Published in *IET Information Security* **1**(3):106–110, 2007.

Strengthening Hardware AES Implementations against Fault Attacks

Marc Joye [1], Pascal Manet [2] and Jean-Baptiste Rigaud [3]

[1] *Thomson R&D France, Technology Group, Corporate Research, Security Laboratory*

1 avenue Belle Fontaine, 35576 Cesson-Sévigné Cedex, France

marc.joye@thomson.net

[2] *CEA-LETI, SESAM Laboratory*

Avenue des anémones, 13541 Gardanne, France

pascal.manet@cea.fr

[3] *Ecole des Mines de St Etienne, Centre Microélectronique de Provence*

Avenue des anémones, 13541 Gardanne, France

rigaud@emse.fr

***Abstract*—Differential fault attacks become a threat of increasing importance against cryptographic devices. One of the most efficient hardware countermeasures for block ciphers to prevent such attacks relies on duplication. In this paper, we propose novel techniques to implement a duplication scheme for the AES. Remarkably, our implementation techniques do not impact the ratio throughput/area and better withstand a large variety of known fault attacks.**

***Index Terms*—AES, Rijndael, differential fault attacks, hardware implementation, duplication.**

I. Introduction

The Rijndael block cipher [1] was standardized as the Advanced Encryption Standard (AES) by NIST in 2001 [2]. It replaces the Data Encryption Standard (DES) for symmetric-key encryption.

In addition of being resistant against cryptanalysis, block ciphers should also be resistant against implementation attacks, including side-channel and fault attacks. Fault attacks were developed by Boneh, DeMillo and Lipton [3] and extended to the symmetric-key setting by Biham and Shamir [4]. The principle behind fault attacks consists in modifying the normal behavior of a cryptographic device in order to get a faulty ciphertext. Then from several faulty ciphertexts, the attacker tries to infer some information about the secret key (see e.g. [5] for practical ways to implement fault attacks). When successful, those attacks are very powerful; for example, applied to the AES, only two pairs of correct/faulty ciphertexts suffice to recover the whole secret key [6].

Of course, numerous countermeasures have been proposed to avoid, prevent or detect fault attacks. We refer the reader to excellent survey paper [7] for a thorough overview of known strategies, with an emphasis on AES, to thwart fault attacks.

In this paper, we propose an hardware AES implementation the cost of which is exactly the cost of duplication as both the data path and the key path are implemented twice. According to [7], this seems to be the minimal price to pay for state-of-the-art countermeasures. Furthermore, duplication presents the advantage of covering permanent (i.e., hardware) faults. The implementation techniques we suggest are completely free: the ratio throughput/area is *unchanged* compared to a straightforward duplicated implementation yet the success probability of an attacker is much lower for a large variety of attacks. Moreover, in a contrast with a straightforward duplicated implementation, the security is not completely jeopardized if a final comparison is bypassed.

We note that our strengthening techniques are simple to implement. They are not restricted to AES and may apply to other cryptosystems as well. They are reminiscent of the infective computation notion, introduced by Yen *et al.* [8] in the context of RSA cryptosystem.

The rest of this paper is organized as follows. In the next section, we describe the AES cryptosystem and introduce the notations used throughout the paper. In Section III, we review known faults attacks against AES. In Section IV, we present our methodology to strengthen hardware AES implementations. Finally, we conclude in Section V.

II. Description of AES

We briefly review the AES block-cipher [2] (see also [1]). The AES supports keys of 128, 192 or 256 bits and consists of 10, 12 or 14 rounds, respectively. The block length is of 128 bits.

An AES encryption consists of an initial `AddRoundKey` operation (namely, a bit-wise XOR), followed by N rounds (10 for this paper). Round r, with $1 \leq r \leq N$, takes on input a 128-bit state $S^{[r-1]}$ and a 128-bit round key $K^{[r]}$, and outputs a 128-bit state $S^{[r]}$. The plaintext is $M \in \{0,1\}^{128}$ and the ciphertext is $C \in \{0,1\}^{128}$. More specifically, we have:

$$\begin{cases} S^{[0]} = M \oplus K^{[0]}\,, \\ S^{[r]} = \big(\texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}(S^{[r-1]})\big) \\ \qquad\qquad \oplus K^{[r]} \text{ for } 1 \leq r \leq N-1\,, \\ C = \big(\texttt{ShiftRows} \circ \texttt{SubBytes}(S^{[N-1]})\big) \oplus K^{[N]}\,. \end{cases}$$

(Note that there is no `MixColumns` operation in the last round.)

