Dummy Rounds as a DPA countermeasure in hardware

Stanislav Jeřábek, Jan Schmidt, Martin Novotný, Vojtěch Miškovský
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
Email: {jerabst1, schmidt, novotnym, miskovoj}@fit.cvut.cz

Abstract—**This paper describes the technique of Dummy Rounds as a countermeasure against DPA in hardware implementation of round-based ciphers. Its principle is inspired by several well-known countermeasures used in hardware as Hiding and Dynamic Reconfiguration as well as countermeasures used in software implementations as Dummy cycles, Random order execution or Hiding in time. Being inspired by countermeasures based on dynamic reconfiguration, this method combines hiding of power consumption with hiding in time. In this work we also discuss the amount of randomness available for the control of the computation.**

Index Terms—**dynamic reconfiguration, hiding, FPGA, DPA, hiding in time, dummy rounds.**

1. Introduction

Dependability requirements are still increasing. Digital systems have to be more and more attack-resistant because of their usage in critical systems. Attackers have many possible methods to disable whole devices or steal some secret data. Therefore one of possible critical threats, which has to be prevented in modern digital systems, is secret data leakage. A method to obtain such data from a secured device is called an *attack*. One of classical *side channel attacks* is based on monitoring power consumption [1] [2] or electromagnetic radiation [3].

The attacked ciphers are commonly iterative. The most common classes of iterative ciphers are Feistel Networks [4] such as DES [5] or, more recently, Substitution-Permutation Networks [6] such as AES [7] or PRESENT [8]. The iterations are called *rounds* and each of them performs similar computations. The similarity greatly simplifies implementation, yet the iterations can be recognized and some distinctive time points in them can be set as targets for cryptanalysis. One possible countermeasure is to *hide* them from an attacker.

The intention of this work is to use additional rounds and randomization as a method of power consumption hiding, preventing attacks by Differential Power Analysis (DPA) [1] [2]. The technique of *Dummy Rounds* appeared before in software implementation [9]. The Dummy Rounds method is similar to some other software countermeasures, such as Dummy Cycles [10], Random Order Execution [11], or Shuffling [12]. Dummy Rounds were studied, in conjunction with other methods, as a countermeasure against fault and combined attacks [9] [13] [14]. As some of these applications were shown to be flawed [15] [16], we limited our study to mere DPA. Usage of Dummy Rounds has been also proposed as a DPA countermeasure [17]. However, the principle is still the same as in the other software implementations – insertion of some round function instructions.

The Dummy Rounds method, as we propose, combines software hiding in time with common hardware hiding of the circuitry power consumption. There are more parts of hardware design which are executed but their outputs are randomly used or not used for computation in every single clock cycle. So, the structure of the design is the same for every clock cycle and power consumption stays the same. Such a behavior can be seen as a kind of dynamic reconfiguration, used also in other methods [18] [19] [20].

While the output of the computation in a single clock cycle changes randomly, the final result stays correct due to round scheduling. Hence, the decision which round to use, although randomized, must follow an algorithm, which we discuss later.

There is detailed description of the Dummy Rounds method in Section II. It is followed with case study on the PRESENT cipher [8] on FPGA and measured results in Section III. Proposed Dummy Rounds modifications, based on the results, are described as future work in Section IV.

2. Dummy Rounds as a DPA countermeasure

2.1. Architecture and operation

Let us assume a round-based cipher with C rounds. Also, let us assume that we can design a hardware implementation of a round so that at least m and no more than M rounds can be executed in a single clock cycle. Then the Dummy Rounds method can be applied as in Fig. 1, where $m = 1$ and $M = 3$. Using a fourth input to the multiplexer, $m = 0$ can be implemented. The round control determines which successive result to use. The unused round results will also cause switching activity in each clock cycle, but their results will not be stored in the result register. Constant switching activity is also the principle of hardware countermeasure called hiding [21] [22] [23]. This method is feasible for both cipher structures, Feistel Networks [4] and Substitution-Permutation Networks [6].

