# Countermeasures against EM Analysis for a Secured FPGA-based AES Implementation

P. Maistri[1], S. Tiran[2], P. Maurine[2], I. Koren[3], R. Leveugle[1]

[1] Univ. Grenoble Alpes, TIMA Laboratory, F-38031 Grenoble
CNRS, TIMA Laboratory, F-38031 Grenoble
{Paolo.Maistri,Regis.Leveugle}@imag.fr

[2] LIRMM (UM2, CNRS) Montpellier, France
{Sebastien.Tiran,Philippe.Maurine}@lirmm.fr

[3] ECE Department University of Massachusetts Amherst, MA, USA
koren@ecs.umass.edu

***Abstract*—Side-channel analysis is one of the most efficient techniques available to an attacker to break the security of a cryptographic device. Started as monitoring of computation time or power, it has evolved into considering several other possible information leakage sources, such as electromagnetic (EM) emissions. EM waves can be a very attractive means to attack a cryptographic implementation: they are contactless, and their intrinsic spatial, temporal, and frequency information can be a source of leakage richer than power consumption. Existing countermeasures may be thus insufficient against an EM attack and new solutions must be found and validated. In this paper, we describe a set of dedicated countermeasures protecting against EM analysis and validate them with real experimental campaigns on a Xilinx FPGA.**

***Keywords*—AES, Advanced Encryption Standard, side-channel analysis, EM, CEMA, CPA, countermeasures, FPGA**

## I. INTRODUCTION

The current trend in consumer products is toward an increasing need of secure protocols and algorithms, which can be often implemented in hardware on reconfigurable platforms, due to their advantageous benefit/cost ratio for low volumes. The Advanced Encryption Standard (AES) [1] is the de facto standard and many FPGA-based implementations have been proposed. In addition, a large effort has been dedicated to analyzing its robustness. In particular, its security can be seriously undermined by implementation attacks. Such side-channel attacks, in fact, are the most efficient way (considering the required resources) for an attacker to extract the secret information. Using Differential Power Analysis (DPA) [2], for instance, the attacker needs a few thousands traces of the power consumption of the device to guess the secret key. Several other attacks have been proposed, either based on more advanced analysis of the collected data [3], or on different channels leaking information, such as Electro-Magnetic (EM) emissions [4]. EM emissions have recently started to be considered as a serious means of attack: they are contactless, thus monitoring them is easier and less detectable than power monitoring; they are a very rich source of information, in the spatial, temporal, and frequency domains; and the required knowledge to mount these attacks is becoming more common.

This work is partly supported by the French Ministry of Research, through the ANR project EMAISECI (act ANR-2010-SEGI-012-03). TIMA is Partner of the Labex PERSYVAL Lab (ANR-11-LABX-0025).

978-1-4799-2079-2/13/$31.00 ©2013 IEEE

Countermeasures aiming at making the physical attacks more difficult are necessary. In general, the idea is to make the correlation between the data and the measured activity (the leaked side-channel information) less evident. One solution is adding some noise: unnecessary computations [5] and random jitter [6] make the statistical post-processing more complex and time consuming. Another approach attempts to hide the data: a random mask applied to the data being computed makes the analysis harder. On the other hand, removing the mask can be non-trivial, and higher-order attacks [7] can still be able to recover the key, at a higher cost. Dual-rail logic [8] encodes each value with a pair of complementary signals, ensuring that all the state transitions are balanced, but it is difficult to design circuits that are accurately balanced [9] so that the power consumption is data independent.

Another issue that requires further study is the quality and quantity of the information coming from the leaking channel as one side-channel may reveal more information than another. For example, power analysis (that has been the classical source of information leakage) can provide only a global view of what is going on in the circuit while EM emissions also embed some spatial information. Existing countermeasures against power analysis may thus be insufficient and further protection would be required.