Each state and the intermediate computations are viewed as a (4×4)-matrix of bytes:

$$\big(s_{i,j}^{[r]}\big)_{\substack{0\le i\le 3\\ 0\le j\le 3}} \ .$$

With this representation, the different transformations operate on each byte as follows.

- `SubBytes()`: This transformation substitutes each byte, $s_{i,j}^{[r]}$, of the state by another byte through a non-linear permutation S-box $\mathsf{S_{RD}}$:

$$\texttt{SubBytes} : s_{i,j}^{[r]} \longleftarrow \mathsf{S_{RD}}(s_{i,j}^{[r]}) \ .$$

- `ShiftRows()`: This transformation cyclically shifts row i, $0 \le i \le 3$, over i bytes to the left:

$$\texttt{ShiftRows} : s_{i,j}^{[r]} \longleftarrow s_{i,(j+i) \bmod 4}^{[r]} \ .$$

- `MixColumns()`: This transformation mixes column j, $0 \le j \le 3$, by considering it as a polynomial over $\mathrm{GF}(2^8)$, say $a(x)$, and by multiplying $a(x)$ modulo (x^4+1) with fixed polynomial $\mathsf{c}(x) = \sum_{0\le l\le 3} \mathsf{c}_l\, x^l := \mathtt{02} + \mathtt{01}\, x + \mathtt{01}\, x^2 + \mathtt{03}\, x^3$:

$$\texttt{MixColumns} : s_{i,j}^{[r]} \longleftarrow \bigoplus_{0\le l\le 3} s_{(i-l) \bmod 4,j}^{[r]} \odot \mathsf{c}_l \ ,$$

  where $\odot$ denotes the multiplication in $\mathrm{GF}(2^8)$ represented as $\mathrm{GF}(2^8) \cong \mathrm{GF}(2)[x]/(x^8+x^4+x^3+x+1)$.

We refer the reader to [2] for a description of the key schedule.

III. Differential Fault Attacks

In this section, we review known differential fault attacks against AES (see [9], [10] for surveys). We make the distinction between attacks occurring at the bit level and at the byte level. However, we do not consider the so-called safe-error attacks (originaly [11] on RSA, [12] on AES) like bit-forcing attacks.

We use the notations of the previous section. Faulty values are indicated with an hat symbol (e.g., if an error is induced on x, we denote by $\hat{x}$ the resulting faulty value.).

A. Single-Bit Errors

In [13], Giraud considers that a *single bit of error* is induced in $S^{[N-1]}$. We let $\widehat{C}$ denote the corresponding corrupt ciphertext. If

$$\hat{s}_{i,j}^{[N-1]} = s_{i,j}^{[N-1]} \oplus \epsilon \ \text{ where } \epsilon = \mathsf{RC}(t) \text{ and } 0 \le t \le 7 \,,$$

that is, if a single bit of byte (i,j) is modified at the output of round $(N-1)$ then it is not difficult to see that the unique non-zero byte of $C \oplus \widehat{C}$ is located in row i and column $(j-i) \bmod 4$. Hence, we get

$$\begin{aligned}\Delta &:= (C \oplus \widehat{C})_{i,(j-i) \bmod 4} = s_{i,(j-i) \bmod 4}^{[N]} \oplus \hat{s}_{i,(j-i) \bmod 4}^{[N]} \\ &= \mathsf{S_{RD}}(s_{i,j}^{[N-1]}) \oplus \mathsf{S_{RD}}(s_{i,j}^{[N-1]} \oplus \epsilon) \ . \end{aligned} \tag{1}$$

The attacker then guesses the value of $\epsilon = \mathsf{RC}(t)$ and finds the set of possible values for $s_{i,j}^{[N-1]}$ verifying Eq. (1). With a few single-bit errors, the attacker can retrieve the value of round-key byte $k_{i,(j-i) \bmod 4}^{[N]}$ as

$$k_{i,(j-i) \bmod 4}^{[N]} = s_{i,(j-i) \bmod 4}^{[N]} \oplus \mathsf{S_{RD}}(s_{i,j}^{[N-1]}) \ .$$

With sufficiently many single-bit faults, the attacker can thus recover the whole round-key $K^{[N]}$ and hence the secret key by reversing the key scheduling.

B. Single-Byte Errors

A weaker fault model is to assume that a fault results in the modification of a *single byte* [14], [15], [13], [6]. We refer the reader to [14], [13] for attacks against the key schedule. We do not consider the case of a fault reducing the number of AES rounds [16]; see [17].