There are two important design parameters in the Dummy Rounds application. The maximum number of rounds per clock cycle M determines clock frequency and influences both time and area overhead. The average number of actually used rounds determines the (constant) number of clock cycles needed for execution, and hence influences time overhead.

The constant number of clock cycles parameter avoids possible information leakage caused by extreme random values. Without the parameter, there would be a very small (but still higher than zero) probability, that the design will compute only one round (or other value of m parameter) in each clock cycle. In that case, the design could be attacked nearly as a design without any countermeasure. The only difference is the power consumption of the next implemented rounds. However, if there are assumptions of the first round values, the additional rounds can be predicted. The case of encryption using maximum possible count of clock cycles with M rounds computed is quite similar. With this parameter and a corresponding controller schedule, such a situation cannot occur.

Let us illustrate the architectural parameters on an implementation of the PRESENT cipher. The cipher has 31 rounds and one extra sub-key, which is considered to be another round, so there are total $C = 32$ rounds. Let us assume the original architecture has one round per clock cycle, which is common. Let us further assume we decided to implement $M = 3$ PRESENT rounds per clock cycle, which is a practical choice in most circumstances. With this hardware architecture, we need $N = 16$ clock cycles with 2 actually used rounds per clock cycle on average. The clock period will be approximately three times longer, but 16 clock cycles instead of 32 will be needed for an encryption. Therefore, the time overhead will be approximately 50%. The round logic dominates the design, so that the upper bound on area overhead is 200%.

2.2. Rounds control

The rounds controller has two tasks. The first one is to assure that the correct number of rounds are executed within the designed clock count. The other one is to prevent uniform computations to occur. In our case, the control is implemented in hardware and should be as simple as possible.

For the first task, the controller has to monitor the number of clock cycles executed and the number of used rounds. Let c_n be the number of rounds accepted up to the step n, $n \leq N$. Then, obviously,

$$c_n \leq Mn \tag{1}$$

$$c_n \geq mn \tag{2}$$

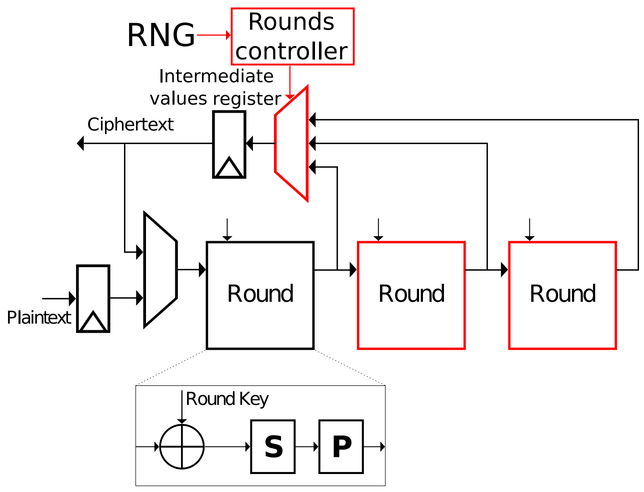

Figure 1. Dummy cycles countermeasure scheme.

To be able to reach precisely C rounds at step N, the following must hold

$$c_n + m(N - n) \leq C \tag{3}$$

$$c_n + M(N - n) \geq C \tag{4}$$

For an example of the space delimited by these inequalities, refer to Figures 2 and 3. Notice how a small change in one parameter ($m = 0$ versus $m = 1$) can cause a large change in the controller state space.

When a controller decides at step n to perform s_n rounds in the next clock cycle, for the resulting number c_{n+1} of accepted rounds, Inequalities 3 and 4 must also hold, so that

$$s_n \leq C - m(N - n - 1) - c_n \tag{5}$$

$$s_n \geq C - M(N - n - 1) - c_n \tag{6}$$

These are minimal correctness ensuring requirements. A simple controller may not utilize the entire space delimited by Inequalities 1 thru 4. A more sophisticated controller can react before Inequalities 3 and 4 apply and only modify the probabilities of future round counts to ensure better randomness of the process.