In this paper we propose several countermeasures against EM attacks for an FPGA implementation of AES. Although these are originally designed as means to protect against EM analysis, they may work against power analysis as well. The main contribution of this paper is a dynamic data relocation scheme working at two levels, one within and one across the AES round computations. This countermeasure is strongly based on a spatial approach, which explains why EM analysis has been chosen as a mean of attack; power analysis is not directly addressed in this paper. Masking schemes are also employed, in order to improve the robustness of the design and illustrate the compatibility of the proposed solution with established protection schemes already known in the literature.

The paper is organized as follows. The next section describes briefly the AES algorithm and the hardware implementation used to illustrate our solution. Section III presents the countermeasures adopted to protect against side-channel analysis, with a brief discussion of the associated costs. Section IV describes the experimental attack campaigns: the

setup, the results, and a brief discussion highlighting the most important observations. Section V concludes the paper.

## II. Architecture Implementation

### A. The Advanced Encryption Standard (AES) Algorithm

AES [1] is a round-iterated cipher which accepts 128-bit input blocks and encrypts them with 128-, 192- or 256-bit keys. The input block is stored into a temporary register, which is organized as a 4-by-4 byte matrix for ease of processing. The rounds are preceded by an initial mixing addition with the secret key; all consecutive iterations are identical and consist of the following operations:

- *SubBytes*: a nonlinear byte substitution.
- *ShiftRows*: the rows of the internal state are rotated by an offset depending on the row index.
- *MixColumns*: each column is processed through polynomial multiplication in the binary field $GF(2^8)^4$.
- *AddRoundKey*: a round-dependent key is added to the internal state using modulo-2 addition.

The last iteration differs from the others, since it lacks the *MixColumns* operation.

### B. Structure of a Round Instance

The basic building block of the implemented architecture is the round. This is shown in Figure 1 and is derived from [10]. It is a 32-bit-wide design, with only four substitution boxes and four scaling units in a single *MixColumns* element. The central block contains the state register, the routing logic used to switch between the inputs, and the key addition logic. A dedicated block is in charge of implementing the *ShiftRows* operation (slightly modified with respect to the original specifications, as described in Section III.A).

The round 4-word state data is loaded column-wise, from the most to the least significant word. If needed, the corresponding round key is added to the input while loading, but this operation is inactive when just transferring data. Then, the data is moved row-wise, in order to implement *ShiftRows* and *SubBytes*, until all the rows have been processed. Due to pipelining, this step takes 6 clock cycles. Then, data is moved again column-wise to compute the *MixColumns* operation and finally sent to the output port. This final step takes 4 clock cycles (one clock cycle per column), thus a whole round is completed in 10 clock cycles. It can be seen that the goal of this design is lower hardware complexity rather than performance.

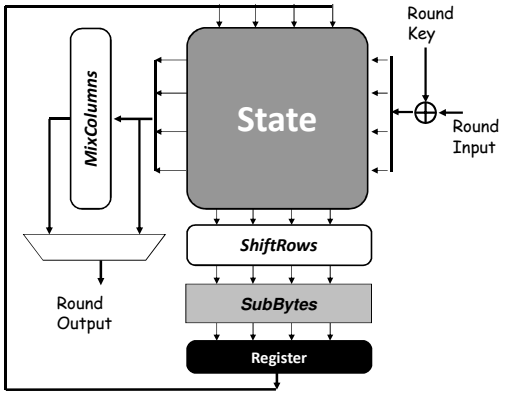


Fig. 1. Implementation of an AES round.

### C. Hardware Implementation

To increase the circuit's global throughput, several round instances are implemented. However, these blocks are not serially pipelined as it is usually done when implementing unrolled ciphers. Thus, there is no first-last relationship among the instances. On the contrary, all instances are hierarchically equal and are connected to a common bus (see Figure 2).

The width of the bus lanes is a parameter that can be chosen by the designer: the wider the lane, the faster the data transfer, but also the higher the cost. In our implementation, we choose 32-bit lanes to be consistent with the round width and be therefore able to transfer one column at a time; larger or narrower widths might be chosen as well.