We follow the presentation of [6]. Suppose that a single-byte fault is induced before the `MixColumns` of round $(N-1)$, for example in $\texttt{SubBytes}(S^{[N-2]})$, say at position (i,j). Again, we let $(C, \widehat{C})$ denote the corresponding pair of correct/faulty ciphertexts. We have:

$$\widehat{\mathsf{S_{RD}}}(s_{i,j}^{[N-2]}) = \mathsf{S_{RD}}(s_{i,j}^{[N-2]}) \oplus \epsilon \ \text{ with } \epsilon \in \mathrm{GF}(2^8) \setminus \{0\} \ .$$

Such a fault affects 4 bytes in the output ciphertext. The 4 non-zero bytes of $C \oplus \widehat{C}$ are located in

$$\begin{gathered}(0,d),\ (1,(d-1) \bmod 4),\\ (2,(d-2) \bmod 4),\ (3,(d-3) \bmod 4)\end{gathered}$$

where $d := (j-i) \bmod 4$. Observe that they are located in the same column before the last `ShiftRows`.

The attacker prepares a list, say $\mathcal{L}_d$, containing $255 \cdot 4 = 1020$ possible differences at the output of `MixColumns` of round $(N-1)$. Next, the attacker makes a guess on the 4 round-key bytes, $(k_{0,d}^{[N]}, k_{1,(d-1) \bmod 4}^{[N]}, k_{2,(d-2) \bmod}^{[N]}, k_{3,(d-3) \bmod 4}^{[N]})$, and checks whether

$$\begin{aligned}\Delta := \texttt{SubBytes}^{-1}\big((C \oplus K^{[N]})_{*,d}\big) \oplus \\ \texttt{SubBytes}^{-1}\big((\widehat{C} \oplus K^{[N]})_{*,d}\big)\end{aligned} \tag{2}$$

is an element of $\mathcal{L}_d$ — and if so, the 4 round-key bytes is a possible candidate. This attack requires on average 8 pairs of correct/faulty ciphertexts (2 for each column) to recover the whole secret key.

This can be reduced to only *2 pairs* by inducing faults before the `MixColumns` of round $(N-2)$ because then a corrupt byte results in 16 faulty bytes in the final output. See [6] for details.

IV. Our Hardware Implementation Techniques

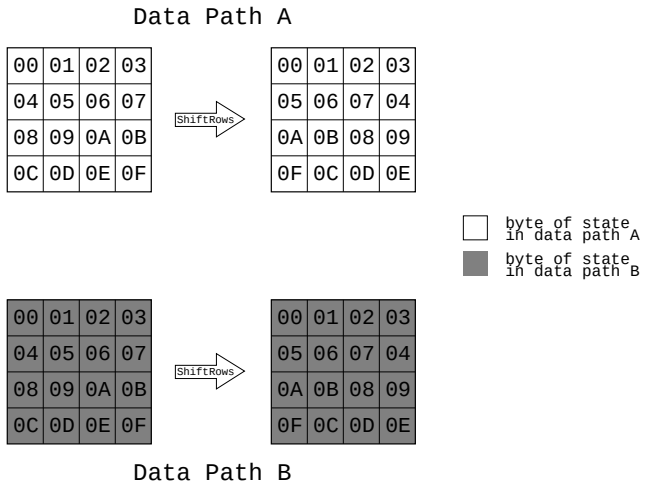
A number of hardware countermeasures have been proposed to protect AES implementations against faults. According to a recent comparative study [7], the cost of robust (hardware) AES implementations is close to duplication. To this end, we devise in this section simple methods which increase the resistance of duplicated AES implementations against faults but do not impact their cost.

A. Description

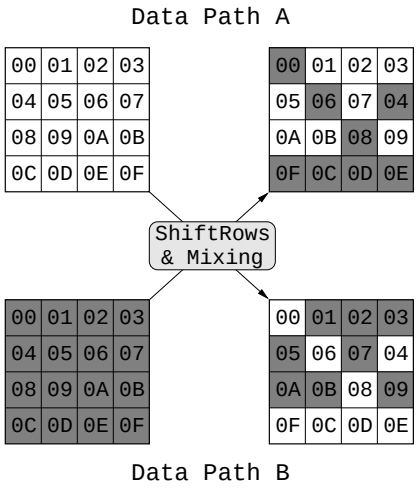
As exemplified in Section III, the attacker needs some knowledge on the fault propagation to succeed in recovering information on the secret key. Intuitively, our methodology consists in changing this propagation path. We present hereafter different realizations.