To continue the PRESENT example, $M = 3$ rounds are computed at each clock cycle. The output from a randomly chosen round (1 to 3) is stored in the output register. The round controller keeps the count of already evaluated rounds and clock cycles, so that for every encryption/decryption, the total clock cycles is 16 (in average, 2 rounds per clock cycle are evaluated). The controller must ensure that the number c_n of rounds accepted up to the step n remains in the permissible points in Figure 2.

With the architectural parameters chosen for the example (3 rounds per clock cycle, 2 rounds on average used in every of 16 clock cycles), there are 5 196 627 ways to evaluate 32 rounds total in 16 clock cycles, while constraint of at least one and at most three rounds is respected. The number of possible combinations has been counted empirically, as it is the count of possible sequences of 16 integers from one to three, where their sum is 32. However, a DPA attack usually targets the first or the last cycle (round) so not all number sequences are different from the attacker's point of view.

Our method shares the need of an RNG with other methods using randomization, e.g. [19]. It does not have any exceptional requirements on the generator. While an RNG itself may be vulnerable to attacks, we have an opportunity to use True Random Number Generator because of hardware implementation.

3. Implementation and Results

As the case study for experimental evaluation, we have chosen the PRESENT cipher with 3 rounds implemented in hardware. The number of clock cycles for execution was set to 16, as in the running example. A 64-bit linear feedback shift register (LFSR) using the generator polynomial $g(x) = x^{64} + x^{63} + x^{61} + x^{60} + 1$ was used as the random number generator. It is seeded with all the bits set to one and counts every clock cycle of the design.

The state of the controller consists of the actual clock cycle number and the number of rounds accepted so far. In each state, the controller checks the uniform random numbers obtained from the generator against Inequalities 5 and 6. Values outside the specified range are replaced by the nearest feasible numbers. The state space is in Figure 2. Notice that there are less states with restricted decision than in Figure 3.

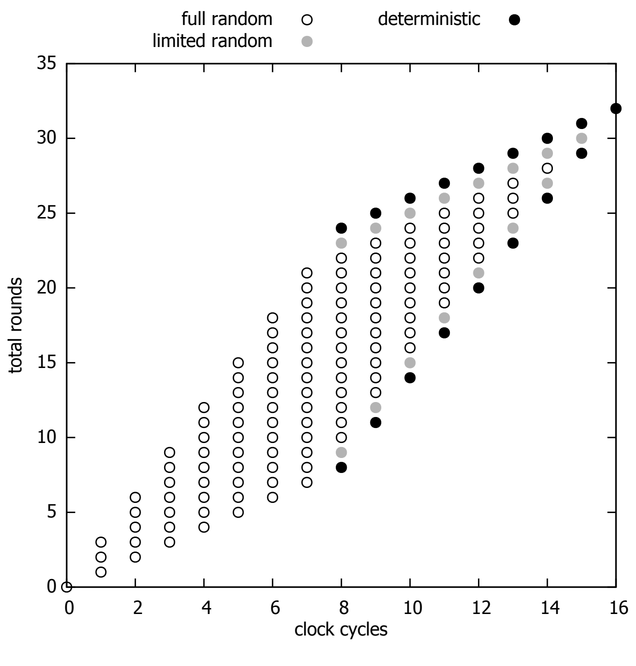

Figure 2. The state space of the rounds controller for $m = 1$ and $M = 3$, with decision types in individual states shown.

The design described above has been evaluated as an FPGA implementation on the SAKURA-G board [24]. We have measured 100 000 power traces and evaluated the traces by first-order univariate non-specific Welch's t-test, as proposed in [25]. T-values of the power traces are shown in Fig. 4, where raising edges of the clock signal are highlighted on the x-axis. T-value says, how much information is leaking from the device power consumption. In our case, the maximum t-value is 346, while usually a threshold 4.5 is defined to regard the design as secured.