The number of lanes depends on the number of round instances in the circuit. Each bus lane acts as the input source for a specific round instance, while any instance (together with the input source) can write on any lane. The description of write operations is detailed in Section III.B.

## III. Specific Countermeasures

### A. Dynamic Column Relocation (DCR)

To implement a dynamic relocation of the columns, the *ShiftRows* has to operate in a slightly unconventional way.

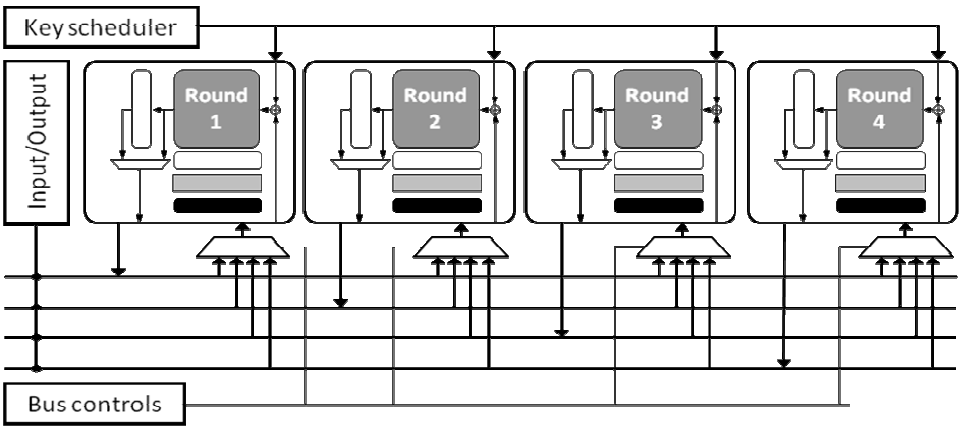


Fig. 2. Global architecture of the analyzed AES implementation (The Pseudo-Random Number Generator is not shown).

Rows are still rotated by an amount proportional to their index; however, the first row is no longer untouched, but it is instead, rotated by a random amount (from 0 to 3). Further rows are rotated accordingly, in order to preserve consistency within the state. Hence, four different configurations are possible after shuffling, as shown in Figure 3. This is done to scramble the state a little bit more than mandated by the AES specifications. The number of possible configurations is limited, due to the constraints imposed by the AES round functions. The *MixColumns* operation, for instance, imposes consistency at the column level. Moreover, columns cannot be shuffled arbitrarily since the row rotations are not independent of each other. These constraints lead to a limited number of four different possibilities. The columns are correctly relocated at the end in order to give a result conforming to the standard.

This technique has already been applied at the software level through register shuffling or renaming [11], but the only other hardware technique similar to this one is the DES Jamming technique [12], where the state register is rotated by a random number of positions (from 0 to 7 steps) in order to achieve resistance against DPA and DEMA (Differential ElectroMagnetic Analysis). In this context, our column relocation has the following characteristics:

1. The number of different configuration is lower, 4 versus 8, due to the constraints imposed by the AES algorithm. This results in a less effective countermeasure due to the limited amount of randomness introduced during the execution;
2. There is no need to extend the functional block with additional resources. In DES, the round operations depend on the position of the data within the state register, while in AES this occurs only at the row level (the rotation by *ShiftRows*, the coefficient of *MixColumns*). As a result, the additional overhead introduced by this countermeasure is small as the relocation scheme works at the column level.

### *B. Dynamic Block Relocation (DBR)*

In order to increase the number of spatial configurations of the encrypted text, data is also moved randomly at each round between the different round instances. This means that after the *MixColumns* has been computed, data is not rerouted within the same round circuit instance, but rather sent on the external bus to be transferred to another instance. The destination is chosen randomly, thus at each round the configuration may change.