A.1 Basic approach

In a duplicated AES hardware implementation, both the data path and the key path are duplicated so that two executions of the AES algorithm run in parallel. We exploit this property to scramble the state bytes between the two executions. Doing so, we ensure that a fault on one data path will likely result in a fault on the other data path. The expected fault-propagation path being modified, it is harder to mount an attack.



(a) Data path duplication



(b) Scrambling bytes

Fig. 1. AES `ShiftRows`.

A natural place to perform this scrambling is during the `ShiftRows` transformation as it can already be considered as a kind of byte scrambling — but on the same data path (i.e., data path A and data path B on Fig. 1-a). It is however worth noting that this technique can easily be implemented at other places as well (including in the key schedule). An application of our basic approach to the `ShiftRows` transformation is depicted on Fig. 1-b. We see that certain state bytes on data path A are swapped with their corresponding state bytes on data path B.

This scrambling technique prevents some attacks (see § III-B) and does not change a correct execution.

A.2 Better approach

The only requirement of the byte scrambling in the basic approach was to keep a non-faulty execution safe. The scrambling can also be done at the bit level. If no fault occurs, everything is unchanged; otherwise, individual bits *inside* the byte states may be affected. Again, this is illustrated for the `ShiftRows` transformation on the next figure.

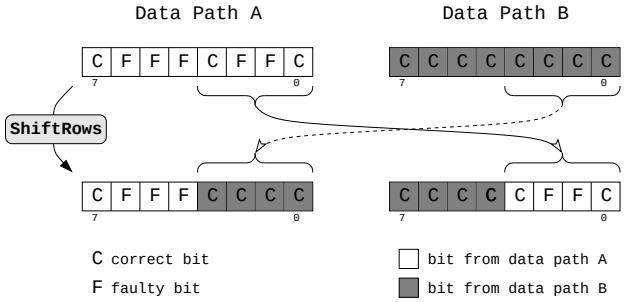


Fig. 2. Bit-level scrambling in `ShiftRows`.

This bit-oriented scrambling has a lot of possible realizations. There are 2^{128} combinations for an AES state since each bit can be swapped or not with the corresponding bit in the other path. Actually, in our FPGA implementation (see § IV-C), we only used a set of four bits for each byte to get a balanced "diffusion", which yields $\binom{8}{4}^{16} \approx 2^{98}$ possible scramblings for an AES state.

Unlike the byte-oriented scrambling, this technique cannot easily be transposed to a software implementation. However, it is fairly easy to emulate it. For example, if the state bytes of data paths A and B are respectively s_{A} and s_{B}, they can be replaced with $s_{\mathrm{A}} \leftarrow s_{\mathrm{A}} \oplus s_{\mathrm{AB}}$ and $s_{\mathrm{B}} \leftarrow s_{\mathrm{B}} \oplus s_{\mathrm{AB}}$ where $s_{\mathrm{AB}} := C \odot (s_{\mathrm{A}} \oplus s_{\mathrm{B}})$ for a byte constant C.

B. Security Analysis

We now evaluate the security of our techniques regarding the attacks described in Section III. Other attacks are treated analogously. For the sake of concreteness, we consider a duplicated AES implementation where *only* the inputs of `ShiftRows` are scrambled byte-wise or bit-wise, in a balanced way. Note that in an actual AES implementation, mounting successful fault attacks is *much harder* as scramblings are applied at other locations, including in the key schedule.

We stress that in the case of an implementation setting a flag when an error is detected, the attacker should in addition be able to bypass this detection to mount the attacks described below.

B.1 Byte-wise scrambling

An inherent limitation of the byte-wise approach is underlined by the flipping-bit attack of § III-A. In this attack, the fault occurs at the beginning of the last round and only affects

one byte at a time. After ShiftRows, either the faulty byte remains unchanged or it is swapped with a correct one (only half of the key is recovered by targeting a single data path).

Concerning the single-byte attack (cf. § III-B) against a byte-wise scrambled implementation, the corrupt data path will not yield the expected ciphertext. However, the attack can be extended at the expense of much more fault inductions and data processing (see the next paragraph for a similar extension).

B.2 Bit-wise scrambling

We will see in this paragraph that the bit-wise approach leads to a more robust implementation. Further, from a hardware viewpoint, it is as easy to implement and does not increase the complexity. This is the scrambling technique we recommend.