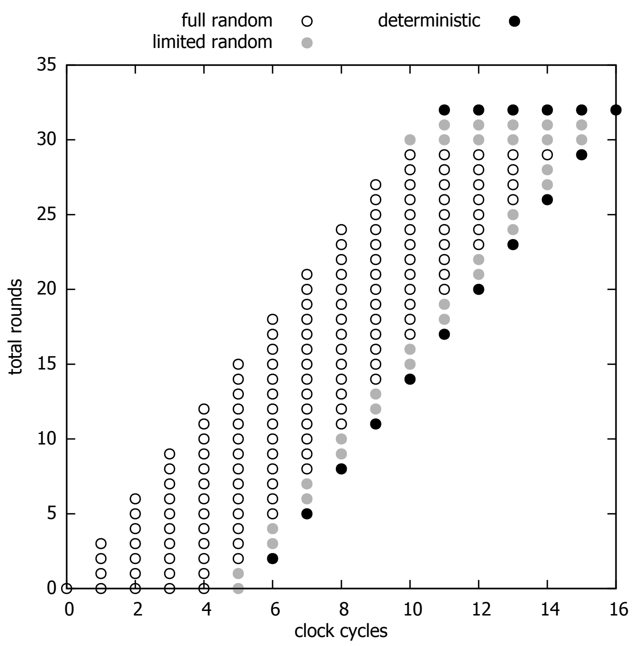

Figure 3. The state space of a modified controller for $m = 0$ and $M = 3$.

The biggest leakage is of course at the beginning of the encryption. There is probability exactly $1/3$, that the intermediate values register contains the output of the first round at the end of the first clock cycle. This intermediate value will have to be hidden using some other countermeasures or at least making the start position of the encryption random.

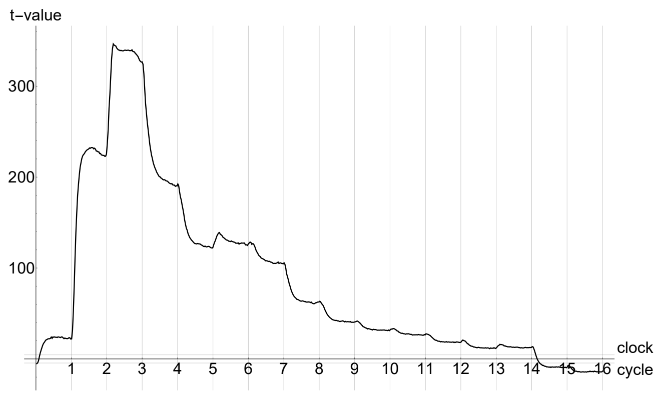

Figure 4. T-values of PRESENT cipher with initial Dummy Rounds countermeasure.

We have also evaluated how the design behavior depends on the counts of computed rounds in individual clock cycles.

Besides the initial version driven with 64-bit LFSR, we have implemented further versions for that purpose:

1) Computing exactly two rounds in every clock cycle.
2) Starting with eight clock cycles having one round per cycle followed by eight clock cycles having three rounds per cycle.
3) Starting with eight clock cycles having three rounds per cycle followed by eight clock cycles having one round per cycle.
4) Alternating one and three rounds per cycle, starting with one round in the first clock cycle.
5) Alternating one and three rounds per cycle, starting with three rounds in the first clock cycle.

The version number 3 (maximal t-value 804), while the version number 1 gives the best results (maximal t-value 267). The t-values diagrams for versions 3 and 1 can be seen in Fig. 5, respectively Fig. 6, respectively. We will further study the effects of *random* values driving the design.