Loading is done statically, i.e., for each encryption, the data is initially stored always into the same register; when encrypting several blocks of plain text, the existing data is shifted to the state register of another round instance. Static loading helps simplify the design and avoids conflicts, such as a new text block overwriting data already stored. After the data is loaded, encryption may start with the initial key addition. Random relocation starts at this point with data sent on the bus to a random destination, where the specific round key word will be added during transfer. Then, regular round computation starts (*ShiftRows*, *SubBytes*, and *MixColumns*). Finally, the data is sent again on the bus to a new destination, along with the round key addition, and a new round. For the final round, *MixColumns* is skipped (as during the loading phase) and data is reordered after the final key addition. *DBR* intervenes thus as soon as the encryption actually starts.

Writing on the bus lanes is managed by a dedicated bus controller that must ensure that all transfers are performed correctly, i.e., no data is lost or overwritten. Tri-state buffers can be used when available; however, our implementation uses a Spartan reconfigurable board [14] that does not provide these, so multiplexers were used instead. The generation of these signals is a nontrivial task: this is equivalent to the generation of a random square matrix, of size equal to the number of lanes, with one and only one enabled control for each row or column. The bus controller may be implemented either as a ROM pre-loaded with all the possible configurations, or as a circuit generating the signals on-the-fly. The choice depends on the available resources and time constraints. In the former case, the size of the table is $n!(n^2)$, which is manageable only for small values of the matrix size n; in the latter, the table can be generated in a number of cycles proportional to $O(n^2)$. On-the-fly generation was chosen for the current design. With four round instances, both solutions are viable, but for larger values of *n,* dynamic generation becomes mandatory. In order to generate a different matrix at each round, the matrix generator

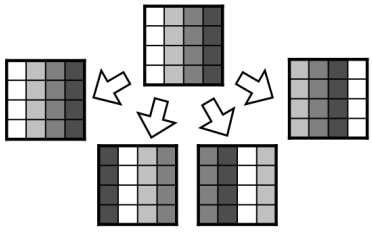

Fig. 3. Dynamic Column Relocation. The columns of the AES state matrix are rotated by a random amount; relative position is preserved.

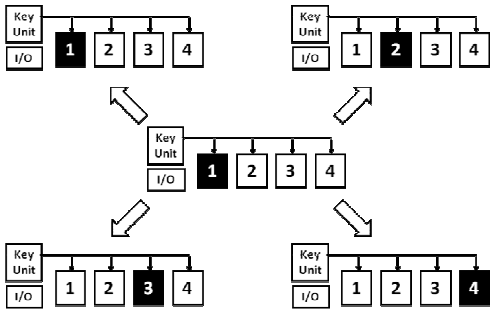


Fig. 4. Dynamic Block Relocation. The output of a round encryption can be sent to any round instance, source included.

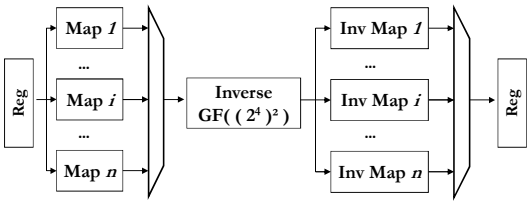


Fig. 5. Dynamic Mapping of a *SubBytes* implementation. Modular inverse is computed in composite field, the destination basis (and thus the to- and from- mappings) is chosen dynamically.

was connected to a faster clock via the Digital Clock Module (DCM) available on the Xilinx boards.

### C. Dynamic Mapping (DynMap)

S-Boxes constitute a large source of information leakage and are often the target of side-channel attacks. An effective protection scheme may be a multiplicative masking, which hides data with a mask that must be suppressed afterwards in order to get the correct result.

In this paper we propose a complementary solution, based on composite fields. These fields are often used to implement small substitution boxes, which can also be easily pipelined to raise the operating frequency. Mapping from the GF($2^8$) field to the composite one (and vice versa) is needed. Many papers deal with composite field implementations of AES S-Boxes, but all focus on the best implementation from the point of view of area or power [15]. In general, the designer can choose from several possible fields and, for each field, from several bases. Usually, several mappings are evaluated and the best one (according to the preferred performance figure) is chosen and implemented statically [16][17]. The approach followed in this paper consists of having several mappings from the regular to the composite field instantiated in the design. Then, at computation time, any of them can be dinamically chosen to compute the *SubBytes* operation (see Figure 5). If a specific composite field (with the respective polynomials) is chosen, then eight different mappings can be computed, depending on the basis selected for the composite field [18]. A similar approach was applied to RSA in [19]: in that case, the countermeasure could benefit from a more suitable context, since RSA architectures are larger and offer more choices for the mappings on GF(p).

