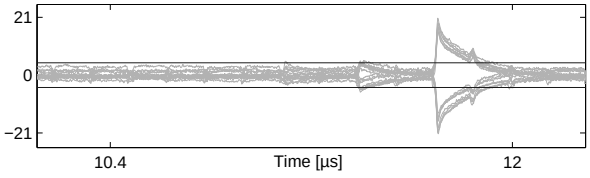
Fig. 6: Group 2 - XOR of round in and round out (64 models)



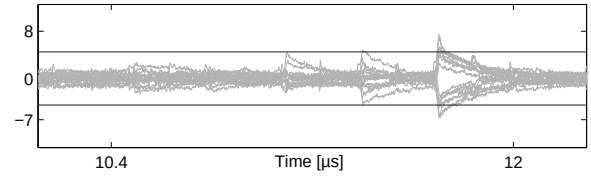
(a) *Profile 0*: No Countermeasure (1mio)



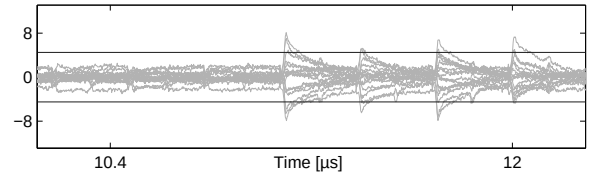
(b) *Profile 1*: S-box Decomposition (1mio)



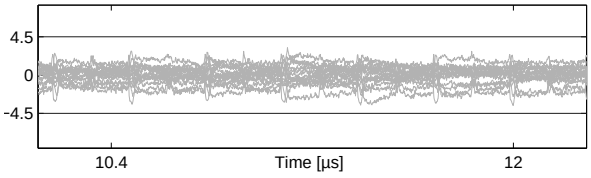
(c) *Profile 2*: Boolean Masking (1mio)



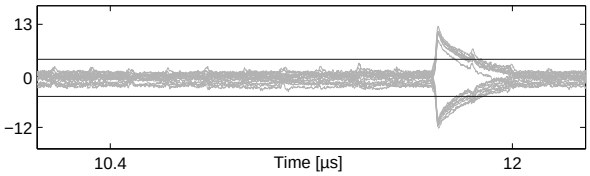
(d) *Profile 3*: Register Precharge (1mio)



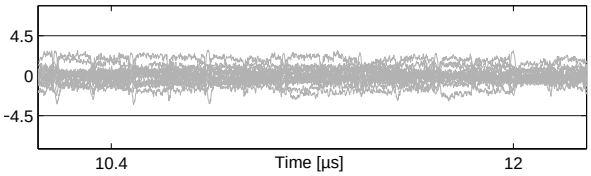
(e) *Profile 4*: Decomposition and Precharge (1mio)



(f) *Profile 5*: Masking and Precharge (1mio)

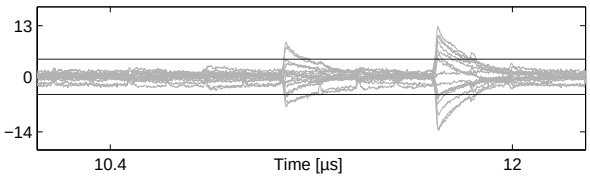


(g) *Profile 6*: Decomposition and Masking (1mio)

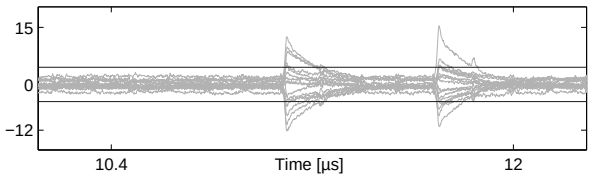


(h) *Profile 7*: Decomposition, Masking and Precharge (10mio)

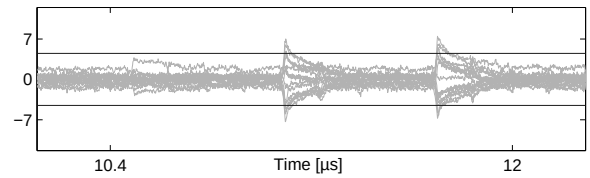
Fig. 7: Group 3 - Output value of S-box S_0 (16 models)



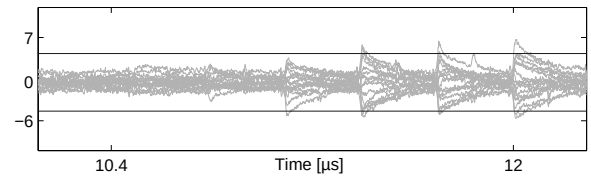
(a) *Profile 0*: No Countermeasure (1mio)



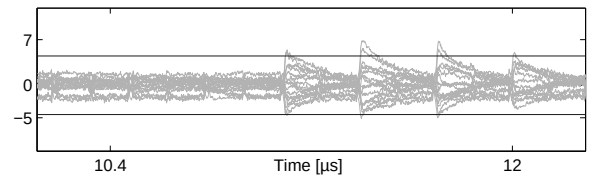
(b) *Profile 1*: S-box Decomposition (1mio)



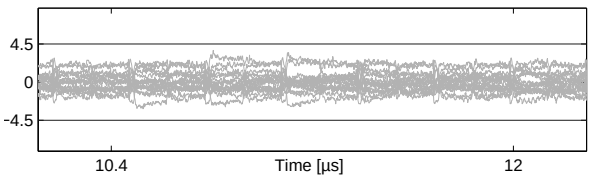
(c) *Profile 2*: Boolean Masking (1mio)



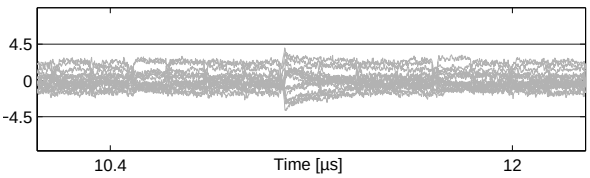
(d) *Profile 3*: Register Precharge (1mio)



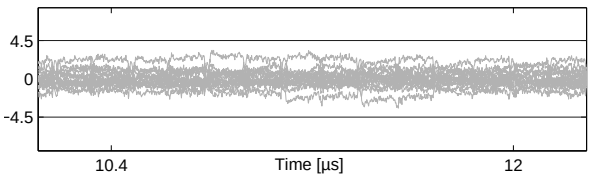
(e) *Profile 4*: Decomposition and Precharge (1mio)



(f) *Profile 5*: Masking and Precharge (1mio)



(g) *Profile 6*: Decomposition and Masking (1mio)



(h) *Profile 7*: Decomposition, Masking and Precharge (10mio)

Fig. 8: Group 4 - Output Value of S-box S_1 (16 Models)