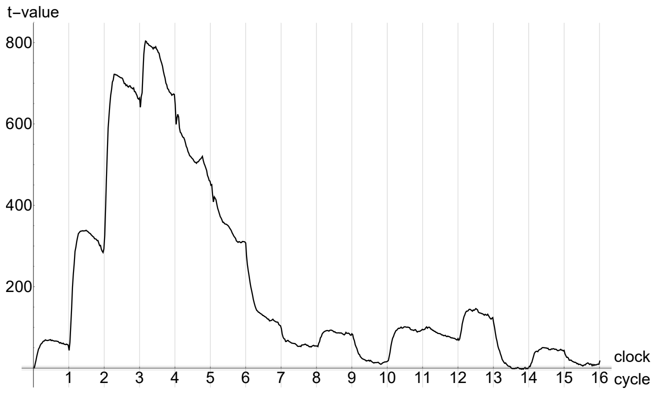

Figure 5. T-values of PRESENT cipher with Dummy Rounds countermeasure starting with 8 clock cycles having three rounds per cycle followed by 8 clock cycles having one round per cycle.

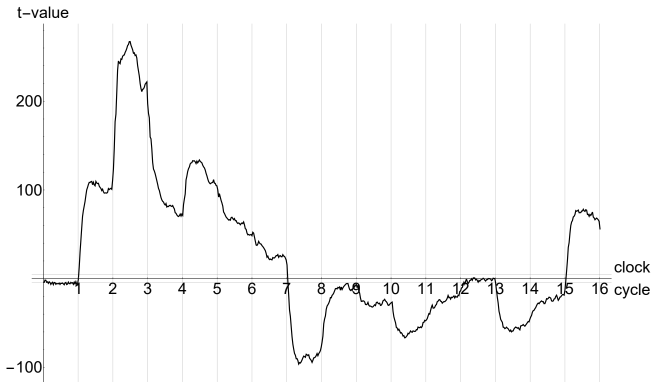

Figure 6. T-values of PRESENT cipher with Dummy Rounds countermeasure, where two rounds in each cycle are computed.

4. Future work

In general, the results with Dummy Rounds as a DPA countermeasure are unsatisfactory right now. We are going to study influence of all the random values driving setups. This section summarizes possible paths.

4.1. Clock cycle count

We have implemented the cipher with the same encryption length in every case, so every encryption or decryption from the attacker's point of view takes 16 clock cycles. This decision was made due to the possible information leakage caused by random values leading to extremely short or long encryption described in subsection 2.1.

However, the architecture parameters can have adverse effect on randomness. In our example, when we compute 32 rounds in 16 clock cycles, we know that the number of clock cycles with three rounds is equal to the number of clock cycles with one round. Moreover, we know that exactly two rounds are computed in each of remaining clock cycles.

4.2. Dummy computation

We could also use more clock cycles for the whole encryption process to hide the real encryption, which could possibly take variable clock cycles count. For example, with standard implementation of PRESENT cipher, every encryption takes 32 cycles. We can use some of 16 spared clock cycles in comparison with initial Dummy Rounds because of faster encryption for some dummy encryption. DPA attacks are generally targeting the first or the last round of the cipher, so it is in practice more difficult to successfully attack a design with some randomly long random values encryption added before the start and after the end of the "real" data encryption. Therefore, we can generate some random value to start the encryption with, load the plaintext into the intermediate register at some moment, perform the encryption itself (taking possibly variable clock cycles count), store the result and then load another random value into the register for another dummy computation at the end. The required data paths are shown in Fig. 7. We just have to achieve a constant length of encryption from the attacker's point of view, which of course includes pre-encryption and post-encryption dummy computations.

Dummy encryption extension (shown in Fig. 7) is also necessary requirement when $m = 0$. Without that, it would be trivial to recognize clock cycle, where no rounds were computed. There would be extremely low power consumption because of the same rounds input during two consecutive clock cycles.