The small number of random possibilities provides only a limited additional protection; however, the design can be made more robust by:

- Implementing dynamic mapping onto different composite fields. This adds further complexity to the S-Box implementation, since the constant value used in a scaling operation (which is actually one of the coefficients of the polynomial) must be replaced by a generic multiplication.
- Combining the dynamic mapping with a multiplicative masking scheme, since the schemes are not exclusive.

It should be observed that, while the multiplicative masking scheme suffers from zero-value attacks [20], so is the dynamic mapping. In particular, it must be noted that, regardless of the mapping chosen, the null and the unity elements will be always unaltered even after the mapping. This may provide some opportunities for an attacker, who can choose the inputs carefully to exploit this weakness. For this reason, this countermeasure should not be implemented as the sole solution, but accompanied by some other countermeasures.

### D. Linear Masking (LinMsk)

The dynamic mapping technique only protects the *SubBytes* data path. The permutation layer of the algorithm (*ShiftRows*, *MixColumns*, and *AddRoundKey*) still operates on unmasked data. For this reason, a simple additive masking is used. This solution is quite common in the literature. Complexity is kept low by using the same mask for each row of the matrix; thus, the mask is not affected by *MixColumns*, since the same byte mask is applied to each byte of the same column. Linear masking also helps protect data transfers on the bus; however, in order to allow masking and unmasking among different round instances, the mask value must be transferred on the bus along with the respective column.

### E. Pseudo Random Number Generator (PRNG)

All the presented countermeasures require several random bits: a random value for linear masking, a few random bits to choose the mapping or the relocation amounts, and so on. A simple Pseudo-Random Number Generator was implemented to produce the random stream starting from a couple of initialization values (key and Initialization Vector). We chose Trivium [21], essentially for its small footprint, ease of integration, and high throughput, which were ideal for our application. However, any other (possibly cheaper) solution is possible, provided that the stream is sufficiently random.

### F. Performance and Cost Evaluation

This basic single-round instance takes 1224 LUTs on a Xilinx Spartan3 and is able to run at about 65 MHz. As already mentioned, the focus of the original design is neither performance nor smallest area, but rather a reasonable tradeoff between the two objectives.

The countermeasures implemented have an associated cost as they require additional resources (see Table I). The first and most apparent cost is related to the number of round instances. *DBR* requires that several round instances are implemented: this multiplies the resources used by a certain factor. However, a larger number of implemented rounds also means higher throughput in burst encryption modes (such as Counter modes [22]) and more possible spatial configurations, yielding an increased security. Moreover, a designer may decide to define an upper bound to resource usage and not use all the available blocks for data encryption, but rather leave some instances as "spares" processing different data (to produce additional uncorrelated noise), or as a backup, in order to deal with computation errors. The number of bus lanes also depends linearly on the number of rounds. Moreover, the width of each lane increases due to the proposed countermeasures: in addition to 32-bit data, the bus must also be able to transfer the column index (which may change due to *DCR*), the block index, and the mask associated with each word. This makes the total width of our bus equal to 44 bits (12-bit header + 32-bit payload).

A non-negligible part of the additional costs comes from the countermeasures implemented at the round level, i.e., *DynMap*, *DCR*, and *LinMsk*. The latter is the cheapest one, but it reduces the overall throughput by about 12% due to the need for unmasking before *SubBytes* and re-masking right after the operation. *DCR* introduces an acceptable overhead (+10%), but it also affects the overall throughput, due to the logic needed for shuffling the data and restoring the correct order.