Given difference Δ, the flipping-bit attack requires a unique solution $s_{i,j}^{[N-1]}$ to Eq. (1) for some $\epsilon = \mathsf{RC}(t)$ with $0 \leq t \leq 7$ (cf. § III-A). If, after SubBytes, the faulty byte $\hat{s}_{i,j}^{[N-1]}$ is scrambled, Eq. (1) will no longer be satisfied; it will only be correct truncated to the four non-swapped bits (remember that we assume a balanced scrambling). It remains for the attacker to recover for *each* byte state the location of the non-swapped bits. This can be achieved by inspection after a few experiments.

We conducted a series of simulations to evaluate the resistance of our implementation against this extended attack. This was done by software. Contrary to [13], we considered random — not necessarily different — faults when evaluating the success probability.* The results are summarized in Table I. *Suc* denotes the success probability of Giraud's attack and *#k* denotes the average number of wrong candidates for a round-key byte.

TABLE I
RESISTANCE AGAINST GIRAUD'S FLIPPING-BIT ATTACK.

	1 fault		2 faults		4 faults		8 faults		16 faults	
	Suc	*#k*	*Suc*	*#k*	*Suc*	*#k*	*Suc*	*#k*	*Suc*	*#k*
Straightfwd impl.	0%	8	71%	1	97%	0	—	—	—	—
Our implementation	0%	105	0%	52	0%	17	33%	3	71%	1

We see for example that the effort to recover a state byte with a success probability of 71% is at least 8 times more demanding than in the original attack.

For the single-byte attack (cf. § III-B), it is seemingly not possible to extend it as the attacker should know how the fault propagates through the device. It appears that the only way to mount a successful attack would require to timely induce the *same* faults in both data paths. Such a security model is nowadays considered as unrealistic.

C. FPGA Implementation

We used the Xilinx ISE framework to design our protected AES. The synthesis was performed with XST application and all simulations (functional, post-synthesis and post-place and route) with Modelsim. The FPGA target was XCV2000E from Xilinx *Virtex-E*™ family. This FPGA was embedded on an Integrator/LM XCV600E+ from ARM Ltd.

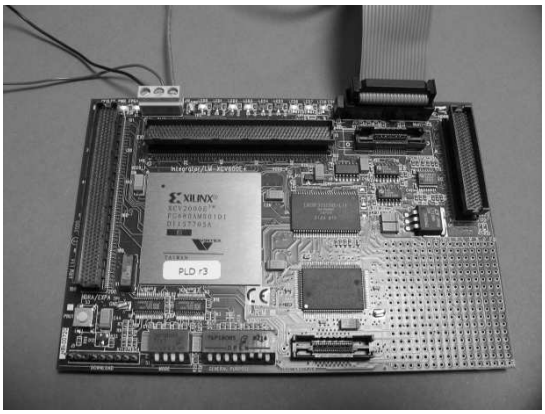


Fig. 3. View of the demonstrator.

C.1 Hardware description

We found an interesting AES structure on opencores website. This implementation makes use of functions to resolve each step of an AES round. We kept the global architecture but implemented each round transformations with dataflow description. Each round is computed within a clock cycle. The S-boxes (SubBytes transformations) are internal dual-port RAM blocks.

Implementing a duplicated path version of this AES was very convenient. The round execution is described using one component for each transformation, so duplication of the path in our case just means duplication of the components. Following our methodology, we replaced the two ShiftRows by a common box with two states for the inputs and outputs. As we described ShiftRows in a material way, we only had to use a bit-wise instead of a byte-wise affectation.

C.2 Performances

The implementation of our unprotected, non-duplicated AES takes 1005 slices for a 60 MHz clock frequency. The unprotected AES also uses ten dual-port RAMs: eight for the 16 datapath S-boxes and two for the 4 key schedules.

The protected version using our strengthening techniques requires 1600 slices† and does not change the throughput. The RAM blocks are doubled, so 20 blocks are used. In comparison with a straightforward implementation of a duplicated AES, our implementation techniques do not oversize the area or slow down the execution. As a result, the global cost of the protection is only the cost of duplication in area.

C.3 Further results

Our implementation was optimized neither for speed nor for area. However as our techniques merely cross wires, they should readily extend to any duplicated implementation without overhead.

* For example, three random single-bit faults yields a success probability of 91% within our model, whereas three *different* single-bit faults yields a success probability of 97% [13].