4.3. Rounds controller

Another thing to consider is behavior of the rounds controller. In the first unsophisticated solution, we implemented the necessary conditions only, that is, the controller modifies the random values only when necessary, as shown in Figs 2 and 3. We could achieve more random distribution, if we generate all the round counts computed in individual clock cycles (how many times there will be one round per cycle

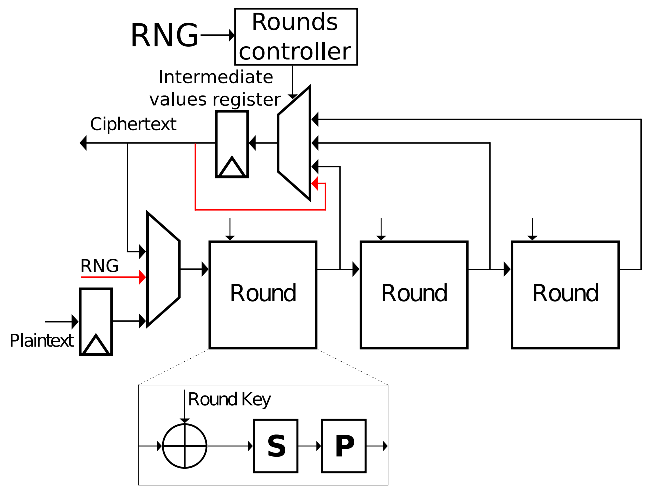

Figure 7. Dummy cycles countermeasure scheme with dummy encryption extension.

computed etc.) at first and permute them randomly before the start of encryption itself. Memory requirements are, of course, a concern.

There are some threats for the principle of Dummy Rounds itself. For example, with localized EM attacks [26] the attacker could focus on only the first round and get much more data leakage. Therefore, a part of the future work is also combining the Dummy Rounds method with other countermeasures described in [19] and an evaluation of their combinations.

5. Conclusion

The Dummy round countermeasure described in this paper is easily applicable to any round based cipher. The designer just needs to copy the round block and add a multiplexor driven by RNG-based controller. It can be combined with other countermeasures. This method strictly depends on security of the random number generator, similarly to other commonly used countermeasures.

Results of t-test statistical evaluation of the Dummy Rounds method in initial version are unsatisfactory yet. The maximum of t-values is 346, while usually a threshold 4.5 is defined to regard the design as secured. However, some of other experiments proposed possible ways how to improve the results by proper random values usage decreasing maximum t-value to 267 for now. Therefore, it is important to implement modifications discussed in Chapter 4 of this paper to estimate the real potential of the method.

Acknowledgment

We thank Prof. Dr.-Ing. Tim Güneysu for helpful comments.

This work was partially funded by the CELSA project “DRASTIC: Dynamically Reconfigurable Architectures for Side-channel analysis protection of Cryptographic implementations” (CELSA/17/033), the grant GA16-05179S of the Czech Grant Agency, “Fault-Tolerant and Attack-Resistant Architectures Based on programmable Devices: Research of Interplay and Common Features” (2016-2018) and CTU project SGS17/213/OHK3/3T/18.

References

[1] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[2] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, “Power analysis, what is now possible...” in *Advances in Cryptology — ASIACRYPT 2000*, T. Okamoto, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 489–502.

[3] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, “The em side—channel(s),” in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski, ç. K. Koç, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 29–45.

[4] H. Feistel, “Cryptography and computer privacy,” *Scientific american*, vol. 228, no. 5, pp. 15–23, 1973.

[5] D. E. Standard *et al.*, “Federal information processing standards publication 46,” *National Bureau of Standards, US Department of Commerce*, vol. 4, 1977.

[6] C. E. Shannon, “Communication theory of secrecy systems,” *The Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, Oct 1949.

[7] J. Daemen and V. Rijmen, “Aes proposal: Rijndael,” 1999.

[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*, P. Paillier and I. Verbauwhede, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 450–466.

[9] B. Gierlichs, J.-M. Schmidt, and M. Tunstall, “Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output,” in *Progress in Cryptology – LATINCRYPT 2012*, A. Hevia and G. Neven, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 305–321.