The remaining overhead is mainly due to the additional mappings computed within the S-Boxes (from 42 to 146 slices for each S-Box), the module generating the bus configuration on-the-fly, the logic responsible for generating and managing the random bits, and some glue logic at the top level (registers, communication bus, and so on), which accounts for about 10% of all the resources. Overall, our design uses 5694 slices on a Spartan3 for a 4-round implementation, while the original design needed about 1200 slices on the same board. Note that

Table I. Costs, performance, and overheads of the most important modules, with and without countermeasures.

| Instance | Slices | | Slice Registers | | Slice LUTs | | Frequency [MHz] | |
|---|---|---|---|---|---|---|---|---|
| Round Implementation | 749 | - | 295 | - | 1224 | - | 65.66 | - |
| + *Dynamic Mapping* | 977 | +30.4% | 307 | +4.1% | 1620 | +32.4% | 60.13 | -8.4% |
| + *Dynamic Relocation (DBR+DCR)* | 829 | +10.7% | 301 | +2.0% | 1375 | +12.3% | 57.61 | -12.3% |
| + *Linear Masking* | 761 | +1.6% | 295 | +0.0% | 1239 | +1.2% | 57.71 | -12.1% |
| + All countermeasures | 1079 | +44.1% | 313 | +6.1% | 1827 | +49.3% | 57.61 | -12.3% |
| Other blocks: | | | | | | | | |
| Control logic | 72 | - | 28 | - | 140 | - | 106.45 | - |
| Key Schedule | 323 | - | 216 | - | 566 | - | 90.81 | - |
| Countermeasure manager (PRNG) | 339 | - | 395 | - | 431 | - | 128.86 | - |
| Bus Controller | 49 | - | 47 | - | 92 | - | 119.15 | - |
| Proposed AES | 5694 | - | 2255 | - | 9457 | - | 54.15 | - |

the larger design can encrypt four blocks at a time, while the simpler one can only process one block at a time.

## IV. Robustness Against EM Analyses

### A. The Setup

In order to verify the efficiency of the different countermeasures, several electromagnetic traces were acquired. The following equipment was used:

- PC providing data to the AES through the serial interface,
- Langer EM probe measuring the vertical magnetic field,
- Low noise 63db amplifier, and
- Lecroy scope to collect EM traces with a sampling rate equal to 20Gs/s.

Several sets of 300k traces were acquired. The first set included traces corresponding to AES encryptions of messages with all countermeasures disabled. It constitutes a reference set to estimate the effects of the different countermeasures on the robustness against side-channel analyses. Additional sets of EM traces correspond to AES encryptions with only one countermeasure activated at a time. Finally, the last set consists of the EM traces obtained with all countermeasures activated.

To test the improvement in terms of resistance against first order side-channel-attacks, Correlation Power Analyses (CPAs) [3] were applied to the three sets of traces. The attacks were targeting the input to the last round of the AES using a Hamming Weight model. All the 16 S-boxes were attacked and we used Guessing Entropy [23] to evaluate the efficiency of the different countermeasures proposed in this paper.

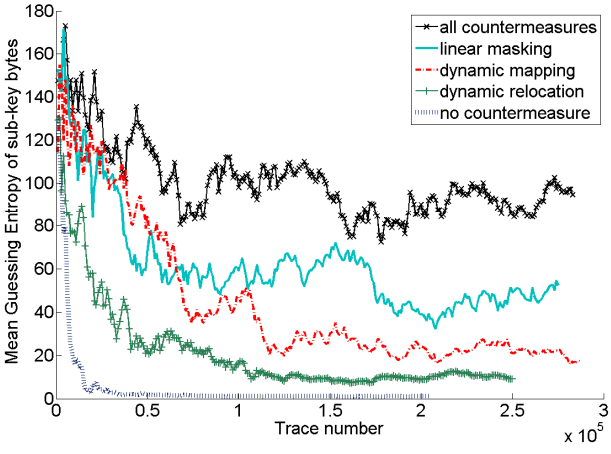


Fig. 6. Mean Guessing Entropy of correct key bytes, i.e., their average position in the ranking given by the CEMA-HW attack, as a function of the number of analyzed traces.