† Note that control and i/o interfaces are not duplicated.

The duplication of the data path increases the power signature and thus may render DPA attacks [18] easier. We tested a complemented duplicated data-path and key expander to have a '1' used for each '0' and vice versa. This DPA countermeasure induces an inverter on each crossed wire, implying a gate overhead of 128 inverters.

V. Concluding Remarks

In this paper, we presented techniques for strengthening the resistance of AES hardware implementations when implemented using duplication. Our techniques are easy to implement and give rise to no overhead in both the area and the throughput. We also gave a security analysis and presented our concrete FPGA implementation.

Our methodology consists in scrambling byte states between the two executions, in *different ways* (byte-wise or bit-wise) and at *different locations*. The next step is to perform the scrambling in a random way. We will evaluate the cost of this extension in a future work.

Acknowledgments

The second and third authors are supported in part by FSE and DGE through the CIMPACA/Micro-PackS BTRS Project.

References

[1] J. Daemen and V. Rijmen, *The Design of Rijndael*. Springer, 2002.

[2] National Institute of Standards and Technology (NIST), "Announcing the Advanced Encryption Standard (AES)," Federal Information Processing Standards Publication, n. 197, Nov. 26, 2001.

[3] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *Advances in Cryptology — EUROCRYPT '97*, ser. Lecture Notes in Computer Science, vol. 1233. Springer, 1997, pp. 37–51.

[4] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology — CRYPTO '97*, ser. Lecture Notes in Computer Science, vol. 1294. Springer, 1997, pp. 513–525.

[5] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems — CHES 2002*, ser. Lecture Notes in Computer Science, vol. 2523. Springer, 2002, pp. 2–12.

[6] G. Piret and J.-J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and Khazad," in *Cryptographic Hardware and Embedded Systems — CHES 2003*, ser. Lecture Notes in Computer Science, vol. 2779. Springer, 2003, pp. 77–88.

[7] T. G. Malkin, F.-X. Standaert, and M. Yung, "A comparative cost/security analysis of fault attack countermeasures," in *Second Workshop on Fault Detection and Tolerance in Cryptography*, Edinburgh, UK, Sept. 2, 2005, pp. 109–123.

[8] S.-M. Yen, S. Moon, and J.-C. Ha, "Hardware fault attacks on RSA with CRT revisited," in *Information Security and Cryptology — ICISC 2002*, ser. Lecture Notes in Computer Science, vol. 2587. Springer, 2003, pp. 374–388.

[9] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," in *First Workshop on Fault Detection and Tolerance in Cryptography (FDTC 2004)*, Florence, Italy, June 1, 2004.

[10] C. Giraud and H. Thiebeauld, "A survey on fault attacks," in *Smart Card Research and Advanced Applications IV*. Kluwer Academic Publishers, 2004, pp. 159–176.

[11] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *IEEE Transactions on Computers*, 49(9):967–970, 2000.

[12] J. Blömer and J.-P. Seifert, "Fault based cryptanalysis of the Advanced Encryption Standard (AES)," in *Financial Cryptography — FC 2003*, ser. Lecture Notes in Computer Science, vol. 2742. Springer, 2003, pp. 162–181.

[13] C. Giraud, "DFA on AES," in *Advanced Encryption Standard — AES*, ser. Lecture Notes in Computer Science, vol. 3373. Springer, 2005, pp. 27–41.

[14] C.-N. Chen and S.-M. Yen, "Differential fault analysis on AES key schedule and some countermeasures," in *Information Security and Privacy — ACISP 2003*, ser. Lecture Notes in Computer Science, vol. 2727. Springer, 2003, pp. 118–129.

[15] P. Dusart, G. Letourneux, and O. Vivilo, "Differential fault analysis on A.E.S." in *Applied Cryptography and Network Security — ACNS 2003*, ser. Lecture Notes in Computer Science, vol. 2846. Springer, 2003, pp. 293–306.

[16] H. Choukri and M. Tunstall, "Round reduction using faults," in *Second Workshop on Fault Detection and Tolerance in Cryptography (FDTC 2005)*, Edinburgh, UK, Sept. 2, 2005.

[17] G. Gaubatz, E. Savaş, and B. Sunar, "Sequential circuit design for embedded cryptographic applications resilient to adversarial faults," *IEEE Transactions on Computers*, to appear.

[18] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO '99*, ser. Lecture Notes in Computer Science, vol. 1666. Springer, 1999, pp. 388–397.