[10] C. Clavier, J.-S. Coron, and N. Dabbous, “Differential power analysis in the presence of hardware countermeasures,” in *Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 252–263.

[11] S. Tillich, C. Herbst, and S. Mangard, “Protecting aes software implementations on 32-bit processors against power analysis,” in *Applied Cryptography and Network Security*, J. Katz and M. Yung, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 141–157.

[12] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert, “Shuffling against side-channel attacks: A comprehensive study with cautionary note,” in *Advances in Cryptology – ASIACRYPT 2012*, X. Wang and K. Sako, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 740–757.

[13] S. Patranabis, A. Chakraborty, and D. Mukhopadhyay, “Fault tolerant infective countermeasure for aes,” in *Security, Privacy, and Applied Cryptography Engineering*, R. S. Chakraborty, P. Schwabe, and J. Solworth, Eds. Cham: Springer International Publishing, 2015, pp. 190–209.

[14] S. Patranabis and D. Mukhopadhyay, *Infective Countermeasures Against Fault Analysis*. Singapore: Springer Singapore, 2018, pp. 197–211. [Online]. Available: https://doi.org/10.1007/978-981-10-1387-4_10

[15] A. Battistello and C. Giraud, “Fault analysis of infective aes computations,” in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Aug 2013, pp. 101–107.

[16] H. Tupsamudre, S. Bisht, and D. Mukhopadhyay, “Destroying fault invariant with randomization,” in *Cryptographic Hardware and Embedded Systems – CHES 2014*, L. Batina and M. Robshaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 93–111.

[17] C. Herbst, E. Oswald, and S. Mangard, “An aes smart card implementation resistant to power analysis attacks,” in *Applied Cryptography and Network Security*, J. Zhou, M. Yung, and F. Bao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 239–252.

[18] 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*, E. Oswald and P. Rohatgi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 346–362.

[19] P. Sasdrich, A. Moradi, O. Mischke, and T. Güneysu, “Achieving side-channel protection with dynamic logic reconfiguration on modern fpgas,” in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2015, pp. 130–136.

[20] P. Sasdrich, A. Moradi, and T. Güneysu, “Hiding higher-order side-channel leakage,” in *Topics in Cryptology – CT-RSA 2017*, H. Handschuh, Ed. Cham: Springer International Publishing, 2017, pp. 131–146.

[21] J. L. Danger, S. Guilley, S. Bhasin, and M. Nassar, “Overview of dual rail with precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors,” in *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*, Nov 2009, pp. 1–8.

[22] D. Suzuki and M. Saeki, “Security evaluation of dpa countermeasures using dual-rail pre-charge logic style,” in *Cryptographic Hardware and Embedded Systems - CHES 2006*, L. Goubin and M. Matsui, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 255–269.

[23] C. Tu, J. Zhou, N. Gao, Z. Liu, Y. Ma, and Z. Liu, “Qrl: A high performance quadruple-rail logic for resisting dpa on fpga implementations,” in *Information and Communications Security*, S. Qing, E. Okamoto, K. Kim, and D. Liu, Eds. Cham: Springer International Publishing, 2016, pp. 184–198.

[24] H. Guntur, J. Ishii, and A. Satoh, “Side-channel attack user reference architecture board sakura-g,” in *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, Oct 2014, pp. 271–274.

[25] T. Schneider and A. Moradi, “Leakage assessment methodology,” *Journal of Cryptographic Engineering*, vol. 6, no. 2, pp. 85–99, Jun 2016. [Online]. Available: https://doi.org/10.1007/s13389-016-0120-y

[26] F. Unterstein, J. Heyszl, F. D. Santis, and R. Specht, “Dissecting leakage resilient prfs with multivariate localized em attacks - a practical security evaluation on fpga,” *IACR Cryptology ePrint Archive*, vol. 2017, p. 272, 2017.