All performed attacks based on the Hamming Distance (HD) model were unsuccessful on this implementation. Only one column at a time is transferred in the state register, with the next column taking the place of the previous one in the state register. Thus, an attacker attempting to use the Hamming Distance Model would have to guess the first four bytes of the key in order to guess the next four ones to mount an efficient attack. This is quite tedious and was not done. Instead, we preferred to collect more traces than typically required to attack unprotected algorithms mapped on this FPGA platform, using the Hamming Weight model.

### B. The Results

The processing of the first set of EM traces (no countermeasures), allowed us to find all the 16 bytes of a sub-key with 155k traces; 15 bytes were even recovered with just 23k traces. When any countermeasure is activated, the whole set of traces allows recovering only partially the sub-key: 8 bytes with linear masking, 9 with dynamic mapping, and 12 bytes with dynamic spatial reallocation. No bytes were discovered when all the countermeasures were activated. This already provides some insights into the efficiency of the proposed countermeasures.

Figure 6 shows the evolution of the mean Guessing Entropy with the number of processed traces, i.e., the evolution of the mean position of the correct sub-key bytes among wrong guesses. A mean Guessing Entropy of 1 means that all sub-key bytes are found by the attack. When no countermeasures are activated, the mean Guessing Entropy of the 16 correct bytes of the sub-key decreases monotonically with a significant slope. When only one countermeasure is activated, we still observe a decrease in the mean Guessing Entropy, but the rate of decrease is lower. The rate depends on the specific countermeasure. After about 300k analyzed traces, the mean guessing entropy is 9 for dynamic reallocation, suggesting that the complete solution is not far from being found. The Guessing Entropy for *DynMap* is 17, showing that this countermeasure is more robust but leakage is still significant.

Table II. Robustness of the proposed countermeasures against EM and power analysis. *D*R* means both *DBR* and *DCR* are active.

| | Power | | | EM | | |
|---|---|---|---|---|---|---|
| **Design** | Key bytes found | Mean Guessing Entropy | # traces ($\times 10^3$) | Key bytes found | Mean Guessing Entropy | # traces ($\times 10^3$) |
| Simple | 15 | 1 | 205 | 16 | 1 | 155 |
| + *LynMsk* | 4 | 54 | 275 | 8 | 52 | 275 |
| + *DynMap* | 5 | 34 | 287 | 9 | 17 | 287 |
| + *D*R* | 7 | 19 | 250 | 12 | 9 | 250 |
| + *All* | 0 | 136 | 283 | 0 | 94 | 283 |

Linear masking is the most robust countermeasure in this scenario, since the guessing entropy is as high as 52. We believe that such a big difference in the robustness may be due to leakage coming from the communication bus. When linear masking is not activated, all the data transfers on the bus are in the clear and thus vulnerable to an attacker. Finally, when all countermeasures are activated we notice that the mean Guessing Entropy approaches the value 94 (and no bytes revealed!), suggesting that a very large number of EM traces is necessary to fully disclose the round key.

Power traces were also collected at the same time of EM analysis, and a correlation attack was mounted on the data. Table II shows the results and gives a direct comparison between the power and the EM attack. Even with spatial oriented countermeasures, the superiority of EM analysis is clear, as it can discover twice the number of bytes than power analysis on average with the same or lower number of traces. The mean Guessing Entropy of the correct key is also significantly higher when using EM instead of power traces.

## V. CONCLUSION

Side-channel analysis is a major threat for secure implementations of cryptographic algorithms. There are several solutions in the literature to counter such attacks, such as dual rail logic or data masking. However, when richer leakage channels are available, these countermeasures may become less effective.

In this paper, we have presented several countermeasure schemes, based on the dynamic relocation of the data within a single encryption round and between different round instances at the same time. This approach is independent of the algorithm and requires only that several instances of the same function operate in parallel (*MixColumns* or rounds in our example). This dynamic relocation has been strengthened by two masking schemes: a traditional additive masking for the linear part of the computation, and a masking scheme for the S-Boxes based on composite mapping.

Our results indicate that the complete set of proposed countermeasures provides very good resistance against EM attacks demonstrating that a very large number of traces is required in order to break the security. Masking is the most efficient countermeasure from the point of view benefits/cost and is required to protect the bus transfers; however, the proposed solutions are complementary, not alternatives, and prove to provide an effective additional layer of security. Future research will consider extending the dynamic mapping to the whole algorithm, and identifying the major source of information leakage (bus or round instances).

## REFERENCES


[1] National Institute Standards and Technology (NIST), "FIPS-197: Advanced Encryption Standard," Nov. 2001.

[2] P. C. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis," CRYPTO 1999, pp. 388-397.

[3] E. Brier, C. Clavier, F. Olivier, "Correlation Power Analysis with a Leakage Model," CHES 2004, pp.16-29.

[4] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): Measures and counter-measures for smard cards," E-smart'2001, vol. 2140 of LNCS, pp. 200-210, Springer-Verlag, 2001.

[5] S. Mangard, E. Oswald, T. Popp, "Power Analysis Attacks - Revealing the Secrets of Smart Cards," ISBN 978-0-387-30857-9, Springer, 2007.

[6] J.-S. Coron and I. Kizhvatov, "An efficient method for random delay generation in embedded software," CHES, vol. 5747 of Lecture Notes in Computer Science, pp. 156-170, Springer, 2009.

[7] T.S. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software," CHES, vol. 1965 of LNCS, pp. 238-251, 2001.

[8] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," DATE 2004, pp. 246-251, Feb. 2004.

[9] T. Popp, S. Mangard, "Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints," CHES 2005, pp. 172-186.

[10] N. Pramstaller, S. Mangard, S. Dominikus, and J. Wolkerstorfer, "Efficient AES Implementations on ASICs and FPGAs," Fourth Int'l Conf. Advanced Encryption Standard (AES '04), pp. 98-112, 2004.

[11] D. May, H.L. Muller, and N.P. Smart, "Random Register Renaming to Foil DPA," CHES 2001, LNCS 2162, pp. 28-38, 2001.

[12] F. Poucheret, L. Barthe, P. Benoit, L. Torres, P. Maurine, M. Robert, "Spatial EM jamming: A countermeasure against EM Analysis?" VLSI-SoC 2010, pp. 105-110.

[13] A. Satoh, S. Morioka, K. Takano, S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," ASIACRYPT 2001, pp. 239-254.

[14] Xilinx, "Spartan-3 FPGA Family Data Sheet," 2009.

[15] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, P. Rohatgi, "Efficient Rijndael Encryption Implementation with Composite Field Arithmetic," CHES 2001, pp. 171-184.

[16] E. Barkan and E. Biham. In How Many Ways Can You Write Rijndael? In ASIACRYPT 2002, vol. 2501 of LNCS, pp. 160–175. Springer, 2002.

[17] H. Raddum. More Dual Rijndaels. In AES Conference 2004, volume 3373 of LNCS, pages 142–147. Springer, 2004.

[18] C. Paar, "Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields," Dissertation, Institute for Experimental Mathematics, Universität Essen, Germany, 1994.

[19] M. Ciet, M. Neve, E. Peeters, and J.-J. Quisquater, "Parallel FPGA implementation of RSA with residue number systems – can side-channel threats be avoided," MWSCAS 2003, 2003.

[20] J. D. Goliç and C. Tymen, "Multiplicative Masking and Power Analysis of AES," CHES 2002, Springer, pp. 198–212, 2003.

[21] C. Paar, J. Pelzl, and B. Preneel, "Understanding Cryptography: A Textbook for Students and Practitioners," Chapter 2, ISBN 978-3-642-04100-6, Springer, 2010.

[22] National Institute Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation," Special Publ. 800-38A, 2001.

[23] F.-X. Standaert, T. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," EUROCRYPT, vol. 5479 of LNCS, page 443-461. Springer, 2009.